

Connectors and APIs

Abstract

This manual describes the Connectors and APIs that can be used with MySQL.

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

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

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

Licensing information. This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL Connectors, see [this document](#) for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL Connectors, see [this document](#) for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Document generated on: 2016-08-18 (revision: 48596)

Table of Contents

Preface and Legal Notices	ix
1 Introduction	1
2 MySQL Connector/J 5.1 Developer Guide	3
2.1 Overview of MySQL Connector/J	3
2.2 Connector/J Versions	4
2.2.1 Connector/J Release Notes and Change History	5
2.2.2 Java Versions Supported	5
2.3 Connector/J Installation	5
2.3.1 Installing Connector/J from a Binary Distribution	6
2.3.2 Installing the Driver and Configuring the <code>CLASSPATH</code>	6
2.3.3 Upgrading from an Older Version	7
2.3.4 Installing from Source	9
2.3.5 Testing Connector/J	12
2.4 Connector/J Examples	14
2.5 Connector/J (JDBC) Reference	14
2.5.1 Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J	15
2.5.2 JDBC API Implementation Notes	48
2.5.3 Java, JDBC and MySQL Types	51
2.5.4 Using Character Sets and Unicode	53
2.5.5 Connecting Securely Using SSL	55
2.5.6 Connecting Using PAM Authentication	57
2.5.7 Using Master/Slave Replication with <code>ReplicationConnection</code>	57
2.5.8 Mapping MySQL Error Numbers to JDBC <code>SQLState</code> Codes	58
2.6 JDBC Concepts	64
2.6.1 Connecting to MySQL Using the JDBC <code>DriverManager</code> Interface	64
2.6.2 Using JDBC <code>Statement</code> Objects to Execute SQL	65
2.6.3 Using JDBC <code>CallableStatements</code> to Execute Stored Procedures	66
2.6.4 Retrieving <code>AUTO_INCREMENT</code> Column Values through JDBC	68
2.7 Connection Pooling with Connector/J	71
2.8 Multi-Host Connections	74
2.8.1 Configuring Server Failover	75
2.8.2 Configuring Load Balancing with Connector/J	77
2.8.3 Configuring Master/Slave Replication with Connector/J	79
2.8.4 Advanced Load-balancing and Failover Configuration	83
2.9 Using the Connector/J Interceptor Classes	84
2.10 Using Connector/J with Tomcat	85
2.11 Using Connector/J with JBoss	86
2.12 Using Connector/J with Spring	87
2.12.1 Using <code>JdbcTemplate</code>	88
2.12.2 Transactional JDBC Access	89
2.12.3 Connection Pooling with Spring	91
2.13 Using Connector/J with GlassFish	91
2.13.1 A Simple JSP Application with GlassFish, Connector/J and MySQL	93
2.13.2 A Simple Servlet with GlassFish, Connector/J and MySQL	95
2.14 Troubleshooting Connector/J Applications	97
2.15 Known Issues and Limitations	104
2.16 Connector/J Support	105
2.16.1 Connector/J Community Support	105
2.16.2 How to Report Connector/J Bugs or Problems	105
3 MySQL Connector/ODBC Developer Guide	107
3.1 Introduction to MySQL Connector/ODBC	107
3.2 Connector/ODBC Versions	108
3.3 General Information About ODBC and Connector/ODBC	109
3.3.1 Connector/ODBC Architecture	110

3.3.2 ODBC Driver Managers	111
3.4 Connector/ODBC Installation	112
3.4.1 Installing Connector/ODBC on Windows	113
3.4.2 Installing Connector/ODBC on Unix-like Systems	118
3.4.3 Installing Connector/ODBC on OS X	120
3.4.4 Building Connector/ODBC from a Source Distribution on Windows	121
3.4.5 Building Connector/ODBC from a Source Distribution on Unix	122
3.4.6 Building Connector/ODBC from a Source Distribution on OS X	124
3.4.7 Installing Connector/ODBC from the Development Source Tree	125
3.5 Configuring Connector/ODBC	125
3.5.1 Overview of Connector/ODBC Data Source Names	125
3.5.2 Connector/ODBC Connection Parameters	126
3.5.3 Configuring a Connector/ODBC DSN on Windows	133
3.5.4 Configuring a Connector/ODBC DSN on OS X	139
3.5.5 Configuring a Connector/ODBC DSN on Unix	142
3.5.6 Connecting Without a Predefined DSN	143
3.5.7 ODBC Connection Pooling	143
3.5.8 Getting an ODBC Trace File	144
3.6 Connector/ODBC Examples	146
3.6.1 Basic Connector/ODBC Application Steps	146
3.6.2 Step-by-step Guide to Connecting to a MySQL Database through Connector/ ODBC	148
3.6.3 Connector/ODBC and Third-Party ODBC Tools	148
3.6.4 Using Connector/ODBC with Microsoft Access	149
3.6.5 Using Connector/ODBC with Microsoft Word or Excel	155
3.6.6 Using Connector/ODBC with Crystal Reports	158
3.6.7 Connector/ODBC Programming	164
3.7 Connector/ODBC Reference	171
3.7.1 Connector/ODBC API Reference	171
3.7.2 Connector/ODBC Data Types	175
3.7.3 Connector/ODBC Error Codes	176
3.8 Connector/ODBC Notes and Tips	177
3.8.1 Connector/ODBC General Functionality	177
3.8.2 Connector/ODBC Application-Specific Tips	178
3.8.3 Connector/ODBC Errors and Resolutions (FAQ)	183
3.9 Connector/ODBC Support	188
3.9.1 Connector/ODBC Community Support	188
3.9.2 How to Report Connector/ODBC Problems or Bugs	188
3.9.3 How to Submit a Connector/ODBC Patch	189
4 MySQL Connector/Net Developer Guide	191
4.1 Introduction to MySQL Connector/Net	191
4.2 Connector/Net Versions	192
4.3 Connector/Net Installation	195
4.3.1 Installing Connector/Net on Windows	195
4.3.2 Installing Connector/Net on Unix with Mono	198
4.3.3 Installing Connector/Net from the Source Code	199
4.4 Connector/Net Tutorials	200
4.4.1 Tutorial: An Introduction to Connector/Net Programming	200
4.4.2 Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider	209
4.4.3 Tutorial: MySQL Connector/Net ASP.NET Profile Provider	214
4.4.4 Tutorial: Web Parts Personalization Provider	216
4.4.5 Tutorial: Simple Membership Web Provider	220
4.4.6 Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source	226
4.4.7 Tutorial: Databinding in ASP.NET Using LINQ on Entities	238
4.4.8 Tutorial: Using SSL with MySQL Connector/Net	242
4.4.9 Tutorial: Using MySqlCommand	244
4.4.10 Tutorial: Generating MySQL DDL from an Entity Framework Model	248
4.5 Connector/Net Programming	248

4.5.1	Connecting to MySQL Using Connector/Net	249
4.5.2	Creating a Connector/Net Connection String	249
4.5.3	Using MySqlCommand	253
4.5.4	Using Connector/Net with Connection Pooling	254
4.5.5	Using the Windows Native Authentication Plugin	255
4.5.6	Writing a Custom Authentication Plugin	255
4.5.7	MySQL Fabric Support	259
4.5.8	Using Connector/Net with Table Caching	264
4.5.9	Using the Connector/Net with Prepared Statements	264
4.5.10	Accessing Stored Procedures with Connector/Net	265
4.5.11	Handling BLOB Data With Connector/Net	268
4.5.12	Asynchronous methods	271
4.5.13	Using the Connector/Net Interceptor Classes	277
4.5.14	Handling Date and Time Information in Connector/Net	279
4.5.15	Using the MySqlBulkLoader Class	280
4.5.16	Using the MySQL Connector/Net Trace Source Object	281
4.5.17	Binary/Nonbinary Issues	286
4.5.18	Character Set Considerations for Connector/Net	287
4.5.19	Using Connector/Net with Crystal Reports	287
4.5.20	ASP.NET Provider Model	291
4.5.21	Working with Partial Trust / Medium Trust	293
4.6	Connector/Net Connection String Options Reference	296
4.7	Connector/Net Support for Windows Store	303
4.8	EF5 Support	304
4.9	EF6 Support	307
4.10	Connector/Net API Reference	312
4.10.1	MySql.Data.MySqlClient Namespace	312
4.10.2	MySql.Data.Types Namespace	410
4.11	Connector/Net Support	453
4.11.1	Connector/Net Community Support	453
4.11.2	How to Report Connector/Net Problems or Bugs	453
5	MySQL Connector/Python Developer Guide	455
5.1	Introduction to MySQL Connector/Python	455
5.2	Guidelines for Python Developers	456
5.3	Connector/Python Versions	457
5.4	Connector/Python Installation	458
5.4.1	Obtaining Connector/Python	459
5.4.2	Installing Connector/Python from a Binary Distribution	459
5.4.3	Installing Connector/Python from a Source Distribution	460
5.4.4	Verifying Your Connector/Python Installation	462
5.5	Connector/Python Coding Examples	462
5.5.1	Connecting to MySQL Using Connector/Python	462
5.5.2	Creating Tables Using Connector/Python	464
5.5.3	Inserting Data Using Connector/Python	466
5.5.4	Querying Data Using Connector/Python	467
5.6	Connector/Python Tutorials	468
5.6.1	Tutorial: Raise Employee's Salary Using a Buffered Cursor	468
5.7	Connector/Python Connection Establishment	469
5.7.1	Connector/Python Connection Arguments	469
5.7.2	Connector/Python Option-File Support	474
5.8	Connector/Python Other Topics	476
5.8.1	Connector/Python Connection Pooling	476
5.8.2	Connector/Python Fabric Support	478
5.8.3	Connector/Python Django Backend	478
5.9	Connector/Python API Reference	479
5.9.1	mysql.connector Module	479
5.9.2	connection.MySQLConnection Class	480
5.9.3	pooling.MySQLConnectionPool Class	492

5.9.4 pooling.PooledMySQLConnection Class	494
5.9.5 cursor.MySQLCursor Class	495
5.9.6 Subclasses cursor.MySQLCursor	503
5.9.7 constants.ClientFlag Class	506
5.9.8 constants.FieldType Class	507
5.9.9 constants.SQLMode Class	507
5.9.10 constants.CharacterSet Class	507
5.9.11 constants.RefreshOption Class	507
5.9.12 Errors and Exceptions	508
6 MySQL and PHP	513
6.1 Introduction to the MySQL PHP API	514
6.2 Overview of the MySQL PHP drivers	515
6.2.1 Introduction	515
6.2.2 Terminology overview	515
6.2.3 Choosing an API	516
6.2.4 Choosing a library	518
6.2.5 Concepts	519
6.3 MySQL Improved Extension	522
6.3.1 Overview	522
6.3.2 Quick start guide	525
6.3.3 Installing/Configuring	547
6.3.4 The mysqli Extension and Persistent Connections	550
6.3.5 Predefined Constants	551
6.3.6 Notes	554
6.3.7 The MySQLi Extension Function Summary	554
6.3.8 Examples	561
6.3.9 The mysqli class	562
6.3.10 The mysqli_stmt class	653
6.3.11 The mysqli_result class	693
6.3.12 The mysqli_driver class	721
6.3.13 The mysqli_warning class	725
6.3.14 The mysqli_sql_exception class	726
6.3.15 Aliases and deprecated Mysqli Functions	727
6.3.16 Changelog	735
6.4 MySQL Functions (PDO_MYSQL)	735
6.4.1 PDO_MYSQL DSN	738
6.5 Original MySQL API	739
6.5.1 Installing/Configuring	739
6.5.2 Changelog	742
6.5.3 Predefined Constants	743
6.5.4 Examples	744
6.5.5 MySQL Functions	744
6.6 MySQL Native Driver	810
6.6.1 Overview	811
6.6.2 Installation	812
6.6.3 Runtime Configuration	813
6.6.4 Incompatibilities	817
6.6.5 Persistent Connections	817
6.6.6 Statistics	818
6.6.7 Notes	831
6.6.8 Memory management	831
6.6.9 MySQL Native Driver Plugin API	833
6.7 MySQLnd replication and load balancing plugin	845
6.7.1 Key Features	845
6.7.2 Limitations	846
6.7.3 On the name	847
6.7.4 Quickstart and Examples	847
6.7.5 Concepts	875

6.7.6 Installing/Configuring	899
6.7.7 Predefined Constants	954
6.7.8 Mysqlnd_ms Functions	956
6.7.9 Change History	978
6.8 Mysqlnd query result cache plugin	986
6.8.1 Key Features	986
6.8.2 Limitations	986
6.8.3 On the name	987
6.8.4 Quickstart and Examples	987
6.8.5 Installing/Configuring	1007
6.8.6 Predefined Constants	1010
6.8.7 mysqlnd_qc Functions	1012
6.8.8 Change History	1034
6.9 Mysqlnd user handler plugin	1036
6.9.1 Security considerations	1037
6.9.2 Documentation note	1037
6.9.3 On the name	1037
6.9.4 Quickstart and Examples	1037
6.9.5 Installing/Configuring	1042
6.9.6 Predefined Constants	1043
6.9.7 The MysqlndUhConnection class	1048
6.9.8 The MysqlndUhPreparedStatement class	1112
6.9.9 Mysqlnd_uh Functions	1115
6.9.10 Change History	1119
6.10 Mysqlnd connection multiplexing plugin	1119
6.10.1 Key Features	1119
6.10.2 Limitations	1120
6.10.3 About the name mysqlnd_mux	1120
6.10.4 Concepts	1120
6.10.5 Installing/Configuring	1121
6.10.6 Predefined Constants	1122
6.10.7 Change History	1122
6.11 Mysqlnd Memcache plugin	1123
6.11.1 Key Features	1124
6.11.2 Limitations	1124
6.11.3 On the name	1124
6.11.4 Quickstart and Examples	1124
6.11.5 Installing/Configuring	1126
6.11.6 Predefined Constants	1127
6.11.7 Mysqlnd_memcache Functions	1128
6.11.8 Change History	1132
6.12 Common Problems with MySQL and PHP	1132
7 MySQL Connector/C Developer Guide	1135
7.1 Introduction to Connector/C	1135
7.2 Connector/C Versions	1135
7.3 Connector/C Distribution Contents	1136
7.4 Installing Connector/C	1137
7.4.1 Installing Connector/C from a Binary Distribution	1137
7.4.2 Installing Connector/C from Source	1138
7.4.3 Postinstallation Steps	1140
7.4.4 Testing Connector/C	1141
7.5 Building Connector/C Applications	1141
8 MySQL Connector/C++ Developer Guide	1143
8.1 Introduction to Connector/C++	1143
8.2 How to Get Connector/C++	1144
8.3 Installing Connector/C++ from a Binary Distribution	1145
8.4 Installing Connector/C++ from Source	1147
8.4.1 Source Installation System Prerequisites	1147

8.4.2 Obtaining and Unpacking a Connector/C++ Source Distribution	1147
8.4.3 Installing Connector/C++ from Source on Unix and Unix-Like Systems	1148
8.4.4 Installing Connector/C++ from Source on Windows	1149
8.4.5 Troubleshooting Connector/C++ Source Installation Problems	1152
8.4.6 Dynamically Linking Connector/C++ Against the MySQL Client Library	1152
8.4.7 Connector/C++ Source-Configuration Options	1153
8.5 Building Connector/C++ Windows Applications with Microsoft Visual Studio	1155
8.6 Building Connector/C++ Linux Applications with NetBeans	1163
8.7 Getting Started with Connector/C++: Usage Examples	1167
8.7.1 Connecting to MySQL	1168
8.7.2 Running a Simple Query	1169
8.7.3 Fetching Results	1169
8.7.4 Using Prepared Statements	1170
8.7.5 Complete Example 1	1170
8.7.6 Complete Example 2	1171
8.8 Connector/C++ Tutorials	1172
8.8.1 Prerequisites and Background Information	1173
8.8.2 Calling Stored Procedures with <code>Statement</code> Objects	1174
8.8.3 Calling Stored Procedures with <code>PreparedStatement</code> Objects	1178
8.9 Connector/C++ Debug Tracing	1181
8.10 Connector/C++ Usage Notes	1182
8.11 Connector/C++ Known Bugs and Issues	1186
8.12 Connector/C++ Support	1187
A Connector Release Notes	1189

Preface and Legal Notices

This manual describes the Connectors and APIs that can be used with MySQL.

Legal Notices

Copyright © 1997, 2016, Oracle and/or its affiliates. All rights reserved.

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

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

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

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

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

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

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

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

Chapter 1 Introduction

MySQL Connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to the MySQL protocol and MySQL resources. Both Connectors and the APIs enable you to connect and execute MySQL statements from another language or environment, including ODBC, Java (JDBC), Perl, Python, PHP, Ruby, and native C and embedded MySQL instances.

Note

Connector version numbers do not correlate with MySQL Server version numbers. See [MySQL Connector Versions and MySQL Server Versions](#).

MySQL Connectors

Oracle develops a number of connectors:

- [Connector/ODBC](#) provides driver support for connecting to MySQL using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix, and OS X platforms.
- [Connector/Net](#) enables developers to create .NET applications that connect to MySQL. Connector/Net implements a fully functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that use Connector/Net can be written in any supported .NET language.

The MySQL [Visual Studio Plugin](#) works with Connector/Net and Visual Studio 2005. The plugin is a MySQL DDEX Provider, which means that you can use the schema and data manipulation tools available in Visual Studio to create and edit objects within a MySQL database.

- [Connector/J](#) provides driver support for connecting to MySQL from Java applications using the standard Java Database Connectivity (JDBC) API.
- [Connector/Python](#) provides driver support for connecting to MySQL from Python applications using an API that is compliant with the [Python DB API version 2.0](#). No additional Python modules or MySQL client libraries are required.
- [Connector/C++](#) enables C++ applications to connect to MySQL.
- [Connector/C](#) is a standalone replacement for the MySQL Client Library ([libmysqlclient](#)), to be used for C applications.

The MySQL C API

For direct access to using MySQL natively within a C application, there are two methods:

- The [C API](#) provides low-level access to the MySQL client/server protocol through the [libmysqlclient](#) client library. This is the primary method used to connect to an instance of the MySQL server, and is used both by MySQL command-line clients and many of the MySQL Connectors and third-party APIs detailed here.

[libmysqlclient](#) is included in MySQL distributions and in Connector/C distributions.

- [libmysqld](#) is an embedded MySQL server library that enables you to embed an instance of the MySQL server into your C applications.

[libmysqld](#) is included in MySQL distributions, but not in Connector/C distributions.

See also [MySQL C API Implementations](#).

To access MySQL from a C application, or to build an interface to MySQL for a language not supported by the Connectors or APIs in this chapter, the [C API](#) is where to start. A number of programmer's utilities are available to help with the process; see [MySQL Program Development Utilities](#).

Third-Party MySQL APIs

The remaining APIs described in this chapter provide an interface to MySQL from specific application languages. These third-party solutions are not developed or supported by Oracle. Basic information on their usage and abilities is provided here for reference purposes only.

All the third-party language APIs are developed using one of two methods, using [libmysqlclient](#) or by implementing a *native driver*. The two solutions offer different benefits:

- Using [libmysqlclient](#) offers complete compatibility with MySQL because it uses the same libraries as the MySQL client applications. However, the feature set is limited to the implementation and interfaces exposed through [libmysqlclient](#) and the performance may be lower as data is copied between the native language, and the MySQL API components.
- *Native drivers* are an implementation of the MySQL network protocol entirely within the host language or environment. Native drivers are fast, as there is less copying of data between components, and they can offer advanced functionality not available through the standard MySQL API. Native drivers are also easier for end users to build and deploy because no copy of the MySQL client libraries is needed to build the native driver components.

[MySQL APIs and Interfaces](#) lists many of the libraries and interfaces available for MySQL. [MySQL Connector Versions and MySQL Server Versions](#) shows which MySQL Server versions each connector supports.

Chapter 2 MySQL Connector/J 5.1 Developer Guide

Table of Contents

2.1 Overview of MySQL Connector/J	3
2.2 Connector/J Versions	4
2.2.1 Connector/J Release Notes and Change History	5
2.2.2 Java Versions Supported	5
2.3 Connector/J Installation	5
2.3.1 Installing Connector/J from a Binary Distribution	6
2.3.2 Installing the Driver and Configuring the CLASSPATH	6
2.3.3 Upgrading from an Older Version	7
2.3.4 Installing from Source	9
2.3.5 Testing Connector/J	12
2.4 Connector/J Examples	14
2.5 Connector/J (JDBC) Reference	14
2.5.1 Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J	15
2.5.2 JDBC API Implementation Notes	48
2.5.3 Java, JDBC and MySQL Types	51
2.5.4 Using Character Sets and Unicode	53
2.5.5 Connecting Securely Using SSL	55
2.5.6 Connecting Using PAM Authentication	57
2.5.7 Using Master/Slave Replication with ReplicationConnection	57
2.5.8 Mapping MySQL Error Numbers to JDBC SQLState Codes	58
2.6 JDBC Concepts	64
2.6.1 Connecting to MySQL Using the JDBC DriverManager Interface	64
2.6.2 Using JDBC Statement Objects to Execute SQL	65
2.6.3 Using JDBC CallableStatements to Execute Stored Procedures	66
2.6.4 Retrieving AUTO_INCREMENT Column Values through JDBC	68
2.7 Connection Pooling with Connector/J	71
2.8 Multi-Host Connections	74
2.8.1 Configuring Server Failover	75
2.8.2 Configuring Load Balancing with Connector/J	77
2.8.3 Configuring Master/Slave Replication with Connector/J	79
2.8.4 Advanced Load-balancing and Failover Configuration	83
2.9 Using the Connector/J Interceptor Classes	84
2.10 Using Connector/J with Tomcat	85
2.11 Using Connector/J with JBoss	86
2.12 Using Connector/J with Spring	87
2.12.1 Using JdbcTemplate	88
2.12.2 Transactional JDBC Access	89
2.12.3 Connection Pooling with Spring	91
2.13 Using Connector/J with GlassFish	91
2.13.1 A Simple JSP Application with GlassFish, Connector/J and MySQL	93
2.13.2 A Simple Servlet with GlassFish, Connector/J and MySQL	95
2.14 Troubleshooting Connector/J Applications	97
2.15 Known Issues and Limitations	104
2.16 Connector/J Support	105
2.16.1 Connector/J Community Support	105
2.16.2 How to Report Connector/J Bugs or Problems	105

2.1 Overview of MySQL Connector/J

MySQL provides connectivity for client applications developed in the Java programming language with MySQL Connector/J, a driver that implements the [Java Database Connectivity \(JDBC\) API](#).

MySQL Connector/J is a JDBC Type 4 driver. Different versions are available that are compatible with the JDBC 3.0 and JDBC 4.x specifications (see [Section 2.2, “Connector/J Versions”](#)). The Type 4 designation means that the driver is a pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries.

For large-scale programs that use common design patterns of data access, consider using one of the popular persistence frameworks such as [Hibernate](#), [Spring’s JDBC templates](#) or [Ibatis SQL Maps](#) to reduce the amount of JDBC code for you to debug, tune, secure, and maintain.

Key Topics

- For installation instructions for Connector/J, see [Section 2.3, “Connector/J Installation”](#).
- For help with connection strings, connection options, and setting up your connection through JDBC, see [Section 2.5.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”](#).
- For information on connection pooling, see [Section 2.7, “Connection Pooling with Connector/J”](#).
- For information on multi-host connections, see [Section 2.8, “Multi-Host Connections”](#).

2.2 Connector/J Versions

Different versions of MySQL Connector/J are available:

- Connector/J 5.1 is a Type 4 pure Java JDBC driver, which conforms to the JDBC 3.0, 4.0, 4.1, and 4.2 specifications. It provides compatibility with all the functionality of MySQL, including 4.1, 5.0, 5.1, 5.5, 5.6, and 5.7. Connector/J 5.1 provides ease of development features, including auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for large update counts, support for local and offset date-time variants from the `java.time` package, support for JDBC-4.x XML processing, support for per connection client information, and support for the `NCHAR`, `NVARCHAR` and `NCLOB` data types. This release also includes all bug fixes up to and including Connector/J 5.0.6.
- Connector/J 5.0 provides support for all the functionality offered by Connector/J 3.1 and includes distributed transaction (XA) support.
- Connector/J 3.1 was designed for connectivity to MySQL 4.1 and MySQL 5.0 servers and provides support for all the functionality in MySQL 5.0 except distributed transaction (XA) support.
- Connector/J 3.0 provides core functionality and was designed for connectivity to MySQL 3.x or MySQL 4.1 servers, although it provides basic compatibility with later versions of MySQL. Connector/J 3.0 does not support server-side prepared statements, and does not support any of the features in versions of MySQL later than 4.1.

The following table summarizes the Connector/J versions available, along with the details of JDBC driver type, what version of the JDBC API it supports, what versions of MySQL Server it works with, and whether it is currently supported or not:

Table 2.1 Summary of Connector/J Versions

Connector/J version	Driver Type	JDBC version	MySQL Server version	Status
5.1	4	3.0, 4.0, 4.1, 4.2	4.1, 5.0, 5.1, 5.5, 5.6, 5.7	Recommended version
5.0	4	3.0	4.1, 5.0	Released version
3.1	4	3.0	4.1, 5.0	Obsolete
3.0	4	3.0	3.x, 4.1	Obsolete

The current recommended version for Connector/J is 5.1. This guide also covers earlier versions of Connector/J, with specific notes given where a setting applies to a specific option.

2.2.1 Connector/J Release Notes and Change History

For details of new features and bug fixes in each Connector/J release, see the [MySQL Connector/J 5.1 Release Notes](#).

2.2.2 Java Versions Supported

The following table summarizes what version of JRE is required to use Connector/J with Java applications, and what version of JDK is required to build Connector/J source code:

Table 2.2 Summary of Java Versions Required by Connector/J 5.1 and Earlier

Connector/J version	JRE Supported	JDK required (to compile source code)
5.1	1.5.x, 1.6.x, 1.7.x*, 1.8.x**	1.5.x and 1.8.x ***
5.0	1.3.x, 1.4.x, 1.5.x, 1.6.x	1.4.2, 1.5.x, 1.6.x
3.1	1.2.x, 1.3.x, 1.4.x, 1.5.x, 1.6.x	1.4.2, 1.5.x, 1.6.x
3.0	1.2.x, 1.3.x, 1.4.x, 1.5.x, 1.6.x	1.4.2, 1.5.x, 1.6.x

*JRE 1.7 support requires Connector/J 5.1.21 and higher.

**JRE 1.8 is required for Connector/J 5.1 to connect to MySQL 5.6.27 and later and 5.7 with SSL/TLS when using some cipher suites.

*** It is also good to have JRE 1.6 for compiling Connector/J 5.1. See [Section 2.3.4, “Installing from Source”](#) for detail.

If you are building Connector/J from source code using the source distribution (see [Section 2.3.4, “Installing from Source”](#)), you must use JDK 1.4.2 or newer to compile the package for Connector/J 5.0 or earlier. For Connector/J 5.1, you must have both JDK-1.6.x AND JDK-1.5.x installed to be able to build the source code.

Several JDBC 4.1 methods were implemented for the first time in Connector/J 5.1.21.

Because of the implementation of `java.sql.Savepoint`, Connector/J 3.1.0 and newer will not run on a Java runtime older than 1.4 unless the class verifier is turned off (by setting the `-Xverify:none` option to the Java runtime). This is because the class verifier will try to load the class definition for `java.sql.Savepoint` even though it is not accessed by the driver unless you actually use savepoint functionality.

Caching functionality provided by Connector/J 3.1.0 or newer is also not available on JVMs older than 1.4.x, as it relies on `java.util.LinkedHashMap`, which was first available in JDK-1.4.0.

MySQL Connector/J does not support JDK-1.1.x or JDK-1.0.x.

2.3 Connector/J Installation

MySQL Connector/J is distributed as a `.zip` or `.tar.gz` archive, available for download from the [Connector/J Download page](#). The archive contains the sources and the JAR archive named `mysql-connector-java-version-bin.jar`.

You can install the Connector/J package using either the binary or source distribution. The binary distribution provides the easiest method for installation; the source distribution lets you customize your installation further. With either solution, you manually add the Connector/J location to your Java `CLASSPATH`.

If you are upgrading from a previous version, read the upgrade information in [Section 2.3.3, “Upgrading from an Older Version”](#) before continuing.

Connector/J is also available as part of the Maven project. For more information and to download the Connector/J JAR files, see the [Maven repository](#).

2.3.1 Installing Connector/J from a Binary Distribution

For the easiest method of installation, use the binary distribution of the Connector/J package. Extract the JAR archive from the tar/gzip or zip archive to a suitable location, then optionally make the information about the JAR archive available by changing your `CLASSPATH` (see [Section 2.3.2, “Installing the Driver and Configuring the CLASSPATH”](#)).

Use the appropriate graphical or command-line utility to extract the distribution (for example, WinZip for the .zip archive, and `tar` for the .tar.gz archive). Because there are potentially long file names in the distribution, we use the GNU tar archive format. Use GNU tar (or an application that understands the GNU tar archive format) to unpack the .tar.gz variant of the distribution.

2.3.2 Installing the Driver and Configuring the `CLASSPATH`

Once you have extracted the distribution archive, you can install the driver by placing `mysql-connector-java-version-bin.jar` in your classpath, either by adding the full path to it to your `CLASSPATH` environment variable, or by directly specifying it with the command line switch `-cp` when starting the JVM.

To use the driver with the JDBC `DriverManager`, use `com.mysql.jdbc.Driver` as the class that implements `java.sql.Driver`.

You can set the `CLASSPATH` environment variable under Unix, Linux, or OS X either locally for a user within their `.profile`, `.login` or other login file. You can also set it globally by editing the global `/etc/profile` file.

For example, add the Connector/J driver to your `CLASSPATH` using one of the following forms, depending on your command shell:

```
# Bourne-compatible shell (sh, ksh, bash, zsh):
shell> export CLASSPATH=/path/mysql-connector-java-ver-bin.jar:$CLASSPATH
# C shell (csh, tcsh):
shell> setenv CLASSPATH /path/mysql-connector-java-ver-bin.jar:$CLASSPATH
```

For Windows platforms, you set the environment variable through the System Control Panel.

To use MySQL Connector/J with an application server such as GlassFish, Tomcat, or JBoss, read your vendor's documentation for more information on how to configure third-party class libraries, as most application servers ignore the `CLASSPATH` environment variable. For configuration examples for some J2EE application servers, see [Section 2.7, “Connection Pooling with Connector/J”](#), [Section 2.8.2, “Configuring Load Balancing with Connector/J”](#), and [Section 2.8.4, “Advanced Load-balancing and Failover Configuration”](#). However, the authoritative source for JDBC connection pool configuration information for your particular application server is the documentation for that application server.

If you are developing servlets or JSPs, and your application server is J2EE-compliant, you can put the driver's `.jar` file in the `WEB-INF/lib` subdirectory of your webapp, as this is a standard location for third party class libraries in J2EE web applications.

You can also use the `MysqlDataSource` or `MysqlConnectionPoolDataSource` classes in the `com.mysql.jdbc.jdbc2.optional` package, if your J2EE application server supports or requires them. Starting with Connector/J 5.0.0, the `javax.sql.XADataSource` interface is implemented using the `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class, which supports XA distributed transactions when used in combination with MySQL server version 5.0 and later.

The various `MySQLDataSource` classes support the following parameters (through standard set mutators):

- `user`
- `password`
- `serverName` (see the previous section about failover hosts)
- `databaseName`
- `port`

2.3.3 Upgrading from an Older Version

This section has information for users who are upgrading from one version of Connector/J to another, or to a new version of the MySQL server that supports a more recent level of JDBC. A newer version of Connector/J might include changes to support new features, improve existing functionality, or comply with new standards.

2.3.3.1 Upgrading to MySQL Connector/J 5.1.x

- In Connector/J 5.0.x and earlier, the alias for a table in a `SELECT` statement is returned when accessing the result set metadata using `ResultSetMetaData.getColumnname()`. This behavior however is not JDBC compliant, and in Connector/J 5.1, this behavior has been changed so that the original table name, rather than the alias, is returned.

The JDBC-compliant behavior is designed to let API users reconstruct the DML statement based on the metadata within `ResultSet` and `ResultSetMetaData`.

You can get the alias for a column in a result set by calling `ResultSetMetaData.getColumnLabel()`. To use the old noncompliant behavior with `ResultSetMetaData.getColumnname()`, use the `useOldAliasMetadataBehavior` option and set the value to `true`.

In Connector/J 5.0.x, the default value of `useOldAliasMetadataBehavior` was `true`, but in Connector/J 5.1 this was changed to a default value of `false`.

2.3.3.2 JDBC-Specific Issues When Upgrading to MySQL Server 4.1 or Newer

- *Using the UTF-8 Character Encoding* - Prior to MySQL server version 4.1, the UTF-8 character encoding was not supported by the server, however the JDBC driver could use it, allowing storage of multiple character sets in `latin1` tables on the server.

Starting with MySQL-4.1, this functionality is deprecated. If you have applications that rely on this functionality, and can not upgrade them to use the official Unicode character support in MySQL server version 4.1 or newer, add the following property to your connection URL:

`useOldUTF8Behavior=true`

- *Server-side Prepared Statements* - Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer). If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the following connection property:

`useServerPrepStmts=false`

2.3.3.3 Upgrading from MySQL Connector/J 3.0 to 3.1

Connector/J 3.1 is designed to be backward-compatible with Connector/J 3.0 as much as possible. Major changes are isolated to new functionality exposed in MySQL-4.1 and newer, which includes

Unicode character sets, server-side prepared statements, [SQLState](#) codes returned in error messages by the server and various performance enhancements that can be enabled or disabled using configuration properties.

- **Unicode Character Sets:** See the next section, as well as [Character Set Support](#), for information on this MySQL feature. If you have something misconfigured, it will usually show up as an error with a message similar to [Illegal mix of collations](#).
- **Server-side Prepared Statements:** Connector/J 3.1 will automatically detect and use server-side prepared statements when they are available (MySQL server version 4.1.0 and newer).

Starting with version 3.1.7, the driver scans SQL you are preparing using all variants of [Connection.prepareStatement\(\)](#) to determine if it is a supported type of statement to prepare on the server side, and if it is not supported by the server, it instead prepares it as a client-side emulated prepared statement. You can disable this feature by passing [emulateUnsupportedPstmts=false](#) in your JDBC URL.

If your application encounters issues with server-side prepared statements, you can revert to the older client-side emulated prepared statement code that is still presently used for MySQL servers older than 4.1.0 with the connection property [useServerPrepStmts=false](#).

- **Datetimes** with all-zero components ([0000-00-00 . . .](#)): These values cannot be represented reliably in Java. Connector/J 3.0.x always converted them to [NULL](#) when being read from a [ResultSet](#).

Connector/J 3.1 throws an exception by default when these values are encountered, as this is the most correct behavior according to the JDBC and SQL standards. This behavior can be modified using the [zeroDateTimeBehavior](#) configuration property. The permissible values are:

- [exception](#) (the default), which throws an [SQLException](#) with an [SQLState](#) of [S1009](#).
- [convertToNull](#), which returns [NULL](#) instead of the date.
- [round](#), which rounds the date to the nearest closest value which is [0001-01-01](#).

Starting with Connector/J 3.1.7, [ResultSet.getString\(\)](#) can be decoupled from this behavior using [noDatetimeStringSync=true](#) (the default value is [false](#)) so that you can retrieve the unaltered all-zero value as a [String](#). Note that this also precludes using any time zone conversions, therefore the driver will not allow you to enable [noDatetimeStringSync](#) and [useTimezone](#) at the same time.

- **New SQLState Codes:** Connector/J 3.1 uses SQL:1999 [SQLState](#) codes returned by the MySQL server (if supported), which are different from the legacy X/Open state codes that Connector/J 3.0 uses. If connected to a MySQL server older than MySQL-4.1.0 (the oldest version to return [SQLStates](#) as part of the error code), the driver will use a built-in mapping. You can revert to the old mapping by using the configuration property [useSqlStateCodes=false](#).
- **ResultSet.getString():** Calling [ResultSet.getString\(\)](#) on a [BLOB](#) column will now return the address of the [byte\[\]](#) array that represents it, instead of a [String](#) representation of the [BLOB](#). [BLOB](#) values have no character set, so they cannot be converted to [java.lang.Strings](#) without data loss or corruption.

To store strings in MySQL with LOB behavior, use one of the [TEXT](#) types, which the driver will treat as a [java.sql.Clob](#).

- **Debug builds:** Starting with Connector/J 3.1.8 a debug build of the driver in a file named [mysql-connector-java-version-bin-g.jar](#) is shipped alongside the normal binary jar file that is named [mysql-connector-java-version-bin.jar](#).

Starting with Connector/J 3.1.9, we do not ship the [.class](#) files unbundled, they are only available in the JAR archives that ship with the driver.

Do not use the debug build of the driver unless instructed to do so when reporting a problem or bug, as it is not designed to be run in production environments, and will have adverse performance impact when used. The debug binary also depends on the Aspect/J runtime library, which is located in the [src/lib/aspectjrt.jar](#) file that comes with the Connector/J distribution.

2.3.4 Installing from Source

Caution

To just get MySQL Connector/J up and running on your system, install Connector/J [using a standard binary release distribution](#). Instructions in this section is only for users who, for various reasons, want to compile Connector/J from source.

The requirements and steps for installing from source Connector/J [5.1.37 or later](#), [5.1.34 to 5.1.36](#), and [5.1.33 or earlier](#) are different; check the section below that is relevant for the version you want.

Installing Connector/J 5.1.37 or later from source. To install MySQL Connector/J from its source tree on GitHub, you need to have the following software on your system:

- A Git client, to check out the sources from our GitHub repository (available from <http://git-scm.com/downloads>).
- Apache Ant version 1.8.2 or newer (available from <http://ant.apache.org/>).
- JDK 1.8.x AND JDK 1.5.x.
- JRE 1.6.x (optional)
- JUnit libraries (available from <https://github.com/junit-team/junit/wiki/Download-and-Install>).
- The required `.jar` files from the Hibernate ORM 4.1 or 4.2 Final release bundle, which is available at <http://sourceforge.net/projects/hibernate/files/hibernate4/>.

To check out and compile MySQL Connector/J, follow these steps:

1. Check out the code from the source code repository for MySQL Connector/J located on GitHub at <https://github.com/mysql/mysql-connector-j>; for the latest release of the Connector/J 5.1 series, use the following command:

```
shell> git clone https://github.com/mysql/mysql-connector-j.git
```

To check out a release other than the latest one, use the `--branch` option to specify the revision tag for it:

```
shell> git clone --branch 5.1 https://github.com/mysql/mysql-connector-j.git
```

Under the current directory, the commands create a `mysql-connector-j` subdirectory, which contains the code you want.

2. Make sure that you have both JDK 1.8.x AND JDK 1.5.x installed. You need both JDKs because besides supporting JDBC from 4.0 to 4.2, Connector/J 5.1 also supports JDBC 3.0, which is an older version and requires the older JDK 1.5.x.
3. Consider also having JRE 1.6.x installed. This is optional: if JRE 1.6.x is not available or not supplied to Ant with the property `com.mysql.jdbc.java6.rtjar`, the Java 8 bootstrap classes will be used. A warning will be returned, saying that the bootstrap class path was not set with the option to compile sources written for Java 6.
4. Place the required `junit.jar` file in a separate directory—for example, `/home/username/ant-extralibs`.

5. In the same directory for extra libraries described in the last step, create a directory named `hibernate4`, and put under it all the `.jar` files you can find under the `/lib/required/` folder in the Hibernate ORM 4 Final release bundle.
6. Change your current working directory to the `mysql-connector-j` directory created in step 1 above.
7. In the directory, create a file named `build.properties` to indicate to Ant the locations of the root directories for your JDK 1.8.x and JDK 1.5.x installations, the location of the `rt.jar` of your JRE 1.6.x (optional), and the location of the extra libraries. The file should contain the following property settings, with the "`path_to_*`" parts replaced by the appropriate filepaths:

```
com.mysql.jdbc.jdk8=path_to_jdk_1.8
com.mysql.jdbc.jdk5=path_to_jdk_1.5
com.mysql.jdbc.java6.rtjar=path_to_rt.jar_under_jre_1.6/rt.jar
com.mysql.jdbc.extra.libs=path_to_folder_for_extra_libraries
```

Alternatively, you can set the values of those properties through the Ant `-D` options.

8. Issue the following command to compile the driver and create a `.jar` file for Connector/J:

```
shell> ant dist
```

This creates a `build` directory in the current directory, where all the build output goes. A directory is created under the `build` directory, whose name includes the version number of the release you are building. That directory contains the sources, the compiled `.class` files, and a `.jar` file for deployment. For more information and other possible targets, including those that create a fully packaged distribution, issue the following command:

```
shell> ant -projecthelp
```

9. Install the newly created `.jar` file for the JDBC driver as you would install a binary `.jar` file you download from MySQL by following the instructions given in [Section 2.3.2, "Installing the Driver and Configuring the CLASSPATH"](#).

Note that a package containing both the binary and source code for Connector/J 5.1 can also be downloaded from the [Connector/J Download page](#).

Installing Connector/J 5.1.34 to 5.1.36 from source. To install MySQL Connector/J 5.1.34 to 5.1.36 from the Connector/J source tree on GitHub, make sure that you have the following software on your system:

- A Git client, to check out the sources from our GitHub repository (available from <http://git-scm.com/downloads>).
- Apache Ant version 1.8.2 or newer (available from <http://ant.apache.org/>).
- JDK 1.6.x AND JDK 1.5.x.
- JUnit libraries (available from <https://github.com/junit-team/junit/wiki/Download-and-Install>).
- The required `.jar` files from the Hibernate ORM 4.1 or 4.2 Final release bundle, which is available at <http://sourceforge.net/projects/hibernate/files/hibernate4/>.

To check out and compile MySQL Connector/J, follow these steps:

1. Check out the code from the source code repository for MySQL Connector/J located on GitHub at <https://github.com/mysql/mysql-connector-j>, using the `--branch` option to specify the revision tag for release 5.1.xx:

```
shell> git clone --branch 5.1.xx https://github.com/mysql/mysql-connector-j.git
```

Under the current directory, the commands create a `mysql-connector-j` subdirectory, which contains the code you want.

2. Make sure that you have both JDK 1.6.x AND JDK 1.5.x installed. You need both JDKs because Connector/J 5.1 supports both JDBC 3.0 (which has existed prior to JDK 1.6.x) and JDBC 4.0.
3. Place the required `junit.jar` file in a separate directory—for example, `/home/username/ant-extralibs`.
4. In the same directory for extra libraries described in the last step, create a directory named `hibernate4`, and put under it all the `.jar` files you can find under the `/lib/required/` folder in the Hibernate ORM 4 Final release bundle.
5. Change your current working directory to the `mysql-connector-j` directory created in step 1 above.
6. In the directory, create a file named `build.properties` to indicate to Ant the locations of the root directories for your JDK 1.5.x and JDK 1.6.x installations, as well as the location of the extra libraries. The file should contain the following property settings, with the “`path_to_*`” parts replaced by the appropriate filepaths:

```
com.mysql.jdbc.jdk5=path_to_jdk_1.5
com.mysql.jdbc.jdk6=path_to_jdk_1.6
com.mysql.jdbc.extra.libs=path_to_folder_for_extra_libraries
```

Alternatively, you can set the values of those properties through the Ant `-D` options.

7. Issue the following command to compile the driver and create a `.jar` file for Connector/J:

```
shell> ant dist
```

This creates a `build` directory in the current directory, where all the build output goes. A directory is created under the `build` directory, whose name includes the version number of the release you are building. That directory contains the sources, the compiled `.class` files, and a `.jar` file for deployment. For more information and other possible targets, including those that create a fully packaged distribution, issue the following command:

```
shell> ant -projecthelp
```

8. Install the newly created `.jar` file for the JDBC driver as you would install a binary `.jar` file you download from MySQL by following the instructions given in [Section 2.3.2, “Installing the Driver and Configuring the CLASSPATH”](#).

Installing Connector/J 5.1.33 or earlier from the source tree. To install MySQL Connector/J 5.1.33 or earlier from the Connector/J source tree on GitHub, make sure that you have the following software on your system:

- A Git client, to check out the source code from our GitHub repository (available from <http://git-scm.com/downloads>).
- Apache Ant version 1.7 or newer (available from <http://ant.apache.org/>).
- JDK 1.6.x AND JDK 1.5.x. Refer to [Section 2.2.2, “Java Versions Supported”](#) for the version of Java you need to build or run any Connector/J release.
- The Ant Contrib (version 1.03b is available from <http://sourceforge.net/projects/ant-contrib/files/ant-contrib/1.0b3/>) and JUnit (available from <https://github.com/junit-team/junit/wiki/Download-and-Install>) libraries.
- The required `.jar` files from the Hibernate ORM 4.1 or 4.2 Final release bundle, which is available at <http://sourceforge.net/projects/hibernate/files/hibernate4/>.

To check out and compile a specific branch of MySQL Connector/J, follow these steps:

1. Check out the code from the source code repository for MySQL Connector/J located on GitHub at <https://github.com/mysql/mysql-connector-j>, using the `--branch` option to specify the revision tag for release 5.1.xx:

```
shell> git clone --branch 5.1.xx https://github.com/mysql/mysql-connector-j.git
```

Under the current directory, the commands create a `mysql-connector-j` subdirectory, which contains the code you want.

2. To build Connector/J 5.1, make sure that you have both JDK 1.6.x AND JDK 1.5.x installed. You need both JDKs because Connector/J 5.1 supports both JDBC 3.0 (which has existed prior to JDK 1.6.x) and JDBC 4.0. Set your `JAVA_HOME` environment variable to the path to the JDK 1.5.x installation.
3. Place the required `ant-contrib.jar` file (in exactly that name, without the version number in it; rename the `jar` file if needed) and `junit.jar` file in a separate directory—for example, `/home/username/ant-extralibs`.
4. In the same directory for extra libraries described in the last step, create a directory named `hibernate4`, and put under it all the `.jar` files you can find under the `/lib/required/` folder in the Hibernate ORM 4 Final release bundle.
5. Change your current working directory to the `mysql-connector-j` directory created in step 1 above.
6. In the directory, create a file named `build.properties` to indicate to Ant the locations of the `Javac` and `rt.jar` of your JDK 1.6.x, as well as the location of the extra libraries. The file should contain the following property settings, with the “`path_to_*`” parts replaced by the appropriate filepaths:

```
com.mysql.jdbc.java6.javac=path_to_javac_1.6/javac
com.mysql.jdbc.java6.rtjar=path_to_rt.jar_under_jdk_1.6/rt.jar
com.mysql.jdbc.extra.libs=path_to_folder_for_extra_libraries
```

Alternatively, you can set the values of those properties through the Ant `-D` options.

7. Issue the following command to compile the driver and create a `.jar` file for Connector/J:

```
shell> ant dist
```

This creates a `build` directory in the current directory, where all the build output goes. A directory is created under the `build` directory, whose name includes the version number of the release you are building. That directory contains the sources, the compiled `.class` files, and a `.jar` file for deployment. For more information and other possible targets, including those that create a fully packaged distribution, issue the following command:

```
shell> ant -projecthelp
```

8. Install the newly created `.jar` file for the JDBC driver as you would install a binary `.jar` file you download from MySQL by following the instructions given in [Section 2.3.2, “Installing the Driver and Configuring the CLASSPATH”](#).

2.3.5 Testing Connector/J

The Connector/J source code repository or packages that are shipped with source code include an extensive test suite, containing test cases that can be executed independently. The test cases are divided into the following categories:

- *Functional or unit tests*: Classes from the package `testsuite.simple`. Include test code for the main features of the Connector/J.
- *Performance tests*: Classes from the package `testsuite.perf`. Include test code to make measurements for the performance of Connector/J.
- *Fabric tests*: Classes from the package `testsuite.fabric`. Includes the code to test Fabric-specific features. These tests require the setting of some special properties that are not documented here. Consult the code or the Fabric-related targets in the bundled Ant build file, `build.xml`.
- *Regression tests*: Classes from the package `testsuite.regression`. Includes code for testing bug and regression fixes.

The bundled Ant build file contains targets like `test` and `test-multijvm`, which can facilitate the process of running the Connector/J tests; see the target descriptions in the build file for details. Besides the requirements for building Connector/J from the source code described in [Section 2.3.4, “Installing from Source”](#), a number of the tests also require the File System Service Provider 1.2 for the Java Naming and Directory Interface (JNDI), available at <http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-java-plat-419418.html>)—place the jar files downloaded from there into the `lib` directory or in the directory pointed to by the property `com.mysql.jdbc.extra.libs`.

To run the test using Ant, in addition to the properties required for [Section 2.3.4, “Installing from Source”](#), you must set the following properties in the `build.properties` file or through the Ant `-D` options:

- `com.mysql.jdbc.testsuite.url`: it specifies the JDBC URL for connection to a MySQL test server; see [Section 2.5.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”](#).
- `com.mysql.jdbc.testsuite.jvm`: the JVM to be used for the tests. If the property is set, the specified JVM will be used for all test cases except if it points to a Java 5 directory, in which case any test cases for JDBC 4.0 and later are run with the JVM supplied with the property `com.mysql.jdbc.jdk8` (for 5.1.36 and earlier, supplied with the property `com.mysql.jdbc.jdk6`). If the property is not set, the JVM supplied with `com.mysql.jdbc.jdk8` will be used to run test cases for JDBC 3.0 and the one supplied with `com.mysql.jdbc.jdk8` (for 5.1.36 and earlier, supplied with the property `com.mysql.jdbc.jdk6`) will be used to run test cases for JDBC 4.0 and later.

After setting these parameters, run the tests with Ant in the following ways:

- Building the `test` target with `ant test` runs all test cases by default on a single server instance. If you want to run a particular test case, put the test's fully qualified class names in the `test` variable; for example:

```
shell > ant -Dtest=testsuite.simple.StringUtilsTest test
```

You can also run individual tests in a test case by specifying the names of the corresponding methods in the `methods` variable, separating multiple methods by commas; for example:

```
shell > ant -Dtest=testsuite.simple.StringUtilsTest -Dmethods=testIndexOfIgnoreCase,testGetBytes test
```

- Building the `test-multijvm` target with `ant test-multijvm` runs all the test cases using multiple JVMs of different versions on multiple server instances. For example, if you want to run the tests using a Java 7 and a Java 8 JVM on three server instances with different configurations, you will need to use the following properties:

```
com.mysql.jdbc.testsuite.jvm.1=path_to_Java_7
com.mysql.jdbc.testsuite.jvm.2=path_to_Java_8
com.mysql.jdbc.testsuite.url.1=URL_to_1st_server
```

```
com.mysql.jdbc.testsuite.url.2=URL_to_2nd_server
com.mysql.jdbc.testsuite.url.3=URL_to_3rd_server
```

Unlike the target `test`, the target `test-multijvm` only recognizes the properties `com.mysql.jdbc.testsuite.jvm.N` and `com.mysql.jdbc.testsuite.url.N`, where `N` is a numeric suffix; the same properties without the suffixes are ignored by `test-multijvm`. As with the target `test`, if any of the `com.mysql.jdbc.testsuite.jvm.N` settings points to Java 5, then Ant relies on the property `com.mysql.jdbc.jdk8` to run the tests specific to JDBC 4.0 and later.

You can choose to run individual test cases or specific tests by using the `test` or `methods` property, as explained in the last bullet for the target `test`. Each test is run once per possible combination of JVMs and server instances (that is, 6 times in total for in this example).

When a test for a certain JVM-server combination has failed, `test-multijvm` does not throw an error, but moves on to the next combination, until all tests for all combinations are finished.

While the test results are partially reported by the console, complete reports in HTML and XML formats are provided:

- For results of `test`: view the HTML report by opening `build/junit/unitregress/report/index.html`. XML version of the reports are located in the folder `build/junit/unitregress`.
- For results of `test-multijvm`: view the HTML report for each JVM-server combination by opening `build/junit/MySQLN.server_version/operating_system_version/jvm-version/unitregress/report/index.html`. XML version of the reports are located in the folder `build/junit/MySQLN.server_version/operating_system_version/jvm-version/unitregress`.

2.4 Connector/J Examples

Examples of using Connector/J are located throughout this document. This section provides a summary and links to these examples.

- [Example 2.1, “Connector/J: Obtaining a connection from the DriverManager”](#)
- [Example 2.2, “Connector/J: Using java.sql.Statement to execute a SELECT query”](#)
- [Example 2.3, “Connector/J: Calling Stored Procedures”](#)
- [Example 2.4, “Connector/J: Using Connection.prepareCall\(\)”](#)
- [Example 2.5, “Connector/J: Registering output parameters”](#)
- [Example 2.6, “Connector/J: Setting CallableStatement input parameters”](#)
- [Example 2.7, “Connector/J: Retrieving results and output parameter values”](#)
- [Example 2.8, “Connector/J: Retrieving AUTO_INCREMENT column values using Statement.getGeneratedKeys\(\)”](#)
- [Example 2.9, “Connector/J: Retrieving AUTO_INCREMENT column values using SELECT LAST_INSERT_ID\(\)”](#)
- [Example 2.10, “Connector/J: Retrieving AUTO_INCREMENT column values in Updatable ResultSets”](#)
- [Example 2.11, “Connector/J: Using a connection pool with a J2EE application server”](#)
- [Example 2.12, “Connector/J: Example of transaction with retry logic”](#)

2.5 Connector/J (JDBC) Reference

This section of the manual contains reference material for MySQL Connector/J.

2.5.1 Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J

The name of the class that implements `java.sql.Driver` in MySQL Connector/J is `com.mysql.jdbc.Driver`. The `org.gjt.mm.mysql.Driver` class name is also usable for backward compatibility with MM.MySQL, the predecessor of Connector/J. Use this class name when registering the driver, or when configuring a software to use MySQL Connector/J.

JDBC URL Format

The general format for a JDBC URL for connecting to a MySQL server is as follows, with items in square brackets ([]) being optional:

```
jdbc:mysql://[host1][:port1][,[host2][:port2]]...[/[database]] »  
[?propertyName1=propertyValue1[&propertyName2=propertyValue2]...]
```

Here is a simple example for a connection URL:

```
jdbc:mysql://localhost:3306/sakila?profileSQL=true
```

Supply multiple hosts for a server failover setup (see [Section 2.8, “Multi-Host Connections”](#) for details):

```
# Connection URL for a server failover setup:  
jdbc:mysql://primaryhost,secondaryhost1,secondaryhost2/test
```

There are specialized URL schemes for configuring Connector/J's multi-host functions like load balancing and replication; here are some examples (see [Section 2.8, “Multi-Host Connections”](#) for details):

```
# Connection URL for load balancing:  
jdbc:mysql:loadbalance://localhost:3306,localhost:3310/sakila  
# Connection URL for server replication:  
jdbc:mysql:replication://master,slave1,slave2,slave3/test
```

Host and Port

If no hosts are not specified, the host name defaults to `127.0.0.1`. If the port for a host is not specified, it defaults to `3306`, the default port number for MySQL servers.

Initial Database for Connection

If the database is not specified, the connection is made with no default database. In this case, either call the `setCatalog()` method on the Connection instance, or fully specify table names using the database name (that is, `SELECT dbname.tablename.colname FROM dbname.tablename...`) in your SQL. Opening a connection without specifying the database to use is generally only useful when building tools that work with multiple databases, such as GUI database managers.

Note

Always use the `Connection.setCatalog()` method to specify the desired database in JDBC applications, rather than the `USE database` statement.

Alternate Format for JDBC URLs

The following is an alternate format for JDBC URLs connecting to a MySQL server, which is mandatory for IPv6 connections, but can also be used with IPv4 (items in square brackets ([]) are optional):

```
jdbc:mysql://address=(key1=value)[(key2=value)]...[,address=(key3=value)[(key4=value)]...]/[database]  
[?propertyName1=propertyValue1[&propertyName2=propertyValue2]...]
```

Supported key-value pairs include:

- (`protocol=tcp`), or (`protocol=pipe`) for named pipes on Windows.
- (`path=path_to_pipe`) for path of named pipes. Default value for the path is `\\.\pipe\MySQL`. Use the key-value pair to specify a custom named pipe.
- (`host=hostname`) for TCP connections.
- (`port=port_number`) for TCP connections.

For example:

```
jdbc:mysql://address=(protocol=tcp)(host=localhost)(port=3306)/db
```

Notice that the alternate format greatly simplifies the URL for a named pipe connection on Windows. For example, to use the default named pipe of `\\.\pipe\MySQL`, just specifies:

```
jdbc:mysql://address=(protocol=pipe)/test
```

To use the custom named pipe of `\\.\pipe\MySQL57`:

```
jdbc:mysql://address=(protocol=pipe)(path=\\.\pipe\MySQL57)/test
```

Keys other than `protocol`, `path`, `host`, and `port` are treated as host-specific configuration properties, which allow per-host overrides of configuration properties set for multi-host connections (that is, when using failover, load balancing, or replication). For example:

```
# Connection URL in the alternate format for a server failover setup:
jdbc:mysql://address=(protocol=tcp)(host=primaryhost)(port=3306),»
address=(protocol=tcp)(host=secondaryhost1)(port=3310)(user=test2)/test
# Connection URL in the alternate format for a load balancing setup:
jdbc:mysql:loadbalance://address=(protocol=tcp)(host=localhost)(port=3306)(user=test1),»
address=(protocol=tcp)(host=localhost)(port=3310)(user=test2)/sakila
# Connection URL in the alternate format for a server replication setup:
jdbc:mysql:replication://address=(protocol=tcp)(host=master)(port=3306)(user=test1),»
address=(protocol=tcp)(host=slave1)(port=3310)(user=test2)/test
```

Limit the overrides to user, password, network timeouts, and statement and metadata cache sizes; the effects of other per-host overrides are not defined.

The ways to set other configuration properties for the connection are the same for the general and the alternate URL formats; see [Setting Configuration Properties](#).

Setting Configuration Properties

Configuration properties define how Connector/J will make a connection to a MySQL server. Unless otherwise noted, properties can be set for a `DataSource` object or for a `Connection` object.

Configuration properties can be set in one of the following ways:

- Using the `set*()` methods on MySQL implementations of `java.sql.DataSource` (which is the preferred method when using implementations of `java.sql.DataSource`):
 - `com.mysql.jdbc.jdbc2.optional.MysqlDataSource`
 - `com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource`
- As a key/value pair in the `java.util.Properties` instance passed to `DriverManager.getConnection()` or `Driver.connect()`
- As a JDBC URL parameter in the URL given to `java.sql.DriverManager.getConnection()`, `java.sql.Driver.connect()` or the MySQL implementations of the `javax.sql.DataSource.setURL()` method. If you specify a configuration property in the URL without providing a value for it, nothing will be set; for example, adding `useServerPrepStmts` alone to the URL does not make Connector/J use server-side prepared statements; you need to add `useServerPrepStmts=true`.

Note

If the mechanism you use to configure a JDBC URL is XML-based, use the XML character literal `&` to separate configuration parameters, as the ampersand is a reserved character for XML.

The properties are listed in the following tables.

Connection/Authentication.

Properties and Descriptions
user The user to connect as Since version: all versions
password The password to use when connecting Since version: all versions
socketFactory The name of the class that the driver should use for creating socket connections to the server. This class must implement the interface 'com.mysql.jdbc.SocketFactory' and have public no-args constructor. Default: com.mysql.jdbc.StandardSocketFactory Since version: 3.0.3
connectTimeout Timeout for socket connect (in milliseconds), with 0 being no timeout. Only works on JDK-1.4 or newer. Defaults to '0'. Default: 0 Since version: 3.0.1
socketTimeout Timeout on network socket operations (0, the default means no timeout). Default: 0 Since version: 3.0.1
connectionLifecycleInterceptors A comma-delimited list of classes that implement "com.mysql.jdbc.ConnectionLifecycleInterceptor" that should notified of connection lifecycle events (creation, destruction, commit, rollback, setCatalog and setAutoCommit) and potentially alter the execution of these commands. ConnectionLifecycleInterceptors are "stackable", more than one interceptor may be specified via the configuration property as a comma-delimited list, with the interceptors executed in order from left to right. Since version: 5.1.4
useConfigs

Properties and Descriptions
Load the comma-delimited list of configuration properties before parsing the URL or applying user-specified properties. These configurations are explained in the 'Configurations' of the documentation. Since version: 3.1.5
authenticationPlugins Comma-delimited list of classes that implement com.mysql.jdbc.AuthenticationPlugin and which will be used for authentication unless disabled by "disabledAuthenticationPlugins" property. Since version: 5.1.19
defaultAuthenticationPlugin Name of a class implementing com.mysql.jdbc.AuthenticationPlugin which will be used as the default authentication plugin (see below). It is an error to use a class which is not listed in "authenticationPlugins" nor it is one of the built-in plugins. It is an error to set as default a plugin which was disabled with "disabledAuthenticationPlugins" property. It is an error to set this value to null or the empty string (i.e. there must be at least a valid default authentication plugin specified for the connection, meeting all constraints listed above). Default: com.mysql.jdbc.authentication.MysqlNativePasswordPlugin Since version: 5.1.19
disabledAuthenticationPlugins Comma-delimited list of classes implementing com.mysql.jdbc.AuthenticationPlugin or mechanisms, i.e. "mysql_native_password". The authentication plugins or mechanisms listed will not be used for authentication which will fail if it requires one of them. It is an error to disable the default authentication plugin (either the one named by "defaultAuthenticationPlugin" property or the hard-coded one if "defaultAuthenticationPlugin" property is not set). Since version: 5.1.19
disconnectOnExpiredPasswords If "disconnectOnExpiredPasswords" is set to "false" and password is expired then server enters "sandbox" mode and sends ERR(08001, ER_MUST_CHANGE_PASSWORD) for all commands that are not needed to set a new password until a new password is set. Default: true Since version: 5.1.23
interactiveClient Set the CLIENT_INTERACTIVE flag, which tells MySQL to timeout connections based on INTERACTIVE_TIMEOUT instead of WAIT_TIMEOUT Default: false Since version: 3.1.0
localSocketAddress Hostname or IP address given to explicitly configure the interface that the driver will bind the client side of the TCP/IP connection to when connecting. Since version: 5.0.5
propertiesTransform

Properties and Descriptions

An implementation of `com.mysql.jdbc.ConnectionPropertiesTransform` that the driver will use to modify URL properties passed to the driver before attempting a connection

Since version: 3.1.4

useCompression

Use zlib compression when communicating with the server (true/false)? Defaults to 'false'.

Default: false

Since version: 3.0.17

Networking.**Properties and Descriptions****socksProxyHost**

Name or IP address of SOCKS host to connect through.

Since version: 5.1.34

socksProxyPort

Port of SOCKS server.

Default: 1080

Since version: 5.1.34

maxAllowedPacket

Maximum allowed packet size to send to server. If not set, the value of system variable 'max_allowed_packet' will be used to initialize this upon connecting. This value will not take effect if set larger than the value of 'max_allowed_packet'. Also, due to an internal dependency with the property "blobSendChunkSize", this setting has a minimum value of "8203" if "useServerPrepStmts" is set to "true".

Default: -1

Since version: 5.1.8

tcpKeepAlive

If connecting using TCP/IP, should the driver set SO_KEEPALIVE?

Default: true

Since version: 5.0.7

tcpNoDelay

If connecting using TCP/IP, should the driver set SO_TCP_NODELAY (disabling the Nagle Algorithm)?

Default: true

Since version: 5.0.7

tcpRcvBuf

If connecting using TCP/IP, should the driver set SO_RCV_BUF to the given value? The default value of '0', means use the platform default value for this property)

Properties and Descriptions
Default: 0 Since version: 5.0.7
tcpSndBuf If connecting using TCP/IP, should the driver set SO_SND_BUF to the given value? The default value of '0', means use the platform default value for this property) Default: 0 Since version: 5.0.7
tcpTrafficClass If connecting using TCP/IP, should the driver set traffic class or type-of-service fields ?See the documentation for java.net.Socket.setTrafficClass() for more information. Default: 0 Since version: 5.0.7

High Availability and Clustering.

Properties and Descriptions
autoReconnect Should the driver try to re-establish stale and/or dead connections? If enabled the driver will throw an exception for a queries issued on a stale or dead connection, which belong to the current transaction, but will attempt reconnect before the next query issued on the connection in a new transaction. The use of this feature is not recommended, because it has side effects related to session state and data consistency when applications don't handle SQLExceptions properly, and is only designed to be used when you are unable to configure your application to handle SQLExceptions resulting from dead and stale connections properly. Alternatively, as a last option, investigate setting the MySQL server variable "wait_timeout" to a high value, rather than the default of 8 hours. Default: false Since version: 1.1
autoReconnectForPools Use a reconnection strategy appropriate for connection pools (defaults to 'false') Default: false Since version: 3.1.3
failOverReadOnly When failing over in autoReconnect mode, should the connection be set to 'read-only'? Default: true Since version: 3.0.12
maxReconnects Maximum number of reconnects to attempt if autoReconnect is true, default is '3'. Default: 3 Since version: 1.1

Properties and Descriptions
reconnectAtTxEnd If <code>autoReconnect</code> is set to true, should the driver attempt reconnections at the end of every transaction? Default: false Since version: 3.0.10
retriesAllDown When using loadbalancing or failover, the number of times the driver should cycle through available hosts, attempting to connect. Between cycles, the driver will pause for 250ms if no servers are available. Default: 120 Since version: 5.1.6
initialTimeout If <code>autoReconnect</code> is enabled, the initial time to wait between re-connect attempts (in seconds, defaults to '2'). Default: 2 Since version: 1.1
roundRobinLoadBalance When <code>autoReconnect</code> is enabled, and <code>failoverReadOnly</code> is false, should we pick hosts to connect to on a round-robin basis? Default: false Since version: 3.1.2
queriesBeforeRetryMaster Number of queries to issue before falling back to the primary host when failed over (when using multi-host failover). Whichever condition is met first, ' <code>queriesBeforeRetryMaster</code> ' or ' <code>secondsBeforeRetryMaster</code> ' will cause an attempt to be made to reconnect to the primary host. Setting both properties to 0 disables the automatic fall back to the primary host at transaction boundaries. Defaults to 50. Default: 50 Since version: 3.0.2
secondsBeforeRetryMaster How long should the driver wait, when failed over, before attempting to reconnect to the primary host? Whichever condition is met first, ' <code>queriesBeforeRetryMaster</code> ' or ' <code>secondsBeforeRetryMaster</code> ' will cause an attempt to be made to reconnect to the master. Setting both properties to 0 disables the automatic fall back to the primary host at transaction boundaries. Time in seconds, defaults to 30 Default: 30 Since version: 3.0.2
allowMasterDownConnections

Properties and Descriptions

By default, a replication-aware connection will fail to connect when configured master hosts are all unavailable at initial connection. Setting this property to 'true' allows to establish the initial connection, by failing over to the slave servers, in read-only state. It won't prevent subsequent failures when switching back to the master hosts i.e. by setting the replication connection to read/write state.

Default: false

Since version: 5.1.27

allowSlaveDownConnections

By default, a replication-aware connection will fail to connect when configured slave hosts are all unavailable at initial connection. Setting this property to 'true' allows to establish the initial connection. It won't prevent failures when switching to slaves i.e. by setting the replication connection to read-only state. The property 'readFromMasterWhenNoSlaves' should be used for this purpose.

Default: false

Since version: 5.1.38

readFromMasterWhenNoSlaves

Replication-aware connections distribute load by using the master hosts when in read/write state and by using the slave hosts when in read-only state. If, when setting the connection to read-only state, none of the slave hosts are available, an SQLException is thrown back. Setting this property to 'true' allows to fail over to the master hosts, while setting the connection state to read-only, when no slave hosts are available at switch instant.

Default: false

Since version: 5.1.38

replicationEnableJMX

Enables JMX-based management of load-balanced connection groups, including live addition/removal of hosts from load-balancing pool.

Default: false

Since version: 5.1.27

selfDestructOnPingMaxOperations

=If set to a non-zero value, the driver will report close the connection and report failure when Connection.ping() or Connection.isValid(int) is called if the connection's count of commands sent to the server exceeds this value.

Default: 0

Since version: 5.1.6

selfDestructOnPingSecondsLifetime

If set to a non-zero value, the driver will report close the connection and report failure when Connection.ping() or Connection.isValid(int) is called if the connection's lifetime exceeds this value.

Default: 0

Since version: 5.1.6

resourceId

Properties and Descriptions

A globally unique name that identifies the resource that this datasource or connection is connected to, used for `XAResource.isSameRM()` when the driver can't determine this value based on hostnames used in the URL

Since version: 5.0.1

Security.**Properties and Descriptions****allowMultiQueries**

Allow the use of ';' to delimit multiple queries during one statement (true/false), defaults to 'false', and does not affect the `addBatch()` and `executeBatch()` methods, which instead rely on `rewriteBatchStatements`.

Default: false

Since version: 3.1.1

useSSL

Use SSL when communicating with the server (true/false), default is 'true' when connecting to MySQL 5.5.45+, 5.6.26+ or 5.7.6+, otherwise default is 'false'

Default: false

Since version: 3.0.2

requireSSL

Require server support of SSL connection if `useSSL=true`? (defaults to 'false').

Default: false

Since version: 3.1.0

verifyServerCertificate

If "useSSL" is set to "true", should the driver verify the server's certificate? When using this feature, the keystore parameters should be specified by the "clientCertificateKeyStore*" properties, rather than system properties. Default is 'false' when connecting to MySQL 5.5.45+, 5.6.26+ or 5.7.6+ and "useSSL" was not explicitly set to "true". Otherwise default is 'true'

Default: true

Since version: 5.1.6

clientCertificateKeyStoreUrl

URL to the client certificate KeyStore (if not specified, use defaults)

Since version: 5.1.0

clientCertificateKeyStoreType

KeyStore type for client certificates (NULL or empty means use the default, which is "JKS". Standard keystore types supported by the JVM are "JKS" and "PKCS12", your environment may have more available depending on what security products are installed and available to the JVM.

Default: JKS

Since version: 5.1.0

Properties and Descriptions
clientCertificateKeyStorePassword Password for the client certificates KeyStore Since version: 5.1.0
trustCertificateKeyStoreUrl URL to the trusted root certificate KeyStore (if not specified, use defaults) Since version: 5.1.0
trustCertificateKeyStoreType KeyStore type for trusted root certificates (NULL or empty means use the default, which is "JKS". Standard keystore types supported by the JVM are "JKS" and "PKCS12", your environment may have more available depending on what security products are installed and available to the JVM. Default: JKS Since version: 5.1.0
trustCertificateKeyStorePassword Password for the trusted root certificates KeyStore Since version: 5.1.0
enabledSSLCipherSuites If "useSSL" is set to "true", overrides the cipher suites enabled for use on the underlying SSL sockets. This may be required when using external JSSE providers or to specify cipher suites compatible with both MySQL server and used JVM. Since version: 5.1.35
allowLoadLocalInfile Should the driver allow use of 'LOAD DATA LOCAL INFILE...' (defaults to 'true'). Default: true Since version: 3.0.3
allowUrlInLocalInfile Should the driver allow URLs in 'LOAD DATA LOCAL INFILE' statements? Default: false Since version: 3.1.4
allowPublicKeyRetrieval Allows special handshake roundtrip to get server RSA public key directly from server. Default: false Since version: 5.1.31
paranoid Take measures to prevent exposure sensitive information in error messages and clear data structures holding sensitive data when possible? (defaults to 'false')

Properties and Descriptions
Default: false
Since version: 3.0.1
passwordCharacterEncoding
What character encoding is used for passwords? Leaving this set to the default value (null), uses the value set in "characterEncoding" if there is one, otherwise uses UTF-8 as default encoding. If the password contains non-ASCII characters, the password encoding must match what server encoding was set to when the password was created. For passwords in other character encodings, the encoding will have to be specified with this property (or with "characterEncoding"), as it's not possible for the driver to auto-detect this.
Since version: 5.1.7
serverRSAPublicKeyFile
File path to the server RSA public key file for sha256_password authentication. If not specified, the public key will be retrieved from the server.
Since version: 5.1.31

Performance Extensions.

Properties and Descriptions
callableStmtCacheSize
If 'cacheCallableStmts' is enabled, how many callable statements should be cached?
Default: 100
Since version: 3.1.2
metadataCacheSize
The number of queries to cache ResultSetMetadata for if cacheResultSetMetaData is set to 'true' (default 50)
Default: 50
Since version: 3.1.1
useLocalSessionState
Should the driver refer to the internal values of autocommit and transaction isolation that are set by Connection.setAutoCommit() and Connection.setTransactionIsolation() and transaction state as maintained by the protocol, rather than querying the database or blindly sending commands to the database for commit() or rollback() method calls?
Default: false
Since version: 3.1.7
useLocalTransactionState
Should the driver use the in-transaction state provided by the MySQL protocol to determine if a commit() or rollback() should actually be sent to the database?
Default: false
Since version: 5.1.7
prepStmtCacheSize

Properties and Descriptions
If prepared statement caching is enabled, how many prepared statements should be cached? Default: 25 Since version: 3.0.10
prepStmtCacheSqlLimit If prepared statement caching is enabled, what's the largest SQL the driver will cache the parsing for? Default: 256 Since version: 3.0.10
parseInfoCacheFactory Name of a class implementing com.mysql.jdbc.CacheAdapterFactory, which will be used to create caches for the parsed representation of client-side prepared statements. Default: com.mysql.jdbc.PerConnectionLRUFactory Since version: 5.1.1
serverConfigCacheFactory Name of a class implementing com.mysql.jdbc.CacheAdapterFactory<String, Map<String, String>>, which will be used to create caches for MySQL server configuration values Default: com.mysql.jdbc.PerVmServerConfigCacheFactory Since version: 5.1.1
alwaysSendSetIsolation Should the driver always communicate with the database when Connection.setTransactionIsolation() is called? If set to false, the driver will only communicate with the database when the requested transaction isolation is different than the whichever is newer, the last value that was set via Connection.setTransactionIsolation(), or the value that was read from the server when the connection was established. Note that useLocalSessionState=true will force the same behavior as alwaysSendSetIsolation=false, regardless of how alwaysSendSetIsolation is set. Default: true Since version: 3.1.7
maintainTimeStats Should the driver maintain various internal timers to enable idle time calculations as well as more verbose error messages when the connection to the server fails? Setting this property to false removes at least two calls to System.currentTimeMillis() per query. Default: true Since version: 3.1.9
useCursorFetch If connected to MySQL > 5.0.2, and setFetchSize() > 0 on a statement, should that statement use cursor-based fetching to retrieve rows? Default: false Since version: 5.0.0

Properties and Descriptions
blobSendChunkSize Chunk size to use when sending BLOB/CLOBs via ServerPreparedStatements. Note that this value cannot exceed the value of "maxAllowedPacket" and, if that is the case, then this value will be corrected automatically. Default: 1048576 Since version: 3.1.9
cacheCallableStmts Should the driver cache the parsing stage of CallableStatements Default: false Since version: 3.1.2
cachePrepStmts Should the driver cache the parsing stage of PreparedStatements of client-side prepared statements, the "check" for suitability of server-side prepared and server-side prepared statements themselves? Default: false Since version: 3.0.10
cacheResultSetMetadata Should the driver cache ResultSetMetaData for Statements and PreparedStatements? (Req. JDK-1.4+, true/false, default 'false') Default: false Since version: 3.1.1
cacheServerConfiguration Should the driver cache the results of 'SHOW VARIABLES' and 'SHOW COLLATION' on a per-URL basis? Default: false Since version: 3.1.5
defaultFetchSize The driver will call setFetchSize(n) with this value on all newly-created Statements Default: 0 Since version: 3.1.9
dontCheckOnDuplicateKeyUpdateInSQL Stops checking if every INSERT statement contains the "ON DUPLICATE KEY UPDATE" clause. As a side effect, obtaining the statement's generated keys information will return a list where normally it wouldn't. Also be aware that, in this case, the list of generated keys returned may not be accurate. The effect of this property is canceled if set simultaneously with 'rewriteBatchedStatements=true'. Default: false Since version: 5.1.32

Properties and Descriptions**dontTrackOpenResources**

The JDBC specification requires the driver to automatically track and close resources, however if your application doesn't do a good job of explicitly calling close() on statements or result sets, this can cause memory leakage. Setting this property to true relaxes this constraint, and can be more memory efficient for some applications. Also the automatic closing of the Statement and current ResultSet in Statement.closeOnCompletion() and Statement.getMoreResults([Statement.CLOSE_CURRENT_RESULT | Statement.CLOSE_ALL_RESULTS]), respectively, ceases to happen. This property automatically sets holdResultsOpenOverStatementClose=true.

Default: false

Since version: 3.1.7

dynamicCalendars

Should the driver retrieve the default calendar when required, or cache it per connection/session?

Default: false

Since version: 3.1.5

elideSetAutoCommits

If using MySQL-4.1 or newer, should the driver only issue 'set autocommit=n' queries when the server's state doesn't match the requested state by Connection.setAutoCommit(boolean)?

Default: false

Since version: 3.1.3

enableEscapeProcessing

Sets the default escape processing behavior for Statement objects. The method Statement.setEscapeProcessing() can be used to specify the escape processing behavior for an individual Statement object. Default escape processing behavior in prepared statements must be defined with the property 'processEscapeCodesForPrepStmts'.

Default: true

Since version: 5.1.37

enableQueryTimeouts

When enabled, query timeouts set via Statement.setQueryTimeout() use a shared java.util.Timer instance for scheduling. Even if the timeout doesn't expire before the query is processed, there will be memory used by the TimerTask for the given timeout which won't be reclaimed until the time the timeout would have expired if it hadn't been cancelled by the driver. High-load environments might want to consider disabling this functionality.

Default: true

Since version: 5.0.6

holdResultsOpenOverStatementClose

Should the driver close result sets on Statement.close() as required by the JDBC specification?

Default: false

Since version: 3.1.7

largeRowSizeThreshold

Properties and Descriptions What size result set row should the JDBC driver consider "large", and thus use a more memory-efficient way of representing the row internally? Default: 2048 Since version: 5.1.1
loadBalanceStrategy If using a load-balanced connection to connect to SQL nodes in a MySQL Cluster/NDB configuration (by using the URL prefix "jdbc:mysql:loadbalance://"), which load balancing algorithm should the driver use: (1) "random" - the driver will pick a random host for each request. This tends to work better than round-robin, as the randomness will somewhat account for spreading loads where requests vary in response time, while round-robin can sometimes lead to overloaded nodes if there are variations in response times across the workload. (2) "bestResponseTime" - the driver will route the request to the host that had the best response time for the previous transaction. Default: random Since version: 5.0.6
locatorFetchBufferSize If 'emulateLocators' is configured to 'true', what size buffer should be used when fetching BLOB data for getBinaryInputStream? Default: 1048576 Since version: 3.2.1
readOnlyPropagatesToServer Should the driver issue appropriate statements to implicitly set the transaction access mode on server side when Connection.setReadOnly() is called? Setting this property to 'true' enables InnoDB read-only potential optimizations but also requires an extra roundtrip to set the right transaction state. Even if this property is set to 'false', the driver will do its best effort to prevent the execution of database-state-changing queries. Requires minimum of MySQL 5.6. Default: true Since version: 5.1.35
rewriteBatchedStatements Should the driver use multiqueries (irregardless of the setting of "allowMultiQueries") as well as rewriting of prepared statements for INSERT into multi-value inserts when executeBatch() is called? Notice that this has the potential for SQL injection if using plain java.sql.Statements and your code doesn't sanitize input correctly. Notice that for prepared statements, server-side prepared statements can not currently take advantage of this rewrite option, and that if you don't specify stream lengths when using PreparedStatement.set*Stream(), the driver won't be able to determine the optimum number of parameters per batch and you might receive an error from the driver that the resultant packet is too large. Statement.getGeneratedKeys() for these rewritten statements only works when the entire batch includes INSERT statements. Please be aware using rewriteBatchedStatements=true with INSERT .. ON DUPLICATE KEY UPDATE that for rewritten statement server returns only one value as sum of all affected (or found) rows in batch and it isn't possible to map it correctly to initial statements; in this case driver returns 0 as a result of each batch statement if total count was 0, and the Statement.SUCCESS_NO_INFO as a result of each batch statement if total count was > 0. Default: false Since version: 3.1.13

Properties and Descriptions
useDirectRowUnpack Use newer result set row unpacking code that skips a copy from network buffers to a MySQL packet instance and instead reads directly into the result set row data buffers. Default: true Since version: 5.1.1
useDynamicCharsetInfo Should the driver use a per-connection cache of character set information queried from the server when necessary, or use a built-in static mapping that is more efficient, but isn't aware of custom character sets or character sets implemented after the release of the JDBC driver? Default: true Since version: 5.0.6
useFastDateParsing Use internal String->Date/Time/Timestamp conversion routines to avoid excessive object creation? This is part of the legacy date-time code, thus the property has an effect only when "useLegacyDatetimeCode=true." Default: true Since version: 5.0.5
useFastIntParsing Use internal String->Integer conversion routines to avoid excessive object creation? Default: true Since version: 3.1.4
useJvmCharsetConverters Always use the character encoding routines built into the JVM, rather than using lookup tables for single-byte character sets? Default: false Since version: 5.0.1
useReadAheadInput Use newer, optimized non-blocking, buffered input stream when reading from the server? Default: true Since version: 3.1.5

Debugging/Profiling.

Properties and Descriptions
logger The name of a class that implements "com.mysql.jdbc.log.Log" that will be used to log messages to. (default is "com.mysql.jdbc.log.StandardLogger", which logs to STDERR) Default: com.mysql.jdbc.log.StandardLogger

Properties and Descriptions
Since version: 3.1.1
gatherPerfMetrics Should the driver gather performance metrics, and report them via the configured logger every 'reportMetricsIntervalMillis' milliseconds? Default: false Since version: 3.1.2
profileSQL Trace queries and their execution/fetch times to the configured logger (true/false) defaults to 'false' Default: false Since version: 3.1.0
profileSql Deprecated, use 'profileSQL' instead. Trace queries and their execution/fetch times on STDERR (true/false) defaults to 'false' Since version: 2.0.14
reportMetricsIntervalMillis If 'gatherPerfMetrics' is enabled, how often should they be logged (in ms)? Default: 30000 Since version: 3.1.2
maxQuerySizeToLog Controls the maximum length/size of a query that will get logged when profiling or tracing Default: 2048 Since version: 3.1.3
packetDebugBufferSize The maximum number of packets to retain when 'enablePacketDebug' is true Default: 20 Since version: 3.1.3
slowQueryThresholdMillis If 'logSlowQueries' is enabled, how long should a query (in ms) before it is logged as 'slow'? Default: 2000 Since version: 3.1.2
slowQueryThresholdNanos If 'useNanosForElapsedTime' is set to true, and this property is set to a non-zero value, the driver will use this threshold (in nanosecond units) to determine if a query was slow. Default: 0

Properties and Descriptions
Since version: 5.0.7 useUsageAdvisor Should the driver issue 'usage' warnings advising proper and efficient usage of JDBC and MySQL Connector/J to the log (true/false, defaults to 'false')? Default: false Since version: 3.1.1
autoGenerateTestcaseScript Should the driver dump the SQL it is executing, including server-side prepared statements to STDERR? Default: false Since version: 3.1.9
autoSlowLog Instead of using <code>slowQueryThreshold*</code> to determine if a query is slow enough to be logged, maintain statistics that allow the driver to determine queries that are outside the 99th percentile? Default: true Since version: 5.1.4
clientInfoProvider The name of a class that implements the <code>com.mysql.jdbc.JDBC4ClientInfoProvider</code> interface in order to support JDBC-4.0's <code>Connection.get/setClientInfo()</code> methods Default: <code>com.mysql.jdbc.JDBC4CommentClientInfoProvider</code> Since version: 5.1.0
dumpMetadataOnColumnNotFound Should the driver dump the field-level metadata of a result set into the exception message when <code>ResultSet.findColumn()</code> fails? Default: false Since version: 3.1.13
dumpQueriesOnException Should the driver dump the contents of the query sent to the server in the message for <code>SQLExceptions</code>? Default: false Since version: 3.1.3
enablePacketDebug When enabled, a ring-buffer of '<code>packetDebugBufferSize</code>' packets will be kept, and dumped when exceptions are thrown in key areas in the driver's code Default: false Since version: 3.1.3

Properties and Descriptions
explainSlowQueries If 'logSlowQueries' is enabled, should the driver automatically issue an 'EXPLAIN' on the server and send the results to the configured log at a WARN level? Default: false Since version: 3.1.2
includeInnodbStatusInDeadlockExceptions Include the output of "SHOW ENGINE INNODB STATUS" in exception messages when deadlock exceptions are detected? Default: false Since version: 5.0.7
includeThreadDumpInDeadlockExceptions Include a current Java thread dump in exception messages when deadlock exceptions are detected? Default: false Since version: 5.1.15
includeThreadNamesAsStatementComment Include the name of the current thread as a comment visible in "SHOW PROCESSLIST", or in Innodb deadlock dumps, useful in correlation with "includeInnodbStatusInDeadlockExceptions=true" and "includeThreadDumpInDeadlockExceptions=true". Default: false Since version: 5.1.15
logSlowQueries Should queries that take longer than 'slowQueryThresholdMillis' be logged? Default: false Since version: 3.1.2
logXaCommands Should the driver log XA commands sent by MysqlXaConnection to the server, at the DEBUG level of logging? Default: false Since version: 5.0.5
profilerEventHandler Name of a class that implements the interface com.mysql.jdbc.profiler.ProfilerEventHandler that will be used to handle profiling/tracing events. Default: com.mysql.jdbc.profiler.LoggingProfilerEventHandler Since version: 5.1.6
resultSetSizeThreshold

Properties and Descriptions

If the usage advisor is enabled, how many rows should a result set contain before the driver warns that it is suspiciously large?

Default: 100

Since version: 5.0.5

traceProtocol

Should trace-level network protocol be logged?

Default: false

Since version: 3.1.2

useNanosForElapsedTime

For profiling/debugging functionality that measures elapsed time, should the driver try to use nanoseconds resolution if available (JDK >= 1.5)?

Default: false

Since version: 5.0.7

Miscellaneous.**Properties and Descriptions****useUnicode**

Should the driver use Unicode character encodings when handling strings? Should only be used when the driver can't determine the character set mapping, or you are trying to 'force' the driver to use a character set that MySQL either doesn't natively support (such as UTF-8), true/false, defaults to 'true'

Default: true

Since version: 1.1g

characterEncoding

If 'useUnicode' is set to true, what character encoding should the driver use when dealing with strings? (defaults is to 'autodetect')

Since version: 1.1g

characterSetResults

Character set to tell the server to return results as.

Since version: 3.0.13

connectionAttributes

A comma-delimited list of user-defined key:value pairs (in addition to standard MySQL-defined key:value pairs) to be passed to MySQL Server for display as connection attributes in the PERFORMANCE_SCHEMA.SESSION_CONNECT_ATTRS table. Example usage: connectionAttributes=key1:value1,key2:value2 This functionality is available for use with MySQL Server version 5.6 or later only. Earlier versions of MySQL Server do not support connection attributes, causing this configuration option to be ignored. Setting connectionAttributes=none will cause connection attribute processing to be bypassed, for situations where Connection creation/initialization speed is critical.

Since version: 5.1.25

Properties and Descriptions
connectionCollation If set, tells the server to use this collation via 'set collation_connection' Since version: 3.0.13
useBlobToStoreUTF8OutsideBMP Tells the driver to treat [MEDIUM/LONG]BLOB columns as [LONG]VARCHAR columns holding text encoded in UTF-8 that has characters outside the BMP (4-byte encodings), which MySQL server can't handle natively. Default: false Since version: 5.1.3
utf8OutsideBmpExcludedColumnNamePattern When "useBlobToStoreUTF8OutsideBMP" is set to "true", column names matching the given regex will still be treated as BLOBs unless they match the regex specified for "utf8OutsideBmpIncludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package. Since version: 5.1.3
utf8OutsideBmpIncludedColumnNamePattern Used to specify exclusion rules to "utf8OutsideBmpExcludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package. Since version: 5.1.3
loadBalanceEnableJMX Enables JMX-based management of load-balanced connection groups, including live addition/removal of hosts from load-balancing pool. Default: false Since version: 5.1.13
loadBalanceHostRemovalGracePeriod Sets the grace period to wait for a host being removed from a load-balanced connection, to be released when it is currently the active host. Default: 15000 Since version: 5.1.39
sessionVariables A comma-separated list of name/value pairs to be sent as SET SESSION ... to the server when the driver connects. Since version: 3.1.8
useColumnNamesInFindColumn Prior to JDBC-4.0, the JDBC specification had a bug related to what could be given as a "column name" to ResultSet methods like findColumn(), or getters that took a String property. JDBC-4.0 clarified "column name" to mean the label, as given in an "AS" clause and returned by ResultSetMetaData.getColumnLabel(), and if no AS clause, the column name. Setting this property to

Properties and Descriptions
"true" will give behavior that is congruent to JDBC-3.0 and earlier versions of the JDBC specification, but which because of the specification bug could give unexpected results. This property is preferred over "useOldAliasMetadataBehavior" unless you need the specific behavior that it provides with respect to ResultSetMetadata. Default: false Since version: 5.1.7
allowNanAndInf Should the driver allow NaN or +/- INF values in PreparedStatement.setDouble()? Default: false Since version: 3.1.5
autoClosePStmtStreams Should the driver automatically call .close() on streams/readers passed as arguments via set*() methods? Default: false Since version: 3.1.12
autoDeserialize Should the driver automatically detect and de-serialize objects stored in BLOB fields? Default: false Since version: 3.1.5
blobsAreStrings Should the driver always treat BLOBs as Strings - specifically to work around dubious metadata returned by the server for GROUP BY clauses? Default: false Since version: 5.0.8
cacheDefaultTimezone Caches client's default time zone. This results in better performance when dealing with time zone conversions in Date and Time data types, however it won't be aware of time zone changes if they happen at runtime. Default: true Since version: 5.1.35
capitalizeTypeNames Capitalize type names in DatabaseMetaData? (usually only useful when using WebObjects, true/false, defaults to 'false') Default: true Since version: 2.0.7
clobCharacterEncoding

Properties and Descriptions
The character encoding to use for sending and retrieving TEXT, MEDIUMTEXT and LONGTEXT values instead of the configured connection characterEncoding Since version: 5.0.0
clobberStreamingResults This will cause a 'streaming' ResultSet to be automatically closed, and any outstanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server. Default: false Since version: 3.0.9
compensateOnDuplicateKeyUpdateCounts Should the driver compensate for the update counts of "ON DUPLICATE KEY" INSERT statements (2 = 1, 0 = 1) when using prepared statements? Default: false Since version: 5.1.7
continueBatchOnError Should the driver continue processing batch commands if one statement fails. The JDBC spec allows either way (defaults to 'true'). Default: true Since version: 3.0.3
createDatabaseIfNotExist Creates the database given in the URL if it doesn't yet exist. Assumes the configured user has permissions to create databases. Default: false Since version: 3.1.9
detectCustomCollations Should the driver detect custom charsets/collations installed on server (true/false, defaults to 'false'). If this option set to 'true' driver gets actual charsets/collations from server each time connection establishes. This could slow down connection initialization significantly. Default: false Since version: 5.1.29
emptyStringsConvertToZero Should the driver allow conversions from empty string fields to numeric values of '0'? Default: true Since version: 3.1.8
emulateLocators Should the driver emulate java.sql.Blobs with locators? With this feature enabled, the driver will delay loading the actual Blob data until the one of the retrieval methods (getInputStream(), getBytes(),

Properties and Descriptions

and so forth) on the blob data stream has been accessed. For this to work, you must use a column alias with the value of the column to the actual name of the Blob. The feature also has the following restrictions: The SELECT that created the result set must reference only one table, the table must have a primary key; the SELECT must alias the original blob column name, specified as a string, to an alternate name; the SELECT must cover all columns that make up the primary key.

Default: false

Since version: 3.1.0

emulateUnsupportedPstmts

Should the driver detect prepared statements that are not supported by the server, and replace them with client-side emulated versions?

Default: true

Since version: 3.1.7

exceptionInterceptors

Comma-delimited list of classes that implement com.mysql.jdbc.ExceptionInterceptor. These classes will be instantiated one per Connection instance, and all SQLExceptions thrown by the driver will be allowed to be intercepted by these interceptors, in a chained fashion, with the first class listed as the head of the chain.

Since version: 5.1.8

functionsNeverReturnBlobs

Should the driver always treat data from functions returning BLOBs as Strings - specifically to work around dubious metadata returned by the server for GROUP BY clauses?

Default: false

Since version: 5.0.8

generateSimpleParameterMetadata

Should the driver generate simplified parameter metadata for PreparedStatement when no metadata is available either because the server couldn't support preparing the statement, or server-side prepared statements are disabled?

Default: false

Since version: 5.0.5

getProceduresReturnsFunctions

Pre-JDBC4 DatabaseMetaData API has only the getProcedures() and getProcedureColumns() methods, so they return metadata info for both stored procedures and functions. JDBC4 was extended with the getFunctions() and getFunctionColumns() methods and the expected behaviours of previous methods are not well defined. For JDBC4 and higher, default 'true' value of the option means that calls of DatabaseMetaData.getProcedures() and DatabaseMetaData.getProcedureColumns() return metadata for both procedures and functions as before, keeping backward compatibility. Setting this property to 'false' decouples Connector/J from its pre-JDBC4 behaviours for DatabaseMetaData.getProcedures() and DatabaseMetaData.getProcedureColumns(), forcing them to return metadata for procedures only.

Default: true

Since version: 5.1.26

Properties and Descriptions
ignoreNonTxTables Ignore non-transactional table warning for rollback? (defaults to 'false'). Default: false Since version: 3.0.9
jdbcCompliantTruncation Should the driver throw java.sql.DataTruncation exceptions when data is truncated as is required by the JDBC specification when connected to a server that supports warnings (MySQL 4.1.0 and newer)? This property has no effect if the server sql-mode includes STRICT_TRANS_TABLES. Default: true Since version: 3.1.2
loadBalanceAutoCommitStatementRegex When load-balancing is enabled for auto-commit statements (via loadBalanceAutoCommitStatementThreshold), the statement counter will only increment when the SQL matches the regular expression. By default, every statement issued matches. Since version: 5.1.15
loadBalanceAutoCommitStatementThreshold When auto-commit is enabled, the number of statements which should be executed before triggering load-balancing to rebalance. Default value of 0 causes load-balanced connections to only rebalance when exceptions are encountered, or auto-commit is disabled and transactions are explicitly committed or rolled back. Default: 0 Since version: 5.1.15
loadBalanceBlacklistTimeout Time in milliseconds between checks of servers which are unavailable, by controlling how long a server lives in the global blacklist. Default: 0 Since version: 5.1.0
loadBalanceConnectionGroup Logical group of load-balanced connections within a classloader, used to manage different groups independently. If not specified, live management of load-balanced connections is disabled. Since version: 5.1.13
loadBalanceExceptionChecker Fully-qualified class name of custom exception checker. The class must implement com.mysql.jdbc.LoadBalanceExceptionChecker interface, and is used to inspect SQLExceptions and determine whether they should trigger fail-over to another host in a load-balanced deployment. Default: com.mysql.jdbc.StandardLoadBalanceExceptionChecker Since version: 5.1.13
loadBalancePingTimeout

Properties and Descriptions

Time in milliseconds to wait for ping response from each of load-balanced physical connections when using load-balanced Connection.

Default: 0

Since version: 5.1.13

loadBalanceSQLExceptionSubclassFailover

Comma-delimited list of classes/interfaces used by default load-balanced exception checker to determine whether a given SQLException should trigger failover. The comparison is done using `Class.isInstance(SQLException)` using the thrown SQLException.

Since version: 5.1.13

loadBalanceSQLStateFailover

Comma-delimited list of SQLState codes used by default load-balanced exception checker to determine whether a given SQLException should trigger failover. The SQLState of a given SQLException is evaluated to determine whether it begins with any value in the comma-delimited list.

Since version: 5.1.13

loadBalanceValidateConnectionOnSwapServer

Should the load-balanced Connection explicitly check whether the connection is live when swapping to a new physical connection at commit/rollback?

Default: false

Since version: 5.1.13

maxRows

The maximum number of rows to return (0, the default means return all rows).

Default: -1

Since version: all versions

netTimeoutForStreamingResults

What value should the driver automatically set the server setting 'net_write_timeout' to when the streaming result sets feature is in use? (value has unit of seconds, the value '0' means the driver will not try and adjust this value)

Default: 600

Since version: 5.1.0

noAccessToProcedureBodies

When determining procedure parameter types for CallableStatements, and the connected user can't access procedure bodies through "SHOW CREATE PROCEDURE" or select on mysql.proc should the driver instead create basic metadata (all parameters reported as IN VARCHARs, but allowing `registerOutParameter()` to be called on them anyway) instead of throwing an exception?

Default: false

Since version: 5.0.3

noDatetimeStringSync

Don't ensure that `ResultSet.getDatetimeType().toString().equals(ResultSet.getString())`

Properties and Descriptions
Default: false Since version: 3.1.7
noTimezoneConversionForDateType Don't convert DATE values using the server time zone if 'useTimezone'='true' or 'useLegacyDatetimeCode'='false' Default: true Since version: 5.1.35
noTimezoneConversionForTimeType Don't convert TIME values using the server time zone if 'useTimezone'='true' Default: false Since version: 5.0.0
nullCatalogMeansCurrent When DatabaseMetadataMethods ask for a 'catalog' parameter, does the value null mean use the current catalog? (this is not JDBC-compliant, but follows legacy behavior from earlier versions of the driver) Default: true Since version: 3.1.8
nullNamePatternMatchesAll Should DatabaseMetaData methods that accept *pattern parameters treat null the same as '%' (this is not JDBC-compliant, however older versions of the driver accepted this departure from the specification) Default: true Since version: 3.1.8
overrideSupportsIntegrityEnhancementFacility Should the driver return "true" for DatabaseMetaData.supportsIntegrityEnhancementFacility() even if the database doesn't support it to workaround applications that require this method to return "true" to signal support of foreign keys, even though the SQL specification states that this facility contains much more than just foreign key support (one such application being OpenOffice)? Default: false Since version: 3.1.12
padCharsWithSpace If a result set column has the CHAR type and the value does not fill the amount of characters specified in the DDL for the column, should the driver pad the remaining characters with space (for ANSI compliance)? Default: false Since version: 5.0.6
pedantic

Properties and Descriptions Follow the JDBC spec to the letter. Default: false Since version: 3.0.0
pinGlobalTxToPhysicalConnection When using XAConnections, should the driver ensure that operations on a given XID are always routed to the same physical connection? This allows the XAConnection to support "XA START ... JOIN" after "XA END" has been called Default: false Since version: 5.0.1
populateInsertRowWithDefaultValues When using ResultSets that are CONCUR_UPDATABLE, should the driver pre-populate the "insert" row with default values from the DDL for the table used in the query so those values are immediately available for ResultSet accessors? This functionality requires a call to the database for metadata each time a result set of this type is created. If disabled (the default), the default values will be populated by the an internal call to refreshRow() which pulls back default values and/or values changed by triggers. Default: false Since version: 5.0.5
processEscapeCodesForPrepStmts Should the driver process escape codes in queries that are prepared? Default escape processing behavior in non-prepared statements must be defined with the property 'enableEscapeProcessing'. Default: true Since version: 3.1.12
queryTimeoutKillsConnection If the timeout given in Statement.setQueryTimeout() expires, should the driver forcibly abort the Connection instead of attempting to abort the query? Default: false Since version: 5.1.9
relaxAutoCommit If the version of MySQL the driver connects to does not support transactions, still allow calls to commit(), rollback() and setAutoCommit() (true/false, defaults to 'false')? Default: false Since version: 2.0.13
retainStatementAfterResultSetClose Should the driver retain the Statement reference in a ResultSet after ResultSet.close() has been called. This is not JDBC-compliant after JDBC-4.0. Default: false Since version: 3.1.11

Properties and Descriptions
rollbackOnPooledClose Should the driver issue a rollback() when the logical connection in a pool is closed? Default: true Since version: 3.0.15
runningCTS13 Enables workarounds for bugs in Sun's JDBC compliance testsuite version 1.3 Default: false Since version: 3.1.7
sendFractionalSeconds Send fractional part from TIMESTAMP seconds. If set to false, the nanoseconds value of TIMESTAMP values will be truncated before sending any data to the server. This option applies only to prepared statements, callable statements or updatable result sets. Default: true Since version: 5.1.37
serverTimezone Override detection/mapping of time zone. Used when time zone from server doesn't map to Java time zone Since version: 3.0.2
statementInterceptors A comma-delimited list of classes that implement "com.mysql.jdbc.StatementInterceptor" that should be placed "in between" query execution to influence the results. StatementInterceptors are "chainable", the results returned by the "current" interceptor will be passed on to the next in the chain, from left-to-right order, as specified in this property. Since version: 5.1.1
strictFloatingPoint Used only in older versions of compliance test Default: false Since version: 3.0.0
strictUpdates Should the driver do strict checking (all primary keys selected) of updatable result sets (true, false, defaults to 'true')? Default: true Since version: 3.0.4
tinyInt1isBit Should the driver treat the datatype TINYINT(1) as the BIT type (because the server silently converts BIT -> TINYINT(1) when creating tables)?

Properties and Descriptions
Default: true
Since version: 3.0.16
transformedBitIsBoolean
If the driver converts TINYINT(1) to a different type, should it use BOOLEAN instead of BIT for future compatibility with MySQL-5.0, as MySQL-5.0 has a BIT type?
Default: false
Since version: 3.1.9
treatUtilDateAsTimestamp
Should the driver treat java.util.Date as a TIMESTAMP for the purposes of PreparedStatement.setObject()?
Default: true
Since version: 5.0.5
ultraDevHack
Create PreparedStatements for prepareCall() when required, because UltraDev is broken and issues a prepareCall() for _all_ statements? (true/false, defaults to 'false')
Default: false
Since version: 2.0.3
useAffectedRows
Don't set the CLIENT_FOUND_ROWS flag when connecting to the server (not JDBC-compliant, will break most applications that rely on "found" rows vs. "affected rows" for DML statements), but does cause "correct" update counts from "INSERT ... ON DUPLICATE KEY UPDATE" statements to be returned by the server.
Default: false
Since version: 5.1.7
useGmtMillisForDatetimes
Convert between session time zone and GMT before creating Date and Timestamp instances (value of 'false' leads to legacy behavior, 'true' leads to more JDBC-compliant behavior)? This is part of the legacy date-time code, thus the property has an effect only when "useLegacyDatetimeCode=true."
Default: false
Since version: 3.1.12
useHostsInPrivileges
Add '@hostname' to users in DatabaseMetaData.getColumn/TablePrivileges() (true/false), defaults to 'true'.
Default: true
Since version: 3.0.2
useInformationSchema

Properties and Descriptions When connected to MySQL-5.0.7 or newer, should the driver use the INFORMATION_SCHEMA to derive information used by DatabaseMetaData? Default: false Since version: 5.0.0
useJDBCCompliantTimezoneShift Should the driver use JDBC-compliant rules when converting TIME/TIMESTAMP/DATETIME values' time zone information for those JDBC arguments which take a java.util.Calendar argument? This is part of the legacy date-time code, thus the property has an effect only when "useLegacyDatetimeCode=true." Default: false Since version: 5.0.0
useLegacyDatetimeCode Use code for DATE/TIME/DATETIME/TIMESTAMP handling in result sets and statements that consistently handles time zone conversions from client to server and back again, or use the legacy code for these datatypes that has been in the driver for backwards-compatibility? Setting this property to 'false' voids the effects of "useTimezone," "useJDBCCompliantTimezoneShift," "useGmtMillisForDatetimes," and "useFastDateParsing." Default: true Since version: 5.1.6
useOldAliasMetadataBehavior Should the driver use the legacy behavior for "AS" clauses on columns and tables, and only return aliases (if any) for ResultSetMetaData.getColumnNames() or ResultSetMetaData.getTableNames() rather than the original column/table name? In 5.0.x, the default value was true. Default: false Since version: 5.0.4
useOldUTF8Behavior Use the UTF-8 behavior the driver did when communicating with 4.0 and older servers Default: false Since version: 3.1.6
useOnlyServerErrorMessages Don't prepend 'standard' SQLState error messages to error messages returned by the server. Default: true Since version: 3.0.15
useSSPSCCompatibleTimezoneShift If migrating from an environment that was using server-side prepared statements, and the configuration property "useJDBCCompliantTimezoneShift" set to "true", use compatible behavior when not using server-side prepared statements when sending TIMESTAMP values to the MySQL server.

Properties and Descriptions
Default: false Since version: 5.0.5
useServerPrepStmts Use server-side prepared statements if the server supports them? Default: false Since version: 3.1.0
useSqlStateCodes Use SQL Standard state codes instead of 'legacy' X/Open/SQL state codes (true/false), default is 'true' Default: true Since version: 3.1.3
useStreamLengthsInPrepStmts Honor stream length parameter in PreparedStatement/ResultSet.setXXXStream() method calls (true/false, defaults to 'true')? Default: true Since version: 3.0.2
useTimezone Convert time/date types between client and server time zones (true/false, defaults to 'false')? This is part of the legacy date-time code, thus the property has an effect only when "useLegacyDatetimeCode=true." Default: false Since version: 3.0.2
useUnbufferedInput Don't use BufferedInputStream for reading data from the server Default: true Since version: 3.0.11
yearIsDateType Should the JDBC driver treat the MySQL type "YEAR" as a java.sql.Date, or as a SHORT? Default: true Since version: 3.1.9
zeroDateTimeBehavior What should happen when the driver encounters DATETIME values that are composed entirely of zeros (used by MySQL to represent invalid dates)? Valid values are "exception", "round" and "convertToNull". Default: exception

Properties and Descriptions

Since version: 3.1.4

Connector/J also supports access to MySQL using named pipes on Windows platforms with the `NamedPipeSocketFactory` as a plugin-socket factory. If you do not use a `namedPipePath` property, the default of `'\\.\pipe\MySQL'` is used. If you use the `NamedPipeSocketFactory`, the host name and port number values in the JDBC URL are ignored. To enable this feature, set the `socketFactory` property:

```
socketFactory=com.mysql.jdbc.NamedPipeSocketFactory
```

Named pipes only work when connecting to a MySQL server on the same physical machine where the JDBC driver is running. In simple performance tests, named pipe access is between 30%-50% faster than the standard TCP/IP access. However, this varies per system, and named pipes are slower than TCP/IP in many Windows configurations.

To create your own socket factories, follow the example code in `com.mysql.jdbc.NamedPipeSocketFactory`, or `com.mysql.jdbc.StandardSocketFactory`.

2.5.1.1 Properties Files for the `useConfigs` Option

The `useConfigs` connection option is convenient shorthand for specifying combinations of options for particular scenarios. The argument values you can use with this option correspond to the names of `.properties` files within the Connector/J `mysql-connector-java-version-bin.jar` JAR file. For example, the Connector/J 5.1.9 driver includes the following configuration properties files:

```
$ unzip mysql-connector-java-5.1.19-bin.jar '*/configs/*'
Archive:  mysql-connector-java-5.1.19-bin.jar
  creating:  com/mysql/jdbc/configs/
  inflating:  com/mysql/jdbc/configs/3-0-Compat.properties
  inflating:  com/mysql/jdbc/configs/5-0-Compat.properties
  inflating:  com/mysql/jdbc/configs/clusterBase.properties
  inflating:  com/mysql/jdbc/configs/coldFusion.properties
  inflating:  com/mysql/jdbc/configs/fullDebug.properties
  inflating:  com/mysql/jdbc/configs/maxPerformance.properties
  inflating:  com/mysql/jdbc/configs/solarisMaxPerformance.properties
```

To specify one of these combinations of options, specify `useConfigs=3-0-Compat`, `useConfigs=maxPerformance`, and so on. The following sections show the options that are part of each `useConfigs` setting. For the details of why each one is included, see the comments in the `.properties` files.

3-0-Compat

```
emptyStringsConvertToZero=true
jdbcCompliantTruncation=false
noDatetimeStringSync=true
nullCatalogMeansCurrent=true
nullNamePatternMatchesAll=true
transformedBitIsBoolean=false
dontTrackOpenResources=true
zeroDateTimeBehavior=convertToNull
useServerPrepStmts=false
autoClosePstmtStreams=true
processEscapeCodesForPrepStmts=false
useFastDateParsing=false
populateInsertRowWithDefaultValues=false
useDirectRowUnpack=false
```

5-0-Compat

```
useDirectRowUnpack=false
```

clusterBase

```
autoReconnect=true  
failOverReadOnly=false  
roundRobinLoadBalance=true
```

coldFusion

```
useDynamicCharsetInfo=false  
alwaysSendSetIsolation=false  
useLocalSessionState=true  
autoReconnect=true
```

fullDebug

```
profileSQL=true  
gatherPerfMetrics=true  
usageAdvisor=true  
logSlowQueries=true  
explainSlowQueries=true
```

maxPerformance

```
cachePrepStmts=true  
cacheCallableStmts=true  
cacheServerConfiguration=true  
useLocalSessionState=true  
elideSetAutoCommits=true  
alwaysSendSetIsolation=false  
enableQueryTimeouts=false
```

solarisMaxPerformance

```
useUnbufferedInput=false  
useReadAheadInput=false  
maintainTimeStats=false
```

2.5.2 JDBC API Implementation Notes

MySQL Connector/J, as a rigorous implementation of the [JDBC API](#), passes all of the tests in the publicly available version of Oracle's JDBC compliance test suite. The JDBC specification is flexible on how certain functionality should be implemented. This section gives details on an interface-by-interface level about implementation decisions that might affect how you code applications with MySQL Connector/J.

- **BLOB**

Starting with Connector/J version 3.1.0, you can emulate BLOBs with locators by adding the property `emulateLocators=true` to your JDBC URL. Using this method, the driver will delay loading the actual BLOB data until you retrieve the other data and then use retrieval methods (`getInputStream()`, `getBytes()`, and so forth) on the BLOB data stream.

You must use a column alias with the value of the column to the actual name of the BLOB, for example:

```
SELECT id, 'data' as blob_data from blobtable
```

You must also follow these rules:

- The `SELECT` must reference only one table. The table must have a [primary key](#).

- The `SELECT` must alias the original BLOB column name, specified as a string, to an alternate name.
- The `SELECT` must cover all columns that make up the primary key.

The BLOB implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, use the corresponding `PreparedStatement.setBlob()` or `ResultSet.updateBlob()` (in the case of updatable result sets) methods to save changes back to the database.

- **CallableStatement**

Starting with Connector/J 3.1.1, stored procedures are supported when connecting to MySQL version 5.0 or newer using the `CallableStatement` interface. Currently, the `getParameterMetaData()` method of `CallableStatement` is not supported.

- **CLOB**

The CLOB implementation does not allow in-place modification (they are copies, as reported by the `DatabaseMetaData.locatorsUpdateCopies()` method). Because of this, use the `PreparedStatement.setClob()` method to save changes back to the database. The JDBC API does not have a `ResultSet.updateClob()` method.

- **Connection**

Unlike the pre-Connector/J JDBC driver (`MM.MySQL`), the `isClosed()` method does not ping the server to determine if it is available. In accordance with the JDBC specification, it only returns true if `closed()` has been called on the connection. If you need to determine if the connection is still valid, issue a simple query, such as `SELECT 1`. The driver will throw an exception if the connection is no longer valid.

- **DatabaseMetaData**

Foreign key information (`getImportedKeys()/getExportedKeys()` and `getCrossReference()`) is only available from `InnoDB` tables. The driver uses `SHOW CREATE TABLE` to retrieve this information, so if any other storage engines add support for foreign keys, the driver would transparently support them as well.

- **PreparedStatement**

PreparedStatements are implemented by the driver, as MySQL does not have a prepared statement feature. Because of this, the driver does not implement `getParameterMetaData()` or `getMetaData()` as it would require the driver to have a complete SQL parser in the client.

Starting with version 3.1.0 MySQL Connector/J, server-side prepared statements and binary-encoded result sets are used when the server supports them.

Take care when using a server-side prepared statement with **large** parameters that are set using `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()`, or `setClob()`. To re-execute the statement with any large parameter changed to a nonlarge parameter, call `clearParameters()` and set all parameters again. The reason for this is as follows:

- During both server-side prepared statements and client-side emulation, large data is exchanged only when `PreparedStatement.execute()` is called.
- Once that has been done, the stream used to read the data on the client side is closed (as per the JDBC spec), and cannot be read from again.

- If a parameter changes from large to nonlarge, the driver must reset the server-side state of the prepared statement to allow the parameter that is being changed to take the place of the prior large value. This removes all of the large data that has already been sent to the server, thus requiring the data to be re-sent, using the `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()` or `setClob()` method.

Consequently, to change the type of a parameter to a nonlarge one, you must call `clearParameters()` and set all parameters of the prepared statement again before it can be re-executed.

- **ResultSet**

By default, ResultSets are completely retrieved and stored in memory. In most cases this is the most efficient way to operate and, due to the design of the MySQL network protocol, is easier to implement. If you are working with ResultSets that have a large number of rows or large values and cannot allocate heap space in your JVM for the memory required, you can tell the driver to stream the results back one row at a time.

To enable this functionality, create a `Statement` instance in the following manner:

```
stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,  
                             java.sql.ResultSet.CONCUR_READ_ONLY);  
stmt.setFetchSize(Integer.MIN_VALUE);
```

The combination of a forward-only, read-only result set, with a fetch size of `Integer.MIN_VALUE` serves as a signal to the driver to stream result sets row-by-row. After this, any result sets created with the statement will be retrieved row-by-row.

There are some caveats with this approach. You must read all of the rows in the result set (or close it) before you can issue any other queries on the connection, or an exception will be thrown.

The earliest the locks these statements hold can be released (whether they be `MyISAM` table-level locks or row-level locks in some other storage engine such as `InnoDB`) is when the statement completes.

If the statement is within scope of a transaction, then locks are released when the transaction completes (which implies that the statement needs to complete first). As with most other databases, statements are not complete until all the results pending on the statement are read or the active result set for the statement is closed.

Therefore, if using streaming results, process them as quickly as possible if you want to maintain concurrent access to the tables referenced by the statement producing the result set.

- **ResultSetMetaData**

The `isAutoIncrement()` method only works when using MySQL servers 4.0 and newer.

- **Statement**

When using versions of the JDBC driver earlier than 3.2.1, and connected to server versions earlier than 5.0.3, the `setFetchSize()` method has no effect, other than to toggle result set streaming as described above.

Connector/J 5.0.0 and later include support for both `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL 5.0.0 or newer server, and require a separate connection to issue the `KILL QUERY` statement. In the case of `setQueryTimeout()`, the implementation creates an additional thread to handle the timeout functionality.

Note

Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeException` rather than failing silently, as there is currently no way to unblock the thread that is executing the query being cancelled due to timeout expiration and have it throw the exception instead.

Note

The MySQL statement `KILL QUERY` (which is what the driver uses to implement `Statement.cancel()`) is non-deterministic; thus, avoid the use of `Statement.cancel()` if possible. If no query is in process, the next query issued will be killed by the server. This race condition is guarded against as of Connector/J 5.1.18.

MySQL does not support SQL cursors, and the JDBC driver doesn't emulate them, so `setCursorName()` has no effect.

Connector/J 5.1.3 and later include two additional methods:

- `setLocalInfileInputStream()` sets an `InputStream` instance that will be used to send data to the MySQL server for a `LOAD DATA LOCAL INFILE` statement rather than a `FileInputStream` or `URLInputStream` that represents the path given as an argument to the statement.

This stream will be read to completion upon execution of a `LOAD DATA LOCAL INFILE` statement, and will automatically be closed by the driver, so it needs to be reset before each call to `execute*()` that would cause the MySQL server to request data to fulfill the request for `LOAD DATA LOCAL INFILE`.

If this value is set to `NULL`, the driver will revert to using a `FileInputStream` or `URLInputStream` as required.

- `getLocalInfileInputStream()` returns the `InputStream` instance that will be used to send data in response to a `LOAD DATA LOCAL INFILE` statement.

This method returns `NULL` if no such stream has been set using `setLocalInfileInputStream()`.

2.5.3 Java, JDBC and MySQL Types

MySQL Connector/J is flexible in the way it handles conversions between MySQL data types and Java data types.

In general, any MySQL data type can be converted to a `java.lang.String`, and any numeric type can be converted to any of the Java numeric types, although round-off, overflow, or loss of precision may occur.

Note

All `TEXT` types return `Types.LONGVARCHAR` with different `getPrecision()` values (65535, 255, 16777215, and 2147483647 respectively) with `getColumnType()` returning `-1`. This behavior is intentional even though `TINYTEXT` does not fall, regarding to its size, within the `LONGVARCHAR` category. This is to avoid different handling inside the same base type. And `getColumnType()` returns `-1` because the internal server handling is of type `TEXT`, which is similar to `BLOB`.

Also note that `getColumnTypeName()` will return `VARCHAR` even though `getColumnType()` returns `Types.LONGVARCHAR`, because `VARCHAR` is the designated column database-specific name for this type.

Starting with Connector/J 3.1.0, the JDBC driver issues warnings or throws `DataTruncation` exceptions as is required by the JDBC specification unless the connection was configured not to do so by using the property `jdbcCompliantTruncation` and setting it to `false`.

The conversions that are always guaranteed to work are listed in the following table. The first column lists one or more MySQL data types, and the second column lists one or more Java types to which the MySQL types can be converted.

Table 2.3 Connection Properties - Miscellaneous

These MySQL Data Types	Can always be converted to these Java types
CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET	java.lang.String, java.io.InputStream, java.io.Reader, java.sql.Blob, java.sql.Clob
FLOAT, REAL, DOUBLE PRECISION, NUMERIC, DECIMAL, TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT	java.lang.String, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Double, java.math.BigDecimal
DATE, TIME, DATETIME, TIMESTAMP	java.lang.String, java.sql.Date, java.sql.Timestamp

Note

Round-off, overflow or loss of precision may occur if you choose a Java numeric data type that has less precision or capacity than the MySQL data type you are converting to/from.

The `ResultSet.getObject()` method uses the type conversions between MySQL and Java types, following the JDBC specification where appropriate. The value returned by `ResultSetMetaData.getColumnClassName()` is also shown below. For more information on the JDBC types, see the reference on the `java.sql.Types` class.

Table 2.4 MySQL Types to Java Types for ResultSet.getObject()

MySQL Type Name	Return value of <code>getColumnClassName</code>	Returned as Java Class
BIT(1) (new in MySQL-5.0)	BIT	java.lang.Boolean
BIT(> 1) (new in MySQL-5.0)	BIT	byte[]
TINYINT	TINYINT	java.lang.Boolean if the configuration property <code>tinyInt1isBit</code> is set to <code>true</code> (the default) and the storage size is 1, or <code>java.lang.Integer</code> if not.
BOOL, BOOLEAN	TINYINT	See TINYINT, above as these are aliases for TINYINT(1), currently.
SMALLINT[(M)] [UNSIGNED]	SMALLINT [UNSIGNED]	java.lang.Integer (regardless of whether it is UNSIGNED or not)
MEDIUMINT[(M)] [UNSIGNED]	MEDIUMINT [UNSIGNED]	java.lang.Integer (regardless of whether it is UNSIGNED or not)
INT, INTEGER[(M)] [UNSIGNED]	INTEGER [UNSIGNED]	java.lang.Integer, if UNSIGNED java.lang.Long

MySQL Type Name	Return value of <code>getColumnClassName</code>	Returned as Java Class
<code>BIGINT[(M)] [UNSIGNED]</code>	<code>BIGINT [UNSIGNED]</code>	<code>java.lang.Long</code> , if <code>UNSIGNED</code> <code>java.math.BigInteger</code>
<code>FLOAT[(M,D)]</code>	<code>FLOAT</code>	<code>java.lang.Float</code>
<code>DOUBLE[(M,B)]</code>	<code>DOUBLE</code>	<code>java.lang.Double</code>
<code>DECIMAL[(M[,D])]</code>	<code>DECIMAL</code>	<code>java.math.BigDecimal</code>
<code>DATE</code>	<code>DATE</code>	<code>java.sql.Date</code>
<code>DATETIME</code>	<code>DATETIME</code>	<code>java.sql.Timestamp</code>
<code>TIMESTAMP[(M)]</code>	<code>TIMESTAMP</code>	<code>java.sql.Timestamp</code>
<code>TIME</code>	<code>TIME</code>	<code>java.sql.Time</code>
<code>YEAR[(2 4)]</code>	<code>YEAR</code>	If <code>yearIsDateType</code> configuration property is set to <code>false</code> , then the returned object type is <code>java.sql.Short</code> . If set to <code>true</code> (the default), then the returned object is of type <code>java.sql.Date</code> with the date set to January 1st, at midnight.
<code>CHAR(M)</code>	<code>CHAR</code>	<code>java.lang.String</code> (unless the character set for the column is <code>BINARY</code> , then <code>byte[]</code> is returned.
<code>VARCHAR(M) [BINARY]</code>	<code>VARCHAR</code>	<code>java.lang.String</code> (unless the character set for the column is <code>BINARY</code> , then <code>byte[]</code> is returned.
<code>BINARY(M)</code>	<code>BINARY</code>	<code>byte[]</code>
<code>VARBINARY(M)</code>	<code>VARBINARY</code>	<code>byte[]</code>
<code>TINYBLOB</code>	<code>TINYBLOB</code>	<code>byte[]</code>
<code>TINYTEXT</code>	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>BLOB</code>	<code>BLOB</code>	<code>byte[]</code>
<code>TEXT</code>	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>MEDIUMBLOB</code>	<code>MEDIUMBLOB</code>	<code>byte[]</code>
<code>MEDIUMTEXT</code>	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>LONGBLOB</code>	<code>LONGBLOB</code>	<code>byte[]</code>
<code>LONGTEXT</code>	<code>VARCHAR</code>	<code>java.lang.String</code>
<code>ENUM('value1', 'value2', ...)</code>	<code>CHAR</code>	<code>java.lang.String</code>
<code>SET('value1', 'value2', ...)</code>	<code>CHAR</code>	<code>java.lang.String</code>

2.5.4 Using Character Sets and Unicode

All strings sent from the JDBC driver to the server are converted automatically from native Java Unicode form to the client character encoding, including all queries sent using `Statement.execute()`, `Statement.executeUpdate()`, `Statement.executeQuery()` as well as all `PreparedStatement` and `CallableStatement` parameters with the exclusion of parameters set using `setBytes()`, `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()` and `setBlob()`.

Number of Encodings Per Connection

In MySQL Server 4.1 and higher, Connector/J supports a single character encoding between client and server, and any number of character encodings for data returned by the server to the client in `ResultSet`.

Prior to MySQL Server 4.1, Connector/J supported a single character encoding per connection, which could either be automatically detected from the server configuration, or could be configured by the user through the [useUnicode](#) and [characterEncoding](#) properties.

Setting the Character Encoding

The character encoding between client and server is automatically detected upon connection. You specify the encoding on the server using the [character_set_server](#) for server versions 4.1.0 and newer, and [character_set](#) system variable for server versions older than 4.1.0. The driver automatically uses the encoding specified by the server. For more information, see [Server Character Set and Collation](#).

For example, to use 4-byte UTF-8 character sets with Connector/J, configure the MySQL server with [character_set_server=utf8mb4](#), and leave [characterEncoding](#) out of the Connector/J connection string. Connector/J will then autodetect the UTF-8 setting.

To override the automatically detected encoding on the client side, use the [characterEncoding](#) property in the URL used to connect to the server.

To allow multiple character sets to be sent from the client, use the UTF-8 encoding, either by configuring [utf8](#) as the default server character set, or by configuring the JDBC driver to use UTF-8 through the [characterEncoding](#) property.

When specifying character encodings on the client side, use Java-style names. The following table lists MySQL character set names and the corresponding Java-style names:

Table 2.5 MySQL to Java Encoding Name Translations

MySQL Character Set Name	Java-Style Character Encoding Name
ascii	US-ASCII
big5	Big5
gbk	GBK
sjis	SJIS (or Cp932 or MS932 for MySQL Server < 4.1.11)
cp932	Cp932 or MS932 (MySQL Server > 4.1.11)
gb2312	EUC_CN
ujis	EUC_JP
euckr	EUC_KR
latin1	Cp1252
latin2	ISO8859_2
greek	ISO8859_7
hebrew	ISO8859_8
cp866	Cp866
tis620	TIS620
cp1250	Cp1250
cp1251	Cp1251
cp1257	Cp1257
macroman	MacRoman
macce	MacCentralEurope
utf8	UTF-8
ucs2	UnicodeBig

Warning

Do not issue the query `set names` with Connector/J, as the driver will not detect that the character set has changed, and will continue to use the character set detected during the initial connection setup.

2.5.5 Connecting Securely Using SSL

SSL in MySQL Connector/J encrypts all data (other than the initial handshake) between the JDBC driver and the server. There is a performance penalty for enabling SSL, the severity of which depends on multiple factors including (but not limited to) the size of the query, the amount of data returned, the server hardware, the SSL library used, the network bandwidth, and so on.

For SSL support to work, you must have the following:

- A JDK that includes JSSE (Java Secure Sockets Extension), like JDK-1.4.1 or newer. SSL does not currently work with a JDK that you can add JSSE to, like JDK-1.2.x or JDK-1.3.x due to the following JSSE bug: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=4273544
- A MySQL server that supports SSL and has been compiled and configured to do so, which is MySQL 4.0.4 or later. For more information, see [Building MySQL with Support for Secure Connections](#).
- A client certificate (covered later in this section)

The system works through two Java truststore files, one file contains the certificate information for the server (`truststore` in the examples below). The other file contains the certificate for the client (`keystore` in the examples below). All Java truststore files are password protected by supplying a suitable password to the `keytool` when you create the files. You need the file names and associated passwords to create an SSL connection.

You will first need to import the MySQL server CA Certificate into a Java truststore. A sample MySQL server CA Certificate is located in the `SSL` subdirectory of the MySQL source distribution. This is what SSL will use to determine if you are communicating with a secure MySQL server. Alternatively, use the CA Certificate that you have generated or been provided with by your SSL provider.

To use Java's `keytool` to create a truststore in the current directory, and import the server's CA certificate (`cacert.pem`), you can do the following (assuming that `keytool` is in your path. The `keytool` is typically located in the `bin` subdirectory of your JDK or JRE):

```
shell> keytool -import -alias mysqlServerCACert \
           -file cacert.pem -keystore truststore
```

Enter the password when prompted for the keystore file. Interaction with `keytool` looks like this:

```
Enter keystore password: *****
Owner: EMAILADDRESS=walrus@example.com, CN=Walrus,
      O=My Company, L=Orenburg, ST=Some-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=Walrus,
        O=My Company, L=Orenburg, ST=Some-State, C=RU
Serial number: 0
Valid from:
  Fri Aug 02 16:55:53 CDT 2002 until: Sat Aug 02 16:55:53 CDT 2003
Certificate fingerprints:
  MD5:  61:91:A0:F2:03:07:61:7A:81:38:66:DA:19:C4:8D:AB
  SHA1: 25:77:41:05:D5:AD:99:8C:14:8C:CA:68:9C:2F:B8:89:C3:34:4D:6C
Trust this certificate? [no]: yes
Certificate was added to keystore
```

You then have two options: either import the client certificate that matches the CA certificate you just imported, or create a new client certificate.

Importing an existing certificate requires the certificate to be in DER format. You can use `openssl` to convert an existing certificate into the new format. For example:

```
shell> openssl x509 -outform DER -in client-cert.pem -out client.cert
```

Now import the converted certificate into your keystore using [keytool](#):

```
shell> keytool -import -file client.cert -keystore keystore -alias mysqlClientCertificate
```

To generate your own client certificate, use [keytool](#) to create a suitable certificate and add it to the [keystore](#) file:

```
shell> keytool -genkey -keyalg rsa \
    -alias mysqlClientCertificate -keystore keystore
```

Keytool will prompt you for the following information, and create a keystore named [keystore](#) in the current directory.

Respond with information that is appropriate for your situation:

```
Enter keystore password: *****
What is your first and last name?
[Unknown]: Matthews
What is the name of your organizational unit?
[Unknown]: Software Development
What is the name of your organization?
[Unknown]: My Company
What is the name of your City or Locality?
[Unknown]: Flossmoor
What is the name of your State or Province?
[Unknown]: IL
What is the two-letter country code for this unit?
[Unknown]: US
Is <CN=Matthews, OU=Software Development, O=My Company,
L=Flossmoor, ST=IL, C=US> correct?
[no]: y
Enter key password for <mysqlClientCertificate>
(RETURN if same as keystore password):
```

Finally, to get JSSE to use the keystore and truststore that you have generated, you need to set the following system properties when you start your JVM, replacing [path_to_keystore_file](#) with the full path to the keystore file you created, [path_to_truststore_file](#) with the path to the truststore file you created, and using the appropriate password values for each property. You can do this either on the command line:

```
-Djavax.net.ssl.keyStore=path\_to\_keystore\_file
-Djavax.net.ssl.keyStorePassword=password
-Djavax.net.ssl.trustStore=path\_to\_truststore\_file
-Djavax.net.ssl.trustStorePassword=password
```

Or you can set the values directly within the application:

```
System.setProperty("javax.net.ssl.keyStore", "path\_to\_keystore\_file");
System.setProperty("javax.net.ssl.keyStorePassword", "password");
System.setProperty("javax.net.ssl.trustStore", "path\_to\_truststore\_file");
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

You will also need to set [useSSL](#) to [true](#) in your connection parameters for MySQL Connector/J, either by adding [useSSL=true](#) to your URL, or by setting the property [useSSL](#) to [true](#) in the [java.util.Properties](#) instance you pass to [DriverManager.getConnection\(\)](#).

You can test that SSL is working by turning on JSSE debugging (as detailed below), and look for the following key events:

```
...
*** ClientHello, v3.1
```

```

RandomCookie: GMT: 1018531834 bytes = { 199, 148, 180, 215, 74, 12, »
    54, 244, 0, 168, 55, 103, 215, 64, 16, 138, 225, 190, 132, 153, 2, »
    217, 219, 239, 202, 19, 121, 78 }
Session ID: {}
Cipher Suites: { 0, 5, 0, 4, 0, 9, 0, 10, 0, 18, 0, 19, 0, 3, 0, 17 }
Compression Methods: { 0 }
***
[write] MD5 and SHA1 hashes: len = 59
0000: 01 00 00 37 03 01 3D B6 90 FA C7 94 B4 D7 4A 0C ...7..=.....J.
0010: 36 F4 00 A8 37 67 D7 40 10 8A E1 BE 84 99 02 D9 6...7g.@.....
0020: DB EF CA 13 79 4E 00 00 10 00 05 00 04 00 09 00 ....yN.....
0030: 0A 00 12 00 13 00 03 00 11 01 00 .....
main, WRITE: SSL v3.1 Handshake, length = 59
main, READ: SSL v3.1 Handshake, length = 74
*** ServerHello, v3.1
RandomCookie: GMT: 1018577560 bytes = { 116, 50, 4, 103, 25, 100, 58, »
    202, 79, 185, 178, 100, 215, 66, 254, 21, 83, 187, 190, 42, 170, 3, »
    132, 110, 82, 148, 160, 92 }
Session ID: {163, 227, 84, 53, 81, 127, 252, 254, 178, 179, 68, 63, »
    182, 158, 30, 11, 150, 79, 170, 76, 255, 92, 15, 226, 24, 17, 177, »
    219, 158, 177, 187, 143}
Cipher Suite: { 0, 5 }
Compression Method: 0
***
%% Created: [Session-1, SSL_RSA_WITH_RC4_128_SHA]
** SSL_RSA_WITH_RC4_128_SHA
[read] MD5 and SHA1 hashes: len = 74
0000: 02 00 00 46 03 01 3D B6 43 98 74 32 04 67 19 64 ...F..=.C.t2.g.d
0010: 3A CA 4F B9 B2 64 D7 42 FE 15 53 BB BE 2A AA 03 :.0..d.B..S.*..
0020: 84 6E 52 94 A0 5C 20 A3 E3 54 35 51 7F FC FE B2 .nR..\ ..T5Q....
0030: B3 44 3F B6 9E 1E 0B 96 4F AA 4C FF 5C 0F E2 18 .D?.....O.L.\...
0040: 11 B1 DB 9E B1 BB 8F 00 05 00 .....
main, READ: SSL v3.1 Handshake, length = 1712
...

```

JSSE provides debugging (to `stdout`) when you set the following system property: - `Djavax.net.debug=all` This will tell you what keystores and truststores are being used, as well as what is going on during the SSL handshake and certificate exchange. It will be helpful when trying to determine what is not working when trying to get an SSL connection to happen.

2.5.6 Connecting Using PAM Authentication

Java applications using Connector/J 5.1.21 and higher can connect to MySQL servers that use the pluggable authentication module (PAM) authentication scheme.

For PAM authentication to work, you must have the following:

- A MySQL server that supports PAM authentication: a commercial distribution of MySQL 5.5.16 or higher. See [The PAM Authentication Plugin](#) for more information. Connector/J implements the same cleartext authentication method as in [The Cleartext Client-Side Authentication Plugin](#).
- SSL capability, as explained in [Section 2.5.5, “Connecting Securely Using SSL”](#). Because the PAM authentication scheme sends the original password to the server, the connection to the server must be encrypted.

PAM authentication support is enabled by default in Connector/J 5.1.21 and up, so no extra configuration is needed.

To disable the PAM authentication feature, specify `mysql_clear_password` (the method) or `com.mysql.jdbc.authentication.MysqlClearPasswordPlugin` (the class name) in the comma-separated list of arguments for the `disabledAuthenticationPlugins` connection option. See [Section 2.5.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”](#) for details about that connection option.

2.5.7 Using Master/Slave Replication with ReplicationConnection

See [Section 2.8.3, “Configuring Master/Slave Replication with Connector/J”](#) for details on the topic.

2.5.8 Mapping MySQL Error Numbers to JDBC SQLState Codes

The table below provides a mapping of the MySQL error numbers to JDBC [SQLState](#) values.

Table 2.6 Mapping of MySQL Error Numbers to SQLStates

MySQL Error Number	MySQL Error Name	Legacy (X/ Open) SQLState	SQL Standard SQLState
1022	ER_DUP_KEY	23000	23000
1037	ER_OUTOFMEMORY	S1001	HY001
1038	ER_OUT_OF_SORTMEMORY	S1001	HY001
1040	ER_CON_COUNT_ERROR	08004	08004
1042	ER_BAD_HOST_ERROR	08004	08S01
1043	ER_HANDSHAKE_ERROR	08004	08S01
1044	ER_DBACCESS_DENIED_ERROR	42000	42000
1045	ER_ACCESS_DENIED_ERROR	28000	28000
1046	ER_NO_DB_ERROR	3D000	3D000
1047	ER_UNKNOWN_COM_ERROR	08S01	08S01
1048	ER_BAD_NULL_ERROR	23000	23000
1049	ER_BAD_DB_ERROR	42000	42000
1050	ER_TABLE_EXISTS_ERROR	42S01	42S01
1051	ER_BAD_TABLE_ERROR	42S02	42S02
1052	ER_NON_UNIQ_ERROR	23000	23000
1053	ER_SERVER_SHUTDOWN	08S01	08S01
1054	ER_BAD_FIELD_ERROR	S0022	42S22
1055	ER_WRONG_FIELD_WITH_GROUP	S1009	42000
1056	ER_WRONG_GROUP_FIELD	S1009	42000
1057	ER_WRONG_SUM_SELECT	S1009	42000
1058	ER_WRONG_VALUE_COUNT	21S01	21S01
1059	ER_TOO_LONG_IDENT	S1009	42000
1060	ER_DUP_FIELDNAME	S1009	42S21
1061	ER_DUP_KEYNAME	S1009	42000
1062	ER_DUP_ENTRY	S1009	23000
1063	ER_WRONG_FIELD_SPEC	S1009	42000
1064	ER_PARSE_ERROR	42000	42000
1065	ER_EMPTY_QUERY	42000	42000
1066	ER_NONUNIQ_TABLE	S1009	42000
1067	ER_INVALID_DEFAULT	S1009	42000
1068	ER_MULTIPLE_PRI_KEY	S1009	42000
1069	ER_TOO_MANY_KEYS	S1009	42000
1070	ER_TOO_MANY_KEY_PARTS	S1009	42000
1071	ER_TOO_LONG_KEY	S1009	42000
1072	ER_KEY_COLUMN_DOES_NOT_EXISTS	S1009	42000

MySQL Error Number	MySQL Error Name	Legacy (X/ Open) SQLState	SQL Standard SQLState
1073	ER_BLOB_USED_AS_KEY	S1009	42000
1074	ER_TOO_BIG_FIELDLENGTH	S1009	42000
1075	ER_WRONG_AUTO_KEY	S1009	42000
1080	ER_FORCING_CLOSE	08S01	08S01
1081	ER_IPSOCK_ERROR	08S01	08S01
1082	ER_NO_SUCH_INDEX	S1009	42S12
1083	ER_WRONG_FIELD_TERMINATORS	S1009	42000
1084	ER_BLOBS_AND_NO_TERMINATED	S1009	42000
1090	ER_CANT_REMOVE_ALL_FIELDS	42000	42000
1091	ER_CANT_DROP_FIELD_OR_KEY	42000	42000
1101	ER_BLOB_CANT_HAVE_DEFAULT	42000	42000
1102	ER_WRONG_DB_NAME	42000	42000
1103	ER_WRONG_TABLE_NAME	42000	42000
1104	ER_TOO_BIG_SELECT	42000	42000
1106	ER_UNKNOWN_PROCEDURE	42000	42000
1107	ER_WRONG_PARAMCOUNT_TO_PROCEDURE	42000	42000
1109	ER_UNKNOWN_TABLE	42S02	42S02
1110	ER_FIELD_SPECIFIED_TWICE	42000	42000
1112	ER_UNSUPPORTED_EXTENSION	42000	42000
1113	ER_TABLE_MUST_HAVE_COLUMNS	42000	42000
1115	ER_UNKNOWN_CHARACTER_SET	42000	42000
1118	ER_TOO_BIG_ROWSIZE	42000	42000
1120	ER_WRONG_OUTER_JOIN	42000	42000
1121	ER_NULL_COLUMN_IN_INDEX	42000	42000
1129	ER_HOST_IS_BLOCKED	08004	HY000
1130	ER_HOST_NOT_PRIVILEGED	08004	HY000
1131	ER_PASSWORD_ANONYMOUS_USER	42000	42000
1132	ER_PASSWORD_NOT_ALLOWED	42000	42000
1133	ER_PASSWORD_NO_MATCH	42000	42000
1136	ER_WRONG_VALUE_COUNT_ON_ROW	21S01	21S01
1138	ER_INVALID_USE_OF_NULL	S1000	42000
1139	ER_REGEXP_ERROR	42000	42000
1140	ER_MIX_OF_GROUP_FUNC_AND_FIELDS	42000	42000
1141	ER_NONEXISTING_GRANT	42000	42000
1142	ER_TABLEACCESS_DENIED_ERROR	42000	42000
1143	ER_COLUMNACCESS_DENIED_ERROR	42000	42000
1144	ER_ILLEGAL_GRANT_FOR_TABLE	42000	42000
1145	ER_GRANT_WRONG_HOST_OR_USER	42000	42000
1146	ER_NO_SUCH_TABLE	42S02	42S02

MySQL Error Number	MySQL Error Name	Legacy (X/ Open) SQLState	SQL Standard SQLState
1147	ER_NONEXISTING_TABLE_GRANT	42000	42000
1148	ER_NOT_ALLOWED_COMMAND	42000	42000
1149	ER_SYNTAX_ERROR	42000	42000
1152	ER_ABORTING_CONNECTION	08S01	08S01
1153	ER_NET_PACKET_TOO_LARGE	08S01	08S01
1154	ER_NET_READ_ERROR_FROM_PIPE	08S01	08S01
1155	ER_NET_FCNTL_ERROR	08S01	08S01
1156	ER_NET_PACKETS_OUT_OF_ORDER	08S01	08S01
1157	ER_NET_UNCOMPRESS_ERROR	08S01	08S01
1158	ER_NET_READ_ERROR	08S01	08S01
1159	ER_NET_READ_INTERRUPTED	08S01	08S01
1160	ER_NET_ERROR_ON_WRITE	08S01	08S01
1161	ER_NET_WRITE_INTERRUPTED	08S01	08S01
1162	ER_TOO_LONG_STRING	42000	42000
1163	ER_TABLE_CANT_HANDLE_BLOB	42000	42000
1164	ER_TABLE_CANT_HANDLE_AUTO_INCREMENT	42000	42000
1166	ER_WRONG_COLUMN_NAME	42000	42000
1167	ER_WRONG_KEY_COLUMN	42000	42000
1169	ER_DUP_UNIQUE	23000	23000
1170	ER_BLOB_KEY_WITHOUT_LENGTH	42000	42000
1171	ER_PRIMARY_CANT_HAVE_NULL	42000	42000
1172	ER_TOO_MANY_ROWS	42000	42000
1173	ER_REQUIRES_PRIMARY_KEY	42000	42000
1176	ER_KEY_DOES_NOT_EXISTS	42000	42000
1177	ER_CHECK_NO_SUCH_TABLE	42000	42000
1178	ER_CHECK_NOT_IMPLEMENTED	42000	42000
1179	ER_CANT_DO_THIS_DURING_AN_TRANSACTION	25000	25000
1184	ER_NEW_ABORTING_CONNECTION	08S01	08S01
1189	ER_MASTER_NET_READ	08S01	08S01
1190	ER_MASTER_NET_WRITE	08S01	08S01
1203	ER_TOO_MANY_USER_CONNECTIONS	42000	42000
1205	ER_LOCK_WAIT_TIMEOUT	40001	40001
1207	ER_READ_ONLY_TRANSACTION	25000	25000
1211	ER_NO_PERMISSION_TO_CREATE_USER	42000	42000
1213	ER_LOCK_DEADLOCK	40001	40001
1216	ER_NO_REFERENCED_ROW	23000	23000
1217	ER_ROW_IS_REFERENCED	23000	23000
1218	ER_CONNECT_TO_MASTER	08S01	08S01
1222	ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT	21000	21000

MySQL Error Number	MySQL Error Name	Legacy (X/ Open) SQLState	SQL Standard SQLState
1226	ER_USER_LIMIT_REACHED	42000	42000
1227	ER_SPECIFIC_ACCESS_DENIED_ERROR	42000	42000
1230	ER_NO_DEFAULT	42000	42000
1231	ER_WRONG_VALUE_FOR_VAR	42000	42000
1232	ER_WRONG_TYPE_FOR_VAR	42000	42000
1234	ER_CANT_USE_OPTION_HERE	42000	42000
1235	ER_NOT_SUPPORTED_YET	42000	42000
1239	ER_WRONG_FK_DEF	42000	42000
1241	ER_OPERAND_COLUMNS	21000	21000
1242	ER_SUBQUERY_NO_1_ROW	21000	21000
1247	ER_ILLEGAL_REFERENCE	42S22	42S22
1248	ER_DERIVED_MUST_HAVE_ALIAS	42000	42000
1249	ER_SELECT_REDUCED	01000	01000
1250	ER_TABLENAME_NOT_ALLOWED_HERE	42000	42000
1251	ER_NOT_SUPPORTED_AUTH_MODE	08004	08004
1252	ER_SPATIAL_CANT_HAVE_NULL	42000	42000
1253	ER_COLLATION_CHARSET_MISMATCH	42000	42000
1261	ER_WARN_TOO_FEW_RECORDS	01000	01000
1262	ER_WARN_TOO_MANY_RECORDS	01000	01000
1263	ER_WARN_NULL_TO_NOTNULL	S1000	01000
1264	ER_WARN_DATA_OUT_OF_RANGE	01000	01000
1265	ER_WARN_DATA_TRUNCATED	01000	01000
1280	ER_WRONG_NAME_FOR_INDEX	42000	42000
1281	ER_WRONG_NAME_FOR_CATALOG	42000	42000
1286	ER_UNKNOWN_STORAGE_ENGINE	42000	42000
1292	ER_TRUNCATED_WRONG_VALUE	22007	22007
1303	ER_SP_NO_RECURSIVE_CREATE	S1000	2F003
1304	ER_SP_ALREADY_EXISTS	42000	42000
1305	ER_SP_DOES_NOT_EXIST	42000	42000
1308	ER_SP_LILABEL_MISMATCH	42000	42000
1309	ER_SP_LABEL_REDEFINE	42000	42000
1310	ER_SP_LABEL_MISMATCH	42000	42000
1311	ER_SP_UNINIT_VAR	01000	01000
1312	ER_SP_BADSELECT	0A000	0A000
1313	ER_SP_BADRETURN	42000	42000
1314	ER_SP_BADSTATEMENT	0A000	0A000
1315	ER_UPDATE_LOG_DEPRECATED_IGNORED	42000	42000
1316	ER_UPDATE_LOG_DEPRECATED_TRANSLATED	42000	42000
1317	ER_QUERY_INTERRUPTED	S1000	70100

MySQL Error Number	MySQL Error Name	Legacy (X/ Open) SQLState	SQL Standard SQLState
1318	ER_SP_WRONG_NO_OF_ARGS	42000	42000
1319	ER_SP_COND_MISMATCH	42000	42000
1320	ER_SP_NORETURN	42000	42000
1321	ER_SP_NORETURNEND	S1000	2F005
1322	ER_SP_BAD_CURSOR_QUERY	42000	42000
1323	ER_SP_BAD_CURSOR_SELECT	42000	42000
1324	ER_SP_CURSOR_MISMATCH	42000	42000
1325	ER_SP_CURSOR_ALREADY_OPEN	24000	24000
1326	ER_SP_CURSOR_NOT_OPEN	24000	24000
1327	ER_SP_UNDECLARED_VAR	42000	42000
1329	ER_SP_FETCH_NO_DATA	S1000	02000
1330	ER_SP_DUP_PARAM	42000	42000
1331	ER_SP_DUP_VAR	42000	42000
1332	ER_SP_DUP_COND	42000	42000
1333	ER_SP_DUP_CURS	42000	42000
1335	ER_SP_SUBSELECT_NYI	0A000	0A000
1336	ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG	0A000	0A000
1337	ER_SP_VARCOND_AFTER_CURSHNDLR	42000	42000
1338	ER_SP_CURSOR_AFTER_HANDLER	42000	42000
1339	ER_SP_CASE_NOT_FOUND	S1000	20000
1365	ER_DIVISION_BY_ZERO	22012	22012
1367	ER_ILLEGAL_VALUE_FOR_TYPE	22007	22007
1370	ER_PROCAccess_DENIED_Error	42000	42000
1397	ER_XAER_NOTA	S1000	XAE04
1398	ER_XAER_INVAL	S1000	XAE05
1399	ER_XAER_RMFAIL	S1000	XAE07
1400	ER_XAER_OUTSIDE	S1000	XAE09
1401	ER_XA_RMERR	S1000	XAE03
1402	ER_XA_RBROLLBACK	S1000	XA100
1403	ER_NONEXISTING_PROC_GRANT	42000	42000
1406	ER_DATA_TOO_LONG	22001	22001
1407	ER_SP_BAD_SQLSTATE	42000	42000
1410	ER_CANT_CREATE_USER_WITH_GRANT	42000	42000
1413	ER_SP_DUP_HANDLER	42000	42000
1414	ER_SP_NOT_VAR_ARG	42000	42000
1415	ER_SP_NO_RESET	0A000	0A000
1416	ER_CANT_CREATE_GEOMETRY_OBJECT	22003	22003
1425	ER_TOO_BIG_SCALE	42000	42000
1426	ER_TOO_BIG_PRECISION	42000	42000

MySQL Error Number	MySQL Error Name	Legacy (X/ Open) SQLState	SQL Standard SQLState
1427	ER_M_BIGGER_THAN_D	42000	42000
1437	ER_TOO_LONG_BODY	42000	42000
1439	ER_TOO_BIG_DISPLAYWIDTH	42000	42000
1440	ER_XAER_DUPID	S1000	XAE08
1441	ER_DATETIME_FUNCTION_OVERFLOW	22008	22008
1451	ER_ROW_IS_REFERENCED_2	23000	23000
1452	ER_NO_REFERENCED_ROW_2	23000	23000
1453	ER_SP_BAD_VAR_SHADOW	42000	42000
1458	ER_SP_WRONG_NAME	42000	42000
1460	ER_SP_NO_AGGREGATE	42000	42000
1461	ER_MAX_PREPARED_STMT_COUNT_REACHED	42000	42000
1463	ER_NON_GROUPING_FIELD_USED	42000	42000
1557	ER_FOREIGN_DUPLICATE_KEY	23000	23000
1568	ER_CANT_CHANGE_TX_ISOLATION	S1000	25001
1582	ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT	42000	42000
1583	ER_WRONG_PARAMETERS_TO_NATIVE_FCT	42000	42000
1584	ER_WRONG_PARAMETERS_TO_STORED_FCT	42000	42000
1586	ER_DUP_ENTRY_WITH_KEY_NAME	23000	23000
1613	ER_XA_RBTIMEOUT	S1000	XA106
1614	ER_XA_RBDEADLOCK	S1000	XA102
1630	ER_FUNC_INEXISTENT_NAME_COLLISION	42000	42000
1641	ER_DUP_SIGNAL_SET	42000	42000
1642	ER_SIGNAL_WARN	01000	01000
1643	ER_SIGNAL_NOT_FOUND	S1000	02000
1645	ER_RESIGNAL_WITHOUT_ACTIVE_HANDLER	S1000	0K000
1687	ER_SPATIAL_MUST_HAVE_GEOM_COL	42000	42000
1690	ER_DATA_OUT_OF_RANGE	22003	22003
1698	ER_ACCESS_DENIED_NO_PASSWORD_ERROR	28000	28000
1701	ER_TRUNCATE_ILLEGAL_FK	42000	42000
1758	ER_DA_INVALID_CONDITION_NUMBER	35000	35000
1761	ER_FOREIGN_DUPLICATE_KEY_WITH_CHILD_INFO	23000	23000
1762	ER_FOREIGN_DUPLICATE_KEY_WITHOUT_CHILD_INFO	23000	23000
1792	ER_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION	S1000	25006
1845	ER_ALTER_OPERATION_NOT_SUPPORTED	0A000	0A000
1846	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON	0A000	0A000
1859	ER_DUP_UNKNOWN_IN_INDEX	23000	23000
1873	ER_ACCESS_DENIED_CHANGE_USER_ERROR	28000	28000
1887	ER_GET_STACKED_DA_WITHOUT_ACTIVE_HANDLER	S1000	0Z002
1903	ER_INVALID_ARGUMENT_FOR_LOGARITHM	S1000	2201E

2.6 JDBC Concepts

This section provides some general JDBC background.

2.6.1 Connecting to MySQL Using the JDBC `DriverManager` Interface

When you are using JDBC outside of an application server, the `DriverManager` class manages the establishment of connections.

Specify to the `DriverManager` which JDBC drivers to try to make Connections with. The easiest way to do this is to use `Class.forName()` on the class that implements the `java.sql.Driver` interface. With MySQL Connector/J, the name of this class is `com.mysql.jdbc.Driver`. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

The following section of Java code shows how you might register MySQL Connector/J from the `main()` method of your application. If testing this code, first read the installation section at [Section 2.3, “Connector/J Installation”](#), to make sure you have connector installed correctly and the `CLASSPATH` set up. Also, ensure that MySQL is configured to accept external TCP/IP connections.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
// Notice, do not import com.mysql.jdbc.*
// or you will have problems!
public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
            // handle the error
        }
    }
}
```

After the driver has been registered with the `DriverManager`, you can obtain a `Connection` instance that is connected to a particular database by calling `DriverManager.getConnection()`:

Example 2.1 Connector/J: Obtaining a connection from the `DriverManager`

If you have not already done so, please review the portion of [Section 2.6.1, “Connecting to MySQL Using the JDBC `DriverManager` Interface”](#) above before working with the example below.

This example shows how you can obtain a `Connection` instance from the `DriverManager`. There are a few different signatures for the `getConnection()` method. Consult the API documentation that comes with your JDK for more specific information on how to use them.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
Connection conn = null;
...
try {
    conn =
        DriverManager.getConnection("jdbc:mysql://localhost/test?" +
                                   "user=minty&password=greatsqldb");
    // Do something with the Connection
    ...
} catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

Once a `Connection` is established, it can be used to create `Statement` and `PreparedStatement` objects, as well as retrieve metadata about the database. This is explained in the following sections.

2.6.2 Using JDBC `Statement` Objects to Execute SQL

`Statement` objects allow you to execute basic SQL queries and retrieve the results through the `ResultSet` class, which is described later.

To create a `Statement` instance, you call the `createStatement()` method on the `Connection` object you have retrieved using one of the `DriverManager.getConnection()` or `DataSource.getConnection()` methods described earlier.

Once you have a `Statement` instance, you can execute a `SELECT` query by calling the `executeQuery(String)` method with the SQL you want to use.

To update data in the database, use the `executeUpdate(String SQL)` method. This method returns the number of rows matched by the update statement, not the number of rows that were modified.

If you do not know ahead of time whether the SQL statement will be a `SELECT` or an `UPDATE/INSERT`, then you can use the `execute(String SQL)` method. This method will return true if the SQL query was a `SELECT`, or false if it was an `UPDATE`, `INSERT`, or `DELETE` statement. If the statement was a `SELECT` query, you can retrieve the results by calling the `getResultSet()` method. If the statement was an `UPDATE`, `INSERT`, or `DELETE` statement, you can retrieve the affected rows count by calling `getUpdateCount()` on the `Statement` instance.

Example 2.2 Connector/J: Using `java.sql.Statement` to execute a `SELECT` query

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
// assume that conn is an already created JDBC connection (see previous examples)
Statement stmt = null;
ResultSet rs = null;
try {
    stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT foo FROM bar");
    // or alternatively, if you don't know ahead of time that
    // the query will be a SELECT...
    if (stmt.execute("SELECT foo FROM bar")) {
        rs = stmt.getResultSet();
    }
    // Now do something with the ResultSet ....
}
catch (SQLException ex){
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
finally {
    // it is a good idea to release
    // resources in a finally{} block
    // in reverse-order of their creation
    // if they are no-longer needed
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException sqlEx) { } // ignore
        rs = null;
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) { } // ignore
    }
}
```

```

        stmt = null;
    }
}

```

2.6.3 Using JDBC `CallableStatements` to Execute Stored Procedures

Starting with MySQL server version 5.0 when used with Connector/J 3.1.1 or newer, the `java.sql.CallableStatement` interface is fully implemented with the exception of the `getParameterMetaData()` method.

For more information on MySQL stored procedures, please refer to [Using Stored Routines \(Procedures and Functions\)](#).

Connector/J exposes stored procedure functionality through JDBC's `CallableStatement` interface.

Note

Current versions of MySQL server do not return enough information for the JDBC driver to provide result set metadata for callable statements. This means that when using `CallableStatement`, `ResultSetMetaData` may return `NULL`.

The following example shows a stored procedure that returns the value of `inOutParam` incremented by 1, and the string passed in using `inputParam` as a `ResultSet`:

Example 2.3 Connector/J: Calling Stored Procedures

```

CREATE PROCEDURE demoSp(IN inputParam VARCHAR(255), \
                        INOUT inOutParam INT)
BEGIN
    DECLARE z INT;
    SET z = inOutParam + 1;
    SET inOutParam = z;
    SELECT inputParam;
    SELECT CONCAT('zyxw', inputParam);
END

```

To use the `demoSp` procedure with Connector/J, follow these steps:

1. Prepare the callable statement by using `Connection.prepareCall()`.

Notice that you have to use JDBC escape syntax, and that the parentheses surrounding the parameter placeholders are not optional:

Example 2.4 Connector/J: Using `Connection.prepareCall()`

```

import java.sql.CallableStatement;
...
//
// Prepare a call to the stored procedure 'demoSp'
// with two parameters
//
// Notice the use of JDBC-escape syntax ({call ...})
//
CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");
cStmt.setString(1, "abcdefg");

```

Note

`Connection.prepareCall()` is an expensive method, due to the metadata retrieval that the driver performs to support output parameters. For performance reasons, minimize unnecessary calls to `Connection.prepareCall()` by reusing `CallableStatement` instances in your code.

2. Register the output parameters (if any exist)

To retrieve the values of output parameters (parameters specified as `OUT` or `INOUT` when you created the stored procedure), JDBC requires that they be specified before statement execution using the various `registerOutputParameter()` methods in the `CallableStatement` interface:

Example 2.5 Connector/J: Registering output parameters

```
import java.sql.Types;
...
//
// Connector/J supports both named and indexed
// output parameters. You can register output
// parameters using either method, as well
// as retrieve output parameters using either
// method, regardless of what method was
// used to register them.
//
// The following examples show how to use
// the various methods of registering
// output parameters (you should of course
// use only one registration per parameter).
//
//
// Registers the second parameter as output, and
// uses the type 'INTEGER' for values returned from
// getObject()
//
cStmt.registerOutParameter(2, Types.INTEGER);
//
// Registers the named parameter 'inOutParam', and
// uses the type 'INTEGER' for values returned from
// getObject()
//
cStmt.registerOutParameter("inOutParam", Types.INTEGER);
...
```

3. Set the input parameters (if any exist)

Input and in/out parameters are set as for `PreparedStatement` objects. However, `CallableStatement` also supports setting parameters by name:

Example 2.6 Connector/J: Setting `CallableStatement` input parameters

```
...
//
// Set a parameter by index
//
cStmt.setString(1, "abcdefg");
//
// Alternatively, set a parameter using
// the parameter name
//
cStmt.setString("inputParameter", "abcdefg");
//
// Set the 'in/out' parameter using an index
//
cStmt.setInt(2, 1);
//
// Alternatively, set the 'in/out' parameter
// by name
//
cStmt.setInt("inOutParam", 1);
...
```

4. Execute the `CallableStatement`, and retrieve any result sets or output parameters.

Although `CallableStatement` supports calling any of the `Statement` execute methods (`executeUpdate()`, `executeQuery()` or `execute()`), the most flexible method to call is `execute()`, as you do not need to know ahead of time if the stored procedure returns result sets:

Example 2.7 Connector/J: Retrieving results and output parameter values

```
...
    boolean hadResults = cStmt.execute();
    //
    // Process all returned result sets
    //
    while (hadResults) {
        ResultSet rs = cStmt.getResultSet();
        // process result set
        ...
        hadResults = cStmt.getMoreResults();
    }
    //
    // Retrieve output parameters
    //
    // Connector/J supports both index-based and
    // name-based retrieval
    //
    int outputValue = cStmt.getInt(2); // index-based
    outputValue = cStmt.getInt("inOutParam"); // name-based
    ...
```

2.6.4 Retrieving `AUTO_INCREMENT` Column Values through JDBC

Before version 3.0 of the JDBC API, there was no standard way of retrieving key values from databases that supported auto increment or identity columns. With older JDBC drivers for MySQL, you could always use a MySQL-specific method on the `Statement` interface, or issue the query `SELECT LAST_INSERT_ID()` after issuing an `INSERT` to a table that had an `AUTO_INCREMENT` key. Using the MySQL-specific method call isn't portable, and issuing a `SELECT` to get the `AUTO_INCREMENT` key's value requires another round-trip to the database, which isn't as efficient as possible. The following code snippets demonstrate the three different ways to retrieve `AUTO_INCREMENT` values. First, we demonstrate the use of the new JDBC 3.0 method `getGeneratedKeys()` which is now the preferred method to use if you need to retrieve `AUTO_INCREMENT` keys and have access to JDBC 3.0. The second example shows how you can retrieve the same value using a standard `SELECT LAST_INSERT_ID()` query. The final example shows how updatable result sets can retrieve the `AUTO_INCREMENT` value when using the `insertRow()` method.

Example 2.8 Connector/J: Retrieving `AUTO_INCREMENT` column values using `Statement.getGeneratedKeys()`

```
Statement stmt = null;
ResultSet rs = null;
try {
    //
    // Create a Statement instance that we can use for
    // 'normal' result sets assuming you have a
    // Connection 'conn' to a MySQL database already
    // available
    stmt = conn.createStatement();
    //
    // Issue the DDL queries for the table for this example
    //
    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");
    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
```

```

//
stmt.executeUpdate(
    "INSERT INTO autoIncTutorial (dataField) "
    + "values ('Can I Get the Auto Increment Field?')",
    Statement.RETURN_GENERATED_KEYS);
//
// Example of using Statement.getGeneratedKeys()
// to retrieve the value of an auto-increment
// value
//
int autoIncKeyFromApi = -1;
rs = stmt.getGeneratedKeys();
if (rs.next()) {
    autoIncKeyFromApi = rs.getInt(1);
} else {
    // throw an exception from here
}
System.out.println("Key returned from getGeneratedKeys():"
    + autoIncKeyFromApi);
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}

```

Example 2.9 Connector/J: Retrieving **AUTO_INCREMENT** column values using **SELECT LAST_INSERT_ID()**

```

Statement stmt = null;
ResultSet rs = null;
try {
    //
    // Create a Statement instance that we can use for
    // 'normal' result sets.
    stmt = conn.createStatement();
    //
    // Issue the DDL queries for the table for this example
    //
    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");
    //
    // Insert one row that will generate an AUTO INCREMENT
    // key in the 'priKey' field
    //
    stmt.executeUpdate(
        "INSERT INTO autoIncTutorial (dataField) "
        + "values ('Can I Get the Auto Increment Field?')");
    //
    // Use the MySQL LAST_INSERT_ID()
    // function to do the same thing as getGeneratedKeys()
    //
    int autoIncKeyFromFunc = -1;
    rs = stmt.executeQuery("SELECT LAST_INSERT_ID()");
    if (rs.next()) {
        autoIncKeyFromFunc = rs.getInt(1);
    } else {
        // throw an exception from here
    }
}

```

```

    }
    System.out.println("Key returned from " +
        "'SELECT LAST_INSERT_ID()': " +
        autoIncKeyFromFunc);
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
}

```

Example 2.10 Connector/J: Retrieving **AUTO_INCREMENT** column values in **Updatable ResultSets**

```

Statement stmt = null;
ResultSet rs = null;
try {
    //
    // Create a Statement instance that we can use for
    // 'normal' result sets as well as an 'updatable'
    // one, assuming you have a Connection 'conn' to
    // a MySQL database already available
    //
    stmt = conn.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
        java.sql.ResultSet.CONCUR_UPDATABLE);

    //
    // Issue the DDL queries for the table for this example
    //
    stmt.executeUpdate("DROP TABLE IF EXISTS autoIncTutorial");
    stmt.executeUpdate(
        "CREATE TABLE autoIncTutorial ("
        + "priKey INT NOT NULL AUTO_INCREMENT, "
        + "dataField VARCHAR(64), PRIMARY KEY (priKey))");

    //
    // Example of retrieving an AUTO INCREMENT key
    // from an updatable result set
    //
    rs = stmt.executeQuery("SELECT priKey, dataField "
        + "FROM autoIncTutorial");
    rs.moveToInsertRow();
    rs.updateString("dataField", "AUTO INCREMENT here?");
    rs.insertRow();
    //
    // the driver adds rows at the end
    //
    rs.last();
    //
    // We should now be on the row we just inserted
    //
    int autoIncKeyFromRS = rs.getInt("priKey");
    System.out.println("Key returned for inserted row: "
        + autoIncKeyFromRS);
} finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
    if (stmt != null) {

```



```
        try {
            stmt.close();
        } catch (SQLException ex) {
            // ignore
        }
    }
}
```

Running the preceding example code should produce the following output:

```
Key returned from getGeneratedKeys(): 1
Key returned from SELECT LAST_INSERT_ID(): 1
Key returned for inserted row: 1
```

At times, it can be tricky to use the `SELECT LAST_INSERT_ID()` query, as that function's value is scoped to a connection. So, if some other query happens on the same connection, the value is overwritten. On the other hand, the `getGeneratedKeys()` method is scoped by the `Statement` instance, so it can be used even if other queries happen on the same connection, but not on the same `Statement` instance.

2.7 Connection Pooling with Connector/J

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any [thread](#) that needs them. Connection pooling can greatly increase the performance of your Java application, while reducing overall resource usage.

How Connection Pooling Works

Most applications only need a thread to have access to a JDBC connection when they are actively processing a [transaction](#), which often takes only milliseconds to complete. When not processing a transaction, the connection sits idle. Connection pooling enables the idle connection to be used by some other thread to do useful work.

In practice, when a thread needs to do work against a MySQL or other database with JDBC, it requests a connection from the pool. When the thread is finished using the connection, it returns it to the pool, so that it can be used by any other threads.

When the connection is loaned out from the pool, it is used exclusively by the thread that requested it. From a programming point of view, it is the same as if your thread called `DriverManager.getConnection()` every time it needed a JDBC connection. With connection pooling, your thread may end up using either a new connection or an already-existing connection.

Benefits of Connection Pooling

The main benefits to connection pooling are:

- Reduced connection creation time.

Although this is not usually an issue with the quick connection setup that MySQL offers compared to other databases, creating new JDBC connections still incurs networking and JDBC driver overhead that will be avoided if connections are recycled.

- Simplified programming model.

When using connection pooling, each individual thread can act as though it has created its own JDBC connection, allowing you to use straightforward JDBC programming techniques.

- Controlled resource usage.

If you create a new connection every time a thread needs one rather than using connection pooling, your application's resource usage can be wasteful, and it could lead to unpredictable behaviors for your application when it is under a heavy load.

Using Connection Pooling with Connector/J

The concept of connection pooling in JDBC has been standardized through the JDBC 2.0 Optional interfaces, and all major application servers have implementations of these APIs that work with MySQL Connector/J.

Generally, you configure a connection pool in your application server configuration files, and access it through the Java Naming and Directory Interface (JNDI). The following code shows how you might use a connection pool from an application deployed in a J2EE application server:

Example 2.11 Connector/J: Using a connection pool with a J2EE application server

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import javax.naming.InitialContext;
import javax.sql.DataSource;
public class MyServletJspOrEjb {
    public void doSomething() throws Exception {
        /*
         * Create a JNDI Initial context to be able to
         * lookup the DataSource
         *
         * In production-level code, this should be cached as
         * an instance or static variable, as it can
         * be quite expensive to create a JNDI context.
         *
         * Note: This code only works when you are using servlets
         * or EJBs in a J2EE application server. If you are
         * using connection pooling in standalone Java code, you
         * will have to create/configure datasources using whatever
         * mechanisms your particular connection pooling library
         * provides.
         */
        InitialContext ctx = new InitialContext();
        /*
         * Lookup the DataSource, which will be backed by a pool
         * that the application server provides. DataSource instances
         * are also a good candidate for caching as an instance
         * variable, as JNDI lookups can be expensive as well.
         */
        DataSource ds =
            (DataSource)ctx.lookup("java:comp/env/jdbc/MySQLDB");
        /*
         * The following code is what would actually be in your
         * Servlet, JSP or EJB 'service' method...where you need
         * to work with a JDBC connection.
         */
        Connection conn = null;
        Statement stmt = null;
        try {
            conn = ds.getConnection();
            /*
             * Now, use normal JDBC programming to work with
             * MySQL, making sure to close each resource when you're
             * finished with it, which permits the connection pool
             * resources to be recovered as quickly as possible
             */
            stmt = conn.createStatement();
            stmt.execute("SOME SQL QUERY");
            stmt.close();
            stmt = null;
            conn.close();
            conn = null;
        } finally {
            /*
             * close any jdbc instances here that weren't
             * explicitly closed during normal code path, so
             * that we don't 'leak' resources...
            */
        }
    }
}
```

```
        */
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException sqlex) {
                // ignore, as we can't do anything about it here
            }
            stmt = null;
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException sqlex) {
                // ignore, as we can't do anything about it here
            }
            conn = null;
        }
    }
}
```

As shown in the example above, after obtaining the JNDI [InitialContext](#), and looking up the [DataSource](#), the rest of the code follows familiar JDBC conventions.

When using connection pooling, always make sure that connections, and anything created by them (such as statements or result sets) are closed. This rule applies no matter what happens in your code (exceptions, flow-of-control, and so forth). When these objects are closed, they can be re-used; otherwise, they will be stranded, which means that the MySQL server resources they represent (such as buffers, locks, or sockets) are tied up for some time, or in the worst case can be tied up forever.

Sizing the Connection Pool

Each connection to MySQL has overhead (memory, CPU, context switches, and so forth) on both the client and server side. Every connection limits how many resources there are available to your application as well as the MySQL server. Many of these resources will be used whether or not the connection is actually doing any useful work! Connection pools can be tuned to maximize performance, while keeping resource utilization below the point where your application will start to fail rather than just run slower.

The optimal size for the connection pool depends on anticipated load and average database transaction time. In practice, the optimal connection pool size can be smaller than you might expect. If you take Oracle's Java Petstore blueprint application for example, a connection pool of 15-20 connections can serve a relatively moderate load (600 concurrent users) using MySQL and Tomcat with acceptable response times.

To correctly size a connection pool for your application, create load test scripts with tools such as Apache JMeter or The Grinder, and load test your application.

An easy way to determine a starting point is to configure your connection pool's maximum number of connections to be unbounded, run a load test, and measure the largest amount of concurrently used connections. You can then work backward from there to determine what values of minimum and maximum pooled connections give the best performance for your particular application.

Validating Connections

MySQL Connector/J can validate the connection by executing a lightweight ping against a server. In the case of load-balanced connections, this is performed against all active pooled internal connections that are retained. This is beneficial to Java applications using connection pools, as the pool can use this feature to validate connections. Depending on your connection pool and configuration, this validation can be carried out at different times:

1. Before the pool returns a connection to the application.

2. When the application returns a connection to the pool.
3. During periodic checks of idle connections.

To use this feature, specify a validation query in your connection pool that starts with `/* ping */`. Note that the syntax must be exactly as specified. This will cause the driver send a ping to the server and return a dummy lightweight result set. When using a [ReplicationConnection](#) or [LoadBalancedConnection](#), the ping will be sent across all active connections.

It is critical that the syntax be specified correctly. The syntax needs to be exact for reasons of efficiency, as this test is done for every statement that is executed:

```
protected static final String PING_MARKER = "/* ping */";
...
if (sql.charAt(0) == '/') {
    if (sql.startsWith(PING_MARKER)) {
        doPingInstead();
    }
    ...
}
```

None of the following snippets will work, because the ping syntax is sensitive to whitespace, capitalization, and placement:

```
sql = "/* PING */ SELECT 1";
sql = "SELECT 1 /* ping*/";
sql = "/*ping*/ SELECT 1";
sql = " /* ping */ SELECT 1";
sql = "/*to ping or not to ping*/ SELECT 1";
```

All of the previous statements will issue a normal [SELECT](#) statement and will **not** be transformed into the lightweight ping. Further, for load-balanced connections, the statement will be executed against one connection in the internal pool, rather than validating each underlying physical connection. This results in the non-active physical connections assuming a stale state, and they may die. If Connector/J then re-balances, it might select a dead connection, resulting in an exception being passed to the application. To help prevent this, you can use [loadBalanceValidateConnectionOnSwapServer](#) to validate the connection before use.

If your Connector/J deployment uses a connection pool that allows you to specify a validation query, take advantage of it, but ensure that the query starts *exactly* with `/* ping */`. This is particularly important if you are using the load-balancing or replication-aware features of Connector/J, as it will help keep alive connections which otherwise will go stale and die, causing problems later.

2.8 Multi-Host Connections

The following sections discuss a number of topics that involve multi-host connections, namely, server load-balancing, failover, and replication.

Developers should know the following things about multi-host connections that are managed through Connector/J:

- Each multi-host connection is a wrapper of the underlying physical connections.
- Each of the underlying physical connections has its own session. Sessions cannot be tracked, shared, or copied, given the MySQL architecture.
- Every switch between physical connections means a switch between sessions.
- Within a transaction boundary, there are no switches between physical connections. Beyond a transaction boundary, there is no guarantee that a switch does not occur.

Note

If an application reuses session-scope data (for example, variables, SSPs) beyond a transaction boundary, failures are possible, as a switch between the physical connections (which is also a switch between sessions) might occur. Therefore, the application should re-prepare the session data and also restart the last transaction in case of an exception, or it should re-prepare session data for each new transaction if it does not want to deal with exception handling.

2.8.1 Configuring Server Failover

MySQL Connector/J supports server failover. A failover happens when connection-related errors occur for an underlying, active connection. The connection errors are, by default, propagated to the client, which has to handle them by, for example, recreating the working objects ([Statement](#), [ResultSet](#), etc.) and restarting the processes. Sometimes, the driver might eventually fall back to the original host automatically before the client application continues to run, in which case the host switch is transparent and the client application will not even notice it.

A connection using failover support works just like a standard connection: the client does not experience any disruptions in the failover process. This means the client can rely on the same connection instance even if two successive statements might be executed on two different physical hosts. However, this does not mean the client does not have to deal with the exception that triggered the server switch.

The failover is configured at the initial setup stage of the server connection by the connection URL (see explanations for its format [here](#)):

```
jdbc:mysql://[primary host][:port],[secondary host 1][:port],[secondary host 2][:port]].../[database]
[?propertyName1=propertyValue1[&propertyName2=propertyValue2]...]
```

The host list in the connection URL comprises of two types of hosts, the primary and the secondary. When starting a new connection, the driver always tries to connect to the primary host first and, if required, fails over to the secondary hosts on the list sequentially when communication problems are experienced. Even if the initial connection to the primary host fails and the driver gets connected to a secondary host, the primary host never loses its special status: for example, it can be configured with an access mode distinct from those of the secondary hosts, and it can be put on a higher priority when a host is to be picked during a failover process.

The failover support is configured by the following connection properties (their functions are explained in the paragraphs below):

- [failOverReadOnly](#)
- [secondsBeforeRetryMaster](#)
- [queriesBeforeRetryMaster](#)
- [retriesAllDown](#)
- [autoReconnect](#)
- [autoReconnectForPools](#)

Configuring Connection Access Mode

As with any standard connection, the initial connection to the primary host is in read/write mode. However, if the driver fails to establish the initial connection to the primary host and it automatically switches to the next host on the list, the access mode now depends on the value of the property

`failOverReadOnly`, which is “true” by default. The same happens if the driver is initially connected to the primary host and, because of some connection failure, it fails over to a secondary host. Every time the connection falls back to the primary host, its access mode will be read/write, irrespective of whether or not the primary host has been connected to before. The connection access mode can be changed any time at runtime by calling the method `Connection.setReadOnly(boolean)`, which partially overrides the property `failOverReadOnly`. When `failOverReadOnly=false` and the access mode is explicitly set to either true or false, it becomes the mode for every connection after a host switch, no matter what host type are we connected to; but, if `failOverReadOnly=true`, changing the access mode to read/write is only possible if the driver is connecting to the primary host; however, even if the access mode cannot be changed for the current connection, the driver remembers the client's last intention and, when falling back to the primary host, that is the mode that will be used. For an illustration, see the following successions of events with a two-host connection.

- Sequence A, with `failOverReadOnly=true`:
 1. Connects to primary host in read/write mode
 2. Sets `Connection.setReadOnly(true)`; primary host now in read-only mode
 3. Failover event; connects to secondary host in read-only mode
 4. Sets `Connection.setReadOnly(false)`; secondary host remains in read-only mode
 5. Falls back to primary host; connection now in read/write mode
- Sequence B, with `failOverReadOnly=false`
 1. Connects to primary host in read/write mode
 2. Sets `Connection.setReadOnly(true)`; primary host now in read-only mode
 3. Failover event; connects to secondary host in read-only mode
 4. Set `Connection.setReadOnly(false)`; connection to secondary host switches to read/write mode
 5. Falls back to primary host; connection now in read/write mode

The difference between the two scenarios is in step 4: the access mode for the secondary host in sequence A does not change at that step, but the driver remembers and uses the set mode when falling back to the primary host, which would be read-only otherwise; but in sequence B, the access mode for the secondary host changes immediately.

Configuring Fallback to Primary Host

As already mentioned, the primary host is special in the failover arrangement when it comes to the host's access mode. Additionally, the driver tries to fall back to the primary host as soon as possible by default, even if no communication exception occurs. Two properties, `secondsBeforeRetryMaster` and `queriesBeforeRetryMaster`, determine when the driver is ready to retry a reconnection to the primary host (the `Master` in the property names stands for the primary host of our connection URL, which is not necessarily a master host in a replication setup; the naming was maintained for back compatibility with Connector/J versions prior to 5.1.35):

- `secondsBeforeRetryMaster` determines how much time the driver waits before trying to fall back to the primary host
- `queriesBeforeRetryMaster` determines the number of queries that are executed before the driver tries to fall back to the primary host. Note that for the driver, each call to a `Statement.execute*()` method increments the query execution counter; therefore, when calls are made to `Statement.executeBatch()` or if `allowMultiQueries` or `rewriteBatchStatements` are enabled, the driver may not have an accurate

count of the actual number of queries executed on the server. Also, the driver calls the `Statement.execute*()` methods internally in several occasions. All these mean you can only use `queriesBeforeRetryMaster` only as a coarse specification for when to fall back to the primary host.

In general, an attempt to fallback to the primary host is made when at least one of the conditions specified by the two properties is met, and the attempt always takes place at transaction boundaries. However, if auto-commit is turned off, the check happens only when the method `Connection.commit()` or `Connection.rollback()` is called. The automatic fallback to the primary host can be turned off by setting simultaneously `secondsBeforeRetryMaster` and `queriesBeforeRetryMaster` to "0". Setting only one of the properties to "0" only disables one part of the check.

Configuring Reconnection Attempts

When establishing a new connection or when a failover event occurs, the driver tries to connect successively to the next candidate on the host list. When the end of the list has been reached, it restarts all over again from the beginning of the list; however, the primary host is skipped over, if (a) NOT all the secondary hosts have already been tested at least once, AND (b) the fallback conditions defined by `secondsBeforeRetryMaster` and `queriesBeforeRetryMaster` are not yet fulfilled. Each run-through of the whole host list, (which is not necessarily completed at the end of the host list) counts as a single connection attempt. The driver tries as many connection attempts as specified by the value of the property `retriesAllDown`.

Seamless Reconnection

Although not recommended, you can make the driver perform failovers without invalidating the active `Statement` or `ResultSet` instances by setting either the parameter `autoReconnect` or `autoReconnectForPools` to `true`. This allows the client to continue using the same object instances after a failover event, without taking any exceptional measures. This, however, may lead to unexpected results: for example, if the driver is connected to the primary host with read/write access mode and it fails-over to a secondary host in read-only mode, further attempts to issue data-changing queries will result in errors, and the client will not be aware of that. This limitation is particularly relevant when using data streaming: after the failover, the `ResultSet` looks to be alright, but the underlying connection may have changed already, and no backing cursor is available anymore.

2.8.2 Configuring Load Balancing with Connector/J

Connector/J has long provided an effective means to distribute read/write load across multiple MySQL server instances for Cluster or master-master replication deployments. Starting with Connector/J 5.1.3, you can now dynamically configure load-balanced connections, with no service outage. In-process transactions are not lost, and no application exceptions are generated if any application is trying to use that particular server instance.

The load balancing is configured at the initial setup stage of the server connection by the following connection URL, which has a similar format as [the general URL for MySQL connection](#), but a specialized scheme:

```
jdbc:mysql:loadbalance://[host1][:port],[host2][:port],[host3][:port]]...[/[database]] »
[?propertyName1=propertyValue1[&propertyName2=propertyValue2]...]
```

There are two configuration properties associated with this functionality:

- `loadBalanceConnectionGroup` – This provides the ability to group connections from different sources. This allows you to manage these JDBC sources within a single class loader in any combination you choose. If they use the same configuration, and you want to manage them as a logical single group, give them the same name. This is the key property for management: if you do not define a name (string) for `loadBalanceConnectionGroup`, you cannot manage the connections. All load-balanced connections sharing the same `loadBalanceConnectionGroup` value, regardless of how the application creates them, will be managed together.

- `loadBalanceEnableJMX` – The ability to manage the connections is exposed when you define a `loadBalanceConnectionGroup`; but if you want to manage this externally, enable JMX by setting this property to `true`. This enables a JMX implementation, which exposes the management and monitoring operations of a connection group. Further, start your application with the - `Dcom.sun.management.jmxremote` JVM flag. You can then perform connect and perform operations using a JMX client such as `jconsole`.

Once a connection has been made using the correct connection properties, a number of monitoring properties are available:

- Current active host count.
- Current active physical connection count.
- Current active logical connection count.
- Total logical connections created.
- Total transaction count.

The following management operations can also be performed:

- Add host.
- Remove host.

The JMX interface, `com.mysql.jdbc.jmx.LoadBalanceConnectionGroupManagerMBean`, has the following methods:

- `int getActiveHostCount(String group);`
- `int getTotalHostCount(String group);`
- `long getTotalLogicalConnectionCount(String group);`
- `long getActiveLogicalConnectionCount(String group);`
- `long getActivePhysicalConnectionCount(String group);`
- `long getTotalPhysicalConnectionCount(String group);`
- `long getTotalTransactionCount(String group);`
- `void removeHost(String group, String host) throws SQLException;`
- `void stopNewConnectionsToHost(String group, String host) throws SQLException;`
- `void addHost(String group, String host, boolean forExisting);`
- `String getActiveHostsList(String group);`
- `String getRegisteredConnectionGroups();`

The `getRegisteredConnectionGroups()` method returns the names of all connection groups defined in that class loader.

You can test this setup with the following code:

```
public class Test {
    private static String URL = "jdbc:mysql:loadbalance://" +
        "localhost:3306,localhost:3310/test?" +
        "loadBalanceConnectionGroup=first&loadBalanceEnableJMX=true";
```



```

public static void main(String[] args) throws Exception {
    new Thread(new Repeater()).start();
    new Thread(new Repeater()).start();
    new Thread(new Repeater()).start();
}
static Connection getNewConnection() throws SQLException, ClassNotFoundException {
    Class.forName("com.mysql.jdbc.Driver");
    return DriverManager.getConnection(URL, "root", "");
}
static void executeSimpleTransaction(Connection c, int conn, int trans){
    try {
        c.setAutoCommit(false);
        Statement s = c.createStatement();
        s.executeQuery("SELECT SLEEP(1) /* Connection: " + conn + ", transaction: " + trans + " */");
        c.commit();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
public static class Repeater implements Runnable {
    public void run() {
        for(int i=0; i < 100; i++){
            try {
                Connection c = getNewConnection();
                for(int j=0; j < 10; j++){
                    executeSimpleTransaction(c, i, j);
                    Thread.sleep(Math.round(100 * Math.random()));
                }
                c.close();
                Thread.sleep(100);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
}

```

After compiling, the application can be started with the `-Dcom.sun.management.jmxremote` flag, to enable remote management. `jconsole` can then be started. The `Test` main class will be listed by `jconsole`. Select this and click **Connect**. You can then navigate to the `com.mysql.jdbc.jmx.LoadBalanceConnectionGroupManager` bean. At this point, you can click on various operations and examine the returned result.

If you now had an additional instance of MySQL running on port 3309, you could ensure that Connector/J starts using it by using the `addHost()`, which is exposed in `jconsole`. Note that these operations can be performed dynamically without having to stop the application running.

For further information on the combination of load balancing and failover, see [Section 2.8.4, “Advanced Load-balancing and Failover Configuration”](#).

2.8.3 Configuring Master/Slave Replication with Connector/J

This section describe a number of features of Connector/J's support for replication-aware deployments.

The replication is configured at the initial setup stage of the server connection by the connection URL, which has a similar format as [the general URL for MySQL connection](#), but a specialized scheme:

```

jdbc:mysql:replication://[master host][:port],[slave host 1][:port],[slave host 2][:port]]...[/[database]]
[?propertyName1=propertyValue1[&propertyName2=propertyValue2]...]

```

Users may specify the property `allowMasterDownConnections=true` to allow `Connection` objects to be created even though no master hosts are reachable. Such `Connection` objects report they are read-only, and `isMasterConnection()` returns false for them. The `Connection` tests for available master hosts when `Connection.setReadOnly(false)` is called, throwing an

SQLException if it cannot establish a connection to a master, or switching to a master connection if the host is available.

For Connector/J 5.1.38 and later, users may specify the property `allowSlavesDownConnections=true` to allow `Connection` objects to be created even though no slave hosts are reachable. A `Connection` then, at runtime, tests for available slave hosts when `Connection.setReadOnly(true)` is called (see explanation for the method below), throwing an SQLException if it cannot establish a connection to a slave, unless the property `readFromMasterWhenNoSlaves` is set to be “true” (see below for a description of the property).

Scaling out Read Load by Distributing Read Traffic to Slaves

Connector/J 3.1.7 and higher includes a variant of the driver that will automatically send queries to a read/write master, or a failover or round-robin loadbalanced set of slaves based on the state of `Connection.getReadOnly()`.

An application signals that it wants a transaction to be read-only by calling `Connection.setReadOnly(true)`. The replication-aware connection will use one of the slave connections, which are load-balanced per slave host using a round-robin scheme. A given connection is sticky to a slave until a transaction boundary command (a commit or rollback) is issued, or until the slave is removed from service. For Connector/J 5.1.38 and later, after calling `Connection.setReadOnly(true)`, if you want to allow connection to a master when no slaves are available, set the property `readFromMasterWhenNoSlaves` to “true.” Notice that the master host will be used in read-only state in those cases, as if it is a slave host. Also notice that setting `readFromMasterWhenNoSlaves=true` might result in an extra load for the master host in a transparent manner.

If you have a write transaction, or if you have a read that is time-sensitive (remember, replication in MySQL is asynchronous), set the connection to be not read-only, by calling `Connection.setReadOnly(false)` and the driver will ensure that further calls are sent to the master MySQL server. The driver takes care of propagating the current state of autocommit, isolation level, and catalog between all of the connections that it uses to accomplish this load balancing functionality.

To enable this functionality, use the `com.mysql.jdbc.ReplicationDriver` class when configuring your application server's connection pool or when creating an instance of a JDBC driver for your standalone application. Because it accepts the same URL format as the standard MySQL JDBC driver, `ReplicationDriver` does not currently work with `java.sql.DriverManager`-based connection creation unless it is the only MySQL JDBC driver registered with the `DriverManager`.

Here is a short example of how `ReplicationDriver` might be used in a standalone application:

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.util.Properties;
import com.mysql.jdbc.ReplicationDriver;

public class ReplicationDriverDemo {
    public static void main(String[] args) throws Exception {
        ReplicationDriver driver = new ReplicationDriver();
        Properties props = new Properties();
        // We want this for failover on the slaves
        props.put("autoReconnect", "true");
        // We want to load balance between the slaves
        props.put("roundRobinLoadBalance", "true");
        props.put("user", "foo");
        props.put("password", "bar");
        //
        // Looks like a normal MySQL JDBC url, with a
        // comma-separated list of hosts, the first
        // being the 'master', the rest being any number
        // of slaves that the driver will load balance against
        //
        Connection conn =
```

```

        driver.connect("jdbc:mysql:replication://master,slave1,slave2,slave3/test",
            props);
        //
        // Perform read/write work on the master
        // by setting the read-only flag to "false"
        //
        conn.setReadOnly(false);
        conn.setAutoCommit(false);
        conn.createStatement().executeUpdate("UPDATE some_table ...");
        conn.commit();
        //
        // Now, do a query from a slave, the driver automatically picks one
        // from the list
        //
        conn.setReadOnly(true);
        ResultSet rs =
            conn.createStatement().executeQuery("SELECT a,b FROM alt_table");
        .....
    }
}

```

Consider using the Load Balancing JDBC Pool ([lbpool](#)) tool, which provides a wrapper around the standard JDBC driver and enables you to use DB connection pools that includes checks for system failures and uneven load distribution. For more information, see [Load Balancing JDBC Driver for MySQL \(mysql-lbpool\)](#).

Support for Multiple-Master Replication Topographies

Since Connector/J 5.1.27, multi-master replication topographies are supported.

The connection URL for replication discussed earlier (i.e., in the format of [jdbc:mysql:replication://master,slave1,slave2,slave3/test](#)) assumes that the first (and only the first) host is the master. Supporting deployments with an arbitrary number of masters and slaves requires a different URL syntax for specifying the hosts and the properties for specific hosts, which is just an expansion of the URL syntax discussed in [Alternate Format for JDBC URLs](#) with the property `type=[master|slave]`; for example:

```
jdbc:mysql://address=(type=master)(host=master1host),address=(type=master)(host=master2host),address=(t
```

Connector/J uses a load-balanced connection internally for management of the master connections, which means that [ReplicationConnection](#), when configured to use multiple masters, exposes the same options to balance load across master hosts as described in [Section 2.8.2, "Configuring Load Balancing with Connector/J"](#).

Live Reconfiguration of Replication Topography

Since Connector/J 5.1.28, live management of replication host (single or multi-master) topographies is also supported. This enables users to promote slaves for Java applications without requiring an application restart.

The replication hosts are most effectively managed in the context of a replication connection group. A `ReplicationConnectionGroup` class represents a logical grouping of connections which can be managed together. There may be one or more such replication connection groups in a given Java class loader (there can be an application with two different JDBC resources needing to be managed independently). This key class exposes host management methods for replication connections, and [ReplicationConnection](#) objects register themselves with the appropriate [ReplicationConnectionGroup](#) if a value for the new `replicationConnectionGroup` property is specified. The [ReplicationConnectionGroup](#) object tracks these connections until they are closed, and it is used to manipulate the hosts associated with these connections.

Some important methods related to host management include:

- [getMasterHosts\(\)](#): Returns a collection of strings representing the hosts configured as masters
- [getSlaveHosts\(\)](#): Returns a collection of strings representing the hosts configured as slaves

- `addSlaveHost(String host)`: Adds new host to pool of possible slave hosts for selection at start of new read-only workload
- `promoteSlaveToMaster(String host)`: Removes the host from the pool of potential slaves for future read-only processes (existing read-only process is allowed to continue to completion) and adds the host to the pool of potential master hosts
- `removeSlaveHost(String host, boolean closeGently)`: Removes the host (host name match must be exact) from the list of configured slaves; if `closeGently` is false, existing connections which have this host as currently active will be closed hardy (application should expect exceptions)
- `removeMasterHost(String host, boolean closeGently)`: Same as `removeSlaveHost()`, but removes the host from the list of configured masters

Some useful management metrics include:

- `getConnectionCountWithHostAsSlave(String host)`: Returns the number of `ReplicationConnection` objects that have the given host configured as a possible slave
- `getConnectionCountWithHostAsMaster(String host)`: Returns the number of `ReplicationConnection` objects that have the given host configured as a possible master
- `getNumberOfSlavesAdded()`: Returns the number of times a slave host has been dynamically added to the group pool
- `getNumberOfSlavesRemoved()`: Returns the number of times a slave host has been dynamically removed from the group pool
- `getNumberOfSlavePromotions()`: Returns the number of times a slave host has been promoted to a master
- `getTotalConnectionCount()`: Returns the number of `ReplicationConnection` objects which have been registered with this group
- `getActiveConnectionCount()`: Returns the number of `ReplicationConnection` objects currently being managed by this group

ReplicationConnectionGroupManager

`com.mysql.jdbc.ReplicationConnectionGroupManager` provides access to the replication connection groups, together with some utility methods.

- `getConnectionGroup(String groupName)`: Returns the `ReplicationConnectionGroup` object matching the `groupName` provided

The other methods in `ReplicationConnectionGroupManager` mirror those of `ReplicationConnectionGroup`, except that the first argument is a `String` group name. These methods will operate on all matching `ReplicationConnectionGroups`, which are helpful for removing a server from service and have it decommissioned across all possible `ReplicationConnectionGroups`.

These methods might be useful for in-JVM management of replication hosts if an application triggers topography changes. For managing host configurations from outside the JVM, JMX can be used.

Using JMX for Managing Replication Hosts

When Connector/J is started with `replicationEnableJMX=true` and a value set for the property `replicationConnectionGroup`, a JMX MBean will be registered, allowing manipulation of replication hosts by a JMX client. The MBean interface is defined in `com.mysql.jdbc.jmx.ReplicationGroupManagerMBean`, and leverages the `ReplicationConnectionGroupManager` static methods:

```

public abstract void addSlaveHost(String groupFilter, String host) throws SQLException;
public abstract void removeSlaveHost(String groupFilter, String host) throws SQLException;
public abstract void promoteSlaveToMaster(String groupFilter, String host) throws SQLException;
public abstract void removeMasterHost(String groupFilter, String host) throws SQLException;
public abstract String getMasterHostsList(String group);
public abstract String getSlaveHostsList(String group);
public abstract String getRegisteredConnectionGroups();
public abstract int getActiveMasterHostCount(String group);
public abstract int getActiveSlaveHostCount(String group);
public abstract int getSlavePromotionCount(String group);
public abstract long getTotalLogicalConnectionCount(String group);
public abstract long getActiveLogicalConnectionCount(String group);

```

2.8.4 Advanced Load-balancing and Failover Configuration

Connector/J provides a useful load-balancing implementation for MySQL Cluster or multi-master deployments, as explained in [Section 2.8.2, “Configuring Load Balancing with Connector/J”](#) and [Support for Multiple-Master Replication Topographies](#). As of Connector/J 5.1.12, this same implementation is used for balancing load between read-only slaves with [ReplicationDriver](#).

When trying to balance workload between multiple servers, the driver has to determine when it is safe to swap servers, doing so in the middle of a transaction, for example, could cause problems. It is important not to lose state information. For this reason, Connector/J will only try to pick a new server when one of the following happens:

1. At transaction boundaries (transactions are explicitly committed or rolled back).
2. A communication exception (SQL State starting with "08") is encountered.
3. When a [SQLException](#) matches conditions defined by user, using the extension points defined by the [loadBalanceSQLStateFailover](#), [loadBalanceSQLExceptionSubclassFailover](#) or [loadBalanceExceptionChecker](#) properties.

The third condition revolves around three new properties introduced with Connector/J 5.1.13. It allows you to control which [SQLExceptions](#) trigger failover.

- [loadBalanceExceptionChecker](#) - The [loadBalanceExceptionChecker](#) property is really the key. This takes a fully-qualified class name which implements the new [com.mysql.jdbc.LoadBalanceExceptionChecker](#) interface. This interface is very simple, and you only need to implement the following method:

```

public boolean shouldExceptionTriggerFailover(SQLException ex)

```

A [SQLException](#) is passed in, and a boolean returned. A value of [true](#) triggers a failover, [false](#) does not.

You can use this to implement your own custom logic. An example where this might be useful is when dealing with transient errors when using MySQL Cluster, where certain buffers may become overloaded. The following code snippet illustrates this:

```

public class NdbLoadBalanceExceptionChecker
    extends StandardLoadBalanceExceptionChecker {
    public boolean shouldExceptionTriggerFailover(SQLException ex) {
        return super.shouldExceptionTriggerFailover(ex)
            || checkNdbException(ex);
    }
    private boolean checkNdbException(SQLException ex){
        // Have to parse the message since most NDB errors
        // are mapped to the same DEMC.
        return (ex.getMessage().startsWith("Lock wait timeout exceeded")) ||
            (ex.getMessage().startsWith("Got temporary error"))
    }
}

```

```
&& ex.getMessage().endsWith("from NDB"))));
}
}
```

The code above extends `com.mysql.jdbc.StandardLoadBalanceExceptionChecker`, which is the default implementation. There are a few convenient shortcuts built into this, for those who want to have some level of control using properties, without writing Java code. This default implementation uses the two remaining properties: `loadBalanceSQLStateFailover` and `loadBalanceSQLExceptionSubclassFailover`.

- `loadBalanceSQLStateFailover` - allows you to define a comma-delimited list of `SQLState` code prefixes, against which a `SQLException` is compared. If the prefix matches, failover is triggered. So, for example, the following would trigger a failover if a given `SQLException` starts with "00", or is "12345":

```
loadBalanceSQLStateFailover=00,12345
```

- `loadBalanceSQLExceptionSubclassFailover` - can be used in conjunction with `loadBalanceSQLStateFailover` or on its own. If you want certain subclasses of `SQLException` to trigger failover, simply provide a comma-delimited list of fully-qualified class or interface names to check against. For example, if you want all `SQLTransientConnectionExceptions` to trigger failover, you would specify:

```
loadBalanceSQLExceptionSubclassFailover=java.sql.SQLTransientConnectionException
```

While the three failover conditions enumerated earlier suit most situations, if `autocommit` is enabled, Connector/J never re-balances, and continues using the same physical connection. This can be problematic, particularly when load-balancing is being used to distribute read-only load across multiple slaves. However, Connector/J can be configured to re-balance after a certain number of statements are executed, when `autocommit` is enabled. This functionality is dependent upon the following properties:

- `loadBalanceAutoCommitStatementThreshold` – defines the number of matching statements which will trigger the driver to potentially swap physical server connections. The default value, 0, retains the behavior that connections with `autocommit` enabled are never balanced.
- `loadBalanceAutoCommitStatementRegex` – the regular expression against which statements must match. The default value, blank, matches all statements. So, for example, using the following properties will cause Connector/J to re-balance after every third statement that contains the string "test":

```
loadBalanceAutoCommitStatementThreshold=3
loadBalanceAutoCommitStatementRegex=.*test.*
```

`loadBalanceAutoCommitStatementRegex` can prove useful in a number of situations. Your application may use temporary tables, server-side session state variables, or connection state, where letting the driver arbitrarily swap physical connections before processing is complete could cause data loss or other problems. This allows you to identify a trigger statement that is only executed when it is safe to swap physical connections.

2.9 Using the Connector/J Interceptor Classes

An interceptor is a software design pattern that provides a transparent way to extend or modify some aspect of a program, similar to a user exit. No recompiling is required. With Connector/J, the interceptors are enabled and disabled by updating the connection string to refer to different sets of interceptor classes that you instantiate.

The connection properties that control the interceptors are explained in [Section 2.5.1, "Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J"](#):

- `connectionLifecycleInterceptors`, where you specify the fully qualified names of classes that implement the `com.mysql.jdbc.ConnectionLifecycleInterceptor` interface. In these kinds of interceptor classes, you might log events such as rollbacks, measure the time between transaction start and end, or count events such as calls to `setAutoCommit()`.
- `exceptionInterceptors`, where you specify the fully qualified names of classes that implement the `com.mysql.jdbc.ExceptionInterceptor` interface. In these kinds of interceptor classes, you might add extra diagnostic information to exceptions that can have multiple causes or indicate a problem with server settings. Because `exceptionInterceptors` classes are only called when handling a `SQLException` thrown from Connector/J code, they can be used even in production deployments without substantial performance overhead.
- `statementInterceptors`, where you specify the fully qualified names of classes that implement the `com.mysql.jdbc.StatementInterceptorV2` interface. In these kinds of interceptor classes, you might change or augment the processing done by certain kinds of statements, such as automatically checking for queried data in a `memcached` server, rewriting slow queries, logging information about statement execution, or route requests to remote servers.

2.10 Using Connector/J with Tomcat

The following instructions are based on the instructions for Tomcat-5.x, available at <http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html> which is current at the time this document was written.

First, install the `.jar` file that comes with Connector/J in `$CATALINA_HOME/common/lib` so that it is available to all applications installed in the container.

Next, configure the JNDI DataSource by adding a declaration resource to `$CATALINA_HOME/conf/server.xml` in the context that defines your web application:

```
<Context ....>
...
<Resource name="jdbc/MySQLDB"
          auth="Container"
          type="javax.sql.DataSource"/>
<ResourceParams name="jdbc/MySQLDB">
  <parameter>
    <name>factory</name>
    <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
  </parameter>
  <parameter>
    <name>maxActive</name>
    <value>10</value>
  </parameter>
  <parameter>
    <name>maxIdle</name>
    <value>5</value>
  </parameter>
  <parameter>
    <name>validationQuery</name>
    <value>SELECT 1</value>
  </parameter>
  <parameter>
    <name>testOnBorrow</name>
    <value>true</value>
  </parameter>
  <parameter>
    <name>testWhileIdle</name>
    <value>true</value>
  </parameter>
  <parameter>
    <name>timeBetweenEvictionRunsMillis</name>
    <value>10000</value>
  </parameter>
</parameter>
```

```
<name>minEvictableIdleTimeMillis</name>
<value>60000</value>
</parameter>
<parameter>
  <name>username</name>
  <value>someuser</value>
</parameter>
<parameter>
  <name>password</name>
  <value>somepass</value>
</parameter>
<parameter>
  <name>driverClassName</name>
  <value>com.mysql.jdbc.Driver</value>
</parameter>
<parameter>
  <name>url</name>
  <value>jdbc:mysql://localhost:3306/test</value>
</parameter>
</ResourceParams>
</Context>
```

Note that Connector/J 5.1.3 introduced a facility whereby, rather than use a `validationQuery` value of `SELECT 1`, it is possible to use `validationQuery` with a value set to `/* ping */`. This sends a ping to the server which then returns a fake result set. This is a lighter weight solution. It also has the advantage that if using `ReplicationConnection` or `LoadBalancedConnection` type connections, the ping will be sent across all active connections. The following XML snippet illustrates how to select this option:

```
<parameter>
  <name>validationQuery</name>
  <value>/* ping */</value>
</parameter>
```

Note that `/* ping */` has to be specified exactly.

In general, follow the installation instructions that come with your version of Tomcat, as the way you configure datasources in Tomcat changes from time to time, and if you use the wrong syntax in your XML file, you will most likely end up with an exception similar to the following:

```
Error: java.sql.SQLException: Cannot load JDBC driver class 'null ' SQL
state: null
```

Note that the auto-loading of drivers having the `META-INF/service/java.sql.Driver` class in JDBC 4.0 and above causes an improper undeployment of the Connector/J driver in Tomcat on Windows. Namely, the Connector/J jar remains locked. This is an initialization problem that is not related to the driver. The possible workarounds, if viable, are as follows: use `"antiResourceLocking=true"` as a Tomcat Context attribute, or remove the `META-INF/` directory.

2.11 Using Connector/J with JBoss

These instructions cover JBoss-4.x. To make the JDBC driver classes available to the application server, copy the `.jar` file that comes with Connector/J to the `lib` directory for your server configuration (which is usually called `default`). Then, in the same configuration directory, in the subdirectory named `deploy`, create a datasource configuration file that ends with `-ds.xml`, which tells JBoss to deploy this file as a JDBC Datasource. The file should have the following contents:

```
<datasources>
  <local-tx-datasource>
    <jndi-name>MySQLDB</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/dbname</connection-url>
```



```

<driver-class>com.mysql.jdbc.Driver</driver-class>
<user-name>user</user-name>
<password>pass</password>
<min-pool-size>5</min-pool-size>
<max-pool-size>20</max-pool-size>
<idle-timeout-minutes>5</idle-timeout-minutes>
<exception-sorter-class-name>
com.mysql.jdbc.integration.jboss.ExtendedMySQLExceptionSorter
</exception-sorter-class-name>
<valid-connection-checker-class-name>
com.mysql.jdbc.integration.jboss.MySQLValidConnectionChecker
</valid-connection-checker-class-name>
</local-tx-datasource>
</datasources>

```

2.12 Using Connector/J with Spring

The Spring Framework is a Java-based application framework designed for assisting in application design by providing a way to configure components. The technique used by Spring is a well known design pattern called Dependency Injection (see [Inversion of Control Containers and the Dependency Injection pattern](#)). This article will focus on Java-oriented access to MySQL databases with Spring 2.0. For those wondering, there is a .NET port of Spring appropriately named Spring.NET.

Spring is not only a system for configuring components, but also includes support for aspect oriented programming (AOP). This is one of the main benefits and the foundation for Spring's resource and transaction management. Spring also provides utilities for integrating resource management with JDBC and Hibernate.

For the examples in this section the MySQL world sample database will be used. The first task is to set up a MySQL data source through Spring. Components within Spring use the “bean” terminology. For example, to configure a connection to a MySQL server supporting the world sample database, you might use:

```

<util:map id="dbProps">
  <entry key="db.driver" value="com.mysql.jdbc.Driver"/>
  <entry key="db.jdbcurl" value="jdbc:mysql://localhost/world"/>
  <entry key="db.username" value="myuser"/>
  <entry key="db.password" value="mypass"/>
</util:map>

```

In the above example, we are assigning values to properties that will be used in the configuration. For the datasource configuration:

```

<bean id="dataSource"
  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="${db.driver}"/>
  <property name="url" value="${db.jdbcurl}"/>
  <property name="username" value="${db.username}"/>
  <property name="password" value="${db.password}"/>
</bean>

```

The placeholders are used to provide values for properties of this bean. This means that you can specify all the properties of the configuration in one place instead of entering the values for each property on each bean. We do, however, need one more bean to pull this all together. The last bean is responsible for actually replacing the placeholders with the property values.

```

<bean
  class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="properties" ref="dbProps"/>

```

```
</bean>
```

Now that we have our MySQL data source configured and ready to go, we write some Java code to access it. The example below will retrieve three random cities and their corresponding country using the data source we configured with Spring.

```
// Create a new application context. this processes the Spring config
ApplicationContext ctx =
    new ClassPathXmlApplicationContext("ex1appContext.xml");
// Retrieve the data source from the application context
DataSource ds = (DataSource) ctx.getBean("dataSource");
// Open a database connection using Spring's DataSourceUtils
Connection c = DataSourceUtils.getConnection(ds);
try {
    // retrieve a list of three random cities
    PreparedStatement ps = c.prepareStatement(
        "select City.Name as 'City', Country.Name as 'Country' " +
        "from City inner join Country on City.CountryCode = Country.Code " +
        "order by rand() limit 3");
    ResultSet rs = ps.executeQuery();
    while(rs.next()) {
        String city = rs.getString("City");
        String country = rs.getString("Country");
        System.out.printf("The city %s is in %s\n", city, country);
    }
} catch (SQLException ex) {
    // something has failed and we print a stack trace to analyse the error
    ex.printStackTrace();
    // ignore failure closing connection
    try { c.close(); } catch (SQLException e) { }
} finally {
    // properly release our connection
    DataSourceUtils.releaseConnection(c, ds);
}
```

This is very similar to normal JDBC access to MySQL with the main difference being that we are using `DataSourceUtils` instead of the `DriverManager` to create the connection.

While it may seem like a small difference, the implications are somewhat far reaching. Spring manages this resource in a way similar to a container managed data source in a J2EE application server. When a connection is opened, it can be subsequently accessed in other parts of the code if it is synchronized with a transaction. This makes it possible to treat different parts of your application as transactional instead of passing around a database connection.

2.12.1 Using JdbcTemplate

Spring makes extensive use of the Template method design pattern (see [Template Method Pattern](#)). Our immediate focus will be on the `JdbcTemplate` and related classes, specifically `NamedParameterJdbcTemplate`. The template classes handle obtaining and releasing a connection for data access when one is needed.

The next example shows how to use `NamedParameterJdbcTemplate` inside of a DAO (Data Access Object) class to retrieve a random city given a country code.

```
public class Ex2JdbcDao {
    /**
     * Data source reference which will be provided by Spring.
     */
    private DataSource dataSource;
    /**
     * Our query to find a random city given a country code. Notice
     * the ":country" parameter toward the end. This is called a
     * named parameter.
     */
    private String queryString = "select Name from City " +
```

```

        "where CountryCode = :country order by rand() limit 1";
    /**
     * Retrieve a random city using Spring JDBC access classes.
     */
    public String getRandomCityByCountryCode(String cntryCode) {
        // A template that permits using queries with named parameters
        NamedParameterJdbcTemplate template =
            new NamedParameterJdbcTemplate(dataSource);
        // A java.util.Map is used to provide values for the parameters
        Map params = new HashMap();
        params.put("country", cntryCode);
        // We query for an Object and specify what class we are expecting
        return (String)template.queryForObject(queryString, params, String.class);
    }
    /**
     * A JavaBean setter-style method to allow Spring to inject the data source.
     * @param dataSource
     */
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }
}

```

The focus in the above code is on the `getRandomCityByCountryCode()` method. We pass a country code and use the `NamedParameterJdbcTemplate` to query for a city. The country code is placed in a Map with the key "country", which is the parameter is named in the SQL query.

To access this code, you need to configure it with Spring by providing a reference to the data source.

```

<bean id="dao" class="code.Ex2JdbcDao">
    <property name="dataSource" ref="dataSource"/>
</bean>

```

At this point, we can just grab a reference to the DAO from Spring and call `getRandomCityByCountryCode()`.

```

// Create the application context
ApplicationContext ctx =
    new ClassPathXmlApplicationContext("ex2appContext.xml");
// Obtain a reference to our DAO
Ex2JdbcDao dao = (Ex2JdbcDao) ctx.getBean("dao");
String countryCode = "USA";
// Find a few random cities in the US
for(int i = 0; i < 4; ++i)
    System.out.printf("A random city in %s is %s\n", countryCode,
        dao.getRandomCityByCountryCode(countryCode));

```

This example shows how to use Spring's JDBC classes to completely abstract away the use of traditional JDBC classes including `Connection` and `PreparedStatement`.

2.12.2 Transactional JDBC Access

You might be wondering how we can add transactions into our code if we do not deal directly with the JDBC classes. Spring provides a transaction management package that not only replaces JDBC transaction management, but also enables declarative transaction management (configuration instead of code).

To use transactional database access, we will need to change the storage engine of the tables in the world database. The downloaded script explicitly creates MyISAM tables which do not support transactional semantics. The InnoDB storage engine does support transactions and this is what we will be using. We can change the storage engine with the following statements.

```

ALTER TABLE City ENGINE=InnoDB;

```

```
ALTER TABLE Country ENGINE=InnoDB;
ALTER TABLE CountryLanguage ENGINE=InnoDB;
```

A good programming practice emphasized by Spring is separating interfaces and implementations. What this means is that we can create a Java interface and only use the operations on this interface without any internal knowledge of what the actual implementation is. We will let Spring manage the implementation and with this it will manage the transactions for our implementation.

First you create a simple interface:

```
public interface Ex3Dao {
    Integer createCity(String name, String countryCode,
        String district, Integer population);
}
```

This interface contains one method that will create a new city record in the database and return the id of the new record. Next you need to create an implementation of this interface.

```
public class Ex3DaoImpl implements Ex3Dao {
    protected DataSource dataSource;
    protected SqlUpdate updateQuery;
    protected SqlFunction idQuery;
    public Integer createCity(String name, String countryCode,
        String district, Integer population) {
        updateQuery.update(new Object[] { name, countryCode,
            district, population });
        return getLastId();
    }
    protected Integer getLastId() {
        return idQuery.run();
    }
}
```

You can see that we only operate on abstract query objects here and do not deal directly with the JDBC API. Also, this is the complete implementation. All of our transaction management will be dealt with in the configuration. To get the configuration started, we need to create the DAO.

```
<bean id="dao" class="code.Ex3DaoImpl">
    <property name="dataSource" ref="dataSource"/>
    <property name="updateQuery">...</property>
    <property name="idQuery">...</property>
</bean>
```

Now you need to set up the transaction configuration. The first thing you must do is create transaction manager to manage the data source and a specification of what transaction properties are required for the **dao** methods.

```
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*" />
    </tx:attributes>
</tx:advice>
```

The preceding code creates a transaction manager that handles transactions for the data source provided to it. The **txAdvice** uses this transaction manager and the attributes specify to create a transaction for all methods. Finally you need to apply this advice with an AOP pointcut.

```
<aop:config>
  <aop:pointcut id="daoMethods"
    expression="execution(* code.Ex3Dao.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="daoMethods"/>
</aop:config>
```

This basically says that all methods called on the [Ex3Dao](#) interface will be wrapped in a transaction. To make use of this, you only have to retrieve the [dao](#) from the application context and call a method on the [dao](#) instance.

```
Ex3Dao dao = (Ex3Dao) ctx.getBean("dao");
Integer id = dao.createCity(name, countryCode, district, pop);
```

We can verify from this that there is no transaction management happening in our Java code and it is all configured with Spring. This is a very powerful notion and regarded as one of the most beneficial features of Spring.

2.12.3 Connection Pooling with Spring

In many situations, such as web applications, there will be a large number of small database transactions. When this is the case, it usually makes sense to create a pool of database connections available for web requests as needed. Although MySQL does not spawn an extra process when a connection is made, there is still a small amount of overhead to create and set up the connection. Pooling of connections also alleviates problems such as collecting large amounts of sockets in the [TIME_WAIT](#) state.

Setting up pooling of MySQL connections with Spring is as simple as changing the data source configuration in the application context. There are a number of configurations that we can use. The first example is based on the [Jakarta Commons DBCP library](#). The example below replaces the source configuration that was based on [DriverManagerDataSource](#) with DBCP's [BasicDataSource](#).

```
<bean id="dataSource" destroy-method="close"
  class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="${db.driver}"/>
  <property name="url" value="${db.jdbcurl}"/>
  <property name="username" value="${db.username}"/>
  <property name="password" value="${db.password}"/>
  <property name="initialSize" value="3"/>
</bean>
```

The configuration of the two solutions is very similar. The difference is that DBCP will pool connections to the database instead of creating a new connection every time one is requested. We have also set a parameter here called [initialSize](#). This tells DBCP that we want three connections in the pool when it is created.

Another way to configure connection pooling is to configure a data source in our J2EE application server. Using JBoss as an example, you can set up the MySQL connection pool by creating a file called [mysql-local-ds.xml](#) and placing it in the server/default/deploy directory in JBoss. Once we have this setup, we can use JNDI to look it up. With Spring, this lookup is very simple. The data source configuration looks like this.

```
<jee:jndi-lookup id="dataSource" jndi-name="java:MySQL_DS"/>
```

2.13 Using Connector/J with GlassFish

This section explains how to use MySQL Connector/J with GlassFish™ Server Open Source Edition 3.0.1. GlassFish can be downloaded from the [GlassFish website](#).

Once GlassFish is installed, make sure it can access MySQL Connector/J. To do this, copy the MySQL Connector/J `jar` file to the `domain-dir/lib` directory. For example, copy `mysql-connector-java-5.1.30-bin.jar` to `C:\glassfish-install-path\domains\domain-name\lib`. Restart the GlassFish Application Server. For more information, see “Integrating the JDBC Driver” in *GlassFish Server Open Source Edition Administration Guide*, available at [GlassFish Server Documentation](#).

You are now ready to create JDBC Connection Pools and JDBC Resources.

Creating a Connection Pool

1. In the GlassFish Administration Console, using the navigation tree navigate to **Resources, JDBC, Connection Pools**.
2. In the **JDBC Connection Pools** frame click **New**. You will enter a two step wizard.
3. In the **Name** field under **General Settings** enter the name for the connection pool, for example enter `MySQLConnPool`.
4. In the **Resource Type** field, select `javax.sql.DataSource` from the drop-down listbox.
5. In the **Database Vendor** field, select `MySQL` from the drop-down listbox. Click **Next** to go to the next page of the wizard.
6. You can accept the default settings for General Settings, Pool Settings and Transactions for this example. Scroll down to Additional Properties.
7. In Additional Properties you will need to ensure the following properties are set:
 - **ServerName** - The server to connect to. For local testing this will be `localhost`.
 - **User** - The user name with which to connect to MySQL.
 - **Password** - The corresponding password for the user.
 - **DatabaseName** - The database to connect to, for example the sample MySQL database `world`.
8. Click **Finish** to exit the wizard. You will be taken to the **JDBC Connection Pools** page where all current connection pools, including the one you just created, will be displayed.
9. In the **JDBC Connection Pools** frame click on the connection pool you just created. Here, you can review and edit information about the connection pool. Because Connector/J does not support optimized validation queries, go to the **Advanced** tab, and under Connection Validation, configure the following settings:
 - **Connection Validation** - select **Required**.
 - **Validation Method** - select **table** from the drop-down menu.
 - **Table Name** - enter `DUAL`.
10. To test your connection pool click the **Ping** button at the top of the frame. A message will be displayed confirming correct operation or otherwise. If an error message is received recheck the previous steps, and ensure that MySQL Connector/J has been correctly copied into the previously specified location.

Now that you have created a connection pool you will also need to create a JDBC Resource (data source) for use by your application.

Creating a JDBC Resource

Your Java application will usually reference a data source object to establish a connection with the database. This needs to be created first using the following procedure.

- Using the navigation tree in the GlassFish Administration Console, navigate to **Resources, JDBC, JDBC Resources**. A list of resources will be displayed in the **JDBC Resources** frame.
- Click **New**. The **New JDBC Resource** frame will be displayed.
- In the **JNDI Name** field, enter the JNDI name that will be used to access this resource, for example enter **jdbc/MySQLDataSource**.
- In the **Pool Name** field, select a connection pool you want this resource to use from the drop-down listbox.
- Optionally, you can enter a description into the **Description** field.
- Additional properties can be added if required.
- Click **OK** to create the new JDBC resource. The **JDBC Resources** frame will list all available JDBC Resources.

2.13.1 A Simple JSP Application with GlassFish, Connector/J and MySQL

This section shows how to deploy a simple JSP application on GlassFish, that connects to a MySQL database.

This example assumes you have already set up a suitable Connection Pool and JDBC Resource, as explained in the preceding sections. It is also assumed you have a sample database installed, such as [world](#).

The main application code, [index.jsp](#) is presented here:

```
<%@ page import="java.sql.*, javax.sql.*, java.io.*, javax.naming.*" %>
<html>
<head><title>Hello world from JSP</title></head>
<body>
<%
    InitialContext ctx;
    DataSource ds;
    Connection conn;
    Statement stmt;
    ResultSet rs;
    try {
        ctx = new InitialContext();
        ds = (DataSource) ctx.lookup("java:comp/env/jdbc/MySQLDataSource");
        //ds = (DataSource) ctx.lookup("jdbc/MySQLDataSource");
        conn = ds.getConnection();
        stmt = conn.createStatement();
        rs = stmt.executeQuery("SELECT * FROM Country");
        while(rs.next()) {
%>
            <h3>Name: <%= rs.getString("Name") %></h3>
            <h3>Population: <%= rs.getString("Population") %></h3>
<%
        }
    }
    catch (SQLException se) {
%>
        <%= se.getMessage() %>
<%
    }
    catch (NamingException ne) {
%>
        <%= ne.getMessage() %>
<%
    }
}
```



```
%>
</body>
</html>
```

In addition two XML files are required: [web.xml](#), and [sun-web.xml](#). There may be other files present, such as classes and images. These files are organized into the directory structure as follows:

```
index.jsp
WEB-INF
|
- web.xml
- sun-web.xml
```

The code for [web.xml](#) is:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  <display-name>HelloWebApp</display-name>
  <distributed/>
  <resource-ref>
    <res-ref-name>jdbc/MySQLDataSource</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
</web-app>
```

The code for [sun-web.xml](#) is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server 8.1 Servlet 2.4//EN" "http:
<sun-web-app>
  <context-root>HelloWebApp</context-root>
  <resource-ref>
    <res-ref-name>jdbc/MySQLDataSource</res-ref-name>
    <jndi-name>jdbc/MySQLDataSource</jndi-name>
  </resource-ref>
</sun-web-app>
```

These XML files illustrate a very important aspect of running JDBC applications on GlassFish. On GlassFish it is important to map the string specified for a JDBC resource to its JNDI name, as set up in the GlassFish administration console. In this example, the JNDI name for the JDBC resource, as specified in the GlassFish Administration console when creating the JDBC Resource, was [jdbc/MySQLDataSource](#). This must be mapped to the name given in the application. In this example the name specified in the application, [jdbc/MySQLDataSource](#), and the JNDI name, happen to be the same, but this does not necessarily have to be the case. Note that the XML element `<res-ref-name>` is used to specify the name as used in the application source code, and this is mapped to the JNDI name specified using the `<jndi-name>` element, in the file [sun-web.xml](#). The resource also has to be created in the [web.xml](#) file, although the mapping of the resource to a JNDI name takes place in the [sun-web.xml](#) file.

If you do not have this mapping set up correctly in the XML files you will not be able to lookup the data source using a JNDI lookup string such as:

```
ds = (DataSource) ctx.lookup("java:comp/env/jdbc/MySQLDataSource");
```

You will still be able to access the data source directly using:


```
ds = (DataSource) ctx.lookup("jdbc/MySQLDataSource");
```

With the source files in place, in the correct directory structure, you are ready to deploy the application:

1. In the navigation tree, navigate to **Applications** - the **Applications** frame will be displayed. Click **Deploy**.
2. You can now deploy an application packaged into a single WAR file from a remote client, or you can choose a packaged file or directory that is locally accessible to the server. If you are simply testing an application locally you can simply point GlassFish at the directory that contains your application, without needing to package the application into a WAR file.
3. Now select the application type from the **Type** drop-down listbox, which in this example is [Web application](#).
4. Click OK.

Now, when you navigate to the **Applications** frame, you will have the option to **Launch**, **Redeploy**, or **Restart** your application. You can test your application by clicking **Launch**. The application will connection to the MySQL database and display the Name and Population of countries in the [Country](#) table.

2.13.2 A Simple Servlet with GlassFish, Connector/J and MySQL

This section describes a simple servlet that can be used in the GlassFish environment to access a MySQL database. As with the previous section, this example assumes the sample database [world](#) is installed.

The project is set up with the following directory structure:

```
index.html
WEB-INF
|
- web.xml
- sun-web.xml
- classes
  |
  - HelloWorldServlet.java
  - HelloWorldServlet.class
```

The code for the servlet, located in [HelloWebServlet.java](#), is as follows:

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
public class HelloWorldServlet extends HttpServlet {
    InitialContext ctx = null;
    DataSource ds = null;
    Connection conn = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    String sql = "SELECT Name, Population FROM Country WHERE Name=?";
    public void init () throws ServletException {
        try {
            ctx = new InitialContext();
            ds = (DataSource) ctx.lookup("java:comp/env/jdbc/MySQLDataSource");
            conn = ds.getConnection();
            ps = conn.prepareStatement(sql);
        }
        catch (SQLException se) {
            System.out.println("SQLException: " + se.getMessage());
        }
    }
}
```

```

    }
    catch (NamingException ne) {
        System.out.println("NamingException: "+ne.getMessage());
    }
}
public void destroy () {
    try {
        if (rs != null)
            rs.close();
        if (ps != null)
            ps.close();
        if (conn != null)
            conn.close();
        if (ctx != null)
            ctx.close();
    }
    catch (SQLException se) {
        System.out.println("SQLException: "+se.getMessage());
    }
    catch (NamingException ne) {
        System.out.println("NamingException: "+ne.getMessage());
    }
}
public void doPost(HttpServletRequest req, HttpServletResponse resp){
    try {
        String country_name = req.getParameter("country_name");
        resp.setContentType("text/html");
        PrintWriter writer = resp.getWriter();
        writer.println("<html><body>");
        writer.println("<p>Country: "+country_name+"</p>");
        ps.setString(1, country_name);
        rs = ps.executeQuery();
        if (!rs.next()){
            writer.println("<p>Country does not exist!</p>");
        }
        else {
            rs.beforeFirst();
            while(rs.next()) {
                writer.println("<p>Name: "+rs.getString("Name")+"</p>");
                writer.println("<p>Population: "+rs.getString("Population")+"</p>");
            }
        }
        writer.println("</body></html>");
        writer.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
public void doGet(HttpServletRequest req, HttpServletResponse resp){
    try {
        resp.setContentType("text/html");
        PrintWriter writer = resp.getWriter();
        writer.println("<html><body>");
        writer.println("<p>Hello from servlet doGet()</p>");
        writer.println("</body></html>");
        writer.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

In the preceding code a basic `doGet()` method is implemented, but is not used in the example. The code to establish the connection with the database is as shown in the previous example, [Section 2.13.1, “A Simple JSP Application with GlassFish, Connector/J and MySQL”](#), and is most conveniently located in the servlet `init()` method. The corresponding freeing of resources is located in the destroy method. The main functionality of the servlet is located in the `doPost()` method. If the user enters into the input form a country name that can be located in the database, the population of

the country is returned. The code is invoked using a POST action associated with the input form. The form is defined in the file `index.html`:

```
<html>
  <head><title>HelloWebServlet</title></head>

  <body>
    <h1>HelloWebServlet</h1>

    <p>Please enter country name:</p>

    <form action="HelloWebServlet" method="POST">
      <input type="text" name="country_name" length="50" />
      <input type="submit" value="Submit" />
    </form>

  </body>
</html>
```

The XML files `web.xml` and `sun-web.xml` are as for the example in the preceding section, [Section 2.13.1, “A Simple JSP Application with GlassFish, Connector/J and MySQL”](#), no additional changes are required.

When compiling the Java source code, you will need to specify the path to the file `javaee.jar`. On Windows, this can be done as follows:

```
shell> javac -classpath c:\glassfishv3\glassfish\lib\javaee.jar HelloWebServlet.java
```

Once the code is correctly located within its directory structure, and compiled, the application can be deployed in GlassFish. This is done in exactly the same way as described in the preceding section, [Section 2.13.1, “A Simple JSP Application with GlassFish, Connector/J and MySQL”](#).

Once deployed the application can be launched from within the GlassFish Administration Console. Enter a country name such as “England”, and the application will return “Country does not exist!”. Enter “France”, and the application will return a population of 59225700.

2.14 Troubleshooting Connector/J Applications

This section explains the symptoms and resolutions for the most commonly encountered issues with applications using MySQL Connector/J.

Questions

- [2.14.1](#): When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What is going on? I can connect just fine with the MySQL command-line client.

- [2.14.2](#): My application throws an `SQLException 'No Suitable Driver'`. Why is this happening?
- [2.14.3](#): I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
```

```
are trying to connect to?
(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

- [2.14.4](#): I have a servlet/application that works fine for a day, and then stops working overnight
- [2.14.5](#): I'm trying to use JDBC 2.0 updatable result sets, and I get an exception saying my result set is not updatable.
- [2.14.6](#): I cannot connect to the MySQL server using Connector/J, and I'm sure the connection parameters are correct.
- [2.14.7](#): My application is deployed through JBoss and I am using transactions to handle the statements on the MySQL database. Under heavy loads, I am getting an error and stack trace, but these only occur after a fixed period of heavy activity.
- [2.14.8](#): When using `gcj`, a `java.io.CharConversionException` exception is raised when working with certain character sequences.
- [2.14.9](#): Updating a table that contains a `primary key` that is either `FLOAT` or compound primary key that uses `FLOAT` fails to update the table and raises an exception.
- [2.14.10](#): You get an `ER_NET_PACKET_TOO_LARGE` exception, even though the binary blob size you want to insert using JDBC is safely below the `max_allowed_packet` size.
- [2.14.11](#): What should you do if you receive error messages similar to the following:
"Communications link failure – Last packet sent to the server was X ms ago"?
- [2.14.12](#): Why does Connector/J not reconnect to MySQL and re-issue the statement after a communication failure, instead of throwing an Exception, even though I use the `autoReconnect` connection string option?
- [2.14.13](#): How can I use 3-byte UTF8 with Connector/J?
- [2.14.14](#): How can I use 4-byte UTF8, `utf8mb4` with Connector/J?
- [2.14.15](#): Using `useServerPrepStmts=false` and certain character encodings can lead to corruption when inserting BLOBs. How can this be avoided?

Questions and Answers

2.14.1: When I try to connect to the database with MySQL Connector/J, I get the following exception:

```
SQLException: Server configuration denies access to data source
SQLState: 08001
VendorError: 0
```

What is going on? I can connect just fine with the MySQL command-line client.

MySQL Connector/J must use TCP/IP sockets to connect to MySQL, as Java does not support Unix Domain Sockets. Therefore, when MySQL Connector/J connects to MySQL, the security manager in MySQL server will use its grant tables to determine whether the connection is permitted.

You must add the necessary security credentials to the MySQL server for this to happen, using the `GRANT` statement to your MySQL Server. See [GRANT Syntax](#), for more information.

Note

Testing your connectivity with the `mysql` command-line client will not work unless you add the "host" flag, and use something other than `localhost`

for the host. The `mysql` command-line client will use Unix domain sockets if you use the special host name `localhost`. If you are testing connectivity to `localhost`, use `127.0.0.1` as the host name instead.

Warning

Changing privileges and permissions improperly in MySQL can potentially cause your server installation to not have optimal security properties.

2.14.2: My application throws an `SQLException` 'No Suitable Driver'. Why is this happening?

There are three possible causes for this error:

- The Connector/J driver is not in your `CLASSPATH`, see [Section 2.3, "Connector/J Installation"](#).
- The format of your connection URL is incorrect, or you are referencing the wrong JDBC driver.
- When using `DriverManager`, the `jdbc.drivers` system property has not been populated with the location of the Connector/J driver.

2.14.3: I'm trying to use MySQL Connector/J in an applet or application and I get an exception similar to:

```
SQLException: Cannot connect to MySQL server on host:3306.
Is there a MySQL server running on the machine/port you
are trying to connect to?
(java.security.AccessControlException)
SQLState: 08S01
VendorError: 0
```

Either you're running an Applet, your MySQL server has been installed with the "skip-networking" option set, or your MySQL server has a firewall sitting in front of it.

Applets can only make network connections back to the machine that runs the web server that served the .class files for the applet. This means that MySQL must run on the same machine (or you must have some sort of port re-direction) for this to work. This also means that you will not be able to test applets from your local file system, you must always deploy them to a web server.

MySQL Connector/J can only communicate with MySQL using TCP/IP, as Java does not support Unix domain sockets. TCP/IP communication with MySQL might be affected if MySQL was started with the "skip-networking" flag, or if it is firewalled.

If MySQL has been started with the "skip-networking" option set (the Debian Linux package of MySQL server does this for example), you need to comment it out in the file `/etc/mysql/my.cnf` or `/etc/my.cnf`. Of course your `my.cnf` file might also exist in the `data` directory of your MySQL server, or anywhere else (depending on how MySQL was compiled for your system). Binaries created by us always look in `/etc/my.cnf` and `datadir/my.cnf`. If your MySQL server has been firewalled, you will need to have the firewall configured to allow TCP/IP connections from the host where your Java code is running to the MySQL server on the port that MySQL is listening to (by default, 3306).

2.14.4: I have a servlet/application that works fine for a day, and then stops working overnight

MySQL closes connections after 8 hours of inactivity. You either need to use a connection pool that handles stale connections or use the `autoReconnect` parameter (see [Section 2.5.1, "Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J"](#)).

Also, catch `SQLExceptions` in your application and deal with them, rather than propagating them all the way until your application exits. This is just good programming practice. MySQL Connector/J will set the `SQLState` (see `java.sql.SQLException.getSQLState()` in your API docs) to `08S01` when it encounters network-connectivity issues during the processing of a query. Attempt to reconnect to MySQL at this point.

The following (simplistic) example shows what code that can handle these exceptions might look like:

Example 2.12 Connector/J: Example of transaction with retry logic

```
public void doBusinessOp() throws SQLException {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    //
    // How many times do you want to retry the transaction
    // (or at least _getting_ a connection)?
    //
    int retryCount = 5;
    boolean transactionCompleted = false;
    do {
        try {
            conn = getConnection(); // assume getting this from a
                                   // javax.sql.DataSource, or the
                                   // java.sql.DriverManager
            conn.setAutoCommit(false);
            //
            // Okay, at this point, the 'retry-ability' of the
            // transaction really depends on your application logic,
            // whether or not you're using autocommit (in this case
            // not), and whether you're using transactional storage
            // engines
            //
            // For this example, we'll assume that it's _not_ safe
            // to retry the entire transaction, so we set retry
            // count to 0 at this point
            //
            // If you were using exclusively transaction-safe tables,
            // or your application could recover from a connection going
            // bad in the middle of an operation, then you would not
            // touch 'retryCount' here, and just let the loop repeat
            // until retryCount == 0.
            //
            retryCount = 0;
            stmt = conn.createStatement();
            String query = "SELECT foo FROM bar ORDER BY baz";
            rs = stmt.executeQuery(query);
            while (rs.next()) {
            }
            rs.close();
            rs = null;
            stmt.close();
            stmt = null;
            conn.commit();
            conn.close();
            conn = null;
            transactionCompleted = true;
        } catch (SQLException sqlEx) {
            //
            // The two SQL states that are 'retry-able' are 08S01
            // for a communications error, and 40001 for deadlock.
            //
            // Only retry if the error was due to a stale connection,
            // communications problem or deadlock
            //
            String sqlState = sqlEx.getSQLState();
            if ("08S01".equals(sqlState) || "40001".equals(sqlState)) {
                retryCount -= 1;
            } else {
                retryCount = 0;
            }
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException sqlEx) {
                    // You'd probably want to log this...
                }
            }
        }
    } while (retryCount > 0);
}
```

```

    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException sqlEx) {
            // You'd probably want to log this as well...
        }
    }
    if (conn != null) {
        try {
            //
            // If we got here, and conn is not null, the
            // transaction should be rolled back, as not
            // all work has been done
            try {
                conn.rollback();
            } finally {
                conn.close();
            }
        } catch (SQLException sqlEx) {
            //
            // If we got an exception here, something
            // pretty serious is going on, so we better
            // pass it up the stack, rather than just
            // logging it...
            throw sqlEx;
        }
    }
}
} while (!transactionCompleted && (retryCount > 0));
}

```

Note

Use of the [autoReconnect](#) option is not recommended because there is no safe method of reconnecting to the MySQL server without risking some corruption of the connection state or database state information. Instead, use a connection pool, which will enable your application to connect to the MySQL server using an available connection from the pool. The [autoReconnect](#) facility is deprecated, and may be removed in a future release.

2.14.5: I'm trying to use JDBC 2.0 updatable result sets, and I get an exception saying my result set is not updatable.

Because MySQL does not have row identifiers, MySQL Connector/J can only update result sets that have come from queries on tables that have at least one [primary key](#), the query must select every primary key column, and the query can only span one table (that is, no joins). This is outlined in the JDBC specification.

Note that this issue only occurs when using updatable result sets, and is caused because Connector/J is unable to guarantee that it can identify the correct rows within the result set to be updated without having a unique reference to each row. There is no requirement to have a unique field on a table if you are using [UPDATE](#) or [DELETE](#) statements on a table where you can individually specify the criteria to be matched using a [WHERE](#) clause.

2.14.6: I cannot connect to the MySQL server using Connector/J, and I'm sure the connection parameters are correct.

Make sure that the [skip-networking](#) option has not been enabled on your server. Connector/J must be able to communicate with your server over TCP/IP; named sockets are not supported. Also ensure that you are not filtering connections through a firewall or other network security system. For more information, see [Can't connect to \[local\] MySQL server](#).

2.14.7: My application is deployed through JBoss and I am using transactions to handle the statements on the MySQL database. Under heavy loads, I am getting an error and stack trace, but these only occur after a fixed period of heavy activity.

This is a JBoss, not Connector/J, issue and is connected to the use of transactions. Under heavy loads the time taken for transactions to complete can increase, and the error is caused because you have exceeded the predefined timeout.

You can increase the timeout value by setting the `TransactionTimeout` attribute to the `TransactionManagerService` within the `/conf/jboss-service.xml` file (pre-4.0.3) or `/deploy/jta-service.xml` for JBoss 4.0.3 or later. See `TransactionTimeout` within the JBoss wiki for more information.

2.14.8: When using `gcj`, a `java.io.CharConversionException` exception is raised when working with certain character sequences.

This is a known issue with `gcj` which raises an exception when it reaches an unknown character or one it cannot convert. Add `useJvmCharsetConverters=true` to your connection string to force character conversion outside of the `gcj` libraries, or try a different JDK.

2.14.9: Updating a table that contains a primary key that is either `FLOAT` or compound primary key that uses `FLOAT` fails to update the table and raises an exception.

Connector/J adds conditions to the `WHERE` clause during an `UPDATE` to check the old values of the primary key. If there is no match, then Connector/J considers this a failure condition and raises an exception.

The problem is that rounding differences between supplied values and the values stored in the database may mean that the values never match, and hence the update fails. The issue will affect all queries, not just those from Connector/J.

To prevent this issue, use a primary key that does not use `FLOAT`. If you have to use a floating point column in your primary key, use `DOUBLE` or `DECIMAL` types in place of `FLOAT`.

2.14.10: You get an `ER_NET_PACKET_TOO_LARGE` exception, even though the binary blob size you want to insert using JDBC is safely below the `max_allowed_packet` size.

This is because the `hexEscapeBlock()` method in `com.mysql.jdbc.PreparedStatement.streamToBytes()` may almost double the size of your data.

2.14.11: What should you do if you receive error messages similar to the following: “Communications link failure – Last packet sent to the server was X ms ago”?

Generally speaking, this error suggests that the network connection has been closed. There can be several root causes:

- Firewalls or routers may clamp down on idle connections (the MySQL client/server protocol does not ping).
- The MySQL Server may be closing idle connections that exceed the `wait_timeout` or `interactive_timeout` threshold.

To help troubleshoot these issues, the following tips can be used. If a recent (5.1.13+) version of Connector/J is used, you will see an improved level of information compared to earlier versions. Older versions simply display the last time a packet was sent to the server, which is frequently 0 ms ago. This is of limited use, as it may be that a packet was just sent, while a packet from the server has not been received for several hours. Knowing the period of time since Connector/J last received a packet from the server is useful information, so if this is not displayed in your exception message, it is recommended that you update Connector/J.

Further, if the time a packet was last sent/received exceeds the `wait_timeout` or `interactive_timeout` threshold, this is noted in the exception message.

Although network connections can be volatile, the following can be helpful in avoiding problems:

- Ensure connections are valid when used from the connection pool. Use a query that starts with `/* ping */` to execute a lightweight ping instead of full query. Note, the syntax of the ping needs to be exactly as specified here.
- Minimize the duration a connection object is left idle while other application logic is executed.
- Explicitly validate the connection before using it if the connection has been left idle for an extended period of time.
- Ensure that `wait_timeout` and `interactive_timeout` are set sufficiently high.
- Ensure that `tcpKeepalive` is enabled.
- Ensure that any configurable firewall or router timeout settings allow for the maximum expected connection idle time.

Note

Do not expect to be able to reuse a connection without problems, if it has being lying idle for a period. If a connection is to be reused after being idle for any length of time, ensure that you explicitly test it before reusing it.

2.14.12: Why does Connector/J not reconnect to MySQL and re-issue the statement after a communication failure, instead of throwing an Exception, even though I use the `autoReconnect` connection string option?

There are several reasons for this. The first is transactional integrity. The MySQL Reference Manual states that “there is no safe method of reconnecting to the MySQL server without risking some corruption of the connection state or database state information”. Consider the following series of statements for example:

```
conn.createStatement().execute(
    "UPDATE checking_account SET balance = balance - 1000.00 WHERE customer='Smith'");
conn.createStatement().execute(
    "UPDATE savings_account SET balance = balance + 1000.00 WHERE customer='Smith'");
conn.commit();
```

Consider the case where the connection to the server fails after the `UPDATE` to `checking_account`. If no exception is thrown, and the application never learns about the problem, it will continue executing. However, the server did not commit the first transaction in this case, so that will get rolled back. But execution continues with the next transaction, and increases the `savings_account` balance by 1000. The application did not receive an exception, so it continued regardless, eventually committing the second transaction, as the commit only applies to the changes made in the new connection. Rather than a transfer taking place, a deposit was made in this example.

Note that running with `autocommit` enabled does not solve this problem. When Connector/J encounters a communication problem, there is no means to determine whether the server processed the currently executing statement or not. The following theoretical states are equally possible:

- The server never received the statement, and therefore no related processing occurred on the server.
- The server received the statement, executed it in full, but the response was not received by the client.

If you are running with `autocommit` enabled, it is not possible to guarantee the state of data on the server when a communication exception is encountered. The statement may have reached the server, or it may not. All you know is that communication failed at some point, before the client received confirmation (or data) from the server. This does not only affect `autocommit` statements though. If the communication problem occurred during `Connection.commit()`, the question arises of whether the transaction was committed on the server before the communication failed, or whether the server received the commit request at all.

The second reason for the generation of exceptions is that transaction-scoped contextual data may be vulnerable, for example:

- Temporary tables.
- User-defined variables.
- Server-side prepared statements.

These items are lost when a connection fails, and if the connection silently reconnects without generating an exception, this could be detrimental to the correct execution of your application.

In summary, communication errors generate conditions that may well be unsafe for Connector/J to simply ignore by silently reconnecting. It is necessary for the application to be notified. It is then for the application developer to decide how to proceed in the event of connection errors and failures.

2.14.13: How can I use 3-byte UTF8 with Connector/J?

To use 3-byte UTF8 with Connector/J set `characterEncoding=utf8` and set `useUnicode=true` in the connection string.

2.14.14: How can I use 4-byte UTF8, `utf8mb4` with Connector/J?

To use 4-byte UTF8 with Connector/J configure the MySQL server with `character_set_server=utf8mb4`. Connector/J will then use that setting as long as `characterEncoding` has not been set in the connection string. This is equivalent to autodetection of the character set.

2.14.15: Using `useServerPrepStmts=false` and certain character encodings can lead to corruption when inserting BLOBs. How can this be avoided?

When using certain character encodings, such as SJIS, CP932, and BIG5, it is possible that BLOB data contains characters that can be interpreted as control characters, for example, backslash, '\'. This can lead to corrupted data when inserting BLOBs into the database. There are two things that need to be done to avoid this:

1. Set the connection string option `useServerPrepStmts` to `true`.
2. Set `SQL_MODE` to `NO_BACKSLASH_ESCAPES`.

2.15 Known Issues and Limitations

The following are some known issues and limitations for MySQL Connector/J:

- When Connector/J retrieves timestamps for a daylight saving time (DST) switch day using the `getTimeStamp()` method on the result set, some of the returned values might be wrong. The errors can be avoided by using the following connection options when connecting to a database:

```
useTimezone=true
useLegacyDatetimeCode=false
serverTimezone=UTC
```

2.16 Connector/J Support

2.16.1 Connector/J Community Support

Oracle provides assistance to the user community by means of its mailing lists. For Connector/J related issues, you can get help from experienced users by using the MySQL and Java mailing list. Archives and subscription information is available online at <http://lists.mysql.com/java>.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [MySQL Mailing Lists](#).

Community support from experienced users is also available through the [JDBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [MySQL Community Support at the MySQL Forums](#).

2.16.2 How to Report Connector/J Bugs or Problems

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

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

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release.

This section will help you write your report correctly so that you do not waste your time doing things that may not help us much or at all.

If you have a repeatable bug report, please report it to the bugs database at <http://bugs.mysql.com/>. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

To report other problems, you can use one of the MySQL mailing lists.

Remember that it is possible for us to respond to a message containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details do not matter.

A good principle is this: If you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of Connector/J or MySQL used, and (b) not fully describing the platform on which Connector/J is installed (including the JVM version, and the platform type and version number that MySQL itself is installed on).

This is highly relevant information, and in 99 cases out of 100, the bug report is useless without it. Very often we get questions like, "Why doesn't this work for me?" Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has already been fixed in newer MySQL versions.

Sometimes the error is platform-dependent; in such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If at all possible, create a repeatable, standalone testcase that doesn't involve any third-party classes.

To streamline this process, we ship a base class for testcases with Connector/J, named `'com.mysql.jdbc.util.BaseBugReport'`. To create a testcase for Connector/J using this class,

create your own class that inherits from `com.mysql.jdbc.util.BaseBugReport` and override the methods `setUp()`, `tearDown()` and `runTest()`.

In the `setUp()` method, create code that creates your tables, and populates them with any data needed to demonstrate the bug.

In the `runTest()` method, create code that demonstrates the bug using the tables and data you created in the `setUp` method.

In the `tearDown()` method, drop any tables you created in the `setUp()` method.

In any of the above three methods, use one of the variants of the `getConnection()` method to create a JDBC connection to MySQL:

- `getConnection()` - Provides a connection to the JDBC URL specified in `getUrl()`. If a connection already exists, that connection is returned, otherwise a new connection is created.
- `getNewConnection()` - Use this if you need to get a new connection for your bug report (that is, there is more than one connection involved).
- `getConnection(String url)` - Returns a connection using the given URL.
- `getConnection(String url, Properties props)` - Returns a connection using the given URL and properties.

If you need to use a JDBC URL that is different from 'jdbc:mysql:///test', override the method `getUrl()` as well.

Use the `assertTrue(boolean expression)` and `assertTrue(String failureMessage, boolean expression)` methods to create conditions that must be met in your testcase demonstrating the behavior you are expecting (vs. the behavior you are observing, which is why you are most likely filing a bug report).

Finally, create a `main()` method that creates a new instance of your testcase, and calls the `run` method:

```
public static void main(String[] args) throws Exception {
    new MyBugReport().run();
}
```

Once you have finished your testcase, and have verified that it demonstrates the bug you are reporting, upload it with your bug report to <http://bugs.mysql.com/>.

Chapter 3 MySQL Connector/ODBC Developer Guide

Table of Contents

3.1 Introduction to MySQL Connector/ODBC	107
3.2 Connector/ODBC Versions	108
3.3 General Information About ODBC and Connector/ODBC	109
3.3.1 Connector/ODBC Architecture	110
3.3.2 ODBC Driver Managers	111
3.4 Connector/ODBC Installation	112
3.4.1 Installing Connector/ODBC on Windows	113
3.4.2 Installing Connector/ODBC on Unix-like Systems	118
3.4.3 Installing Connector/ODBC on OS X	120
3.4.4 Building Connector/ODBC from a Source Distribution on Windows	121
3.4.5 Building Connector/ODBC from a Source Distribution on Unix	122
3.4.6 Building Connector/ODBC from a Source Distribution on OS X	124
3.4.7 Installing Connector/ODBC from the Development Source Tree	125
3.5 Configuring Connector/ODBC	125
3.5.1 Overview of Connector/ODBC Data Source Names	125
3.5.2 Connector/ODBC Connection Parameters	126
3.5.3 Configuring a Connector/ODBC DSN on Windows	133
3.5.4 Configuring a Connector/ODBC DSN on OS X	139
3.5.5 Configuring a Connector/ODBC DSN on Unix	142
3.5.6 Connecting Without a Predefined DSN	143
3.5.7 ODBC Connection Pooling	143
3.5.8 Getting an ODBC Trace File	144
3.6 Connector/ODBC Examples	146
3.6.1 Basic Connector/ODBC Application Steps	146
3.6.2 Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC	148
3.6.3 Connector/ODBC and Third-Party ODBC Tools	148
3.6.4 Using Connector/ODBC with Microsoft Access	149
3.6.5 Using Connector/ODBC with Microsoft Word or Excel	155
3.6.6 Using Connector/ODBC with Crystal Reports	158
3.6.7 Connector/ODBC Programming	164
3.7 Connector/ODBC Reference	171
3.7.1 Connector/ODBC API Reference	171
3.7.2 Connector/ODBC Data Types	175
3.7.3 Connector/ODBC Error Codes	176
3.8 Connector/ODBC Notes and Tips	177
3.8.1 Connector/ODBC General Functionality	177
3.8.2 Connector/ODBC Application-Specific Tips	178
3.8.3 Connector/ODBC Errors and Resolutions (FAQ)	183
3.9 Connector/ODBC Support	188
3.9.1 Connector/ODBC Community Support	188
3.9.2 How to Report Connector/ODBC Problems or Bugs	188
3.9.3 How to Submit a Connector/ODBC Patch	189

3.1 Introduction to MySQL Connector/ODBC

The MySQL Connector/ODBC is the name for the family of MySQL ODBC drivers (previously called MyODBC drivers) that provide access to a MySQL database using the industry standard Open Database Connectivity (ODBC) API. This reference covers Connector/ODBC 5.2, which includes the functionality of the Unicode driver and the ANSI driver, which formerly were split between Connector/ODBC 5.1 and Connector/ODBC 3.51.

MySQL Connector/ODBC provides both driver-manager based and native interfaces to the MySQL database, with full support for MySQL functionality, including stored procedures, [transactions](#) and, with Connector/ODBC 5.1 and higher, full Unicode compliance.

For more information on the ODBC API standard and how to use it, refer to <http://support.microsoft.com/kb/110093>.

The application development section of the ODBC API reference assumes a good working knowledge of C, general DBMS, and a familiarity with MySQL. For more information about MySQL functionality and its syntax, refer to <http://dev.mysql.com/doc/>.

Typically, you need to install Connector/ODBC only on Windows machines. For Unix and OS X, you can use the native MySQL network or named pipes to communicate with your MySQL database. You may need Connector/ODBC for Unix or OS X if you have an application that requires an ODBC interface to communicate with the database. Applications that require ODBC to communicate with MySQL include ColdFusion, Microsoft Office, and Filemaker Pro.

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

Key connector/ODBC topics include:

- Installing Connector/ODBC: [Section 3.4, “Connector/ODBC Installation”](#).
- The configuration options: [Section 3.5.2, “Connector/ODBC Connection Parameters”](#).
- An example that connects to a MySQL database from a Windows host: [Section 3.6.2, “Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC”](#).
- An example that uses Microsoft Access as an interface to a MySQL database: [Section 3.6.4, “Using Connector/ODBC with Microsoft Access”](#).
- General tips and notes, including how to obtain the last auto-increment ID: [Section 3.8.1, “Connector/ODBC General Functionality”](#).
- Application-specific usage tips and notes: [Section 3.8.2, “Connector/ODBC Application-Specific Tips”](#).
- A FAQ (Frequently Asked Questions) list: [Section 3.8.3, “Connector/ODBC Errors and Resolutions \(FAQ\)”](#).
- Additional Connector/ODBC support options: [Section 3.9, “Connector/ODBC Support”](#).

3.2 Connector/ODBC Versions

These are the versions of Connector/ODBC that are currently available:

- Connector/ODBC 5.3 series, now in GA status, is suitable for use with any MySQL version since 4.1 (it will not work with 4.0 or earlier releases). It conforms to the ODBC 3.8 specification and contains key ODBC 3.8 features including self-identification as a ODBC 3.8 driver, streaming of output parameters (supported for binary types only), and support of the SQL_ATTR_RESET_CONNECTION connection attribute (for the Unicode driver only). Connector/ODBC 5.3 also introduces a GTK+-based setup library, providing GUI DSN setup dialog on some Unix-based systems. The library is currently included in the Oracle Linux 6 and Debian 6 binary packages. Other new features in the 5.3 series include file DSN and bookmark support; see the [release notes for the 5.3 series](#) for details.
- Connector/ODBC 5.2 upgrades the ANSI driver of Connector/ODBC 3.51 to the 5.x code base. It also includes new features, such as enabling server-side prepared statements by default. At installation time, you can choose the Unicode driver for the broadest compatibility with data sources using various character sets, or the ANSI driver for optimal performance with a more limited range of character sets. It works with MySQL versions 4.1.1 and higher.

- Connector/ODBC 5.1, is a partial rewrite of the of the 3.51 code base, and is designed to work with MySQL versions 4.1.1 and newer.

Connector/ODBC 5.1 also includes the following changes and improvements over the 3.51 release:

- Improved support on Windows 64-bit platforms.
- Full Unicode support at the driver level. This includes support for the `SQL_WCHAR` data type, and support for Unicode login, password and DSN configurations. For more information, see [Microsoft Knowledgebase Article #716246](#).
- Support for the `SQL_NUMERIC_STRUCT` data type, which provides easier access to the precise definition of numeric values. For more information, see [Microsoft Knowledgebase Article #714556](#)
- Native Windows setup library. This replaces the Qt library based interface for configuring DSN information within the ODBC Data Sources application.
- Support for the ODBC descriptor, which improves the handling and metadata of columns and parameter data. For more information, see [Microsoft Knowledgebase Article #716339](#).
- Connector/ODBC 3.51, also known as the MySQL ODBC 3.51 driver, is a 32-bit ODBC driver. Connector/ODBC 3.51 has support for ODBC 3.5x specification level 1 (complete core API + level 2 features) to continue to provide all functionality of ODBC for accessing MySQL.

The manual for versions of Connector/ODBC older than 5.3 can be located in the corresponding binary or source distribution.

Note

Versions of Connector/ODBC earlier than the 3.51 revision were not fully compliant with the ODBC specification.

Note

From this section onward, the primary focus of this guide is the Connector/ODBC 5.3 driver.

Note

Version numbers for MySQL products are formatted as X.X.X. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.XX.XX. For example, the official MySQL formatted version number 5.0.9 may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display formats are different.

3.3 General Information About ODBC and Connector/ODBC

ODBC (Open Database Connectivity) provides a way for client programs to access a wide range of databases or data sources. ODBC is a standardized API that enables connections to SQL database servers. It was developed according to the specifications of the SQL Access Group and defines a set of function calls, error codes, and data types that can be used to develop database-independent applications. ODBC usually is used when database independence or simultaneous access to different data sources is required.

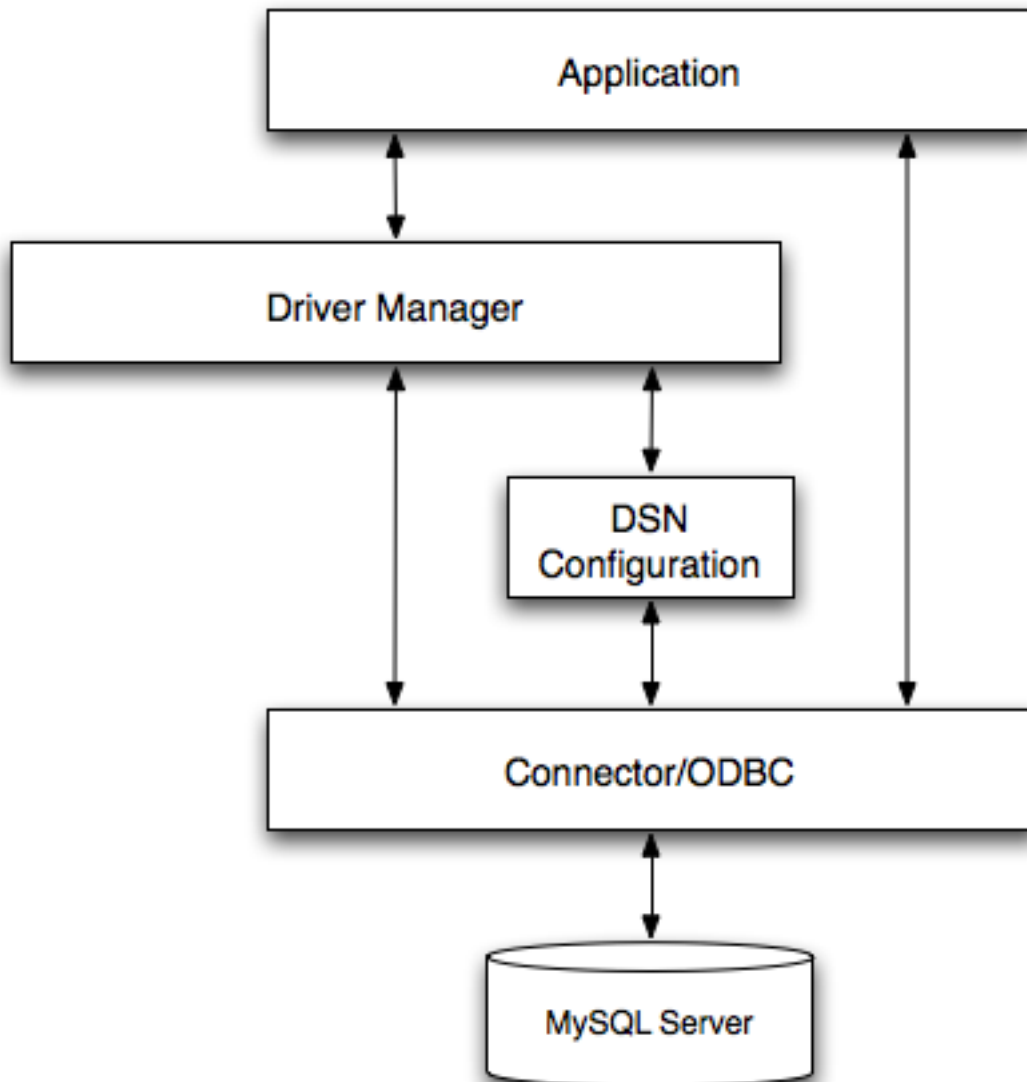
For more information about ODBC, refer to <http://support.microsoft.com/kb/110093>.

Open Database Connectivity (ODBC) is a widely accepted application-programming interface (API) for database access. It is based on the Call-Level Interface (CLI) specifications from X/Open and ISO/IEC for database APIs and uses Structured Query Language (SQL) as its database access language.

A survey of ODBC functions supported by Connector/ODBC is given at [Section 3.7.1, “Connector/ODBC API Reference”](#). For general information about ODBC, see <http://support.microsoft.com/kb/110093>.

3.3.1 Connector/ODBC Architecture

The Connector/ODBC architecture is based on five components, as shown in the following diagram:



- **Application:**

The Application uses the ODBC API to access the data from the MySQL server. The ODBC API in turn communicates with the Driver Manager. The Application communicates with the Driver Manager using the standard ODBC calls. The Application does not care where the data is stored, how it is stored, or even how the system is configured to access the data. It needs to know only the Data Source Name (DSN).

A number of tasks are common to all applications, no matter how they use ODBC. These tasks are:

- Selecting the MySQL server and connecting to it.
- Submitting SQL statements for execution.

- Retrieving results (if any).
- Processing errors.
- [Committing](#) or [rolling back](#) the [transaction](#) enclosing the SQL statement.
- Disconnecting from the MySQL server.

Because most data access work is done with SQL, the primary tasks for applications that use ODBC are submitting SQL statements and retrieving any results generated by those statements.

- **Driver manager:**

The Driver Manager is a library that manages communication between application and driver or drivers. It performs the following tasks:

- Resolves Data Source Names (DSN). The DSN is a configuration string that identifies a given database driver, database, database host and optionally authentication information that enables an ODBC application to connect to a database using a standardized reference.

Because the database connectivity information is identified by the DSN, any ODBC-compliant application can connect to the data source using the same DSN reference. This eliminates the need to separately configure each application that needs access to a given database; instead you instruct the application to use a pre-configured DSN.

- Loading and unloading of the driver required to access a specific database as defined within the DSN. For example, if you have configured a DSN that connects to a MySQL database then the driver manager will load the Connector/ODBC driver to enable the ODBC API to communicate with the MySQL host.
- Processes ODBC function calls or passes them to the driver for processing.

- **Connector/ODBC Driver:**

The Connector/ODBC driver is a library that implements the functions supported by the ODBC API. It processes ODBC function calls, submits SQL requests to MySQL server, and returns results back to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by MySQL.

- **DSN Configuration:**

The ODBC configuration file stores the driver and database information required to connect to the server. It is used by the Driver Manager to determine which driver to be loaded according to the definition in the DSN. The driver uses this to read connection parameters based on the DSN specified. For more information, [Section 3.5, “Configuring Connector/ODBC”](#).

- **MySQL Server:**

The MySQL database where the information is stored. The database is used as the source of the data (during queries) and the destination for data (during inserts and updates).

3.3.2 ODBC Driver Managers

An ODBC Driver Manager is a library that manages communication between the ODBC-aware application and any drivers. Its main functionality includes:

- Resolving Data Source Names (DSN).
- Driver loading and unloading.
- Processing ODBC function calls or passing them to the driver.

Most ODBC Driver Manager implementations also include an administration application that makes the configuration of DSN and drivers easier. Examples and information on ODBC Driver Managers for different operating systems are listed below:

- Windows: Microsoft Windows ODBC Driver Manager ([odbc32.dll](#)). It is included in the Windows operating system. See <http://support.microsoft.com/kb/110093> for more information.
- OS X: ODBC Administrator is a GUI application for OS X. It provides a simplified configuration mechanism for the iODBC Driver Manager. You can configure DSN and driver information either through ODBC Administrator or through the iODBC configuration files. This also means that you can test ODBC Administrator configurations using the [iodbctest](#) command. See <http://support.apple.com/kb/DL895> for more information.
- Unix:
 - [unixODBC](#) Driver Manager for Unix ([libodbc.so](#)). See <http://www.unixodbc.org>, for more information.
 - [iODBC](#) Driver Manager for Unix ([libiodbc.so](#)). See <http://www.iodbc.org>, for more information.

3.4 Connector/ODBC Installation

This section explains where to download Connector/ODBC, and how to run the installer, copy the files manually, or build from source.

Where to Get Connector/ODBC

You can get a copy of the latest version of Connector/ODBC binaries and sources from our Web site at <http://dev.mysql.com/downloads/connector/odbc/>.

For more information about Connector/ODBC, visit <http://www.mysql.com/products/myodbc/>.

For more information about licensing, visit <http://www.mysql.com/company/legal/licensing/>.

Choosing Unicode or ANSI Driver

Connector/ODBC offers the flexibility to handle data using any character set through its **Unicode-enabled** driver, or the maximum raw speed for a more limited range of character sets through its **ANSI** driver. Some users postponed their upgrade to Connector/ODBC 5.1, remaining with the older 3.51 version to keep this performance edge. As of Connector/ODBC 5.2, both kinds of drivers are available based on the 5.x code base: you can choose either a Unicode-enabled driver or an ANSI driver on the download page. The Unicode-enabled driver, recommended for most users, has no special qualifier in the download filename. The ANSI driver includes **-ansi-** in the download filename.

Note

You can install either the Unicode driver on a Windows system, or the ANSI driver, or both. The drivers are distinguished in the list of installed software and in the names of libraries and directories by a **w** (for “wide characters”) for the Unicode driver, and **a** in the ANSI driver.

Choosing Binary or Source Installation Method

You can install the Connector/ODBC drivers using two different methods:

- The **binary installation** is the easiest and most straightforward method of installation. You receive all the necessary libraries and other files pre-built, with an installer program or batch script to perform all necessary copying and configuration.

- The **source installation** method is intended for platforms where a binary installation package is not available, or in situations where you want to customize or modify the installation process or Connector/ODBC drivers before installation.

If a binary distribution is not available for a particular platform, and you build the driver from the original source code, you can contribute the binaries you create to MySQL by sending a mail message to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com), so that it becomes available for other users.

Supported Platforms

Connector/ODBC can be used on all major platforms supported by MySQL. You can install it on:

- Windows XP, 2003, Vista and 7.
- All Unix-like Operating Systems, including: Amiga, BSDI, DEC, FreeBSD, HP-UX 10/11, Linux, NetBSD, OpenBSD, OS/2, SGI Irix, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64 Unix.
- OS X and OS X Server.

Note

On all non-Windows platforms except OS X, the driver is built against [unixODBC](#) and is expecting a 2-byte [SQLWCHAR](#), not 4 bytes as [iODBC](#) is using. For this reason, the binaries are **only** compatible with [unixODBC](#); recompile the driver against [iODBC](#) to use them together. For further information, see [Section 3.3.2, “ODBC Driver Managers”](#).

For further instructions, consult the documentation corresponding to the platform where you are installing and whether you are running a binary installer or building from source:

Platform	Binary Installer	Build from Source
Windows	Installation Instructions	Build Instructions
Unix/Linux	Installation Instructions	Build Instructions
OS X	Installation Instructions	

3.4.1 Installing Connector/ODBC on Windows

Before installing the Connector/ODBC drivers on Windows:

- Make sure your Microsoft Data Access Components (MDAC) are up to date. You can obtain the latest version from the [Microsoft Data Access and Storage](#) Web site.
- Make sure you have the Microsoft Visual C++ 2013 Redistributable Package installed on your system. The package is available at the [Microsoft Download Center](#). Use the version of the package that matches the system type of your Connector/ODBC driver: use the 64-bit version (marked by “x64” in the package’s title and filename) if you are running a 64-bit driver, and use the 32-bit version (marked by “x86” in the package’s title and filename) if you are running a 32-bit driver.

There are different distribution types to use when installing for Windows. The software that is installed is identical in each case, only the installation method is different.

- The **zipped installer** consists of a zipped package containing a standalone installation application. To install from this package, unzip the installer, and then run the installation application.

The **MSI installer** is an installation file that can be used with the installer included in Windows 2000, Windows XP and Windows Server 2003.

See [Section 3.4.1.1, “Installing the Windows Connector/ODBC Driver Using an Installer”](#) for the steps to complete the installation with either of these installers.

- The **zipped DLL** package contains DLL files that must be manually installed. See [Section 3.4.1.2, “Installing the Windows Connector/ODBC Driver Using the Zipped DLL Package”](#) for the steps to complete this type of installation.

Note

An OLEDB/ODBC driver for Windows 64-bit is available from [Microsoft Downloads](#).

3.4.1.1 Installing the Windows Connector/ODBC Driver Using an Installer

The zipped or MSI installer packages offer a very simple method for installing the Connector/ODBC drivers. If you have downloaded the zipped installer, extract the installer application. The basic installation process is identical for both installers.

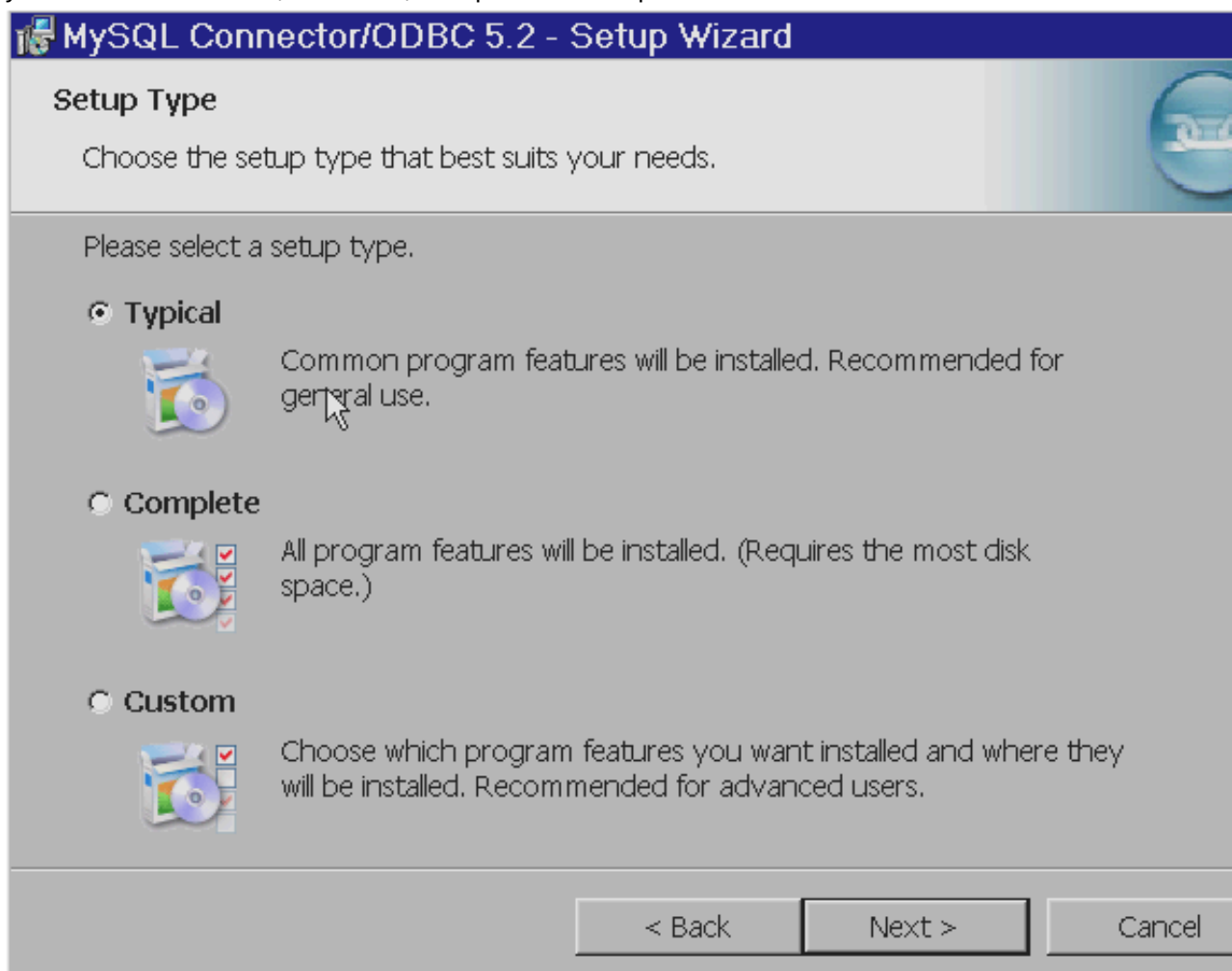
Follow these steps to complete the installation:

1. Double-click the standalone installer that you extracted, or the MSI file you downloaded.
2. The MySQL Connector/ODBC Setup Wizard starts. Click the **Next** button to begin the installation process.

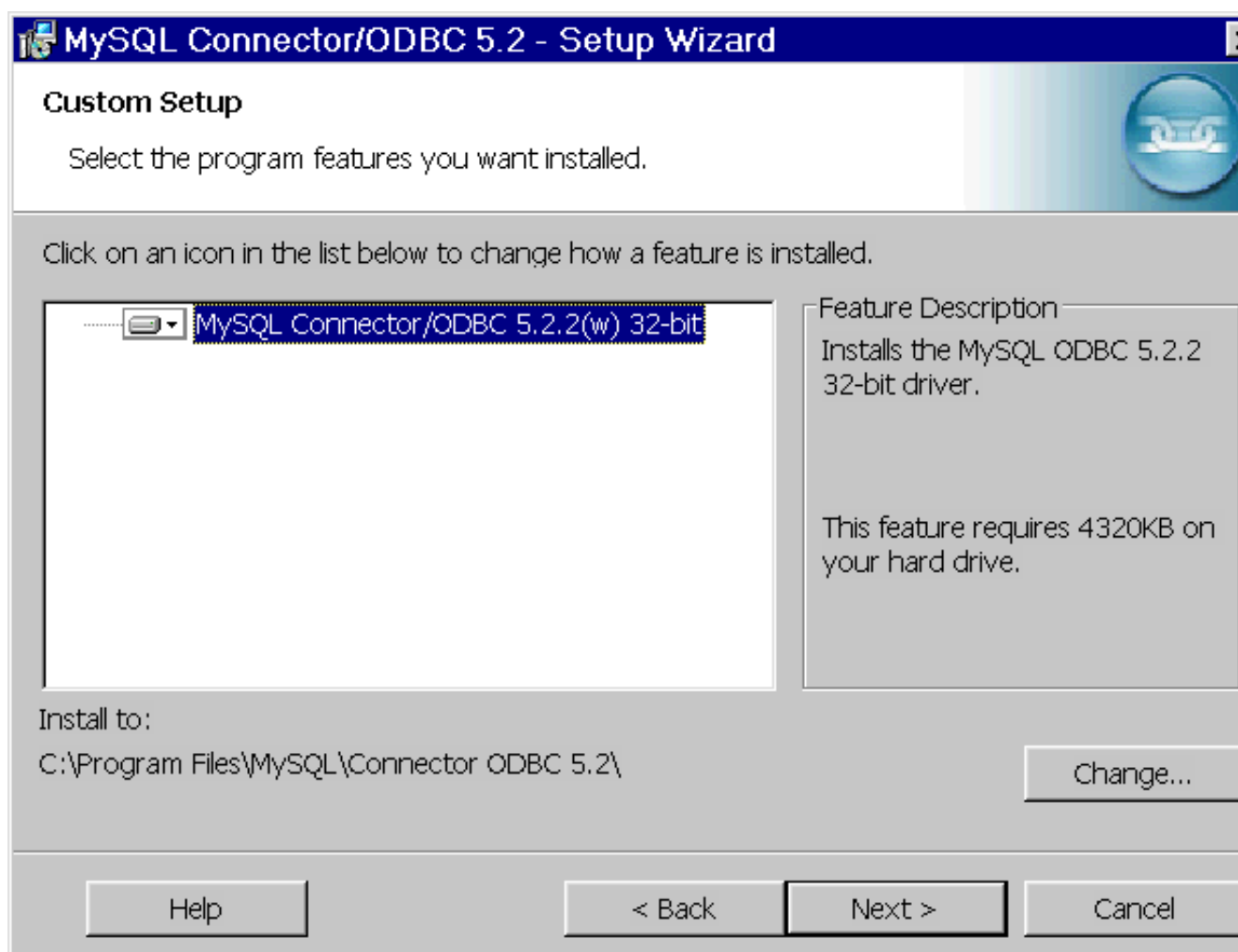


3. Choose the installation type. The **Typical** installation provides the standard files needed to connect to a MySQL database using ODBC. The **Complete** option installs all the available files, including debug and utility components. Oracle recommends choosing one of these two options to complete the installation. If you choose one of these methods, click **Next**, then proceed to step 5.

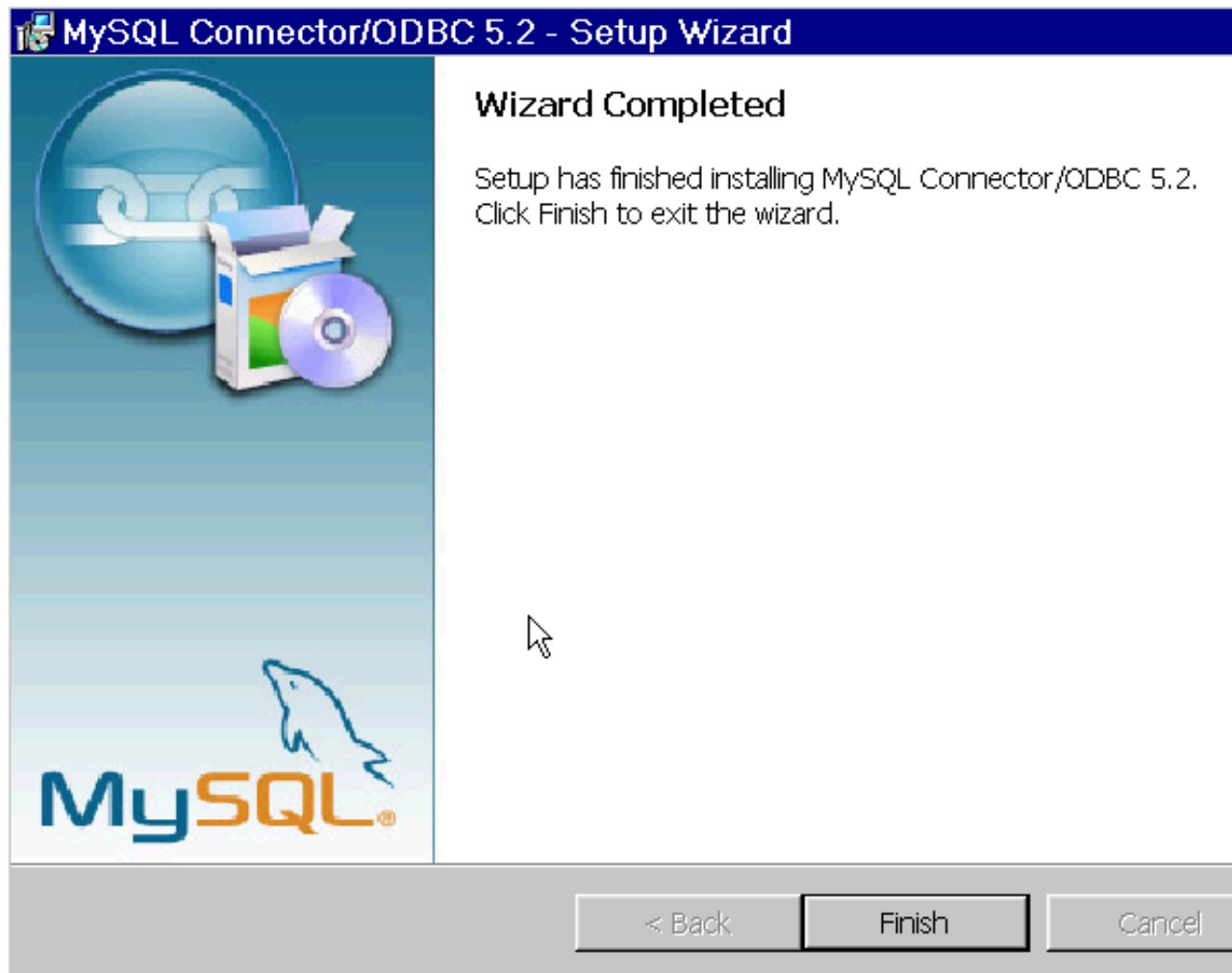
You can also choose a **Custom** installation, where you select the individual components to install. If you choose this method, click **Next**, then proceed to step 4.



4. If you have chosen a custom installation, use the pop-ups to select which components to install, then click **Next** to install the necessary files.



5. Once the files are copied to their final locations, the installation is complete. Click **Finish** to exit the installer.



Now that the installation is complete, configure your ODBC connections using [Section 3.5, “Configuring Connector/ODBC”](#).

3.4.1.2 Installing the Windows Connector/ODBC Driver Using the Zipped DLL Package

If you have downloaded the zipped DLL package:

1. Unzip the installation files.
2. Run the included batch file to perform an installation to the default locations.
3. Alternatively, install the individual files required for Connector/ODBC operation manually.

Note

The following instructions only work for 32-bit Windows systems. If you have a 64-bit Windows system, use the MSI installer, which installs both the 32-bit and 64-bit drivers to the correct locations.

To install using the **batch file**:

1. Unzip the Connector/ODBC zipped DLL package.
2. Open a command prompt.
3. Change to the directory created when you unzipped the Connector/ODBC zipped DLL package.
4. Run `Install.bat`:

```
C:\> Install.bat
```

This copies the necessary files into the default location, and then registers the Connector/ODBC driver with the Windows ODBC manager.

Note

Changing or adding a new DSN (data source name) may be accomplished using either the GUI, or from the command-line using `myodbc-installer.exe`.

Although Oracle recommends installing these files in the standard location, you can also copy the files by hand to an alternative location - for example, to run or test different versions of the Connector/ODBC driver on the same machine. To **copy the files** to a location of your choice, use the following steps:

1. Unzip the Connector/ODBC zipped DLL package.
2. Open a command prompt.
3. Change to the directory created when you unzipped the Connector/ODBC zipped DLL package.
4. Copy the library files to a suitable directory. The default location is the default Windows system directory `\Windows\System32`:

```
C:\> copy lib\myodbc5S.dll \Windows\System32
C:\> copy lib\myodbc5S.lib \Windows\System32
If installing the Unicode-enabled driver:
C:\> copy lib\myodbc5w.dll \Windows\System32
C:\> copy lib\myodbc5w.lib \Windows\System32
If installing the ANSI driver:
C:\> copy lib\myodbc5a.dll \Windows\System32
C:\> copy lib\myodbc5a.lib \Windows\System32
```

5. Copy the Connector/ODBC tools. These must be placed in a directory that is in the system `%PATH%`. The default is to install these into the Windows system directory `\Windows\System32`:

```
C:\> copy bin\myodbc-installer.exe \Windows\System32
```

6. Optionally, copy the help files. For these files to be accessible through the help system, they must be installed in the Windows system directory:

```
C:\> copy doc\*.hlp \Windows\System32
```

7. Finally, register the Connector/ODBC driver with the ODBC manager:

```
For Unicode-enabled driver:
C:\> myodbc-installer -a -d -t"MySQL ODBC 5.2 Driver;\
DRIVER=myodbc5w.dll;SETUP=myodbc5S.dll"
For ANSI driver:
C:\> myodbc-installer -a -d -t"MySQL ODBC 5.2 Driver;\
DRIVER=myodbc5a.dll;SETUP=myodbc5S.dll"
```

If you installed these files into a non-default location, change the references to the DLL files and command location in the above statement

3.4.2 Installing Connector/ODBC on Unix-like Systems

There are three methods available for installing Connector/ODBC on a Unix-like system from a binary distribution. For most Unix environments, you will use the **tarball** distribution. For Linux systems, **RPM** distributions are available, through the [MySQL Yum repository](#) (for some platforms) or direct download.

Note

Installing Connector/ODBC 5.x on Unix requires unixODBC 2.2.12 or later to be installed.

3.4.2.1 Installing Connector/ODBC Using the MySQL Yum Repository

For EL5, EL6, or EL7-based platforms and Fedora 19 or 20, you can install the Connector/ODBC using the [MySQL Yum repository](#). You must have the MySQL Yum repository on your system's repository list (see [Adding the MySQL Yum Repository](#) for details). Make sure your Yum repository setup is up-to-date by running:

```
shell> su root
shell> yum update mysql-community-release
```

You can then install Connector/ODBC by the following command:

```
shell> yum install mysql-connector-odbc
```

See [Installing Additional MySQL Products and Components with Yum](#) for more details.

3.4.2.2 Installing Connector/ODBC from a Binary Tarball Distribution

To install the driver from a tarball distribution (`.tar.gz` file), download the latest version of the driver for your operating system and follow these steps, substituting the appropriate file and directory names based on the package you download (some of the steps below might require superuser privileges):

1. Extract the archive:

```
shell> gunzip mysql-connector-odbc-5.2.2-i686-pc-linux.tar.gz
shell> tar xvf mysql-connector-odbc-5.2.2-i686-pc-linux.tar
```

2. The extra directory contains two subdirectories, `lib` and `bin`. Copy their contents to the proper locations on your system (we use `/usr/local/bin` and `/usr/local/lib` in this example; replace them with the destinations of your choice):

```
shell> cp bin/* /usr/local/bin
shell> cp lib/* /usr/local/lib
```

The last command copies both the Connector/ODBC ANSI and the Unicode drivers from `lib` into `/usr/local/lib`; if you do not need both, you can just copy the one you want. See [Choosing Unicode or ANSI Driver](#) for details.

3. Finally, register the driver version of your choice (the ANSI version, the Unicode version, or both) with your system's ODBC manager (for example, iODBC or unixodbc) using the `myodbc-installer` tool that was included in the package under the `bin` subdirectory (and is now under the `/usr/local/bin` directory, if the last step was followed); for example, this registers the Unicode driver with the ODBC manager:

```
shell> myodbc-installer -a -d -n "MySQL ODBC 5.2 Driver" -t "Driver=/usr/local/lib/libmyodbc5w.so"
```

This registers the ANSI driver:

```
shell> myodbc-installer -a -d -n "MySQL ODBC 5.2 Driver" -t "Driver=/usr/local/lib/libmyodbc5a.so"
```

4. Verify that the driver is installed and registered using the ODBC manager, or the `myodbc-installer` utility:

```
shell> myodbc-installer -d -l
```

Next, see [Section 3.5.5, “Configuring a Connector/ODBC DSN on Unix”](#) on how to configure a DSN for Connector/ODBC.

3.4.2.3 Installing Connector/ODBC from an RPM Distribution

To install or upgrade Connector/ODBC from an RPM distribution on Linux, simply download the RPM distribution of the latest version of Connector/ODBC and follow the instructions below. Use `su root` to become `root`, then install the RPM file.

If you are installing for the first time:

```
shell> su root
shell> rpm -ivh mysql-connector-odbc-5.2.2.i386.rpm
```

If the driver exists, upgrade it like this:

```
shell> su root
shell> rpm -Uvh mysql-connector-odbc-5.2.2.i386.rpm
```

If there is any dependency error for MySQL client library, `libmysqlclient`, simply ignore it by supplying the `--nodeps` option, and then make sure the MySQL client shared library is in the path or set through `LD_LIBRARY_PATH`.

This installs the driver libraries and related documents to `/usr/local/lib` and `/usr/share/doc/MyODBC`, respectively. See [Section 3.5.5, “Configuring a Connector/ODBC DSN on Unix”](#) for the post-installation configuration steps.

To **uninstall** the driver, become `root` and execute an `rpm` command:

```
shell> su root
shell> rpm -e mysql-connector-odbc
```

3.4.3 Installing Connector/ODBC on OS X

OS X is based on the FreeBSD operating system, and you can normally use the MySQL network port for connecting to MySQL servers on other hosts. Installing the Connector/ODBC driver lets you connect to MySQL databases on any platform through the ODBC interface. If your application requires an ODBC interface, install the Connector/ODBC driver. Applications that require or can use ODBC (and therefore the Connector/ODBC driver) include ColdFusion, Filemaker Pro, 4th Dimension and many other applications.

On OS X, the ODBC Administrator, based on the `iODBC` manager, provides easy administration of ODBC drivers and configuration, allowing the updates of the underlying `iODBC` configuration files through a GUI tool. The tool is included in OS X v10.5 and earlier; users of later versions of OS X need to download and install it manually.

There are two ways to install Connector/ODBC on OS X. You can use either the package provided in a compressed tar archive that you manually install, or use a compressed disk image (`.dmg`) file, which includes an installer.

To install using the compressed tar archive (some of the steps below might require superuser privileges):

1. Download the compressed tar archive.
2. Extract the archive:

```
shell> tar xvzf mysql-connector-odbc-5.x.y-osx10.z-x86-(32|64)bit.tar.gz
```

3. The directory created contains two subdirectories, `lib` and `bin`. Copy these to a suitable location such as `/usr/local`:

```
shell> cp bin/* /usr/local/bin
shell> cp lib/* /usr/local/lib
```

4. Finally, register the driver with iODBC using the `myodbc-installer` tool that was included in the package:

```
shell> myodbc-installer -a -d -n "MySQL ODBC 5.2 Driver" -t "Driver=/usr/local/lib/libmyodbc5w.so"
```

To install using the a compressed disk image (`.dmg`) file:

1. Download the disk image.
2. Double click the disk image to open it. You see the Connector/ODBC installer inside.
3. Double click the Connector/ODBC installer, and you will be guided through the rest of the installation process. You need superuser privileges to finish the installation.

To verify the installed drivers, either use the ODBC Administrator application or the `myodbc-installer` utility:

```
shell> myodbc-installer -d -l
```

3.4.4 Building Connector/ODBC from a Source Distribution on Windows

You only need to build Connector/ODBC from source on Windows to modify the source or installation location. If you are unsure whether to install from source, please use the binary installation detailed in [Section 3.4.1, "Installing Connector/ODBC on Windows"](#).

Building Connector/ODBC from source on Windows requires a number of different tools and packages:

- MDAC, Microsoft Data Access SDK from <http://support.microsoft.com/kb/110093>.
- Suitable C compiler, such as Microsoft Visual C++ or the C compiler included with Microsoft Visual Studio.

Microsoft Visual Studio 7 and 8 are preferred, and well-tested.

- Connector/ODBC 5.2: `cmake`.
- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because Connector/ODBC uses calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit <http://dev.mysql.com/downloads/>.

Build Steps

Set the environment variables for the Visual Studio toolchain. Visual Studio includes a batch file to set these for you, and installs a **Start** menu shortcut that opens a command prompt with these variables set.

Set `MYSQL_DIR` to the MySQL server installation path, while using the short-style file names. For example:

```
C:\> set MYSQL_DIR=C:\PROGRA~1\MySQL\MYSQLS~1.0
```

Build Connector/ODBC using the `cmake` command-line tool by executing the following from the source root directory (in a command prompt window):

```
C:\> cmake -G "Visual Studio 8 2005"
```

This produces a project file that you can open with Visual Studio, or build from the command line with either of the following commands:

```
C:\> devenv.com MySQL_Connector_ODBC.sln /build Release
C:\> devenv.com MySQL_Connector_ODBC.sln /build RelWithDebInfo
```

By default, Connector/ODBC is linked statically with the MySQL client library `mysqlclient.lib`. If you want to link dynamically or to another MySQL client library, use the `MYSQLCLIENT_LIB_NAME` option to supply the client library's name:

```
C:\> cmake -G "Visual Studio 8 2005" -DMYSQLCLIENT_LIB_NAME=client_lib_name_with_extension
```

To compile a debug build, set the `cmake` build type so that the correct versions of the MySQL client libraries are used:

```
C:\> cmake -G "Visual Studio 8 2005" -DWITH_DEBUG=1
C:\> devenv.com MySQL_Connector_ODBC.sln /build Debug
```

Upon completion, the executables are in the `bin/` and `lib/` subdirectories.

See [Section 3.4.1.2, "Installing the Windows Connector/ODBC Driver Using the Zipped DLL Package"](#) for the `copy` commands to complete the installation.

3.4.5 Building Connector/ODBC from a Source Distribution on Unix

You need the following tools to build MySQL from source on Unix:

- A working ANSI C++ compiler. GCC 4.2.1 or later, Sun Studio 10 or later, Visual Studio 2008 or later, and many current vendor-supplied compilers are known to work.
- `cmake`.
- MySQL client libraries and include files from MySQL 4.0.0 or higher (preferably 4.0.16 or higher). This is required because Connector/ODBC uses calls and structures that exist only starting from that version of the library. To get the client libraries and include files, visit <http://dev.mysql.com/downloads/>.
- A compatible ODBC manager must be installed. Connector/ODBC is known to work with the `iODBC` and `unixODBC` managers. See [Section 3.3.2, "ODBC Driver Managers"](#) for more information.
- If you are using a character set that is not compiled into the MySQL client library, install the MySQL character definitions from the `charsets` directory into `SHAREDIR` (by default, `/usr/local/mysql/share/mysql/charsets`). These should be in place if you have installed the MySQL server on the same machine. See [Character Set Support](#) for more information on character set support.

Once you have all the required files, unpack the source files to a separate directory, then run `cmake` with the following command:

```
shell> cmake -G "Unix Makefiles"
```

Typical `cmake` Parameters and Options

You might need to help `cmake` find the MySQL headers and libraries by setting the environment variables `MYSQL_INCLUDE_DIR`, `MYSQL_LIB_DIR`, and `MYSQL_DIR` to the appropriate locations; for example:

```
shell> export MYSQL_INCLUDE_DIR=/usr/local/mysql/include
shell> export MYSQL_LIB_DIR=/usr/local/mysql/lib
shell> export MYSQL_DIR=/usr/local/mysql
```

When you run `cmake`, you might add options to the command line. Here are some examples:

- `-DODBC_INCLUDES=dir_name`: Use when the ODBC include directory is not found within the system `$PATH`.
- `-DODBC_LIB_DIR=dir_name`: Use when the ODBC library directory is not found within the system `$PATH`.
- `-DWITH_UNIXODBC=1`: Enables unixODBC support. `iODBC` is the default ODBC library used by Connector/ODBC. Alternatively, `unixODBC` may be used by setting this option to "1".
- `-DMYSQLCLIENT_LIB_NAME=client_lib_name_with_extension`: By default, Connector/ODBC is linked statically with the MySQL client library `libmysqlclient.a`. If you want to link dynamically or to another MySQL client library, use this option to supply the client library's name.
- `-DMYSQL_CONFIG_EXECUTABLE=/path/to/mysql_config`: Specifies location of the utility `mysql_config`, which is used to fetch values of the variables `MYSQL_INCLUDE_DIR`, `MYSQL_LIB_DIR`, `MYSQL_LINK_FLAGS`, and `MYSQL_CXXFLAGS`. Values fetched by `mysql_config` can be overridden by values provided directly to `cmake` as parameters.
- `-DMYSQL_LINK_FLAGS=MySQL link flags`
- `-DMYSQL_CXXFLAGS=MySQL C++ linkage flags`
- `-DMYSQL_CXX_LINKAGE=1`: Enables C++ linkage to MySQL client library. By default, `MYSQL_CXX_LINKAGE` is enabled for MySQL 5.6.4 or later. For MySQL 5.6.3 and earlier, this option must be set explicitly to `1`.

Build Steps for Unix

To build the driver libraries, execute `make`:

```
shell> make
```

If any errors occur, correct them and continue with the build process. If you are not able to finish the build, see [Section 3.9.1, "Connector/ODBC Community Support"](#).

Installing Driver Libraries

To install the driver libraries, execute the following command:

```
shell> make install
```

For more information on build process, refer to the `BUILD` file that comes with the source distribution.

Testing Connector/ODBC on Unix

Some tests for Connector/ODBC are provided in the distribution with the libraries that you built. To run the tests:

1. Make sure you have an `odbc.ini` file in place, by which you can configure your DSN entries. A sample `odbc.ini` file is generated by the build process under the `test` folder. Set the environment variable `ODBCINI` to the location of your `odbc.ini` file.
2. Set up a test DSN in your `odbc.ini` file (see [Section 3.5.5, "Configuring a Connector/ODBC DSN on Unix"](#) for details). A sample DSN entry, which you can use for your tests, can be found in the sample `odbc.ini` file.

3. Set the environment variable `TEST_DSN` to the name of your test DSN.
4. Set the environment variable `TEST_UID` and perhaps also `TEST_PASSWORD` to the user name and password for the tests, if needed. By default, the tests use “root” as the user and do not enter a password; if you want the tests to use another user name or password, set `TEST_UID` and `TEST_PASSWORD` accordingly.
5. Make sure that your MySQL server is running.
6. Run the following command:

```
shell> make test
```

3.4.6 Building Connector/ODBC from a Source Distribution on OS X

To build the driver on OS X (Darwin), make use of the following `configure` example:

```
shell> ./configure --prefix=/usr/local
--with-unixODBC=/usr/local
--with-mysql-path=/usr/local/mysql
--disable-shared
--enable-gui=no
--host=powerpc-apple
```

The command assumes that the `unixODBC` and MySQL are installed in the default locations. If not, configure accordingly.

On OS X, `--enable-shared` builds `.dylib` files by default. You can build `.so` files like this:

```
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error
-o .libs/libmyodbc3-3.51.01.so *.o
-L/usr/local/mysql/lib/
-L/usr/local/iodbc/lib
-liodbcinst -lmysqlclient -lz -lc
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
$CC -bundle -flat_namespace -undefined error
-o .libs/libmyodbc3-3.51.01.so *.o
-L/usr/local/mysql/lib/
-L/usr/local/iodbc/lib
-liodbcinst -lmysqlclient_r -lz -lc -lpthread
```

Make sure to change the `-liodbcinst` to `-lodbcinst` in case of using `unixODBC` instead of `iodbc` and configure the libraries path accordingly.

By default, Connector/ODBC is linked statically with the MySQL client library `libmysqlclient.a`. If you want to link dynamically or to another MySQL client library, use the `mysqlclient_lib_name` option to supply the client library's name.

In Apple's version of GCC, both `cc` and `gcc` are actually symbolic links to `gcc3`.

Copy this library to the `$prefix/lib` directory and symlink to `libmyodbc3.so`.

You can cross-check the output shared-library properties using this command:

```
shell> otool -LD .libs/libmyodbc3-3.51.01.so
```

3.4.7 Installing Connector/ODBC from the Development Source Tree

Caution

This section is only for users who are interested in helping us test our new code. To just get MySQL Connector/ODBC up and running on your system, use a standard release distribution.

To obtain the most recent development source tree, first download and install Bazaar from the [Bazaar VCS Web site](#). Bazaar is supported by any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows, or OS X host. Instructions for downloading and installing Bazaar on the different platforms are available on the Bazaar Web site.

The most recent development source tree is available on LaunchPad at <https://code.launchpad.net/myodbc>.

To check out the Connector/ODBC sources, change to the directory where you want the copy of the Connector/ODBC tree to be stored, then use the following command:

```
shell> bzip branch lp:myodbc/5.2
```

You should now have a copy of the entire Connector/ODBC source tree in the directory `connector-odbc5`. To build and then install the driver libraries from this source tree on Unix or Linux, use the same steps outlined in [Section 3.4.5, “Building Connector/ODBC from a Source Distribution on Unix”](#).

If you have gotten to the `make` stage and the distribution does not compile, please report it to <myodbc@lists.mysql.com>.

On Windows, make use of Windows Makefiles `WIN-Makefile` and `WIN-Makefile_debug` in building the driver. For more information, see [Section 3.4.4, “Building Connector/ODBC from a Source Distribution on Windows”](#).

After the initial checkout operation to get the source tree, run `bzip pull` periodically to update your source according to the latest version.

3.5 Configuring Connector/ODBC

Before you connect to a MySQL database using the Connector/ODBC driver, you configure an ODBC Data Source Name (DSN). The DSN associates the various configuration parameters required to communicate with a database to a specific name. You use the DSN in an application to communicate with the database, rather than specifying individual parameters within the application itself. DSN information can be user-specific, system-specific, or provided in a special file. ODBC data source names are configured in different ways, depending on your platform and ODBC driver.

3.5.1 Overview of Connector/ODBC Data Source Names

A Data Source Name associates the configuration parameters for communicating with a specific database. Generally, a DSN consists of the following parameters:

- Name
- Host Name
- Database Name
- Login
- Password

In addition, different ODBC drivers, including Connector/ODBC, may accept additional driver-specific options and parameters.

There are three types of DSN:

- A *System DSN* is a global DSN definition that is available to any user and application on a particular system. A System DSN can normally only be configured by a systems administrator, or by a user who has specific permissions that let them create System DSNs.
- A *User DSN* is specific to an individual user, and can be used to store database connectivity information that the user regularly uses.
- A *File DSN* uses a simple file to define the DSN configuration. File DSNs can be shared between users and machines and are therefore more practical when installing or deploying DSN information as part of an application across many machines.

DSN information is stored in different locations depending on your platform and environment.

3.5.2 Connector/ODBC Connection Parameters

You can specify the parameters in the following tables for Connector/ODBC when configuring a DSN:

- [Table 3.1, “Connector/ODBC DSN Configuration Options”](#)
- [Table 3.2, “Connector/ODBC Option Parameters”](#)

Users on Windows can use the Options and Advanced panels when configuring a DSN to set these parameters; see [Table 3.1, “Connector/ODBC DSN Configuration Options”](#) for information on which options are related to which fields and check boxes. On Unix and OS X, use the parameter name and value as the keyword/value pair in the DSN configuration. Alternatively, you can set these parameters within the `InConnectionString` argument in the `SQLDriverConnect()` call.

Table 3.1 Connector/ODBC DSN Configuration Options

Parameter	Default Value	Comment
<code>user</code>	ODBC	The user name used to connect to MySQL.
<code>uid</code>	ODBC	Synonymous with <code>user</code> . Added in 3.51.16.
<code>server</code>	<code>localhost</code>	The host name of the MySQL server.
<code>database</code>		The default database.
<code>option</code>	0	Options that specify how Connector/ODBC works. See Table 3.2, “Connector/ODBC Option Parameters” and Table 3.4, “Recommended Connector/ODBC Option Values for Different Configurations” .
<code>port</code>	3306	The TCP/IP port to use if <code>server</code> is not <code>localhost</code> .
<code>initstmt</code>		Initial statement. A statement to execute when connecting to MySQL. In version 3.51 the parameter is called <code>stmt</code> . The driver supports the initial statement being executed only at the time of the initial connection.
<code>password</code>		The password for the <code>user</code> account on <code>server</code> .
<code>pwd</code>		Synonymous with <code>password</code> . Added in 3.51.16.
<code>socket</code>		The Unix socket file or Windows named pipe to connect to if <code>server</code> is <code>localhost</code> .
<code>sslca</code>		The path to a file with a list of trust SSL CAs. Added in 3.51.16.
<code>sslcapath</code>		The path to a directory that contains trusted SSL CA certificates in PEM format. Added in 3.51.16.
<code>sslcert</code>		The name of the SSL certificate file to use for establishing a secure connection. Added in 3.51.16.

Parameter	Default Value	Comment
sslcipher		A list of permissible ciphers to use for SSL encryption. The cipher list has the same format as the openssl ciphers command. Added in 3.51.16.
sslkey		The name of the SSL key file to use for establishing a secure connection. Added in 3.51.16.
rsakey		The full-path name of the PEM file that contains the RSA public key for using the SHA256 authentication plugin of MySQL. Added in 5.3.4.
sslverify		If set to 1, the SSL certificate will be verified when used with the MySQL connection. If not set, then the default behavior is to ignore SSL certificate verification.
charset		The character set to use for the connection. Added in 3.51.17.
readtimeout		The timeout in seconds for attempts to read from the server. Each attempt uses this timeout value and there are retries if necessary, so the total effective timeout value is three times the option value. You can set the value so that a lost connection can be detected earlier than the TCP/IP Close_Wait_Timeout value of 10 minutes. This option works only for TCP/IP connections, and only for Windows prior to MySQL 5.1.12. Corresponds to the MYSQL_OPT_READ_TIMEOUT option of the MySQL Client Library. Added in 3.51.27.
writetimeout		The timeout in seconds for attempts to write to the server. Each attempt uses this timeout value and there are net_retry_count retries if necessary, so the total effective timeout value is net_retry_count times the option value. This option works only for TCP/IP connections, and only for Windows prior to MySQL 5.1.12. Corresponds to the MYSQL_OPT_WRITE_TIMEOUT option of the MySQL Client Library. Added in 3.51.27.
interactive		If set to 1, the CLIENT_INTERACTIVE connection option of mysql_real_connect is enabled.
prefetch	0	When set to a non-zero value <i>N</i>, causes all queries in the connection to return <i>N</i> rows at a time rather than the entire result set. Useful for queries against very large tables where it is not practical to retrieve the whole result set at once. You can scroll through the result set, <i>N</i> records at a time. This option works only with forward-only cursors. It does not work when the option parameter MULTI_STATEMENTS is set. It can be used in combination with the option parameter NO_CACHE. Its behavior in ADO applications is undefined: the prefetching might or might not occur.
no_ssps	0	In Connector/ODBC 5.2, by default, server-side prepared statements are used. When this option is set to a non-zero value, prepared statements are emulated on the client side, which is the same behavior as in 5.1 and 3.51. Added in 5.2.
can_handle_expired_pwd		Indicates that the application can deal with an expired password, which is signalled by an SQL state of 08004 ("Server rejected the connection") and a native error code ER_MUST_CHANGE_PASSWORD_LOGIN (1862). The connection is "sandboxed", and can do nothing other than issue a SET PASSWORD statement. To establish a connection in this case, your application must either use the initstmt connection option

Parameter	Default Value	Comment
		to set a new password at the start, or issue a SET PASSWORD statement immediately after connecting. Once the expired password is reset, the restrictions on the connection are lifted. See ALTER USER Syntax for details about password expiration for MySQL server accounts. Added in 5.2.4.

Note

The SSL configuration parameters can also be automatically loaded from a [my.ini](#) or [my.cnf](#) file. See [Using Option Files](#).

The behavior of Connector/ODBC can be modified by using special option parameters listed in [Table 3.2, “Connector/ODBC Option Parameters”](#), specified in the connection string or through the GUI dialog box. Most of the connection parameters also have their own numeric constant values, which can be added up as a combined value for the [option](#) parameter for specifying those options. However, the numerical [option](#) value in the connection string can only enable, but not disable parameters enabled on the DSN, which can only be overridden by specifying the option parameters using their text names in the connection string.

Note

While the combined numerical value for the [option](#) parameter can be easily constructed by addition of the options' constant values, decomposing the value to verify if particular options are enabled can be difficult. We recommend using the options' parameter names instead in the connection string, because they are self-explanatory. Also notice that not every option parameter has a constant value.

Table 3.2 Connector/ODBC Option Parameters

Parameter Name	GUI Option	Constant Value	Description
FOUND_ROWS	Return matched rows instead of affected rows	2	The client cannot handle when MySQL returns the true value of affected rows. If this flag is set, MySQL returns “found rows” instead. You must have MySQL 3.21.14 or newer for this to work.
BIG_PACKETS	Allow big result set	8	Do not set any packet limit for results and bind parameters. Without this option, parameter binding will be truncated to 255 characters.
NO_PROMPT	Don't prompt when connecting	16	Do not prompt for questions even if driver would like to prompt.
DYNAMIC_CURSOR	Enable Dynamic Cursors	32	Enable or disable the dynamic cursor support.
NO_SCHEMA	Ignore schema in column specifications	64	Ignore use of database name in db_name.tbl_name.col_name .
NO_DEFAULT_CURSOR	Disable driver-provided cursor support	128	Force use of ODBC manager cursors (experimental).
NO_LOCALE	Don't use setlocale()	256	Disable the use of extended fetch (experimental).

Parameter Name	GUI Option	Constant Value	Description
PAD_SPACE	Pad CHAR to full length with space	512	Pad CHAR columns to full column length.
FULL_COLUMN_NAMES	Include table name in SQLDescribeCol()	1024	SQLDescribeCol() returns fully-qualified column names.
COMPRESSED_PROTO	Use compression	2048	Use the compressed client/server protocol.
IGNORE_SPACE	Ignore space after function names	4096	Tell server to ignore space after function name and before "(" (needed by PowerBuilder). This makes all function names keywords.
NAMED_PIPE	Named Pipe	8192	Connect with named pipes to a mysqld server running on NT.
NO_BIGINT	Treat BIGINT columns as INT columns	16384	Change BIGINT columns to INT columns (some applications cannot handle BIGINT).
NO_CATALOG	Disable catalog support	32768	Forces results from the catalog functions, such as SQLTables , to always return NULL and the driver to report that catalogs are not supported.
USE_MYCNF	Read options from my.cnf	65536	Read parameters from the [client] and [odbc] groups from my.cnf .
SAFE	Enable safe options	131072	Add some extra safety checks.
NO_TRANSACTIONS	Disable transaction support	262144	Disable transactions.
LOG_QUERY	Log queries to %TEMP%\myodbc.sql	524288	Enable query logging to c:\myodbc.sql(/tmp/myodbc.sql) file. (Enabled only in debug mode.)
NO_CACHE	Don't cache results of forward-only cursors	1048576	Do not cache the results locally in the driver, instead read from server (mysql_use_result()). This works only for forward-only cursors. This option is very important in dealing with large tables when you do not want the driver to cache the entire result set.
FORWARD_CURSOR	Force use of forward-only cursors	2097152	Force the use of Forward-only cursor type. In cases of applications setting the default static/dynamic cursor type and one wants the driver to use noncache result sets, this option ensures the forward-only cursor behavior.

Parameter Name	GUI Option	Constant Value	Description
AUTO_RECONNECT	Enable automatic reconnect	4194304	Enables auto-reconnection functionality. Do not use this option with transactions , since an auto-reconnection during a incomplete transaction may cause corruption. An auto-reconnected connection will not inherit the same settings and environment as the original connection. Added in 3.51.13.
AUTO_IS_NULL	Enable SQL_AUTO_IS_NULL	8388608	When AUTO_IS_NULL is set, the driver does not change the default value of sql_auto_is_null, leaving it at 1, so you get the MySQL default, not the SQL standard behavior. When AUTO_IS_NULL is not set, the driver changes the default value of SQL_AUTO_IS_NULL to 0 after connecting, so you get the SQL standard, not the MySQL default behavior. Thus, omitting the flag disables the compatibility option and forces SQL standard behavior. See IS NULL. Added in 3.51.13.
ZERO_DATE_TO_MIN	Return SQL_NULL_DATA for zero date	16777216	Translates zero dates (XXXX-00-00) into the minimum date values supported by ODBC, XXXX-01-01 . This resolves an issue where some statements will not work because the date returned and the minimum ODBC date value are incompatible. Added in 3.51.17.
MIN_DATE_TO_ZERO	Bind minimal date as zero date	33554432	Translates the minimum ODBC date value (XXXX-01-01) to the zero date format supported by MySQL (XXXX-00-00). This resolves an issue where some statements will not work because the date returned and the minimum ODBC date value are incompatible. Added in 3.51.17.
MULTI_STATEMENTS	Allow multiple statements	67108864	Enables support for batched statements. Added in 3.51.18.
COLUMN_SIZE_S32	Limit column size to signed 32-bit range	134217728	Limits the column size to a signed 32-bit value to prevent problems with larger column

Parameter Name	GUI Option	Constant Value	Description
			sizes in applications that do not support them. This option is automatically enabled when working with ADO applications. Added in 3.51.22.
NO_BINARY_RESULT	Always handle binary function results as character data	268435456	When set, this option disables charset 63 for columns with an empty org_table . Added in 3.51.26.
DFLT_BIGINT_BIND_STR	[This option is not on the GUI dialog box]	536870912	Causes BIGINT parameters to be bound as strings. Microsoft Access treats BIGINT as a string on linked tables. The value is read correctly, but bound as a string. This option is used automatically if the driver is used by Microsoft Access. Added in 5.1.3.
NO_INFORMATION_SCHEMA	Don't use INFORMATION_SCHEMA for metadata	1073741824	Tells catalog functions not to use INFORMATION_SCHEMA , but rather use legacy algorithms. The trade-off here is usually speed for information quality. Using INFORMATION_SCHEMA is often slow, but the information obtained is more complete. Added in 5.1.7.
INTERACTIVE	Interactive Client	-	Makes the client interactive and uses interactive_timeout instead of wait_timeout . Added in 5.1.7.
CAN_HANDLE_EXP_PWD	Can Handle Expired Password	-	Enables handling of expired password. Added in 5.2.4.
ENABLE_CLEARTEXT_PLUGIN	Enable Cleartext Authentication	-	Enables cleartext authentication. Added in 5.1.13 and 5.2.5.
NO_SSPS	Prepare statements on the client	-	Prepares statements on the client instead of the server. Added in 5.2.0.
PREFETCH	Prefetch from server by <i>N</i> rows at a time	-	Specifies the number of rows (<i>N</i>) to prefetch from the server for queries without LIMIT . It turns, for example, SELECT * FROM table into SELECT * FROM table LIMIT 0, N . If the client wants to read more rows than specified in the PREFETCH parameter, the driver runs another query SELECT * FROM table LIMIT N+1, N*2 . Added in 5.1.11.
SSL_ENFORCE	Enforce SSL	-	Enforce the requirement to use SSL for connections to server.

Parameter Name	GUI Option	Constant Value	Description
			See Table 3.3, “Combined Effects of SSL_ENFORCE and DISABLE_SSL_DEFAULT” . Added in 5.3.6.
DISABLE_SSL_DEFAULT	Disable default SSL	-	Disable the default requirement to use SSL for connections to server. When set to “0” [default], Connector/ODBC tries to connect with SSL first, and falls back to unencrypted connection if it is not possible to establish an SSL connection. When set to “1,” Connection with SSL is not attempted, and unencrypted connection is used, unless SSL_ENFORCE is also set to “1.” See Table 3.3, “Combined Effects of SSL_ENFORCE and DISABLE_SSL_DEFAULT” . Added in 5.3.6.

Table 3.3 Combined Effects of [SSL_ENFORCE](#) and [DISABLE_SSL_DEFAULT](#)

	DISABLE_SSL_DEFAULT = 0	DISABLE_SSL_DEFAULT = 1
SSL_ENFORCE = 0	(Default) Connection with SSL is attempted first; if not possible, fall back to unencrypted connection.	Connection with SSL is not attempted; use unencrypted connection.
SSL_ENFORCE = 1	Connect with SSL; throw an error if an SSL connection cannot be established.	Connect with SSL; throw an error if an SSL connection cannot be established. DISABLE_SSL_DEFAULT=1 is overridden.

[Table 3.4, “Recommended Connector/ODBC Option Values for Different Configurations”](#) shows some recommended parameter settings and their corresponding [option](#) values for various configurations:

Table 3.4 Recommended Connector/ODBC Option Values for Different Configurations

Configuration	Parameter Settings	Option Value
Microsoft Access, Visual Basic	FOUND_ROWS=1;	2
Microsoft Access (with improved DELETE queries)	FOUND_ROWS=1;DYNAMIC_CURSOR=1;	34
Microsoft SQL Server	COLUMN_SIZE_S32=1;	134217728
Large tables with too many rows	COMPRESSED_PROTO=1;	2048
Sybase PowerBuilder	IGNORE_SPACE=1;FLAG_SAFE=1;	135168
Query log generation (Debug mode)	LOG_QUERY=1;	524288
Large tables with no-cache results	NO_CACHE=1;FORWARD_CURSOR=1;	3145728
Applications that run full-table "SELECT * FROM ... " query, but read only a small number (<i>N</i>) of rows from the result	PREFETCH= <i>N</i>	Not Applicable

3.5.3 Configuring a Connector/ODBC DSN on Windows

The [ODBC Data Source Administrator](#) within Windows lets you create DSNs, check driver installation and configure ODBC systems such as tracing (used for debugging) and connection pooling.

Different editions and versions of Windows store the [ODBC Data Source Administrator](#) in different locations depending on the version of Windows that you are using.

To open the [ODBC Data Source Administrator](#) in Windows Server 2003:

Tip

To identify whether a DSN was created using the 32-bit or the 64-bit driver, include the driver being used within the DSN identifier. This will help you to identify the right DSN to use with applications such as Excel that are only compatible with the 32-bit driver. For example, you might add [Using32bitCOBDC](#) to the DSN identifier for the 32-bit interface and [Using64bitCOBDC](#) for those using the 64-bit Connector/ODBC driver.

1. On the [Start](#) menu, choose [Administrative Tools](#), and then click [Data Sources \(ODBC\)](#).

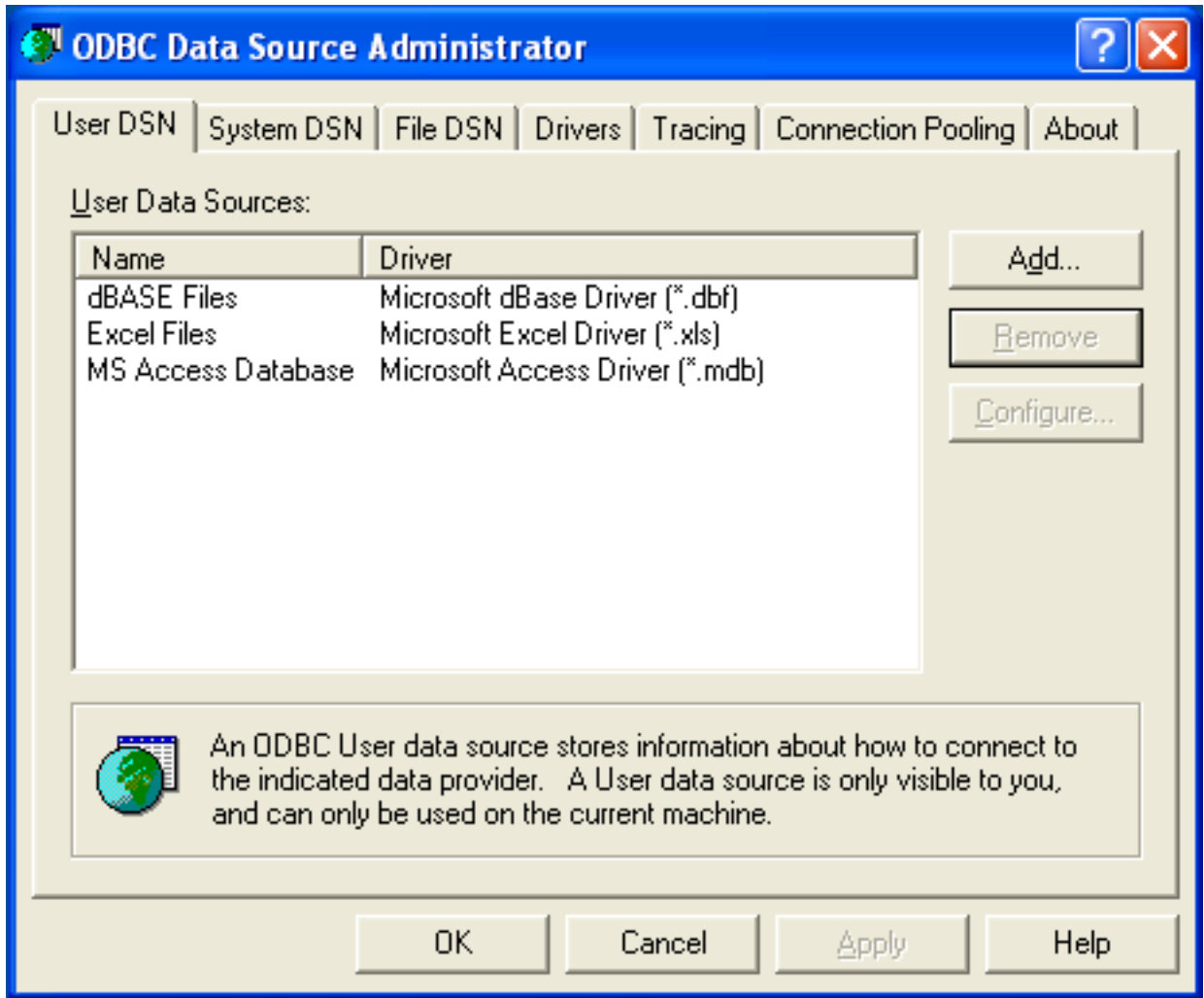
To open the [ODBC Data Source Administrator](#) in Windows 2000 Server or Windows 2000 Professional:

1. On the [Start](#) menu, choose [Settings](#), and then click [Control Panel](#).
2. In [Control Panel](#), click [Administrative Tools](#).
3. In [Administrative Tools](#), click [Data Sources \(ODBC\)](#).

To open the [ODBC Data Source Administrator](#) on Windows XP:

1. On the [Start](#) menu, click [Control Panel](#).
2. In the [Control Panel](#) when in [Category View](#) click [Performance and Maintenance](#) and then click [Administrative Tools](#).. If you are viewing the [Control Panel](#) in [Classic View](#), click [Administrative Tools](#).
3. In [Administrative Tools](#), click [Data Sources \(ODBC\)](#).

Irrespective of your Windows version, you should be presented the [ODBC Data Source Administrator](#) window:



Within Windows XP, you can add the [Administrative Tools](#) folder to your **Start** menu to make it easier to locate the ODBC Data Source Administrator. To do this:

1. Right-click the **Start** menu.
2. Select [Properties](#).
3. Click **Customize...**
4. Select the **Advanced** tab.
5. Within [Start menu items](#), within the [System Administrative Tools](#) section, select [Display on the All Programs menu](#).

Within both Windows Server 2003 and Windows XP, consider permanently adding the [ODBC Data Source Administrator](#) to your **Start** menu. To do this, locate the [Data Sources \(ODBC\)](#) icon using the methods shown, then right-click on the icon and then choose **Pin to Start Menu**.

The interfaces for the 3.51 and 5.x versions of the Connector/ODBC driver are different, although the fields and information that you need to enter remain the same.

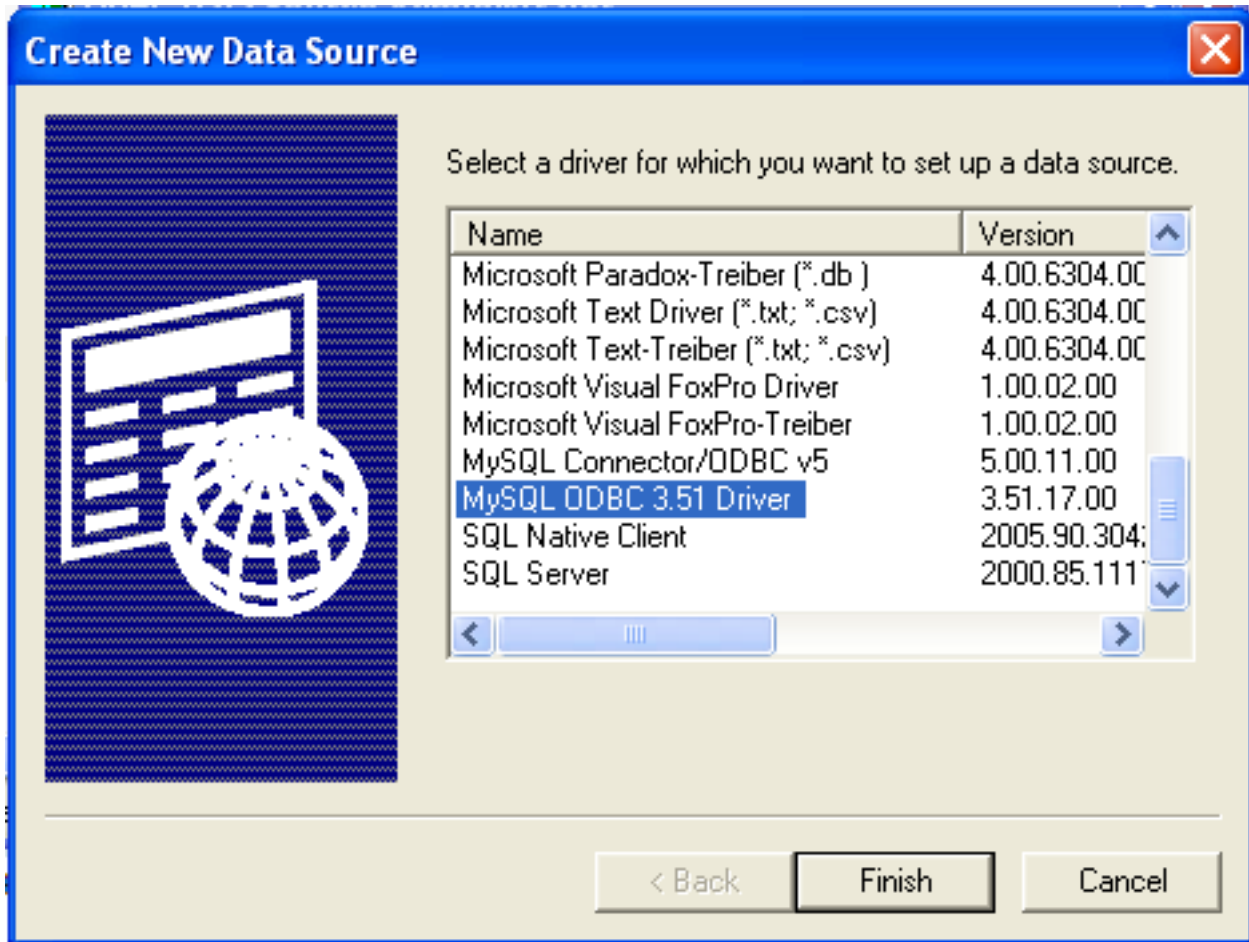
To configure a DSN using Connector/ODBC 5.2, see [Section 3.5.3.1, "Configuring a Connector/ODBC 5.x DSN on Windows"](#).

3.5.3.1 Configuring a Connector/ODBC 5.x DSN on Windows

Due to the native Unicode support within Connector/ODBC, you do not need to specify the initial character set to be used with your connection.

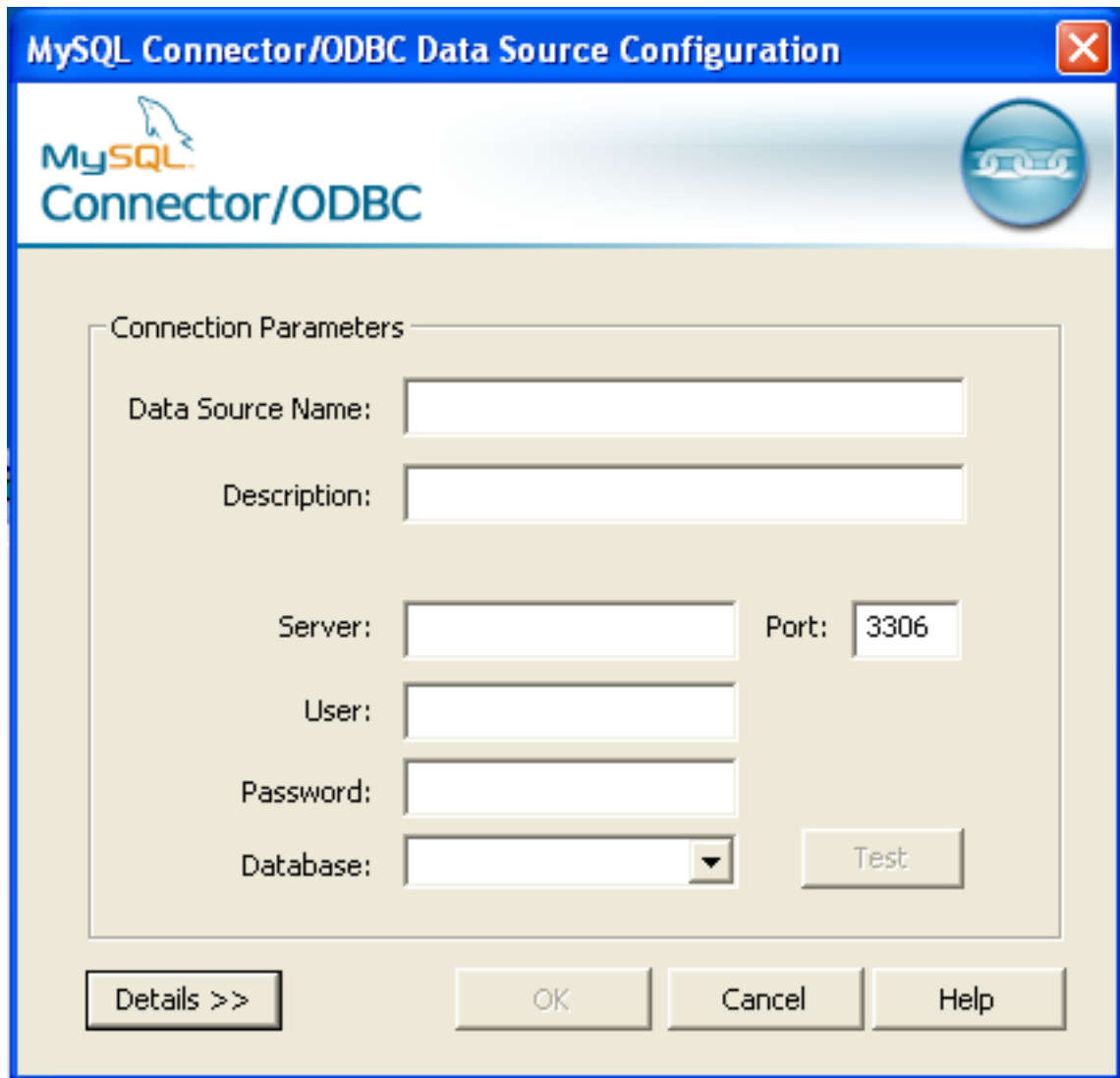
To add or configure a Connector/ODBC 5.x DSN on Windows, use either the command-line, or the [ODBC Data Source Administrator](#) GUI.

1. Open the [ODBC Data Source Administrator](#).
2. To create a System DSN (which will be available to all users), select the **System DSN** tab. To create a User DSN, which will be unique only to the current user, click the **Add...** button.
3. Select the ODBC driver for this DSN.



Select [MySQL ODBC 5.x Driver](#) for the appropriate level of Connector/ODBC, then click **Finish**.

4. You now need to configure the specific fields for the DSN you are creating through the [Connection Parameters](#) dialog.

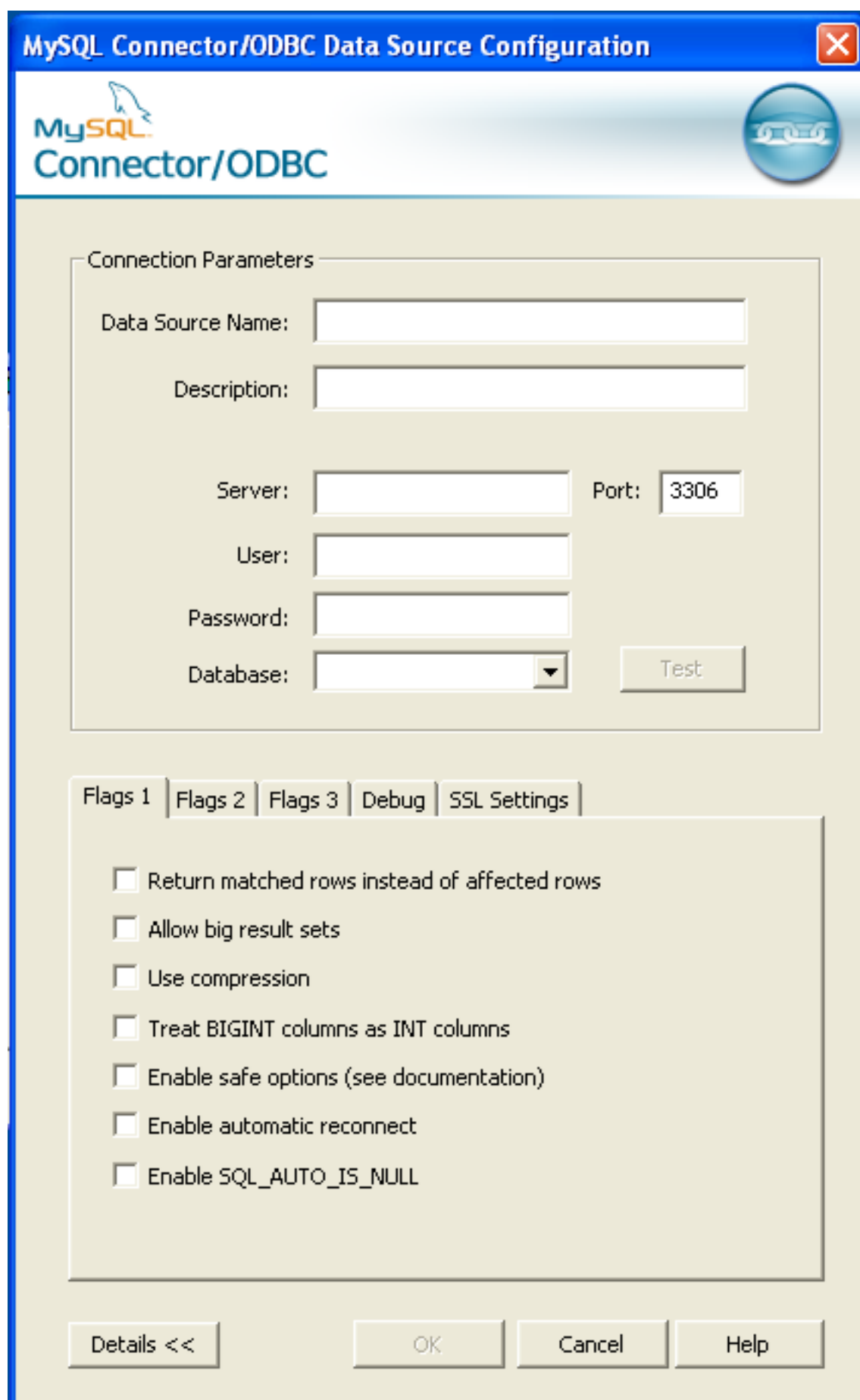
The image shows a Windows dialog box titled "MySQL Connector/ODBC Data Source Configuration". It features the MySQL logo and a blue circular icon with a bridge-like structure. The dialog is divided into a "Connection Parameters" section and a bottom section with buttons. The "Connection Parameters" section contains fields for "Data Source Name", "Description", "Server", "Port" (set to 3306), "User", "Password", and "Database" (a dropdown menu). A "Test" button is located to the right of the "Database" field. The bottom section contains a "Details >>" button, an "OK" button, a "Cancel" button, and a "Help" button.

In the **Data Source Name** box, enter the name of the data source to access. It can be any valid name that you choose.

5. In the **Description** box, enter some text to help identify the connection.
6. In the **Server** field, enter the name of the MySQL server host to access. By default, it is `localhost`.
7. In the **User** field, enter the user name to use for this connection.
8. In the **Password** field, enter the corresponding password for this connection.
9. The **Database** pop-up should automatically populate with the list of databases that the user has permissions to access.
10. To communicate over a different TCP/IP port than the default (3306), change the value of the **Port**.
11. Click **OK** to save the DSN.

To verify the connection using the parameters you have entered, click the **Test** button. If the connection could be made successfully, you will be notified with a `Success; connection was made!` dialog.

You can configure a number of options for a specific DSN by using the **Details** button.



The image shows the 'MySQL Connector/ODBC Data Source Configuration' dialog box. It has a blue title bar with the text 'MySQL Connector/ODBC Data Source Configuration' and a close button. Below the title bar is a header area with the MySQL logo and the text 'Connector/ODBC'. The main area is divided into two sections. The top section, titled 'Connection Parameters', contains several input fields: 'Data Source Name', 'Description', 'Server', 'Port' (with '3306' entered), 'User', 'Password', and 'Database' (a dropdown menu). A 'Test' button is located to the right of the 'Database' field. The bottom section, titled 'Flags', contains a list of checkboxes: 'Return matched rows instead of affected rows', 'Allow big result sets', 'Use compression', 'Treat BIGINT columns as INT columns', 'Enable safe options (see documentation)', 'Enable automatic reconnect', and 'Enable SQL_AUTO_IS_NULL'. At the bottom of the dialog are four buttons: 'Details <<', 'OK', 'Cancel', and 'Help'.

MySQL Connector/ODBC Data Source Configuration

MySQL Connector/ODBC

Connection Parameters

Data Source Name:

Description:

Server: Port:

User:

Password:

Database:

Flags 1 | Flags 2 | Flags 3 | Debug | SSL Settings

☐ Return matched rows instead of affected rows

☐ Allow big result sets

☐ Use compression

☐ Treat BIGINT columns as INT columns

☐ Enable safe options (see documentation)

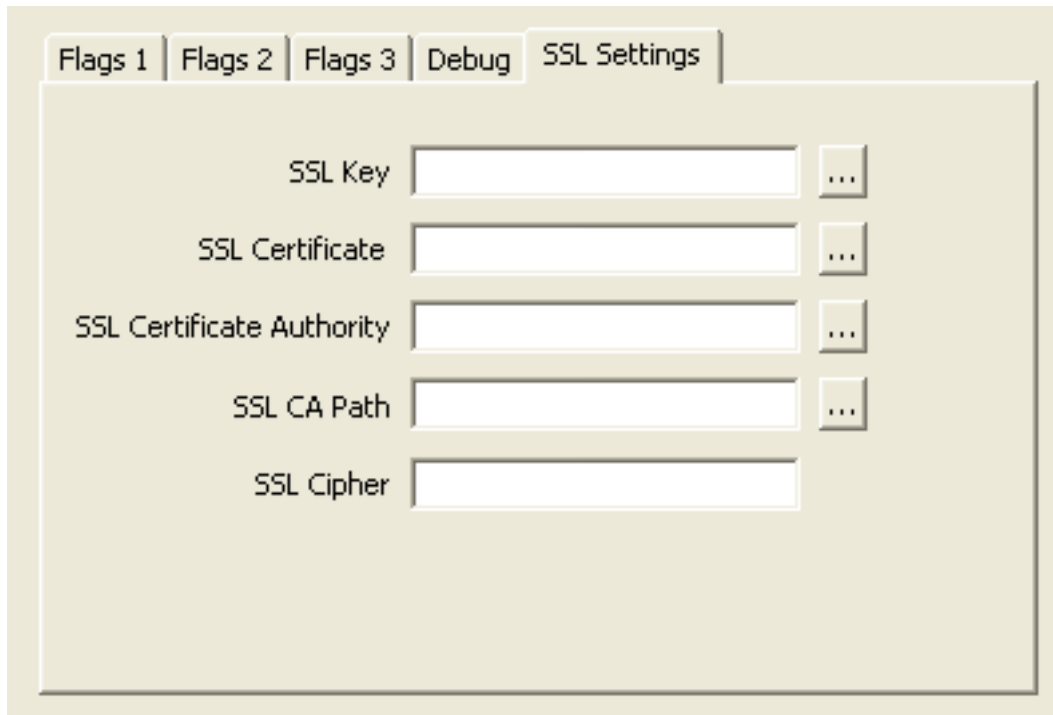
☐ Enable automatic reconnect

☐ Enable SQL_AUTO_IS_NULL

The **Details** button opens a tabbed display where you set additional options:

- **Flags 1**, **Flags 2**, and **Flags 3** enable you to select the additional flags for the DSN connection. For more information on these flags, see [Section 3.5.2, “Connector/ODBC Connection Parameters”](#).
- **Debug** lets you turn on ODBC debugging to record the queries you execute through the DSN to the `myodbc.sql` file. For more information, see [Section 3.5.8, “Getting an ODBC Trace File”](#).
- **SSL Settings** configures the additional options required for using the Secure Sockets Layer (SSL) when communicating with MySQL server.

You must also enable SSL and configure the MySQL server with suitable certificates to communicate over SSL.



The **Advanced** tab lets you configure Connector/ODBC connection parameters. Refer to [Section 3.5.2, “Connector/ODBC Connection Parameters”](#), for information about the meaning of these options.

3.5.3.2 Configuring a Connector/ODBC 5.x DSN on Windows, Using the Command Line

Use `myodbc-installer.exe` when configuring Connector/ODBC 5.1 or later from the command-line.

Execute `myodbc-installer.exe` without arguments to view a list of available options.

3.5.3.3 Troubleshooting ODBC Connection Problems

This section answers Connector/ODBC connection-related questions.

- **While configuring a Connector/ODBC DSN, a `Could Not Load Translator or Setup Library` error occurs**

For more information, refer to [MS KnowledgeBase Article\(Q260558\)](#). Also, make sure you have the latest valid `ct13d32.dll` in your system directory.

- On Windows, the default `myodbc5w.dll` (Unicode) or `myodbc5a.dll` (ANSI) is compiled for optimal performance. To debug Connector/ODBC (for example, to enable tracing), instead use `myodbc5d.dll`. To install this file, copy `myodbc5d.dll` over the installed `myodbc5w.dll` or

`myodbc5a.dll` file. Make sure to revert back to the release version of the driver DLL once you are done with the debugging, because the debug version may cause performance issues.

3.5.4 Configuring a Connector/ODBC DSN on OS X

To configure a DSN on OS X, you can either use the command-line utility (`myodbc-installer`), edit the `odbc.ini` file within the `Library/ODBC` directory of the user, or use the ODBC Administrator GUI. If you have OS X 10.2 or earlier, refer to [Section 3.5.5, “Configuring a Connector/ODBC DSN on Unix”](#). Select whether to create a User DSN or a System DSN. When adding a System DSN, you might need to authenticate with the system. Click the padlock and enter a user and password with administrator privileges.

For correct operation of ODBC Administrator, ensure that the `/Library/ODBC/odbc.ini` file used to set up ODBC connectivity and DSNs are writable by the `admin` group. If this file is not writable by this group, then the ODBC Administrator may fail, or may appear to work but not generate the correct entry.

Warning

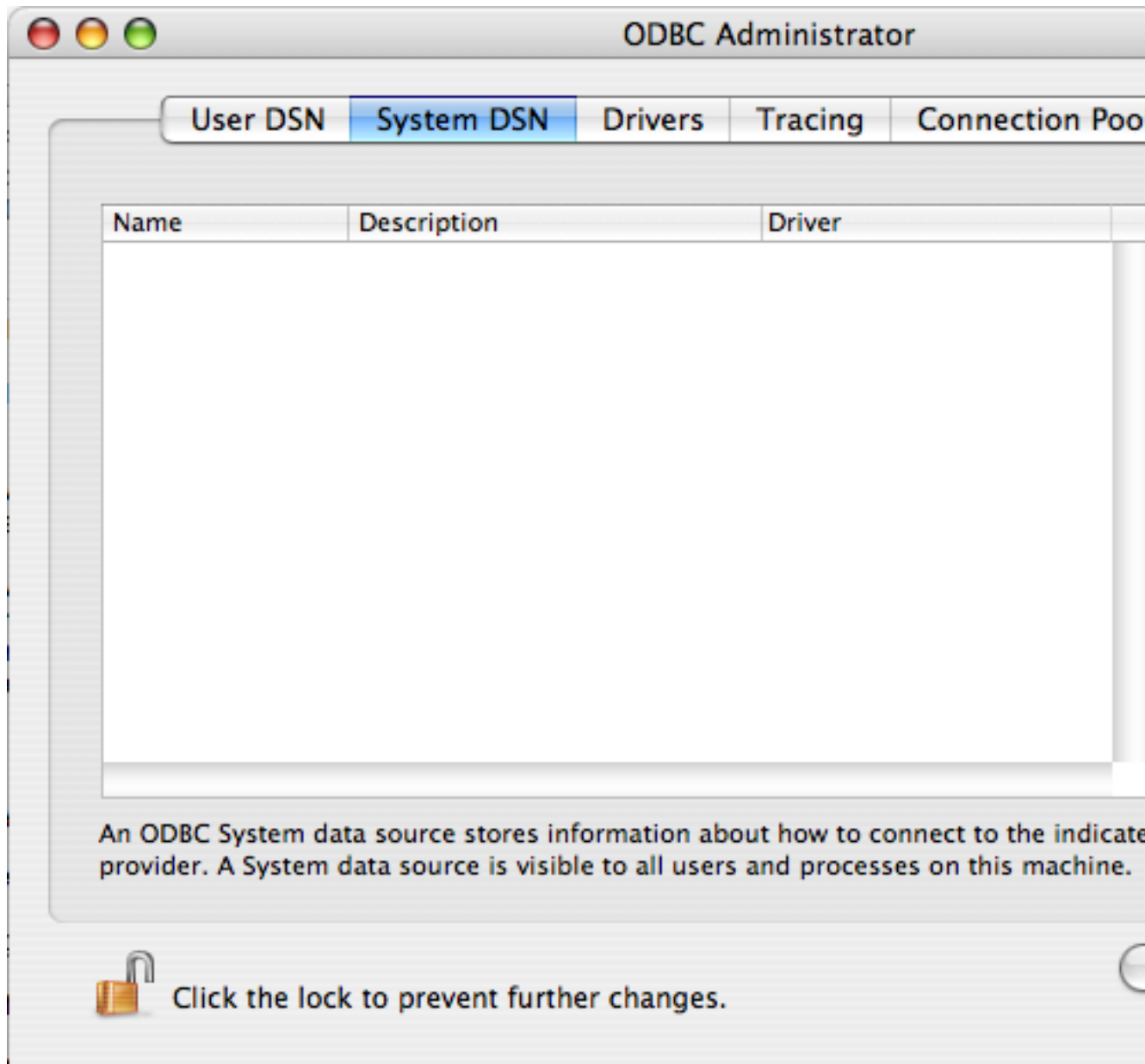
There are known issues with the OS X ODBC Administrator and Connector/ODBC that may prevent you from creating a DSN using this method. In this case, use the command line or edit the `odbc.ini` file directly. Existing DSNs or those that you created using the `myodbc-installer` tool can still be checked and edited using ODBC Administrator.

To create a DSN using the `myodbc-installer` utility, you need only specify the DSN type and the DSN connection string. For example:

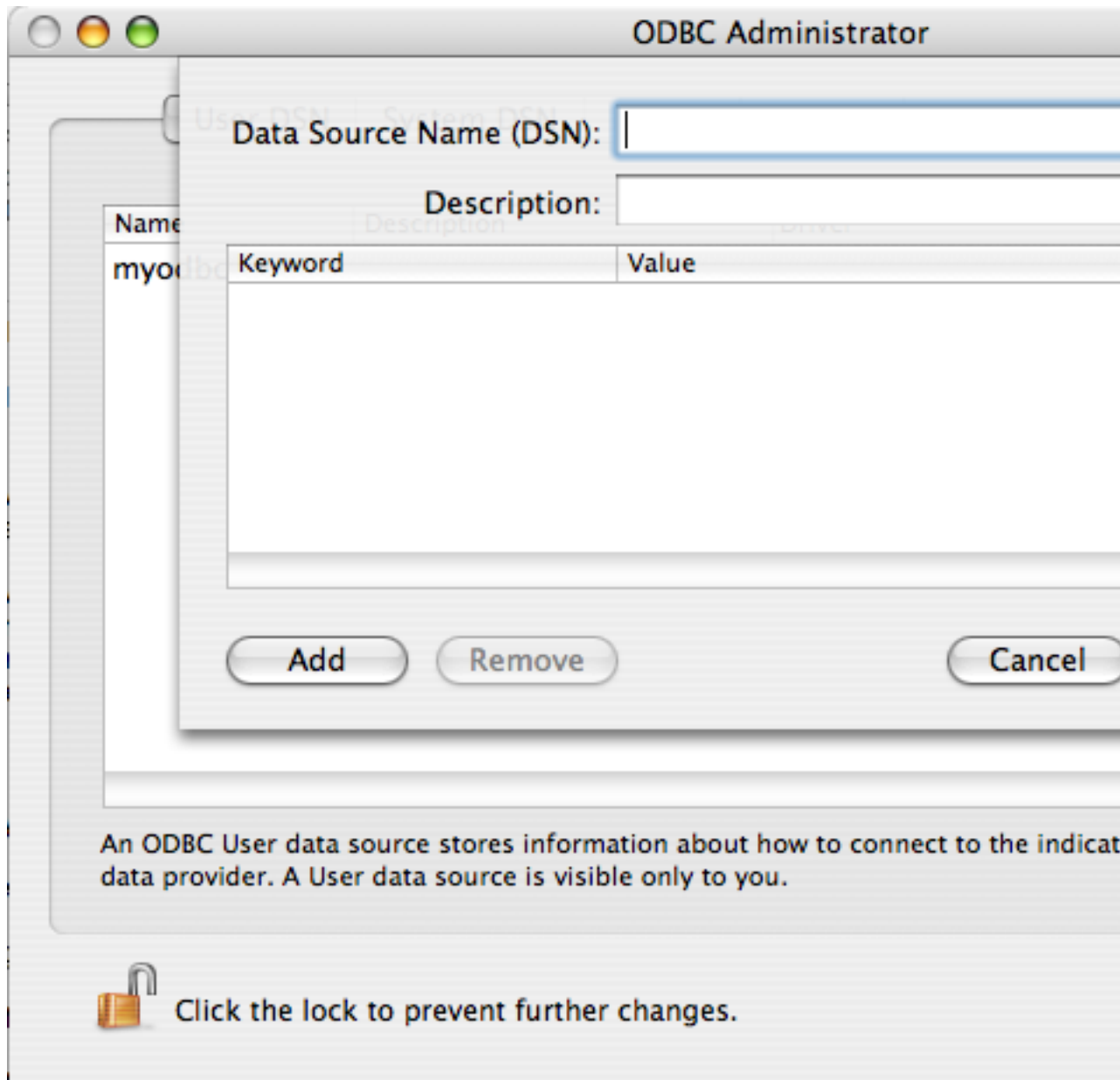
```
shell> myodbc-installer -a -s -t"DSN=mydb;DRIVER=MySQL ODBC 5.2 Driver;SERVER=mysql;USER=username;PASSW
```

To use ODBC Administrator:

1. Open the ODBC Administrator from the [Utilities](#) folder in the [Applications](#) folder.



2. On the User DSN or System DSN panel, click **Add**.
3. Select the Connector/ODBC driver and click **OK**.
4. You will be presented with the **Data Source Name** dialog. Enter the **Data Source Name** and an optional **Description** for the DSN.



5. Click **Add** to add a new keyword/value pair to the panel. Configure at least four pairs to specify the `server`, `username`, `password` and `database` connection parameters. See [Section 3.5.2, “Connector/ODBC Connection Parameters”](#).
6. Click **OK** to add the DSN to the list of configured data source names.

A completed DSN configuration may look like this:

Data Source Name (DSN): **WorldSample**

Description: **Connection to sample World database**

Keyword	Value
server	mysql
user	sakila
password	Sample
database	test_world

Buttons: Add, Remove, Cancel, OK

You can configure other ODBC options in your DSN by adding further keyword/value pairs and setting the corresponding values. See [Section 3.5.2, “Connector/ODBC Connection Parameters”](#).

3.5.5 Configuring a Connector/ODBC DSN on Unix

On **Unix**, you configure DSN entries directly in the `odbc.ini` file. Here is a typical `odbc.ini` file that configures `myodbc3` as the DSN name for Connector/ODBC 3.51:

```
;
; odbc.ini configuration for Connector/ODBC and Connector/ODBC 3.51 drivers
;
[ODBC Data Sources]
myodbc3      = MyODBC 3.51 Driver DSN
[myodbc3]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = Connector/ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password     =
Database     = test
OPTION      = 3
SOCKET      =
[Default]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = Connector/ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password     =
Database     = test
OPTION      = 3
SOCKET      =
```

Refer to the [Section 3.5.2, “Connector/ODBC Connection Parameters”](#), for the list of connection parameters that can be supplied.

Note

If you are using [unixODBC](#), you can use the following tools to set up the DSN:

- [ODBCConfig](#) GUI tool ([HOWTO: ODBCConfig](#))
- [odbcinst](#)

In some cases when using [unixODBC](#), you might get this error:

```
Data source name not found and no default driver specified
```

If this happens, make sure the [ODBCINI](#) and [ODBCSYSINI](#) environment variables are pointing to the right [odbc.ini](#) file. For example, if your [odbc.ini](#) file is located in `/usr/local/etc`, set the environment variables like this:

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

3.5.6 Connecting Without a Predefined DSN

You can connect to the MySQL server using [SQLDriverConnect](#), by specifying the [DRIVER](#) name field. Here are the connection strings for Connector/ODBC using DSN-less connections:

For Connector/ODBC 5.3:

```
ConnectionString = "DRIVER={MySQL ODBC 5.3 Driver};\
SERVER=localhost;\
DATABASE=test;\
USER=venu;\
PASSWORD=venu;\
OPTION=3;"
```

Substitute “MySQL ODBC 5.3 Driver” with the name by which you have registered your Connector/ODBC driver with the ODBC driver manager, if it is different. If your programming language converts backslash followed by whitespace to a space, it is preferable to specify the connection string as a single long string, or to use a concatenation of multiple strings that does not add spaces in between. For example:

```
ConnectionString = "DRIVER={MySQL ODBC 5.3 Driver};"
"SERVER=localhost;"
"DATABASE=test;"
"USER=venu;"
"PASSWORD=venu;"
"OPTION=3;"
```

Note. On OS X, you might need to specify the full path to the Connector/ODBC driver library.

Refer to [Section 3.5.2, “Connector/ODBC Connection Parameters”](#) for the list of connection parameters that can be supplied.

3.5.7 ODBC Connection Pooling

Connection pooling enables the ODBC driver to re-use existing connections to a given database from a pool of connections, instead of opening a new connection each time the database is accessed. By enabling connection pooling you can improve the overall performance of your application by lowering the time taken to open a connection to a database in the connection pool.

For more information about connection pooling: <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169470>.

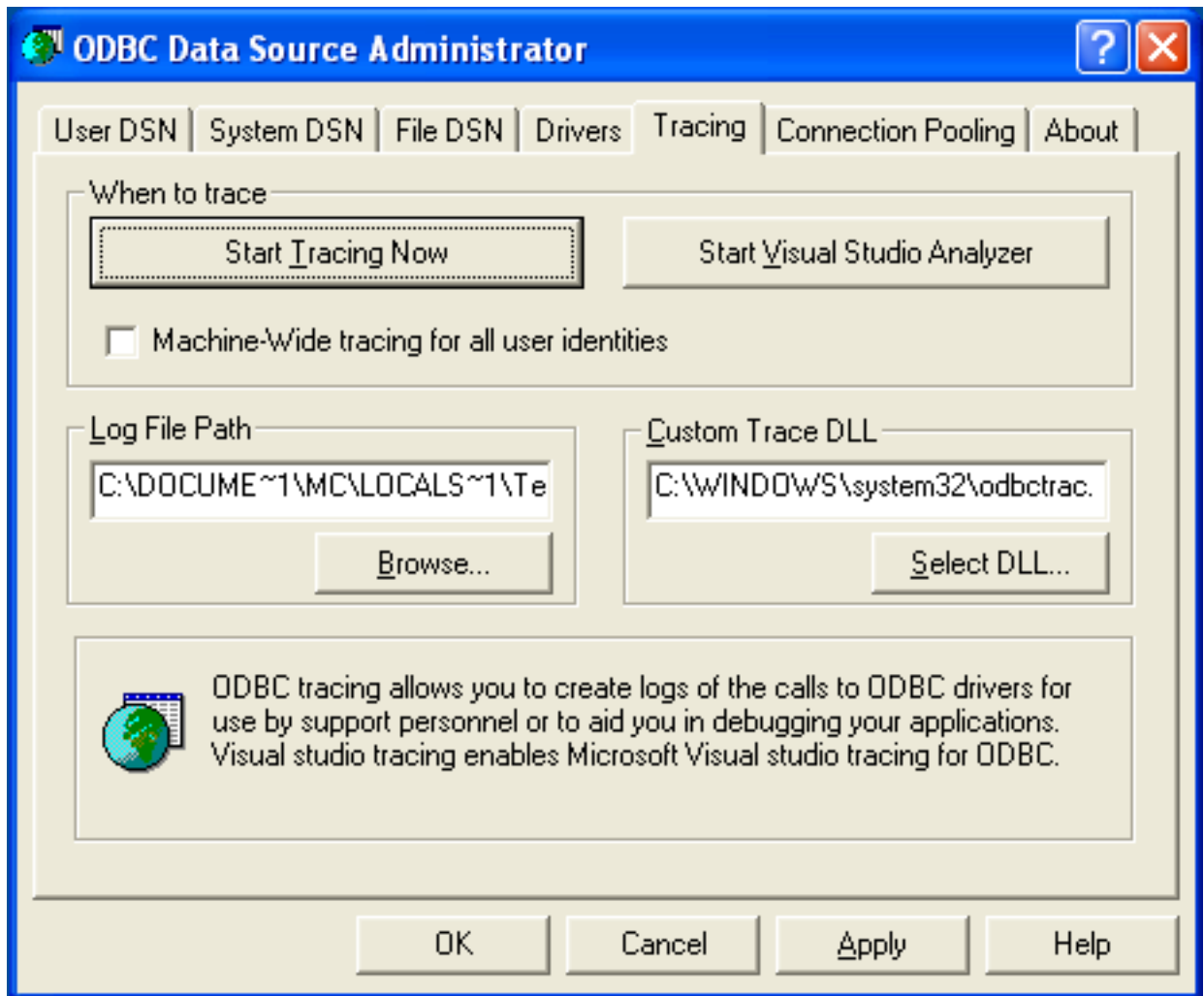
3.5.8 Getting an ODBC Trace File

If you encounter difficulties or problems with Connector/ODBC, start by making a log file from the [ODBC Manager](#) and Connector/ODBC. This is called *tracing*, and is enabled through the ODBC Manager. The procedure for this differs for Windows, OS X and Unix.

3.5.8.1 Enabling ODBC Tracing on Windows

To enable the trace option on Windows:

1. The [Tracing](#) tab of the ODBC Data Source Administrator dialog box lets you configure the way ODBC function calls are traced.



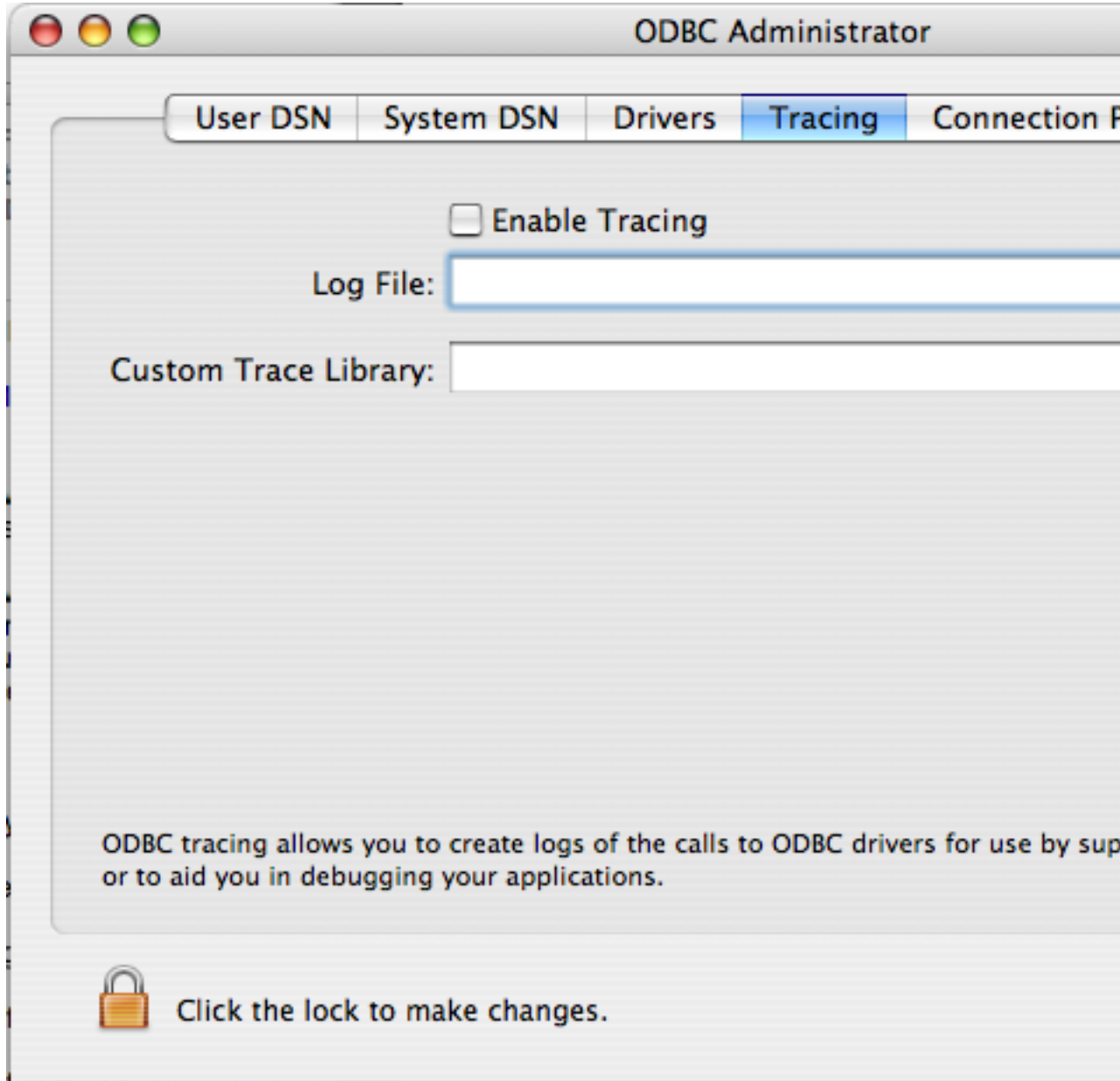
2. When you activate tracing from the [Tracing](#) tab, the [Driver Manager](#) logs all ODBC function calls for all subsequently run applications.
3. ODBC function calls from applications running before tracing is activated are not logged. ODBC function calls are recorded in a log file you specify.
4. Tracing ceases only after you click [Stop Tracing Now](#). Remember that while tracing is on, the log file continues to increase in size and that tracing affects the performance of all your ODBC applications.

3.5.8.2 Enabling ODBC Tracing on OS X

To enable the trace option on OS X 10.3 or later, use the [Tracing](#) tab within ODBC Administrator .

1. Open the ODBC Administrator.

2. Select the **Tracing** tab.



3. Select the **Enable Tracing** check box.
4. Enter the location to save the Tracing log. To append information to an existing log file, click the **Choose...** button.

3.5.8.3 Enabling ODBC Tracing on Unix

To enable the trace option on OS X 10.2 (or earlier) or Unix, add the **trace** option to the ODBC configuration:

1. On Unix, explicitly set the **Trace** option in the **ODBC.INI** file.

Set the tracing **ON** or **OFF** by using **TraceFile** and **Trace** parameters in **odbc.ini** as shown below:

```
TraceFile = /tmp/odbc.trace
Trace     = 1
```

`TraceFile` specifies the name and full path of the trace file and `Trace` is set to `ON` or `OFF`. You can also use `1` or `YES` for `ON` and `0` or `NO` for `OFF`. If you are using `ODBCConfig` from `unixODBC`, then follow the instructions for tracing `unixODBC` calls at [HOWTO-ODBCConfig](#).

3.5.8.4 Enabling a Connector/ODBC Log

To generate a Connector/ODBC log, do the following:

1. Within Windows, enable the `Trace Connector/ODBC` option flag in the Connector/ODBC connect/configure screen. The log is written to file `C:\myodbc.log`. If the trace option is not remembered when you are going back to the above screen, it means that you are not using the `myodbcd.dll` driver, see [Section 3.5.3.3, "Troubleshooting ODBC Connection Problems"](#).

On OS X, Unix, or if you are using a DSN-less connection, either supply `OPTION=4` in the connection string, or set the corresponding keyword/value pair in the DSN.

2. Start your application and try to get it to fail. Then check the Connector/ODBC trace file to find out what could be wrong.

If you need help determining what is wrong, see [Section 3.9.1, "Connector/ODBC Community Support"](#).

3.6 Connector/ODBC Examples

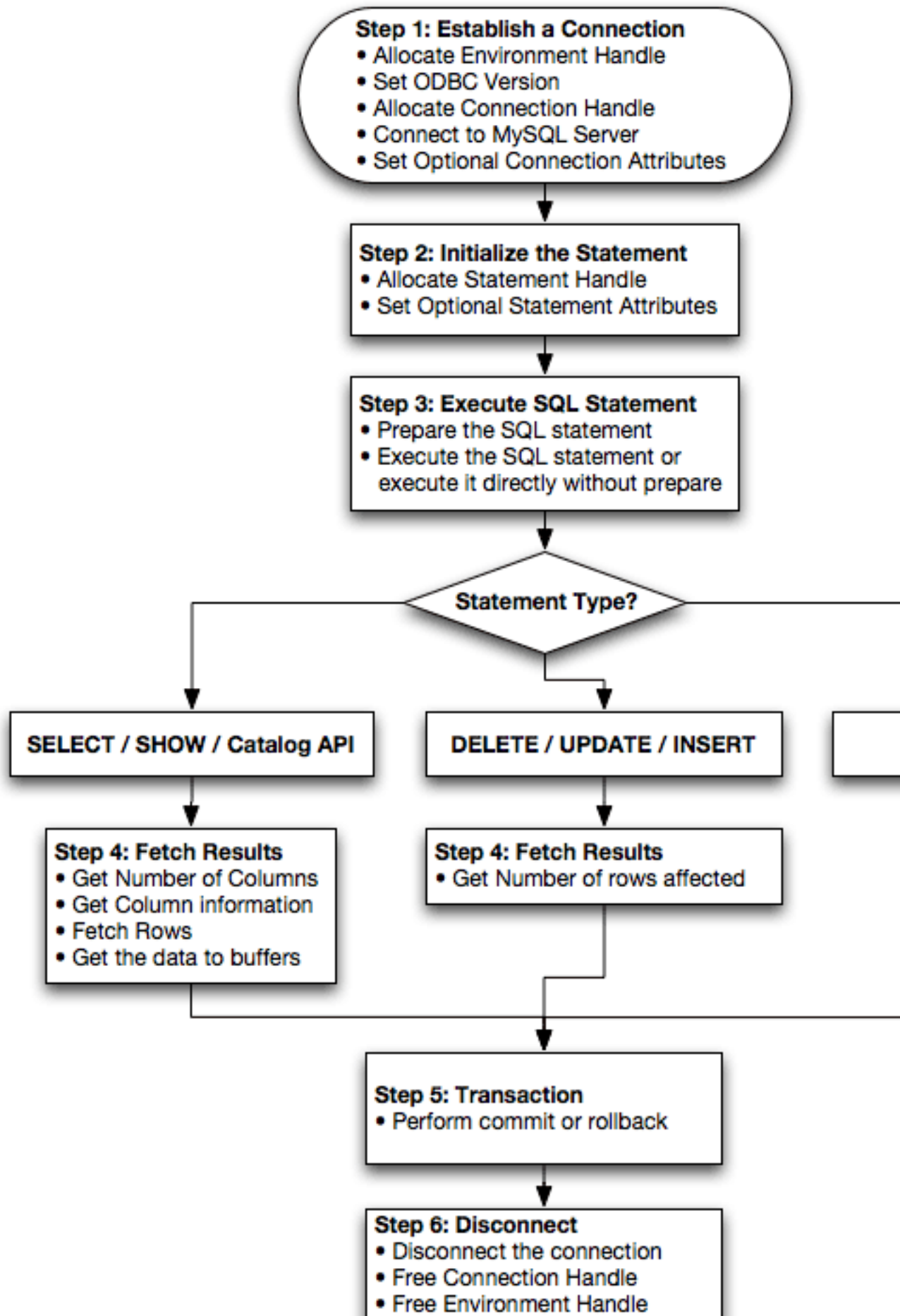
Once you have configured a DSN to provide access to a database, how you access and use that connection is dependent on the application or programming language. As ODBC is a standardized interface, any application or language that supports ODBC can use the DSN and connect to the configured database.

3.6.1 Basic Connector/ODBC Application Steps

Interacting with a MySQL server from an applications using the Connector/ODBC typically involves the following operations:

- Configure the Connector/ODBC DSN.
- Connect to MySQL server.
- Initialization operations.
- Execute SQL statements.
- Retrieve results.
- Perform [transactions](#).
- Disconnect from the server.

Most applications use some variation of these steps. The basic application steps are shown in the following diagram:



3.6.2 Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC

A typical situation where you would install Connector/ODBC is to access a database on a Linux or Unix host from a Windows machine.

As an example of the process required to set up access between two machines, the steps below take you through the basic steps. These instructions assume that you connect to system ALPHA from system BETA with a user name and password of `myuser` and `mypassword`.

On system ALPHA (the MySQL server) follow these steps:

1. Start the MySQL server.
2. Use `GRANT` to set up an account with a user name of `myuser` that can connect from system BETA using a password of `myuser` to the database `test`:

```
GRANT ALL ON test.* to 'myuser'@'BETA' IDENTIFIED BY 'mypassword';
```

For more information about MySQL privileges, refer to [MySQL User Account Management](#).

On system BETA (the Connector/ODBC client), follow these steps:

1. Configure a Connector/ODBC DSN using parameters that match the server, database and authentication information that you have just configured on system ALPHA.

Parameter	Value	Comment
DSN	remote_test	A name to identify the connection.
SERVER	ALPHA	The address of the remote server.
DATABASE	test	The name of the default database.
USER	myuser	The user name configured for access to this database.
PASSWORD	mypassword	The password for <code>myuser</code> .

2. Using an ODBC-capable application, such as Microsoft Office, connect to the MySQL server using the DSN you have just created. If the connection fails, use tracing to examine the connection process. See [Section 3.5.8, "Getting an ODBC Trace File"](#), for more information.

3.6.3 Connector/ODBC and Third-Party ODBC Tools

Once you have configured your Connector/ODBC DSN, you can access your MySQL database through any application that supports the ODBC interface, including programming languages and third-party applications. This section contains guides and help on using Connector/ODBC with various ODBC-compatible tools and applications, including Microsoft Word, Microsoft Excel and Adobe/Macromedia ColdFusion.

Connector/ODBC has been tested with the following applications:

Publisher	Application	Notes
Adobe	ColdFusion	Formerly Macromedia ColdFusion
Borland	C++ Builder	
	Builder 4	
	Delphi	
Business Objects	Crystal Reports	
Claris	Filemaker Pro	

Publisher	Application	Notes
Corel	Paradox	
Computer Associates	Visual Objects	Also known as CAVO
	AllFusion ERwin Data Modeler	
Gupta	Team Developer	Previously known as Centura Team Developer; Gupta SQL/Windows
Gensym	G2-ODBC Bridge	
Inline	iHTML	
Lotus	Notes	Versions 4.5 and 4.6
Microsoft	Access	
	Excel	
	Visio Enterprise	
	Visual C++	
	Visual Basic	
	ODBC.NET	Using C#, Visual Basic, C++
	FoxPro	
	Visual Interdev	
OpenOffice.org	OpenOffice.org	
Perl	DBD::ODBC	
Pervasive Software	DataJunction	
Sambar Technologies	Sambar Server	
SPSS	SPSS	
SoftVelocity	Clarion	
SQLExpress	SQLExpress for Xbase+ +	
Sun	StarOffice	
SunSystems	Vision	
Sybase	PowerBuilder	
	PowerDesigner	
theKompany.com	Data Architect	

If you know of any other applications that work with Connector/ODBC, please send mail to <myodbc@lists.mysql.com> about them.

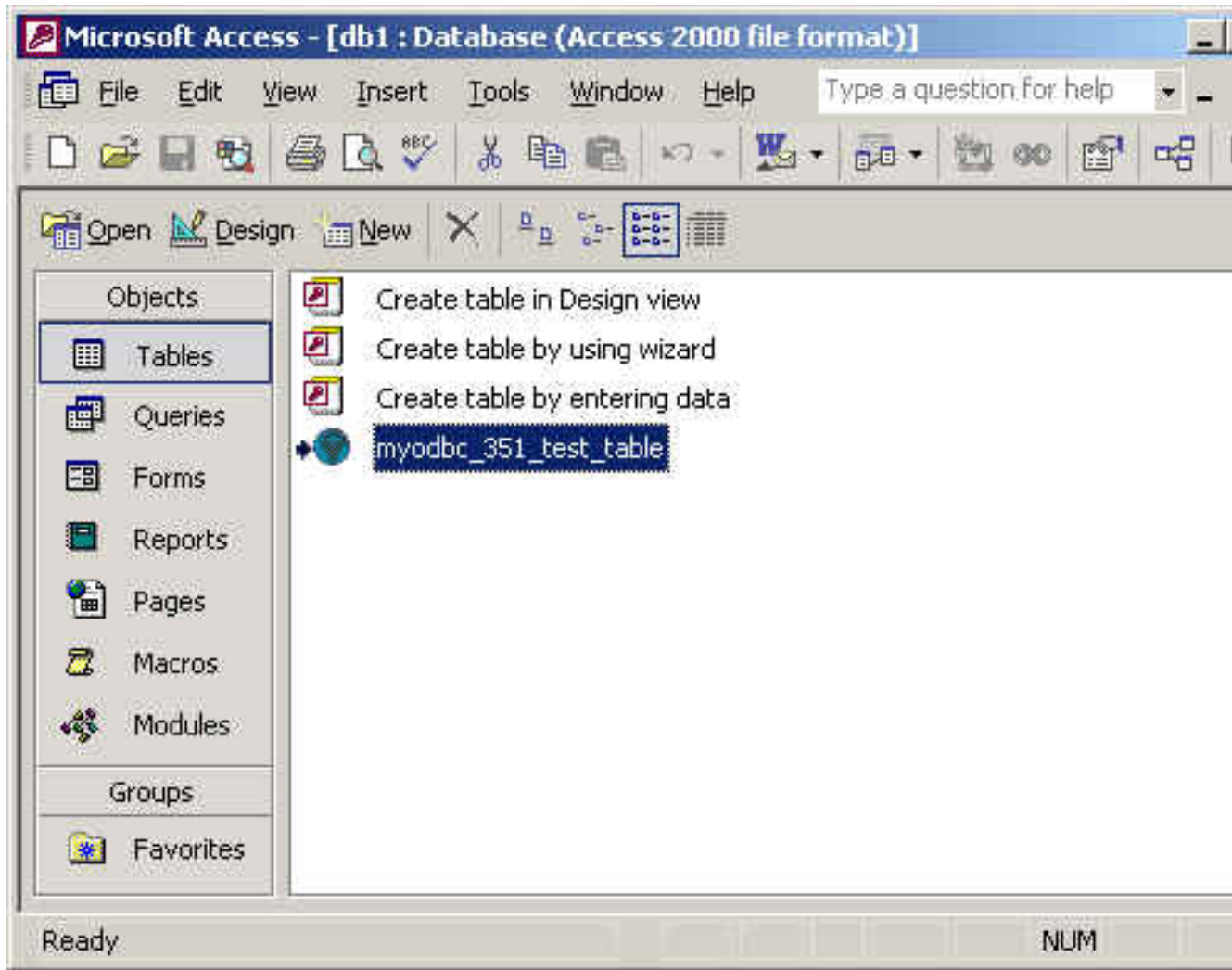
3.6.4 Using Connector/ODBC with Microsoft Access

You can use MySQL database with Microsoft Access using Connector/ODBC. The MySQL database can be used as an import source, an export source, or as a linked table for direct use within an Access application, so you can use Access as the front-end interface to a MySQL database.

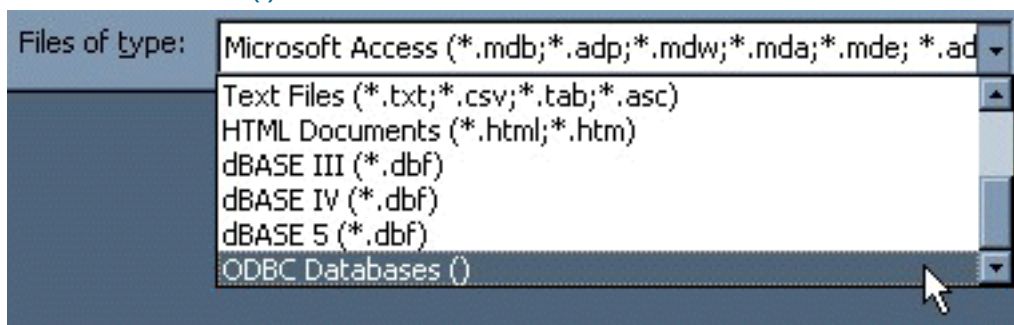
3.6.4.1 Exporting Access Data to MySQL

To export a table of data from an Access database to MySQL, follow these instructions:

1. When you open an Access database or an Access project, a Database window appears. It displays shortcuts for creating new database objects and opening existing objects.



2. Click the name of the [table](#) or [query](#) to export, and then in the [File](#) menu, select [Export](#).
3. In the [Export Object Type Object name To](#) dialog box, in the [Save As Type](#) box, select [ODBC Databases \(\)](#) as shown here:



4. In the [Export](#) dialog box, enter a name for the file (or use the suggested name), and then select [OK](#).
5. The [Select Data Source](#) dialog box is displayed; it lists the defined data sources for any ODBC drivers installed on your computer. Click either the [File Data Source](#) or [Machine Data Source](#) tab, and then double-click the [Connector/ODBC](#) or [Connector/ODBC 3.51](#) data source to export to. To define a new data source for [Connector/ODBC](#), please [Section 3.5.3, "Configuring a Connector/ODBC DSN on Windows"](#).

Note

Ensure that the information that you are exporting to the MySQL table is valid for the corresponding MySQL data types. Values that are outside of the supported range of the MySQL data type but valid within Access may trigger an “overflow” error during the export.

Microsoft Access connects to the MySQL Server through this data source and exports new tables and or data.

3.6.4.2 Importing MySQL Data to Access

To import a table or tables from MySQL to Access, follow these instructions:

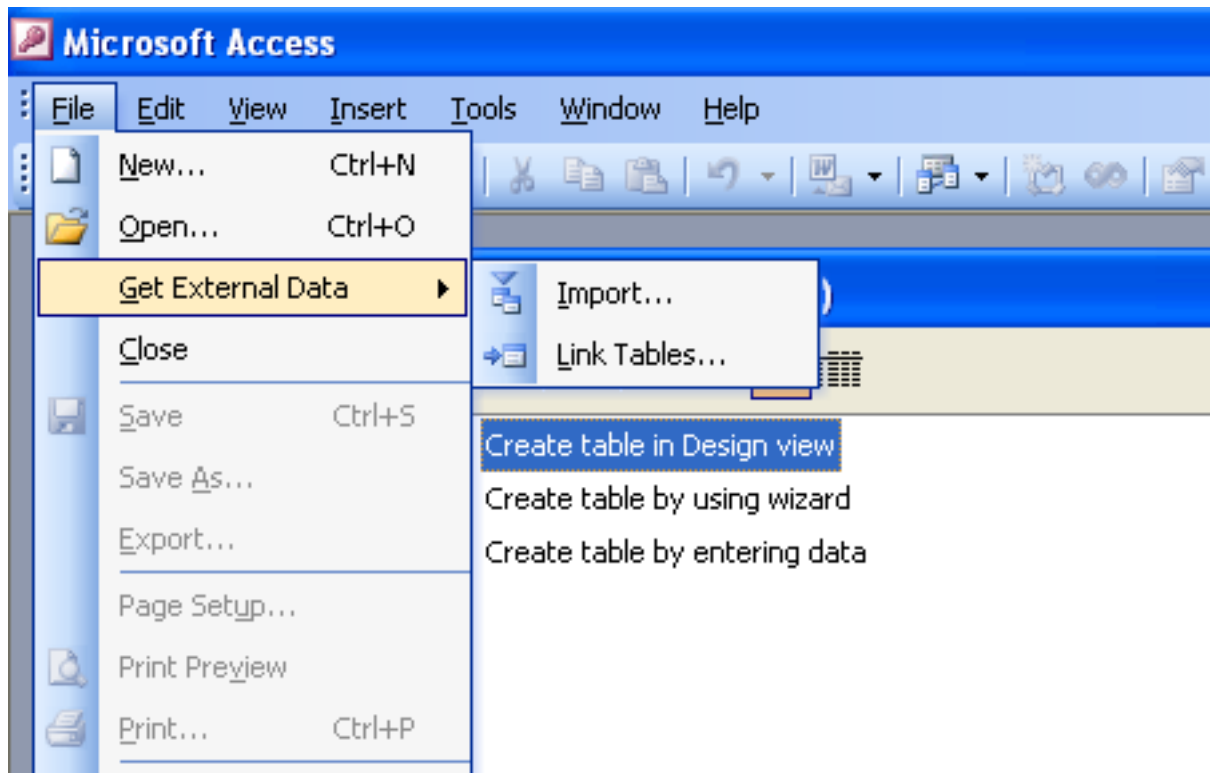
1. Open a database, or switch to the Database window for the open database.
2. To import tables, on the **File** menu, point to **Get External Data**, and then click **Import**.
3. In the **Import** dialog box, in the Files Of Type box, select **ODBC Databases ()**. The Select Data Source dialog box lists the defined data sources **The Select Data Source** dialog box is displayed; it lists the defined data source names.
4. If the ODBC data source that you selected requires you to log on, enter your login ID and password (additional information might also be required), and then click **OK**.
5. Microsoft Access connects to the MySQL server through **ODBC data source** and displays the list of tables that you can **import**.
6. Click each table to **import**, and then click **OK**.

3.6.4.3 Using Microsoft Access as a Front-end to MySQL

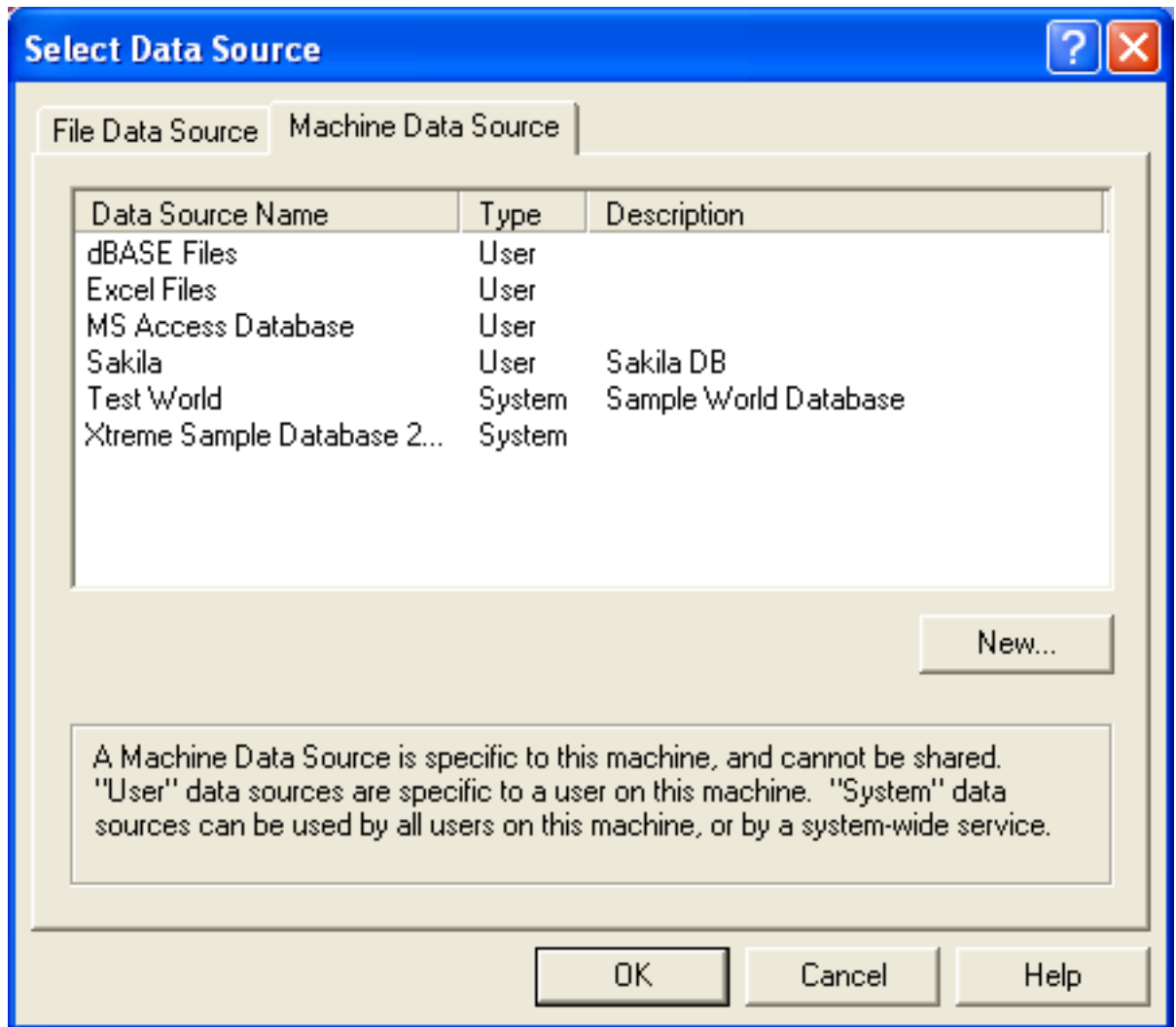
You can use Microsoft Access as a front end to a MySQL database by linking tables within your Microsoft Access database to tables that exist within your MySQL database. When a query is requested on a table within Access, ODBC is used to execute the queries on the MySQL database instead.

To create a linked table:

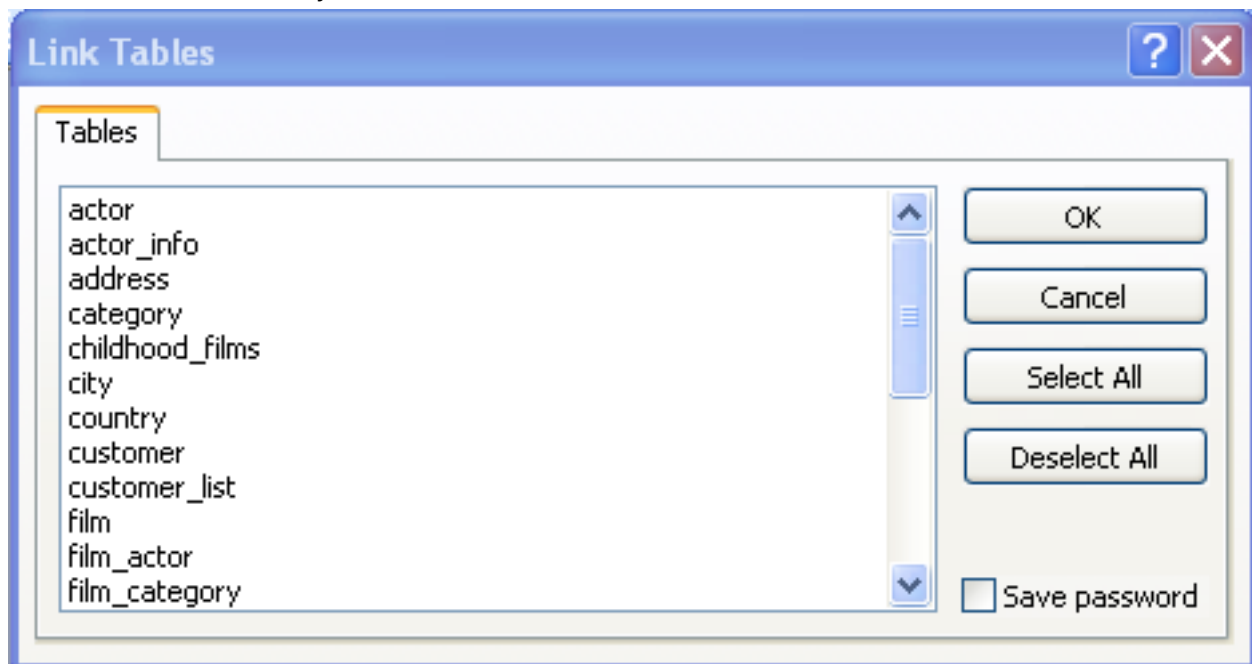
1. Open the Access database that you want to link to MySQL.
2. From the **File**, choose **Get External Data->Link Tables**.



3. From the browser, choose **ODBC Databases ()** from the **Files of type** pop-up.
4. In the **Select Data Source** window, choose an existing DSN, either from a **File Data Source** or **Machine Data Source**. You can also create a new DSN using the **New...** button. For more information on creating a DSN see [Section 3.5.3, "Configuring a Connector/ODBC DSN on Windows"](#).



5. In the **Link Tables** dialog, select one or more tables from the MySQL database. A link will be created to each table that you select from this list.



6. If Microsoft Access is unable to determine the unique record identifier for a table automatically then it may ask you to confirm the column, or combination of columns, to be used to uniquely identify each row from the source table. Select the columns to use and click **OK**.



Once the process has been completed, you can now build interfaces and queries to the linked tables just as you would for any Access database.

Use the following procedure to view or to refresh links when the structure or location of a linked table has changed. The Linked Table Manager lists the paths to all currently linked tables.

To view or refresh links:

1. Open the database that contains links to MySQL tables.
2. On the **Tools** menu, point to **Add-ins (Database Utilities)** in Access 2000 or newer, and then click **Linked Table Manager**.
3. Select the check box for the tables whose links you want to refresh.
4. Click **OK** to refresh the links.

Microsoft Access confirms a successful refresh or, if the table wasn't found, displays the **Select New Location of <table name>** dialog box in which you can specify its the table's new location. If several selected tables have moved to the new location that you specify, the Linked Table Manager searches that location for all selected tables, and updates all links in one step.

To change the path for a set of linked tables:

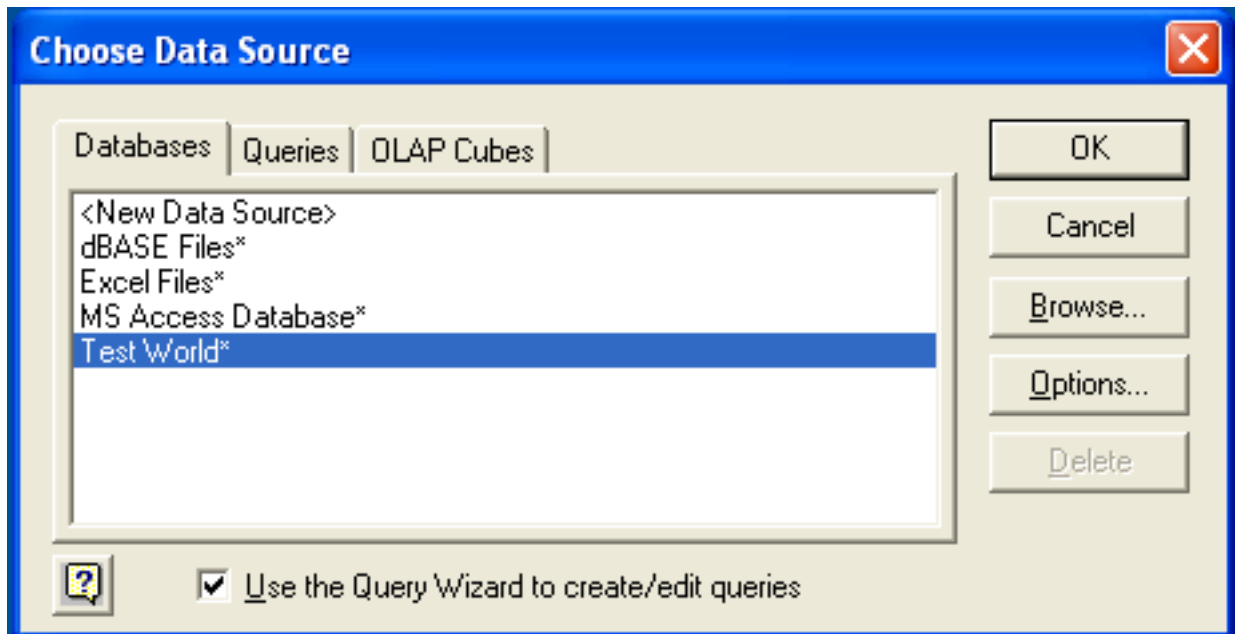
1. Open the database that contains links to tables.
2. On the **Tools** menu, point to **Add-ins (Database Utilities)** in Access 2000 or newer, and then click **Linked Table Manager**.
3. Select the **Always Prompt For A New Location** check box.
4. Select the check box for the tables whose links you want to change, and then click **OK**.
5. In the **Select New Location of <table name>** dialog box, specify the new location, click **Open**, and then click **OK**.

3.6.5 Using Connector/ODBC with Microsoft Word or Excel

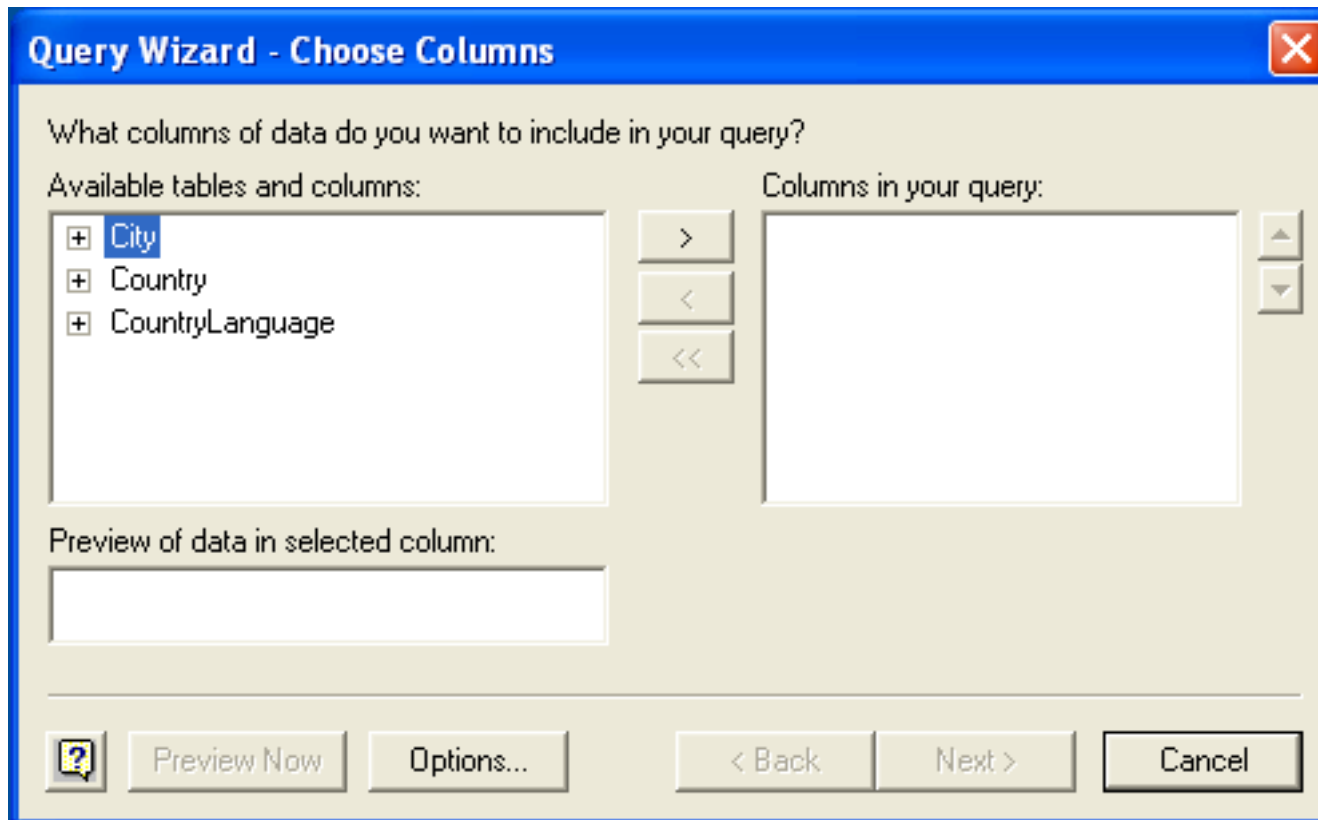
You can use Microsoft Word and Microsoft Excel to access information from a MySQL database using Connector/ODBC. Within Microsoft Word, this facility is most useful when importing data for mailmerge, or for tables and data to be included in reports. Within Microsoft Excel, you can execute queries on your MySQL server and import the data directly into an Excel Worksheet, presenting the data as a series of rows and columns.

With both applications, data is accessed and imported into the application using Microsoft Query, which lets you execute a query through an ODBC source. You use Microsoft Query to build the SQL statement to be executed, selecting the tables, fields, selection criteria and sort order. For example, to insert information from a table in the World test database into an Excel spreadsheet, using the DSN samples shown in [Section 3.5, “Configuring Connector/ODBC”](#):

1. Create a new Worksheet.
2. From the **Data** menu, choose **Import External Data**, and then select **New Database Query**.
3. Microsoft Query will start. First, you need to choose the data source, by selecting an existing Data Source Name.



4. Within the **Query Wizard**, choose the columns to import. The list of tables available to the user configured through the DSN is shown on the left, the columns that will be added to your query are shown on the right. The columns you choose are equivalent to those in the first section of a **SELECT** query. Click **Next** to continue.



Query Wizard - Choose Columns

What columns of data do you want to include in your query?

Available tables and columns:

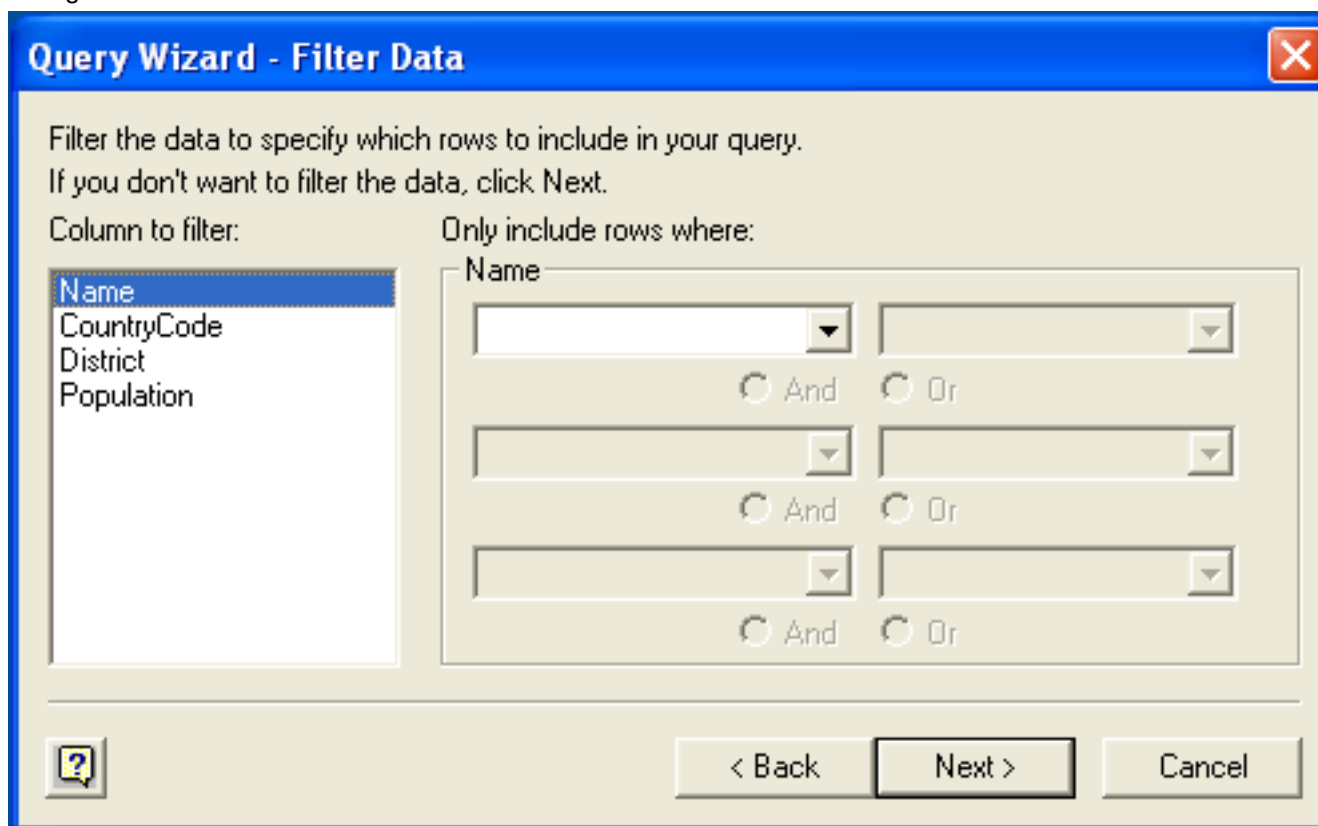
- + City
- + Country
- + CountryLanguage

Columns in your query:

Preview of data in selected column:

Buttons:  Preview Now Options... < Back Next > Cancel

5. You can filter rows from the query (the equivalent of a **WHERE** clause) using the **Filter Data** dialog. Click **Next** to continue.



Query Wizard - Filter Data

Filter the data to specify which rows to include in your query.
If you don't want to filter the data, click Next.

Column to filter:

- Name
- CountryCode
- District
- Population

Only include rows where:

Name

☐ And ☐ Or

☐ And ☐ Or

☐ And ☐ Or

Buttons:  < Back Next > Cancel

6. Select an (optional) sort order for the data. This is equivalent to using a **ORDER BY** clause in your SQL query. You can select up to three fields for sorting the information returned by the query. Click **Next** to continue.

Query Wizard - Sort Order

Specify how you want your data sorted.
If you don't want to sort the data, click Next.

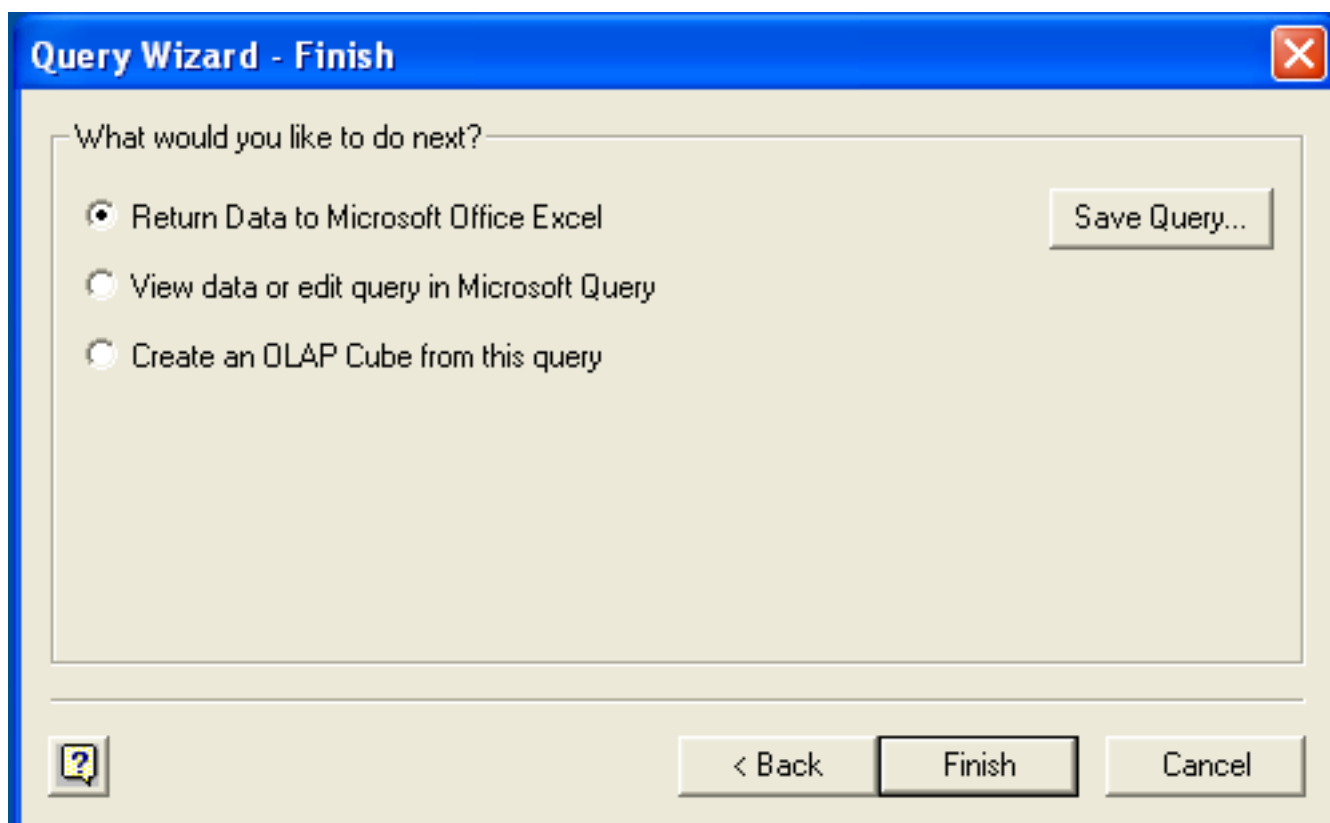
Sort by
Name ☒ Ascending ☐ Descending

Then by
☐ Ascending ☐ Descending

Then by
☐ Ascending ☐ Descending

 < Back Next > Cancel

7. Select the destination for your query. You can select to return the data Microsoft Excel, where you can choose a worksheet and cell where the data will be inserted; you can continue to view the query and results within Microsoft Query, where you can edit the SQL query and further filter and sort the information returned; or you can create an OLAP Cube from the query, which can then be used directly within Microsoft Excel. Click **Finish**.



The same process can be used to import data into a Word document, where the data will be inserted as a table. This can be used for mail merge purposes (where the field data is read from a Word table), or where you want to include data and reports within a report or other document.

3.6.6 Using Connector/ODBC with Crystal Reports

Crystal Reports can use an ODBC DSN to connect to a database from which you to extract data and information for reporting purposes.

Note

There is a known issue with certain versions of Crystal Reports where the application is unable to open and browse tables and fields through an ODBC connection. Before using Crystal Reports with MySQL, please ensure that you have update to the latest version, including any outstanding service packs and hotfixes. For more information on this issue, see the [Business\) Objects Knowledgebase](#) for more information.

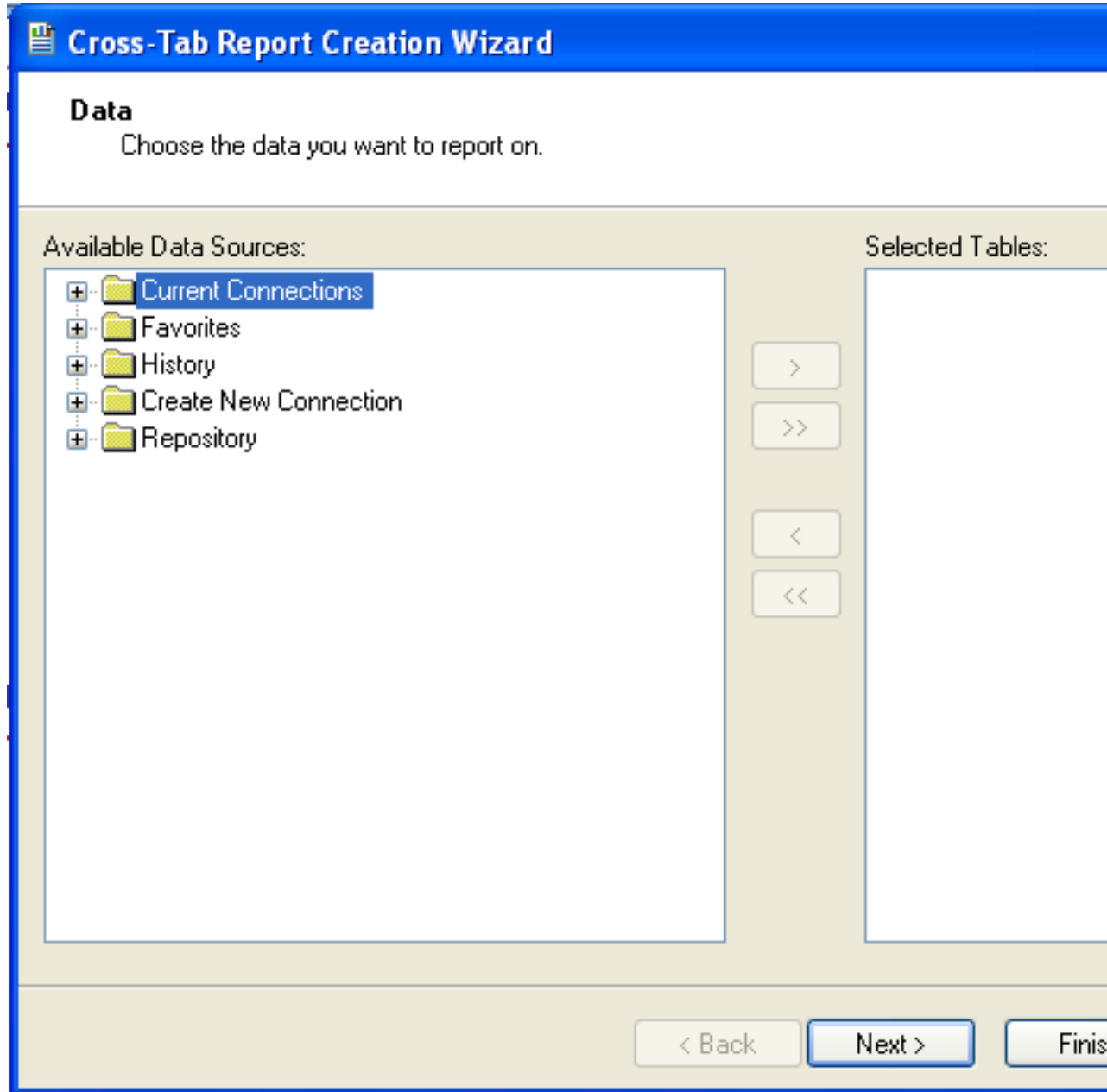
For example, to create a simple crosstab report within Crystal Reports XI, follow these steps:

1. Create a DSN using the [Data Sources \(ODBC\)](#) tool. You can either specify a complete database, including user name and password, or you can build a basic DSN and use Crystal Reports to set the user name and password.

For the purposes of this example, a DSN that provides a connection to an instance of the MySQL Sakila sample database has been created.

2. Open Crystal Reports and create a new project, or an open an existing reporting project into which you want to insert data from your MySQL data source.
3. Start the Cross-Tab Report Wizard, either by clicking the option on the Start Page. Expand the **Create New Connection** folder, then expand the **ODBC (RDO)** folder to obtain a list of ODBC data sources.

You will be asked to select a data source.



4. When you first expand the **ODBC (RDO)** folder you will be presented the Data Source Selection screen. From here you can select either a pre-configured DSN, open a file-based DSN or enter and manual connection string. For this example, the **Sakila** DSN will be used.

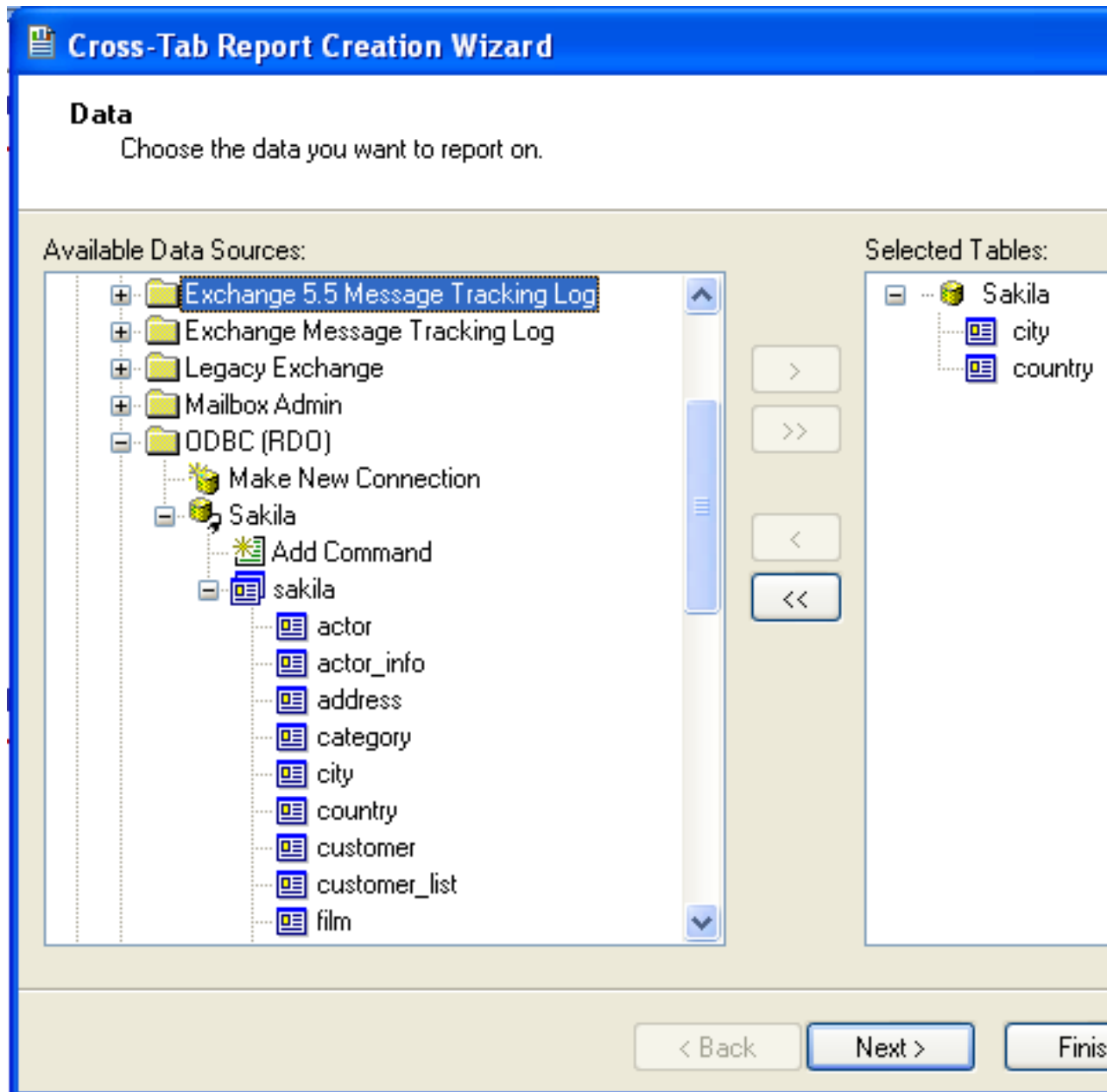
If the DSN contains a user name/password combination, or you want to use different authentication credentials, click **Next** to enter the user name and password that you want to use. Otherwise, click **Finish** to continue the data source selection wizard.

The screenshot shows the 'ODBC (RDO)' dialog box with the 'Data Source Selection' tab active. The dialog has a blue title bar with a close button. Below the title bar, the text 'Data Source Selection' is followed by the instruction 'Choose a data source from the list or open a file dsn from the browse button'. There are three radio buttons: 'Select Data Source:' (which is selected), 'Find File DSN:', and 'Enter Connection String:'. The 'Select Data Source:' option is selected, and a list box shows the following data sources: dBASE Files, Excel Files, MS Access Database, MySQLTest, Sakila, Test World, and Xtreme Sample Database 11. Below the list box, there is a text field for 'File DSN:' with a browse button (...). Below that, there is a text field for 'Connection String:'. At the bottom of the dialog, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

5. You will be returned the Cross-Tab Report Creation Wizard. You now need to select the database and tables that you want to include in your report. For our example, we will expand the selected Sakila database. Click the **city** table and use the > button to add the table to the report. Then repeat the action with the **country** table. Alternatively you can select multiple tables and add them to the report.

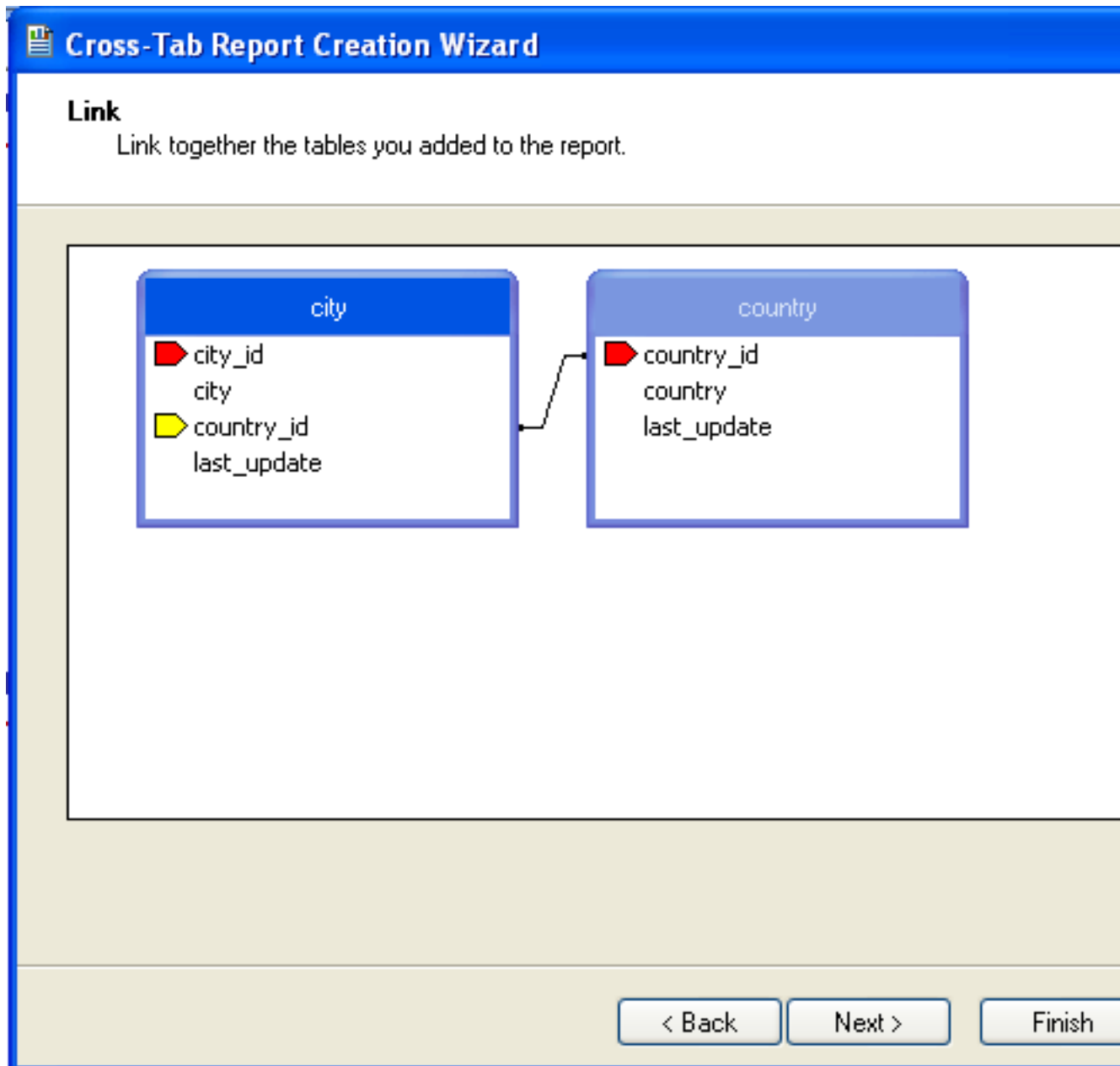
Finally, you can select the parent **Sakila** resource and add of the tables to the report.

Once you have selected the tables you want to include, click **Next** to continue.



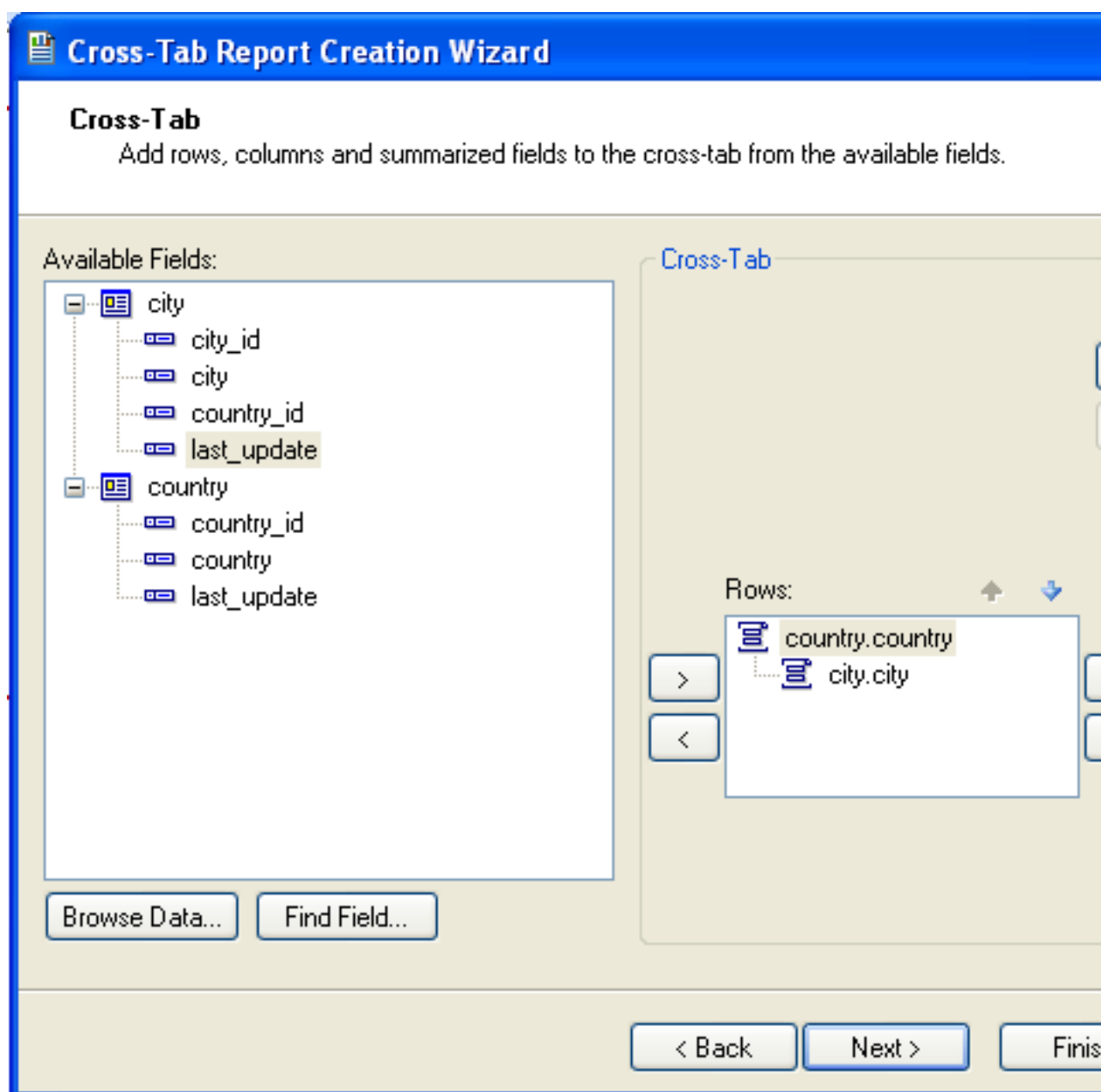
6. Crystal Reports will now read the table definitions and automatically identify the links between the tables. The identification of links between tables enables Crystal Reports to automatically lookup and summarize information based on all the tables in the database according to your query. If Crystal Reports is unable to perform the linking itself, you can manually create the links between fields in the tables you have selected.

Click **Next** to continue the process.



7. You can now select the columns and rows that to include within the Cross-Tab report. Drag and drop or use the > buttons to add fields to each area of the report. In the example shown, we will report on cities, organized by country, incorporating a count of the number of cities within each country. If you want to browse the data, select a field and click the **Browse Data...** button.

Click **Next** to create a graph of the results. Since we are not creating a graph from this data, click **Finish** to generate the report.



8. The finished report will be shown, a sample of the output from the Sakila sample database is shown below.

		Total
Total		600
Afghanistan	Total	1
	Kabul	1
Algeria	Total	3
	Batna	1
	Bchar	1
	Skikda	1
American Samoa	Total	1
	Tafuna	1
Angola	Total	2
	Benguela	1
	Namibe	1
Anguilla	Total	1
	South Hill	1
Argentina	Total	13
	Almirante Brow	1

Once the ODBC connection has been opened within Crystal Reports, you can browse and add any fields within the available tables into your reports.

3.6.7 Connector/ODBC Programming

With a suitable ODBC Manager and the Connector/ODBC driver installed, any programming language or environment that can support ODBC can connect to a MySQL database through Connector/ODBC.

This includes, but is not limited to, Microsoft support languages (including Visual Basic, C# and interfaces such as ODBC.NET), Perl (through the DBI module, and the DBD::ODBC driver).

3.6.7.1 Using Connector/ODBC with Visual Basic Using ADO, DAO and RDO

This section contains simple examples of the use of Connector/ODBC with ADO, DAO and RDO.

ADO: `rs.addNew`, `rs.delete`, and `rs.update`

The following ADO (ActiveX Data Objects) example creates a table `my_ado` and demonstrates the use of `rs.addNew`, `rs.delete`, and `rs.update`.

```
Private Sub myodbc_ado_Click()
Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim fld As ADODB.Field
Dim sql As String
'connect to MySQL server using Connector/ODBC
Set conn = New ADODB.Connection
conn.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"
conn.Open
'create table
conn.Execute "DROP TABLE IF EXISTS my_ado"
conn.Execute "CREATE TABLE my_ado(id int not null primary key, name varchar(20)," _
& "txt text, dt date, tm time, ts timestamp)"
'direct insert
conn.Execute "INSERT INTO my_ado(id,name,txt) values(1,100,'venu')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(2,200,'MySQL')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(3,300,'Delete')"
Set rs = New ADODB.Recordset
rs.CursorLocation = adUseServer
'fetch the initial table ..
rs.Open "SELECT * FROM my_ado", conn
Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Initial my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print
Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close
'rs insert
rs.Open "select * from my_ado", conn, adOpenDynamic, adLockOptimistic
rs.AddNew
rs!Name = "Monty"
rs!txt = "Insert row"
rs.Update
rs.Close
'rs update
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-row"
rs.Update
rs.Close
'rs update second time..
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-second-time"
rs.Update
rs.Close
'rs delete
rs.Open "SELECT * FROM my_ado"
rs.MoveNext
rs.MoveNext
rs.Delete
rs.Close
'fetch the updated table ..
rs.Open "SELECT * FROM my_ado", conn
```

```

Debug.Print rs.RecordCount
rs.MoveFirst
Debug.Print String(50, "-") & "Updated my_ado Result Set " & String(50, "-")
For Each fld In rs.Fields
Debug.Print fld.Name,
Next
Debug.Print
Do Until rs.EOF
For Each fld In rs.Fields
Debug.Print fld.Value,
Next
rs.MoveNext
Debug.Print
Loop
rs.Close
conn.Close
End Sub

```

DAO: **rs.addNew**, **rs.update**, and Scrolling

The following DAO (Data Access Objects) example creates a table **my_dao** and demonstrates the use of **rs.addNew**, **rs.update**, and result set scrolling.

```

Private Sub myodbc_dao_Click()
Dim ws As Workspace
Dim conn As Connection
Dim queryDef As queryDef
Dim str As String
'connect to MySQL using MySQL ODBC 3.51 Driver
Set ws = DBEngine.CreateWorkspace("", "venu", "venu", dbUseODBC)
str = "odbc;DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"
Set conn = ws.OpenConnection("test", dbDriverNoPrompt, False, str)
'Create table my_dao
Set queryDef = conn.CreateQueryDef("", "drop table if exists my_dao")
queryDef.Execute
Set queryDef = conn.CreateQueryDef("", "create table my_dao(Id INT AUTO_INCREMENT PRIMARY KEY, " _
& "Ts TIMESTAMP(14) NOT NULL, Name varchar(20), Id2 INT)")
queryDef.Execute
'Insert new records using rs.addNew
Set rs = conn.OpenRecordset("my_dao")
Dim i As Integer
For i = 10 To 15
rs.AddNew
rs!Name = "insert record" & i
rs!Id2 = i
rs.Update
Next i
rs.Close
'rs update..
Set rs = conn.OpenRecordset("my_dao")
rs.Edit
rs!Name = "updated-string"
rs.Update
rs.Close
'fetch the table back...
Set rs = conn.OpenRecordset("my_dao", dbOpenDynamic)
str = "Results:"
rs.MoveFirst
While Not rs.EOF
str = " " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print "DATA:" & str
rs.MoveNext
Wend
'rs Scrolling
rs.MoveFirst
str = " FIRST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str
rs.MoveLast

```



```

str = " LAST ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str
rs.MovePrevious
str = " LAST-1 ROW: " & rs!Id & " , " & rs!Name & " , " & rs!Ts & " , " & rs!Id2
Debug.Print str
'free all resources
rs.Close
queryDef.Close
conn.Close
ws.Close
End Sub

```

RDO: `rs.addNew` and `rs.update`

The following RDO (Remote Data Objects) example creates a table `my_rdo` and demonstrates the use of `rs.addNew` and `rs.update`.

```

Dim rs As rdoResultset
Dim cn As New rdoConnection
Dim cl As rdoColumn
Dim SQL As String
'cn.Connect = "DSN=test;"
cn.Connect = "DRIVER={MySQL ODBC 3.51 Driver};"_
& "SERVER=localhost;"_
& " DATABASE=test;"_
& "UID=venu;PWD=venu; OPTION=3"
cn.CursorDriver = rdUseOdbc
cn.EstablishConnection rdDriverPrompt
'drop table my_rdo
SQL = "drop table if exists my_rdo"
cn.Execute SQL, rdExecDirect
'create table my_rdo
SQL = "create table my_rdo(id int, name varchar(20))"
cn.Execute SQL, rdExecDirect
'insert - direct
SQL = "insert into my_rdo values (100,'venu')"
cn.Execute SQL, rdExecDirect
SQL = "insert into my_rdo values (200,'MySQL')"
cn.Execute SQL, rdExecDirect
'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 300
rs!Name = "Insert1"
rs.Update
rs.Close
'rs insert
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.AddNew
rs!id = 400
rs!Name = "Insert 2"
rs.Update
rs.Close
'rs update
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
rs.Edit
rs!id = 999
rs!Name = "updated"
rs.Update
rs.Close
'fetch back...
SQL = "select * from my_rdo"
Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
Do Until rs.EOF
For Each cl In rs.rdoColumns
Debug.Print cl.Value,
Next
rs.MoveNext

```

```

Debug.Print
Loop
Debug.Print "Row count="; rs.RowCount
'close
rs.Close
cn.Close
End Sub

```

3.6.7.2 Using Connector/ODBC with .NET

This section contains simple examples that demonstrate the use of Connector/ODBC drivers with ODBC.NET.

Using Connector/ODBC with ODBC.NET and C# (C sharp)

The following sample creates a table `my_odbc_net` and demonstrates its use in C#.

```

/**
 * @sample      : mycon.cs
 * @purpose     : Demo sample for ODBC.NET using Connector/ODBC
 * @author      : Venu, <myodbc@lists.mysql.com>
 *
 * (C) Copyright MySQL AB, 1995-2006
 */
/* build command
 *
 * csc /t:exe
 *      /out:mycon.exe mycon.cs
 *      /r:Microsoft.Data.Odbc.dll
 */
using Console = System.Console;
using Microsoft.Data.Odbc;
namespace myodbc3
{
    class mycon
    {
        static void Main(string[] args)
        {
            try
            {
                //Connection string for Connector/ODBC 3.51
                string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
                    "SERVER=localhost;" +
                    "DATABASE=test;" +
                    "UID=venu;" +
                    "PASSWORD=venu;" +
                    "OPTION=3";
                //Connect to MySQL using Connector/ODBC
                OdbcConnection MyConnection = new OdbcConnection(MyConString);
                MyConnection.Open();
                Console.WriteLine("\n !!! success, connected successfully !!!\n");
                //Display connection information
                Console.WriteLine("Connection Information:");
                Console.WriteLine("\tConnection String:" +
                    MyConnection.ConnectionString);
                Console.WriteLine("\tConnection Timeout:" +
                    MyConnection.ConnectionTimeout);
                Console.WriteLine("\tDatabase:" +
                    MyConnection.Database);
                Console.WriteLine("\tDataSource:" +
                    MyConnection.DataSource);
                Console.WriteLine("\tDriver:" +
                    MyConnection.Driver);
                Console.WriteLine("\tServerVersion:" +
                    MyConnection.ServerVersion);
                //Create a sample table
                OdbcCommand MyCommand =
                    new OdbcCommand("DROP TABLE IF EXISTS my_odbc_net",
                        MyConnection);
            }
            catch { }
        }
    }
}

```

```

MyCommand.ExecuteNonQuery();
MyCommand.CommandText =
    "CREATE TABLE my_odbc_net(id int, name varchar(20), idb bigint)";
MyCommand.ExecuteNonQuery();
//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(10,'venu', 300)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());
//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(20,'mysql',400)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());
//Insert
MyCommand.CommandText =
    "INSERT INTO my_odbc_net VALUES(20,'mysql',500)";
Console.WriteLine("INSERT, Total rows affected:" +
    MyCommand.ExecuteNonQuery());
//Update
MyCommand.CommandText =
    "UPDATE my_odbc_net SET id=999 WHERE id=20";
Console.WriteLine("Update, Total rows affected:" +
    MyCommand.ExecuteNonQuery());
//COUNT(*)
MyCommand.CommandText =
    "SELECT COUNT(*) as TRows FROM my_odbc_net";
Console.WriteLine("Total Rows:" +
    MyCommand.ExecuteScalar());
//Fetch
MyCommand.CommandText = "SELECT * FROM my_odbc_net";
OdbcDataReader MyDataReader;
MyDataReader = MyCommand.ExecuteReader();
while (MyDataReader.Read())
{
    if(string.Compare(MyConnection.Driver,"myodbc3.dll") == 0) {
        //Supported only by Connector/ODBC 3.51
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
            MyDataReader.GetString(1) + " " +
            MyDataReader.GetInt64(2));
    }
    else {
        //BIGINTs not supported by Connector/ODBC
        Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
            MyDataReader.GetString(1) + " " +
            MyDataReader.GetInt32(2));
    }
}
//Close all resources
MyDataReader.Close();
MyConnection.Close();
}
catch (OdbcException MyOdbcException) //Catch any ODBC exception ..
{
    for (int i=0; i < MyOdbcException.Errors.Count; i++)
    {
        Console.WriteLine("ERROR #" + i + "\n" +
            "Message: " +
            MyOdbcException.Errors[i].Message + "\n" +
            "Native: " +
            MyOdbcException.Errors[i].NativeError.ToString() + "\n" +
            "Source: " +
            MyOdbcException.Errors[i].Source + "\n" +
            "SQL: " +
            MyOdbcException.Errors[i].SQLState + "\n");
    }
}
}
}
}
}

```

Using Connector/ODBC with ODBC.NET and Visual Basic

The following sample creates a table `my_vb_net` and demonstrates the use in VB.

```
' @sample      : myvb.vb
' @purpose     : Demo sample for ODBC.NET using Connector/ODBC
' @author      : Venu, <myodbc@lists.mysql.com>
'
' (C) Copyright MySQL AB, 1995-2006
'
'
' build command
'
' vbc /target:exe
'     /out:myvb.exe
'     /r:Microsoft.Data.Odbc.dll
'     /r:System.dll
'     /r:System.Data.dll
Imports Microsoft.Data.Odbc
Imports System
Module myvb
  Sub Main()
    Try
      'Connector/ODBC 3.51 connection string
      Dim MyConString As String = "DRIVER={MySQL ODBC 3.51 Driver};" & _
        "SERVER=localhost;" & _
        "DATABASE=test;" & _
        "UID=venu;" & _
        "PASSWORD=venu;" & _
        "OPTION=3;"
      'Connection
      Dim MyConnection As New OdbcConnection(MyConString)
      MyConnection.Open()
      Console.WriteLine("Connection State: " & MyConnection.State.ToString)
      'Drop
      Console.WriteLine("Dropping table")
      Dim MyCommand As New OdbcCommand()
      MyCommand.Connection = MyConnection
      MyCommand.CommandText = "DROP TABLE IF EXISTS my_vb_net"
      MyCommand.ExecuteNonQuery()
      'Create
      Console.WriteLine("Creating...")
      MyCommand.CommandText = "CREATE TABLE my_vb_net(id int, name varchar(30))"
      MyCommand.ExecuteNonQuery()
      'Insert
      MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(10,'venu')"
      Console.WriteLine("INSERT, Total rows affected: " & _
        MyCommand.ExecuteNonQuery())
      'Insert
      MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
      Console.WriteLine("INSERT, Total rows affected: " & _
        MyCommand.ExecuteNonQuery())
      'Insert
      MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
      Console.WriteLine("INSERT, Total rows affected: " & _
        MyCommand.ExecuteNonQuery())
      'Insert
      MyCommand.CommandText = "INSERT INTO my_vb_net(id) VALUES(30)"
      Console.WriteLine("INSERT, Total rows affected: " & _
        MyCommand.ExecuteNonQuery())
      'Update
      MyCommand.CommandText = "UPDATE my_vb_net SET id=999 WHERE id=20"
      Console.WriteLine("Update, Total rows affected: " & _
        MyCommand.ExecuteNonQuery())
      'COUNT(*)
      MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_vb_net"
      Console.WriteLine("Total Rows: " & MyCommand.ExecuteScalar())
      'Select
      Console.WriteLine("Select * FROM my_vb_net")
    End Try
  End Sub
End Module
```

```

MyCommand.CommandText = "SELECT * FROM my_vb_net"
Dim MyDataReader As OdbcDataReader
MyDataReader = MyCommand.ExecuteReader
While MyDataReader.Read
    If MyDataReader("name") Is DBNull.Value Then
        Console.WriteLine("id = " & _
            CStr(MyDataReader("id")) & " name = " & _
            "NULL")
    Else
        Console.WriteLine("id = " & _
            CStr(MyDataReader("id")) & " name = " & _
            CStr(MyDataReader("name")))
    End If
End While
'Catch ODBC Exception
Catch MyOdbcException As OdbcException
    Dim i As Integer
    Console.WriteLine(MyOdbcException.ToString)
'Catch program exception
Catch MyException As Exception
    Console.WriteLine(MyException.ToString)
End Try
End Sub

```

3.7 Connector/ODBC Reference

This section provides reference material for the Connector/ODBC API, showing supported functions and methods, supported MySQL column types and the corresponding native type in Connector/ODBC, and the error codes returned by Connector/ODBC when a fault occurs.

3.7.1 Connector/ODBC API Reference

This section summarizes ODBC routines, categorized by functionality.

For the complete ODBC API reference, please refer to the ODBC Programmer's Reference at <http://msdn.microsoft.com/en-us/library/ms714177.aspx>.

An application can call [SQLGetInfo](#) function to obtain conformance information about Connector/ODBC. To obtain information about support for a specific function in the driver, an application can call [SQLGetFunctions](#).

Note

For backward compatibility, the Connector/ODBC driver supports all deprecated functions.

The following tables list Connector/ODBC API calls grouped by task:

Table 3.5 ODBC API Calls for Connecting to a Data Source

Function Name	Connector/ODBC Supports?	Standard	Purpose
SQLAllocHandle	Yes	ISO 92	Obtains an environment, connection, statement, or descriptor handle.
SQLConnect	Yes	ISO 92	Connects to a specific driver by data source name, user ID, and password.
SQLDriverConnect	Yes	ODBC	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
SQLAllocEnv	Yes	Deprecated	Obtains an environment handle allocated from driver.

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLAllocConnect	Yes	Deprecated	Obtains a connection handle

Table 3.6 ODBC API Calls for Obtaining Information about a Driver and Data Source

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLDataSources	No	ISO 92	Returns the list of available data sources, handled by the Driver Manager
SQLDrivers	No	ODBC	Returns the list of installed drivers and their attributes, handles by Driver Manager
SQLGetInfo	Yes	ISO 92	Returns information about a specific driver and data source.
SQLGetFunctions	Yes	ISO 92	Returns supported driver functions.
SQLGetTypeInfo	Yes	ISO 92	Returns information about supported data types.

Table 3.7 ODBC API Calls for Setting and Retrieving Driver Attributes

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLSetConnectAttr	Yes	ISO 92	Sets a connection attribute.
SQLGetConnectAttr	Yes	ISO 92	Returns the value of a connection attribute.
SQLSetConnectOption	Yes	Deprecated	Sets a connection option
SQLGetConnectOption	Yes	Deprecated	Returns the value of a connection option
SQLSetEnvAttr	Yes	ISO 92	Sets an environment attribute.
SQLGetEnvAttr	Yes	ISO 92	Returns the value of an environment attribute.
SQLSetStmtAttr	Yes	ISO 92	Sets a statement attribute.
SQLGetStmtAttr	Yes	ISO 92	Returns the value of a statement attribute.
SQLSetStmtOption	Yes	Deprecated	Sets a statement option
SQLGetStmtOption	Yes	Deprecated	Returns the value of a statement option

Table 3.8 ODBC API Calls for Preparing SQL Requests

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLAllocStmt	Yes	Deprecated	Allocates a statement handle
SQLPrepare	Yes	ISO 92	Prepares an SQL statement for later execution.
SQLBindParameter	Yes	ODBC	Assigns storage for a parameter in an SQL statement. Connector/ODBC 5.2 adds support for “out” and “inout” parameters, through the SQL_PARAM_OUTPUT or SQL_PARAM_INPUT_OUTPUT type specifiers. (“Out” and “inout” parameters are not supported for LONGTEXT and LONGBLOB columns.)
SQLGetCursorName	Yes	ISO 92	Returns the cursor name associated with a statement handle.

Function Name	Connector/ODBC Supports?	Standard	Purpose
SQLSetCursorName	Yes	ISO 92	Specifies a cursor name.
SQLSetScrollOptions	Yes	ODBC	Sets options that control cursor behavior.

Table 3.9 ODBC API Calls for Submitting Requests

Function Name	Connector/ODBC Supports?	Standard	Purpose
SQLExecute	Yes	ISO 92	Executes a prepared statement.
SQLExecDirect	Yes	ISO 92	Executes a statement
SQLNativeSql	Yes	ODBC	Returns the text of an SQL statement as translated by the driver.
SQLDescribeParam	No	ODBC	Returns the description for a specific parameter in a statement. Not supported by Connector/ODBC —the returned results should not be trusted.
SQLNumParams	Yes	ISO 92	Returns the number of parameters in a statement.
SQLParamData	Yes	ISO 92	Used in conjunction with SQLPutData to supply parameter data at execution time. (Useful for long data values.)
SQLPutData	Yes	ISO 92	Sends part or all of a data value for a parameter. (Useful for long data values.)

Table 3.10 ODBC API Calls for Retrieving Results and Information about Results

Function Name	Connector/ODBC Supports?	Standard	Purpose
SQLRowCount	Yes	ISO 92	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResultCols	Yes	ISO 92	Returns the number of columns in the result set.
SQLDescribeCol	Yes	ISO 92	Describes a column in the result set.
SQLColAttribute	Yes	ISO 92	Describes attributes of a column in the result set.
SQLColAttributes	Yes	Deprecated	Describes attributes of a column in the result set.
SQLFetch	Yes	ISO 92	Returns multiple result rows.
SQLFetchScroll	Yes	ISO 92	Returns scrollable result rows.
SQLExtendedFetch	Yes	Deprecated	Returns scrollable result rows.
SQLSetPos	Yes	ODBC	Positions a cursor within a fetched block of data and enables an application to refresh data in the rowset or to update or delete data in the result set.
SQLBulkOperations	Yes	ODBC	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.

Table 3.11 ODBC API Calls for Retrieving Error or Diagnostic Information

Function Name	Connector/ODBC Supports?	Standard	Purpose
SQLError	Yes	Deprecated	Returns additional error or status information

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLGetDiagField	Yes	ISO 92	Returns additional diagnostic information (a single field of the diagnostic data structure).
SQLGetDiagRec	Yes	ISO 92	Returns additional diagnostic information (multiple fields of the diagnostic data structure).

Table 3.12 ODBC API Calls for Obtaining Information about the Data Source's System Tables (Catalog Functions) Item

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLColumnPrivileges	Yes	ODBC	Returns a list of columns and associated privileges for one or more tables.
SQLColumns	Yes	X/Open	Returns the list of column names in specified tables.
SQLForeignKeys	Yes	ODBC	Returns a list of column names that make up foreign keys, if they exist for a specified table.
SQLPrimaryKeys	Yes	ODBC	Returns the list of column names that make up the primary key for a table.
SQLSpecialColumns	Yes	X/Open	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	Yes	ISO 92	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges	Yes	ODBC	Returns a list of tables and the privileges associated with each table.
SQLTables	Yes	X/Open	Returns the list of table names stored in a specific data source.

Table 3.13 ODBC API Calls for Performing Transactions

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLTransact	Yes	Deprecated	Commits or rolls back a transaction
SQLEndTran	Yes	ISO 92	Commits or rolls back a transaction .

Table 3.14 ODBC API Calls for Terminating a Statement

Function Name	Connector/ ODBC Supports?	Standard	Purpose
SQLFreeStmt	Yes	ISO 92	Ends statement processing, discards pending results, and, optionally, frees all resources associated with the statement handle.
SQLCloseCursor	Yes	ISO 92	Closes a cursor that has been opened on a statement handle.
SQLCancel	Yes	ISO 92	Cancels an SQL statement.

Table 3.15 ODBC API Calls for Terminating a Connection

Function Name	Connector/ODBC Supports?	Standard	Purpose
SQLDisconnect	Yes	ISO 92	Closes the connection.
SQLFreeHandle	Yes	ISO 92	Releases an environment, connection, statement, or descriptor handle.
SQLFreeConnect	Yes	Deprecated	Releases connection handle.
SQLFreeEnv	Yes	Deprecated	Releases an environment handle.

3.7.2 Connector/ODBC Data Types

The following table illustrates how Connector/ODBC maps the server data types to default SQL and C data types.

Table 3.16 How Connector/ODBC Maps MySQL Data Types to SQL and C Data Types

Native Value	SQL Type	C Type
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bit	SQL_BIT	SQL_C_BIT
bit	SQL_CHAR	SQL_C_CHAR
blob	SQL_LONGVARBINARY	SQL_C_BINARY
bool	SQL_CHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
date	SQL_DATE	SQL_C_DATE
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
decimal	SQL_DECIMAL	SQL_C_CHAR
double precision	SQL_DOUBLE	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
enum	SQL_VARCHAR	SQL_C_CHAR
float	SQL_REAL	SQL_C_FLOAT
int unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
integer unsigned	SQL_INTEGER	SQL_C_ULONG
integer	SQL_INTEGER	SQL_C_SLONG
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
longblob	SQL_LONGVARBINARY	SQL_C_BINARY
longtext	SQL_LONGVARCHAR	SQL_C_CHAR
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
mediumint	SQL_INTEGER	SQL_C_SLONG
mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR

Native Value	SQL Type	C Type
real	SQL_FLOAT	SQL_C_DOUBLE
set	SQL_VARCHAR	SQL_C_CHAR
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT
smallint	SQL_SMALLINT	SQL_C_SSHORT
text	SQL_LONGVARCHAR	SQL_C_CHAR
time	SQL_TIME	SQL_C_TIME
timestamp	SQL_TIMESTAMP	SQL_C_TIMESTAMP
tinyblob	SQL_LONGVARBINARY	SQL_C_BINARY
tinyint unsigned	SQL_TINYINT	SQL_C_UTINYINT
tinyint	SQL_TINYINT	SQL_C_STINYINT
tinytext	SQL_LONGVARCHAR	SQL_C_CHAR
varchar	SQL_VARCHAR	SQL_C_CHAR
year	SQL_SMALLINT	SQL_C_SHORT

3.7.3 Connector/ODBC Error Codes

The following tables lists the error codes returned by Connector/ODBC apart from the server errors.

Table 3.17 Special Error Codes Returned by Connector/ODBC

Native Code	SQLSTATE 2	SQLSTATE 3	Error Message
500	01000	01000	General warning
501	01004	01004	String data, right truncated
502	01S02	01S02	Option value changed
503	01S03	01S03	No rows updated/deleted
504	01S04	01S04	More than one row updated/deleted
505	01S06	01S06	Attempt to fetch before the result set returned the first row set
506	07001	07002	SQLBindParameter not used for all parameters
507	07005	07005	Prepared statement not a cursor-specification
508	07009	07009	Invalid descriptor index
509	08002	08002	Connection name in use
510	08003	08003	Connection does not exist
511	24000	24000	Invalid cursor state
512	25000	25000	Invalid transaction state
513	25S01	25S01	Transaction state unknown
514	34000	34000	Invalid cursor name
515	S1000	HY000	General driver defined error
516	S1001	HY001	Memory allocation error
517	S1002	HY002	Invalid column number
518	S1003	HY003	Invalid application buffer type
519	S1004	HY004	Invalid SQL data type
520	S1009	HY009	Invalid use of null pointer

Native Code	SQLSTATE 2	SQLSTATE 3	Error Message
521	S1010	HY010	Function sequence error
522	S1011	HY011	Attribute can not be set now
523	S1012	HY012	Invalid transaction operation code
524	S1013	HY013	Memory management error
525	S1015	HY015	No cursor name available
526	S1024	HY024	Invalid attribute value
527	S1090	HY090	Invalid string or buffer length
528	S1091	HY091	Invalid descriptor field identifier
529	S1092	HY092	Invalid attribute/option identifier
530	S1093	HY093	Invalid parameter number
531	S1095	HY095	Function type out of range
532	S1106	HY106	Fetch type out of range
533	S1117	HY117	Row value out of range
534	S1109	HY109	Invalid cursor position
535	S1C00	HYC00	Optional feature not implemented
0	21S01	21S01	Column count does not match value count
0	23000	23000	Integrity constraint violation
0	42000	42000	Syntax error or access violation
0	42S02	42S02	Base table or view not found
0	42S12	42S12	Index not found
0	42S21	42S21	Column already exists
0	42S22	42S22	Column not found
0	08S01	08S01	Communication link failure

3.8 Connector/ODBC Notes and Tips

Here are some common notes and tips for using Connector/ODBC within different environments, applications and tools. The notes provided here are based on the experiences of Connector/ODBC developers and users.

3.8.1 Connector/ODBC General Functionality

This section provides help with common queries and areas of functionality in MySQL and how to use them with Connector/ODBC.

3.8.1.1 Obtaining Auto-Increment Values

Obtaining the value of column that uses **AUTO_INCREMENT** after an **INSERT** statement can be achieved in a number of different ways. To obtain the value immediately after an **INSERT**, use a **SELECT** query with the **LAST_INSERT_ID()** function.

For example, using Connector/ODBC you would execute two separate statements, the **INSERT** statement and the **SELECT** query to obtain the auto-increment value.

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

If you do not require the value within your application, but do require the value as part of another **INSERT**, the entire process can be handled by executing the following statements:

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
INSERT INTO tbl2 (id,text) VALUES(LAST_INSERT_ID(), 'text');
```

Certain ODBC applications (including Delphi and Access) may have trouble obtaining the auto-increment value using the previous examples. In this case, try the following statement as an alternative:

```
SELECT * FROM tbl WHERE auto IS NULL;
```

This alternative method requires that `sql_auto_is_null` variable is not set to 0. See [Server System Variables](#).

See also [How to Get the Unique ID for the Last Inserted Row](#).

3.8.1.2 Dynamic Cursor Support

Support for the **dynamic cursor** is provided in Connector/ODBC 3.51, but dynamic cursors are not enabled by default. You can enable this function within Windows by selecting the **Enable Dynamic Cursor** check box within the ODBC Data Source Administrator.

On other platforms, you can enable the dynamic cursor by adding **32** to the **OPTION** value when creating the DSN.

3.8.1.3 Connector/ODBC Performance

The Connector/ODBC driver has been optimized to provide very fast performance. If you experience problems with the performance of Connector/ODBC, or notice a large amount of disk activity for simple queries, there are a number of aspects to check:

- Ensure that **ODBC Tracing** is not enabled. With tracing enabled, a lot of information is recorded in the tracing file by the ODBC Manager. You can check, and disable, tracing within Windows using the **Tracing** panel of the ODBC Data Source Administrator. Within OS X, check the **Tracing** panel of ODBC Administrator. See [Section 3.5.8, “Getting an ODBC Trace File”](#).
- Make sure you are using the standard version of the driver, and not the debug version. The debug version includes additional checks and reporting measures.
- Disable the Connector/ODBC driver trace and query logs. These options are enabled for each DSN, so make sure to examine only the DSN that you are using in your application. Within Windows, you can disable the Connector/ODBC and query logs by modifying the DSN configuration. Within OS X and Unix, ensure that the driver trace (option value 4) and query logging (option value 524288) are not enabled.

3.8.1.4 Setting ODBC Query Timeout in Windows

For more information on how to set the query timeout on Microsoft Windows when executing queries through an ODBC connection, read the Microsoft knowledgebase document at <http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B153756>.

3.8.2 Connector/ODBC Application-Specific Tips

Most programs should work with Connector/ODBC, but for each of those listed here, there are specific notes and tips to improve or enhance the way you work with Connector/ODBC and these applications.

With all applications, ensure that you are using the latest Connector/ODBC drivers, ODBC Manager and any supporting libraries and interfaces used by your application. For example, on Windows, using the latest version of Microsoft Data Access Components (MDAC) will improve the compatibility with ODBC in general, and with the Connector/ODBC driver.

3.8.2.1 Using Connector/ODBC with Microsoft Applications

The majority of Microsoft applications have been tested with Connector/ODBC, including Microsoft Office, Microsoft Access and the various programming languages supported within ASP and Microsoft Visual Studio.

Microsoft Access

To improve the integration between Microsoft Access and MySQL through Connector/ODBC:

- For all versions of Access, enable the Connector/ODBC **Return matching rows** option. For Access 2.0, also enable the **Simulate ODBC 1.0** option.
- Include a **TIMESTAMP** column in all tables that you want to be able to update. For maximum portability, do not use a length specification in the column declaration (which is unsupported within MySQL in versions earlier than 4.1).
- Include a **primary key** in each MySQL table you want to use with Access. If not, new or updated rows may show up as **#DELETED#**.
- Use only **DOUBLE** float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as **#DELETED#** or that you cannot find or update rows.
- If you are using Connector/ODBC to link to a table that has a **BIGINT** column, the results are displayed as **#DELETED#**. The work around solution is:
 - Have one more dummy column with **TIMESTAMP** as the data type.
 - Select the **Change BIGINT columns to INT** option in the connection dialog in ODBC DSN Administrator.
 - Delete the table link from Access and re-create it.

Old records may still display as **#DELETED#**, but newly added/updated records are displayed properly.

- If you still get the error **Another user has changed your data** after adding a **TIMESTAMP** column, the following trick may help you:

Do not use a **table** data sheet view. Instead, create a form with the fields you want, and use that **form** data sheet view. Set the **DefaultValue** property for the **TIMESTAMP** column to **NOW()**. Consider hiding the **TIMESTAMP** column from view so your users are not confused.

- In some cases, Access may generate SQL statements that MySQL cannot understand. You can fix this by selecting **"Query|SQLSpecific|Pass-Through"** from the Access menu.
- On Windows NT, Access reports **BLOB** columns as **OLE OBJECTS**. If you want to have **MEMO** columns instead, change **BLOB** columns to **TEXT** with **ALTER TABLE**.
- Access cannot always handle the MySQL **DATE** column properly. If you have a problem with these, change the columns to **DATETIME**.
- If you have in Access a column defined as **BYTE**, Access tries to export this as **TINYINT** instead of **TINYINT UNSIGNED**. This gives you problems if you have values larger than 127 in the column.
- If you have very large (long) tables in Access, it might take a very long time to open them. Or you might run low on virtual memory and eventually get an **ODBC Query Failed** error and the table cannot open. To deal with this, select the following options:
 - Return Matching Rows (2)

- Allow BIG Results (8).

These add up to a value of 10 (`OPTION=10`).

Some external articles and tips that may be useful when using Access, ODBC and Connector/ODBC:

- Read [How to Trap ODBC Login Error Messages in Access](#)
- Optimizing Access ODBC Applications
 - [Optimizing for Client/Server Performance](#)
 - [Tips for Converting Applications to Using ODBCDirect](#)
 - [Tips for Optimizing Queries on Attached SQL Tables](#)
- For a list of tools that can be used with Access and ODBC data sources, refer to <http://www.mysql.com/portal/software/convertors/>.

Microsoft Excel and Column Types

If you have problems importing data into Microsoft Excel, particularly numeric, date, and time values, this is probably because of a bug in Excel, where the column type of the source data is used to determine the data type when that data is inserted into a cell within the worksheet. The result is that Excel incorrectly identifies the content and this affects both the display format and the data when it is used within calculations.

To address this issue, use the `CONCAT()` function in your queries. The use of `CONCAT()` forces Excel to treat the value as a string, which Excel will then parse and usually correctly identify the embedded information.

However, even with this option, some data may be incorrectly formatted, even though the source data remains unchanged. Use the [Format Cells](#) option within Excel to change the format of the displayed information.

Microsoft Visual Basic

To be able to update a table, you must define a [primary key](#) for the table.

Visual Basic with ADO cannot handle big integers. This means that some queries like `SHOW PROCESSLIST` do not work properly. The fix is to use `OPTION=16384` in the ODBC connect string or to select the [Change BIGINT columns to INT](#) option in the Connector/ODBC connect screen. You may also want to select the [Return matching rows](#) option.

Microsoft Visual InterDev

If you have a `BIGINT` in your result, you may get the error `[Microsoft][ODBC Driver Manager] Driver does not support this parameter`. Try selecting the [Change BIGINT columns to INT](#) option in the Connector/ODBC connect screen.

Visual Objects

Select the [Don't optimize column widths](#) option.

Microsoft ADO

When you are coding with the ADO API and Connector/ODBC, you need to pay attention to some default properties that aren't supported by the MySQL server. For example, using the [CursorLocation Property](#) as `adUseServer` returns a result of -1 for the [RecordCount Property](#). To have the right value, you need to set this property to `adUseClient`, as shown in the VB code here:

```
Dim myconn As New ADODB.Connection
Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long
myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount
myrs.Close
myconn.Close
```

Another workaround is to use a `SELECT COUNT(*)` statement for a similar query to get the correct row count.

To find the number of rows affected by a specific SQL statement in ADO, use the `RecordsAffected` property in the ADO execute method. For more information on the usage of execute method, refer to <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/htm/mdmthcnnextecute.asp>.

For information, see [ActiveX Data Objects\(ADO\) Frequently Asked Questions](#).

Using Connector/ODBC with Active Server Pages (ASP)

Select the `Return matching rows` option in the DSN.

For more information about how to access MySQL through ASP using Connector/ODBC, refer to the following articles:

- [Using MyODBC To Access Your MySQL Database Via ASP](#)
- [ASP and MySQL at DWAM.NT](#)

A Frequently Asked Questions list for ASP can be found at <http://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp>.

Using Connector/ODBC with Visual Basic (ADO, DAO and RDO) and ASP

Some articles that may help with Visual Basic and ASP:

- [MySQL BLOB columns and Visual Basic 6](#) by Mike Hillyer (<mike@openwin.org>).
- [How to map Visual basic data type to MySQL types](#) by Mike Hillyer (<mike@openwin.org>).

3.8.2.2 Using Connector/ODBC with Borland Applications

With all Borland applications where the Borland Database Engine (BDE) is used, follow these steps to improve compatibility:

- Update to BDE 3.2 or newer.
- Enable the `Don't optimize column widths` option in the DSN.
- Enabled the `Return matching rows` option in the DSN.

Using Connector/ODBC with Borland Builder 4

When you start a query, you can use the `Active` property or the `Open` method.

The `Active` property starts by automatically issuing a `SELECT * FROM ...` query. That may affect performance for large tables.

Using Connector/ODBC with Delphi

Also, here is some potentially useful Delphi code that sets up both an ODBC entry and a BDE entry for Connector/ODBC. The BDE entry requires a BDE Alias Editor that is free at a Delphi Super Page near you. (Thanks to Bryan Brunton <bryan@flesherfab.com> for this):

```
fReg:= TRegistry.Create;
fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
fReg.WriteString('Database', 'Documents');
fReg.WriteString('Description', ' ');
fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
fReg.WriteString('Flag', '1');
fReg.WriteString('Password', '');
fReg.WriteString('Port', ' ');
fReg.WriteString('Server', 'xmark');
fReg.WriteString('User', 'winuser');
fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
fReg.WriteString('DocumentsFab', 'MySQL');
fReg.CloseKey;
fReg.Free;
Memo1.Lines.Add('DATABASE NAME=');
Memo1.Lines.Add('USER NAME=');
Memo1.Lines.Add('ODBC DSN=DocumentsFab');
Memo1.Lines.Add('OPEN MODE=READ/WRITE');
Memo1.Lines.Add('BATCH COUNT=200');
Memo1.Lines.Add('LANGDRIVER=');
Memo1.Lines.Add('MAX ROWS=-1');
Memo1.Lines.Add('SCHEMA CACHE DIR=');
Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');
AliasEditor.Add('DocumentsFab', 'MySQL', Memo1.Lines);
```

Using Connector/ODBC with C++ Builder

Tested with BDE 3.0. The only known problem is that when the table schema changes, query fields are not updated. BDE, however, does not seem to recognize primary keys, only the index named **PRIMARY**, although this has not been a problem.

3.8.2.3 Using Connector/ODBC with ColdFusion

The following information is taken from the ColdFusion documentation:

Use the following information to configure ColdFusion Server for Linux to use the [unixODBC](#) driver with Connector/ODBC for MySQL data sources. You can download Connector/ODBC at <http://dev.mysql.com/downloads/connector/odbc/>.

ColdFusion version 4.5.1 lets you use the ColdFusion Administrator to add the MySQL data source. However, the driver is not included with ColdFusion version 4.5.1. Before the MySQL driver appears in the ODBC data sources drop-down list, build and copy the Connector/ODBC driver to [/opt/coldfusion/lib/libmyodbc.so](#).

The Contrib directory contains the program [mydsn-xxx.zip](#) which lets you build and remove the DSN registry file for the Connector/ODBC driver on ColdFusion applications.

For more information and guides on using ColdFusion and Connector/ODBC, see the following external sites:

- [Troubleshooting Data Sources and Database Connectivity for Unix Platforms](#).

3.8.2.4 Using Connector/ODBC with OpenOffice.org

Open Office (<http://www.openoffice.org>) [How-to: MySQL + OpenOffice](#). [How-to: OpenOffice + MyODBC + unixODBC](#).

3.8.2.5 Using Connector/ODBC with Sambar Server

Sambar Server (<http://www.sambarserver.info>) [How-to: MyODBC + SambarServer + MySQL](#).

3.8.2.6 Using Connector/ODBC with Pervasive Software DataJunction

You have to change it to output `VARCHAR` rather than `ENUM`, as it exports the latter in a manner that causes MySQL problems.

3.8.2.7 Using Connector/ODBC with SunSystems Vision

Select the `Return matching rows` option.

3.8.3 Connector/ODBC Errors and Resolutions (FAQ)

The following section details some common errors and their suggested fix or alternative solution. If you are still experiencing problems, use the Connector/ODBC mailing list; see [Section 3.9.1, "Connector/ODBC Community Support"](#).

Many problems can be resolved by upgrading your Connector/ODBC drivers to the latest available release. On Windows, make sure that you have the latest versions of the Microsoft Data Access Components (MDAC) installed.

64-Bit Windows and ODBC Data Source Administrator

I have installed Connector/ODBC on Windows XP x64 Edition or Windows Server 2003 R2 x64. The installation completed successfully, but the Connector/ODBC driver does not appear in [ODBC Data Source Administrator](#).

This is not a bug, but is related to the way Windows x64 editions operate with the ODBC driver. On Windows x64 editions, the Connector/ODBC driver is installed in the `%SystemRoot%\SysWOW64` folder. However, the default [ODBC Data Source Administrator](#) that is available through the [Administrative Tools](#) or [Control Panel](#) in Windows x64 Editions is located in the `%SystemRoot%\system32` folder, and only searches this folder for ODBC drivers.

On Windows x64 editions, use the ODBC administration tool located at `%SystemRoot%\SysWOW64\odbcad32.exe`, this will correctly locate the installed Connector/ODBC drivers and enable you to create a Connector/ODBC DSN.

This issue was originally reported as Bug #20301.

Error 10061 (Cannot connect to server)

When connecting or using the **Test** button in [ODBC Data Source Administrator](#) I get error 10061 (Cannot connect to server)

This error can be raised by a number of different issues, including server problems, network problems, and firewall and port blocking problems. For more information, see [Can't connect to \[local\] MySQL server](#).

"Transactions are not enabled" Error

The following error is reported when using transactions: `Transactions are not enabled`

This error indicates that you are trying to use [transactions](#) with a MySQL table that does not support transactions. Transactions are supported within MySQL when using the [InnoDB](#) database engine,

which is the default storage engine in MySQL 5.5 and higher. In versions of MySQL before MySQL 5.1, you may also use the [BDB](#) engine.

Check the following before continuing:

- Verify that your MySQL server supports a transactional database engine. Use [SHOW ENGINES](#) to obtain a list of the available engine types.
- Verify that the tables you are updating use a transactional database engine.
- Ensure that you have not enabled the [disable transactions](#) option in your DSN.

#DELETED# Records Reported by Access

Access reports records as [#DELETED#](#) when inserting or updating records in linked tables.

If the inserted or updated records are shown as [#DELETED#](#) in Access, then:

- If you are using Access 2000, get and install the newest (version 2.6 or higher) Microsoft MDAC ([Microsoft Data Access Components](#)) from <http://support.microsoft.com/kb/110093>. This fixes a bug in Access that when you export data to MySQL, the table and column names aren't specified.

Also, get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5), which can be found at <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114>. This fixes some cases where columns are marked as [#DELETED#](#) in Access.

- For all versions of Access, enable the Connector/ODBC [Return matching rows](#) option. For Access 2.0, also enable the [Simulate ODBC 1.0](#) option.
- Include a [TIMESTAMP](#) in all tables that you want to be able to update.
- Include a [primary key](#) in the table. If not, new or updated rows may show up as [#DELETED#](#).
- Use only [DOUBLE](#) float fields. Access fails when comparing with single-precision floats. The symptom usually is that new or updated rows may show up as [#DELETED#](#) or that you cannot find or update rows.
- If you are using Connector/ODBC to link to a table that has a [BIGINT](#) column, the results are displayed as [#DELETED](#). The work around solution is:
 - Have one more dummy column with [TIMESTAMP](#) as the data type.
 - Select the [Change BIGINT columns to INT](#) option in the connection dialog in ODBC DSN Administrator.
 - Delete the table link from Access and re-create it.

Old records still display as [#DELETED#](#), but newly added/updated records are displayed properly.

Write Conflicts or Row Location Errors

How do I handle Write Conflicts or Row Location errors?

If you see the following errors, select the [Return Matching Rows](#) option in the DSN configuration dialog, or specify [OPTION=2](#), as the connection parameter:

```
Write Conflict. Another user has changed your data.
Row cannot be located for updating. Some values may have been changed
since it was last read.
```

Importing from Access 97

Exporting data from Access 97 to MySQL reports a [Syntax Error](#).

This error is specific to Access 97 and versions of Connector/ODBC earlier than 3.51.02. Update to the latest version of the Connector/ODBC driver to resolve this problem.

Importing from Microsoft DTS

Exporting data from Microsoft DTS to MySQL reports a [Syntax Error](#).

This error occurs only with MySQL tables using the [TEXT](#) or [VARCHAR](#) data types. You can fix this error by upgrading your Connector/ODBC driver to version 3.51.02 or higher.

SQL_NO_DATA Exception from ODBC.NET

Using ODBC.NET with Connector/ODBC, while fetching empty string (0 length), it starts giving the [SQL_NO_DATA](#) exception.

You can get the patch that addresses this problem from <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q319243>.

Error with SELECT COUNT(*)

Using [SELECT COUNT\(*\) FROM tbl_name](#) within Visual Basic and ASP returns an error.

This error occurs because the [COUNT\(*\)](#) expression is returning a [BIGINT](#), and ADO cannot make sense of a number this big. Select the [Change BIGINT columns to INT](#) option (option value 16384).

Multiple-Step Operation Error

Using the [AppendChunk\(\)](#) or [GetChunk\(\)](#) ADO methods, the [Multiple-step operation generated errors. Check each status value](#) error is returned.

The [GetChunk\(\)](#) and [AppendChunk\(\)](#) methods from ADO do not work as expected when the cursor location is specified as [adUseServer](#). On the other hand, you can overcome this error by using [adUseClient](#).

A simple example can be found from http://www.dwam.net/iishelp/ado/docs/adomth02_4.htm

Modified Record Error

Access returns [Another user had modified the record that you have modified](#) while editing records on a Linked Table.

In most cases, this can be solved by doing one of the following things:

- Add a [primary key](#) for the table if one doesn't exist.
- Add a timestamp column if one doesn't exist.
- Only use double-precision float fields. Some programs may fail when they compare single-precision floats.

If these strategies do not help, start by making a log file from the ODBC manager (the log you get when requesting logs from ODBCADMIN) and a Connector/ODBC log to help you figure out why things go wrong. For instructions, see [Section 3.5.8, "Getting an ODBC Trace File"](#).

Direct Application Linking Under Unix or Linux

When linking an application directly to the Connector/ODBC library under Unix or Linux, the application crashes.

Connector/ODBC under Unix or Linux is not compatible with direct application linking. To connect to an ODBC source, use a driver manager, such as [iODBC](#) or [unixODBC](#).

Microsoft Office and DATE or TIMESTAMP Columns

Applications in the Microsoft Office suite cannot update tables that have [DATE](#) or [TIMESTAMP](#) columns.

This is a known issue with Connector/ODBC. Ensure that the field has a default value (rather than [NULL](#)) and that the default value is nonzero (that is, something other than [0000-00-00 00:00:00](#)).

INFORMATION_SCHEMA Database

When connecting Connector/ODBC 5.x to a MySQL 4.x server, the error [1044 Access denied for user 'xxx'@'%' to database 'information_schema'](#) is returned.

Connector/ODBC 5.x is designed to work with MySQL 5.0 or later, taking advantage of the [INFORMATION_SCHEMA](#) database to determine data definition information. Support for MySQL 4.1 is planned for the final release.

S1T00 Error

When calling [SQLTables](#), the error [S1T00](#) is returned, but I cannot find this in the list of error numbers for Connector/ODBC.

The [S1T00](#) error indicates that a general timeout has occurred within the ODBC system and is not a MySQL error. Typically it indicates that the connection you are using is stale, the server is too busy to accept your request or that the server has gone away.

"Table does not exist" Error in Access 2000

When linking to tables in Access 2000 and generating links to tables programmatically, rather than through the table designer interface, you may get errors about tables not existing.

There is a known issue with a specific version of the [msjet40.dll](#) that exhibits this issue. The version affected is 4.0.9025.0. Reverting to an older version will enable you to create the links. If you have recently updated your version, check your [WINDOWS](#) directory for the older version of the file and copy it to the drivers directory.

Batched Statements

When I try to use batched statements, the execution of the batched statements fails.

Batched statement support was added in 3.51.18. Support for batched statements is not enabled by default. Enable option [FLAG_MULTI_STATEMENTS](#), value 67108864, or select the **Allow multiple statements** flag within a GUI configuration.

Packet Errors with ADODB and Excel

When connecting to a MySQL server using ADODB and Excel, occasionally the application fails to communicate with the server and the error [Got an error reading communication packets](#) appears in the error log.

This error may be related to Keyboard Logger 1.1 from PanteraSoft.com, which is known to interfere with the network communication between MySQL Connector/ODBC and MySQL.

Outer Join Error

When using some applications to access a MySQL server using Connector/ODBC and outer joins, an error is reported regarding the Outer Join Escape Sequence.

This is a known issue with MySQL Connector/ODBC which is not correctly parsing the "Outer Join Escape Sequence", as per the specs at [Microsoft ODBC Specs](#). Currently, Connector/ODBC will return a value > 0 when asked for [SQL_OJ_CAPABILITIES](#) even though no parsing takes place in the driver to handle the outer join escape sequence.

Hebrew/CJK Characters

I can correctly store extended characters in the database (Hebrew/CJK) using Connector/ODBC 5.1, but when I retrieve the data, the text is not formatted correctly and I get garbled characters.

When using ASP and UTF8 characters, add the following to your ASP files to ensure that the data returned is correctly encoded:

```
Response.CodePage = 65001
Response.CharSet = "utf-8"
```

Duplicate Entry in Installed Programs List

I have a duplicate MySQL Connector/ODBC entry within my **Installed Programs** list, but I cannot delete one of them.

This problem can occur when you upgrade an existing Connector/ODBC installation, rather than removing and then installing the updated version.

Warning

To fix the problem, use any working uninstallers to remove existing installations; then may have to edit the contents of the registry. Make sure you have a backup of your registry information before attempting any editing of the registry contents.

Values Truncated to 255 Characters

When submitting queries with parameter binding using [UPDATE](#), my field values are being truncated to 255 characters.

Ensure that the [FLAG_BIG_PACKETS](#) option is set for your connection. This removes the 255 character limitation on bound parameters.

Disabling Data-At-Execution

Is it possible to disable data-at-execution using a flag?

If you do not want to use data-at-execution, remove the corresponding calls. For example:

```
SQLLEN ylen = SQL_LEN_DATA_AT_EXEC(10);
SQLBindCol(hstmt, 2, SQL_C_BINARY, buf, 10, &ylen);
```

Would become:

```
SQLBindCol(hstmt, 2, SQL_C_BINARY, buf, 10, NULL);
```

This example also replaced &ylen with NULL in the call to [SQLBindCol\(\)](#).

For further information, refer to the [MSDN documentation](#) for [SQLBindCol\(\)](#).

NULLABLE Attribute for AUTO_INCREMENT Columns

When you call [SQLColumns\(\)](#) for a table column that is [AUTO_INCREMENT](#), the [NULLABLE](#) column of the result set is always [SQL_NULLABLE \(1\)](#).

This is because MySQL reports the [DEFAULT](#) value for such a column as [NULL](#). It means, if you insert a [NULL](#) value into the column, you will get the next integer value for the table's [auto_increment](#) counter.

3.9 Connector/ODBC Support

There are many different places where you can get support for using Connector/ODBC. Always try the Connector/ODBC Mailing List or Connector/ODBC Forum. See [Section 3.9.1, “Connector/ODBC Community Support”](#), for help before reporting a specific bug or issue to MySQL.

3.9.1 Connector/ODBC Community Support

Oracle provides assistance to the user community by means of its mailing lists. For Connector/ODBC-related issues, you can get help from experienced users by using the [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com) mailing list. Archives are available online at <http://lists.mysql.com/myodbc>.

For information about subscribing to MySQL mailing lists or to browse list archives, visit <http://lists.mysql.com/>. See [MySQL Mailing Lists](#).

Community support from experienced users is also available through the [ODBC Forum](#). You may also find help from other users in the other MySQL Forums, located at <http://forums.mysql.com>. See [MySQL Community Support at the MySQL Forums](#).

3.9.2 How to Report Connector/ODBC Problems or Bugs

If you encounter difficulties or problems with Connector/ODBC, start by making a log file from the [ODBC Manager](#) (the log you get when requesting logs from [ODBC ADMIN](#)) and Connector/ODBC. The procedure for doing this is described in [Section 3.5.8, “Getting an ODBC Trace File”](#).

Check the Connector/ODBC trace file to find out what could be wrong. Determine what statements were issued by searching for the string `>mysql_real_query` in the `myodbc.log` file.

Also, try issuing the statements from the `mysql` client program or from `admdemo`. This helps you determine whether the error is in Connector/ODBC or MySQL.

If you find out something is wrong, please only send the relevant rows (maximum 40 rows) to the `myodbc` mailing list. See [MySQL Mailing Lists](#). Please never send the whole Connector/ODBC or ODBC log file!

Ideally, include the following information with the email:

- Operating system and version
- Connector/ODBC version
- ODBC Driver Manager type and version
- MySQL server version
- ODBC trace from Driver Manager
- Connector/ODBC log file from Connector/ODBC driver
- Simple reproducible sample

Remember that the more information you can supply to us, the more likely it is that we can fix the problem!

Also, before posting the bug, check the MyODBC mailing list archive at <http://lists.mysql.com/myodbc>.

If you are unable to find out what is wrong, the last option is to create an archive in `tar` or `zip` format that contains a Connector/ODBC trace file, the ODBC log file, and a `README` file that explains the problem. Initiate a bug report for our bugs database at <http://bugs.mysql.com/>, then click the Files tab in the bug report for instructions on uploading the archive to the bugs database. Only MySQL engineers have access to the files you upload, and we are very discreet with the data.

If you can create a program that also demonstrates the problem, please include it in the archive as well.

If the program works with another SQL server, include an ODBC log file where you perform exactly the same SQL statements so that we can compare the results between the two systems.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

3.9.3 How to Submit a Connector/ODBC Patch

You can send a patch or suggest a better solution for any existing code or problems by sending a mail message to [<myodbc@lists.mysql.com>](mailto:myodbc@lists.mysql.com).

Chapter 4 MySQL Connector/Net Developer Guide

Table of Contents

4.1 Introduction to MySQL Connector/Net	191
4.2 Connector/Net Versions	192
4.3 Connector/Net Installation	195
4.3.1 Installing Connector/Net on Windows	195
4.3.2 Installing Connector/Net on Unix with Mono	198
4.3.3 Installing Connector/Net from the Source Code	199
4.4 Connector/Net Tutorials	200
4.4.1 Tutorial: An Introduction to Connector/Net Programming	200
4.4.2 Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider	209
4.4.3 Tutorial: MySQL Connector/Net ASP.NET Profile Provider	214
4.4.4 Tutorial: Web Parts Personalization Provider	216
4.4.5 Tutorial: Simple Membership Web Provider	220
4.4.6 Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source	226
4.4.7 Tutorial: Databinding in ASP.NET Using LINQ on Entities	238
4.4.8 Tutorial: Using SSL with MySQL Connector/Net	242
4.4.9 Tutorial: Using MySqlScript	244
4.4.10 Tutorial: Generating MySQL DDL from an Entity Framework Model	248
4.5 Connector/Net Programming	248
4.5.1 Connecting to MySQL Using Connector/Net	249
4.5.2 Creating a Connector/Net Connection String	249
4.5.3 Using MySqlCommand	253
4.5.4 Using Connector/Net with Connection Pooling	254
4.5.5 Using the Windows Native Authentication Plugin	255
4.5.6 Writing a Custom Authentication Plugin	255
4.5.7 MySQL Fabric Support	259
4.5.8 Using Connector/Net with Table Caching	264
4.5.9 Using the Connector/Net with Prepared Statements	264
4.5.10 Accessing Stored Procedures with Connector/Net	265
4.5.11 Handling BLOB Data With Connector/Net	268
4.5.12 Asynchronous methods	271
4.5.13 Using the Connector/Net Interceptor Classes	277
4.5.14 Handling Date and Time Information in Connector/Net	279
4.5.15 Using the MySqlBulkLoader Class	280
4.5.16 Using the MySQL Connector/Net Trace Source Object	281
4.5.17 Binary/Nonbinary Issues	286
4.5.18 Character Set Considerations for Connector/Net	287
4.5.19 Using Connector/Net with Crystal Reports	287
4.5.20 ASP.NET Provider Model	291
4.5.21 Working with Partial Trust / Medium Trust	293
4.6 Connector/Net Connection String Options Reference	296
4.7 Connector/Net Support for Windows Store	303
4.8 EF5 Support	304
4.9 EF6 Support	307
4.10 Connector/Net API Reference	312
4.10.1 MySql.Data.MySqlClient Namespace	312
4.10.2 MySql.Data.Types Namespace	410
4.11 Connector/Net Support	453
4.11.1 Connector/Net Community Support	453
4.11.2 How to Report Connector/Net Problems or Bugs	453

4.1 Introduction to MySQL Connector/Net

Connector/Net lets you easily develop .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET-aware tools. Developers can build applications using their choice of .NET languages. Connector/Net is a fully managed ADO.NET driver written in 100% pure C#. It does not use the MySQL C client library.

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

Connector/Net includes full support for:

- Features provided by MySQL Server up to and including MySQL Server 5.7.
- Large-packet support for sending and receiving rows and **BLOB** values up to 2 gigabytes in size.
- Protocol compression, which enables compressing the data stream between the client and server.
- Connections using TCP/IP sockets, named pipes, or shared memory on Windows.
- Connections using TCP/IP sockets or Unix sockets on Unix.
- The Open Source Mono framework developed by Novell.
- Microsoft Entity Framework.
- Microsoft Windows RT.

This document is intended as a user's guide to Connector/Net and includes a full syntax reference. Syntax information is also included within the `MySQL.Data.chm` file included with the Connector/Net distribution.

If you are using MySQL 5.0 or later, and Visual Studio as your development environment, you can also use the MySQL Visual Studio Plugin. The plugin acts as a DDEX (Data Designer Extensibility) provider: you can use the data design tools within Visual Studio to manipulate the schema and objects within a MySQL database. For more information, see [MySQL for Visual Studio](#).

Note

From Connector/Net 5.1 through 6.6, the Visual Studio Plugin is part of the main Connector/Net package. Starting with 6.7, the Visual Studio Plugin has been separated out into its own product. The MySQL for Visual Studio Plugin release notes can be found at [MySQL Connector/Net Release Notes](#).

MySQL Connector/Net supports full versions of Visual Studio 2008, 2010, 2012, 2013, and 2015, although the extent of support may be limited depending on your versions of MySQL Connector/Net and Visual Studio. For details, see [MySQL for Visual Studio](#).

MySQL Connector/Net does not support Express versions of Microsoft products, including Microsoft Visual Web Developer.

Key Topics

- For connection string properties when using the `MySqlConnection` class, see [Section 4.6](#), “Connector/Net Connection String Options Reference”.

4.2 Connector/Net Versions

There are several versions of Connector/Net available:

- **Connector/Net 6.9** includes new features such as a MySQL Personalization provider, SiteMap Web provider, a simple Membership Web provider, and support for MySQL for Visual Studio 1.2.
- **Connector/Net 6.8** includes new features such as Entity Framework 6 support, added idempotent script for Entity Framework 6 migrations, changed EF migration history table to use a

single column as primary key, removed installer validation when MySQL for Visual Studio is installed, and support for MySQL for Visual Studio 1.1.

- [Connector/Net 6.7](#) includes new features such as Entity Framework 5 support, built-in Load Balancing (to be used with a backend implementing either MySQL Replication or MySQL Clustering), a Memcached client (compatible with Innodb Memcached plugin) and support for Windows Runtime (WinRT) to write store apps. This version also removes all features related to Visual Studio Integration, which are provided in a separate product, [MySQL for Visual Studio](#).
- [Connector/Net 6.6](#) includes new features such as stored procedure debugging in Microsoft Visual Studio, support for pluggable authentication including the ability to write your own authentication plugins, Entity Framework 4.3 Code First support, and enhancements to partial trust support to allow hosting services to deploy applications without installing the Connector/Net library in the GAC.
- [Connector/Net 6.5](#) includes new features such as interceptor classes for exceptions and commands, support for the MySQL 5.6+ fractional seconds feature, better partial-trust support, and better IntelliSense, including auto-completion when editing stored procedures or `.mysql` files.
- [Connector/Net 6.4](#) includes new features such as support for Windows authentication (when connecting to MySQL Server 5.5+), table caching on the client side, simple connection fail-over support, and improved SQL generation from the Entity Framework provider.

This version of Connector/Net is no longer supported.

- [Connector/Net 6.3](#) includes new features such as integration with Visual Studio 2010, such as the availability of DDL T4 template for Entity Framework, and a custom MySQL SQL Editor. Other features include refactored transaction scope: Connector/Net now supports nested transactions in a scope where they use the same connection string.

This version of Connector/Net is no longer supported.

- [Connector/Net 6.2](#) includes new features such as a new logging system and client SSL certificates.

This version of Connector/Net is no longer supported.

- [Connector/Net 6.1](#) includes new features such as the MySQL Website Configuration Tool, and a Session State Provider.

This version of Connector/Net is no longer supported.

- [Connector/Net 6.0](#) includes support for UDF schema collection, Initial Entity Framework, and use of the traditional SQL Server buttons in Visual Studio for keys, indexes, and so on.

This version of Connector/Net is no longer supported.

- [Connector/Net 5.2](#) includes support for a new membership/role provider, Compact Framework 2.0, a new stored procedure parser and improvements to [GetSchema](#). Connector/Net 5.2 also includes the Visual Studio Plugin as a standard installable component.

This version of Connector/Net is no longer supported.

- [Connector/Net 5.1](#) includes support for a new membership/role provider, Compact Framework 2.0, a new stored procedure parser and improvements to [GetSchema](#). Connector/Net 5.1 also includes the Visual Studio Plugin as a standard installable component.

This version of Connector/Net is no longer supported.

- [Connector/Net 5.0](#) includes full support for the ADO.Net 2.0 interfaces and subclasses, includes support for the usage advisor and performance monitor (PerfMon) hooks.

This version of Connector/Net is no longer supported.

- **Connector/Net 1.0** includes full compatibility with the ADO.NET driver interface.

This version of Connector/Net is no longer supported.

The following table shows the .NET Framework version required, and the MySQL Server version supported by Connector/Net:

Table 4.1 Connector/Net Requirements for Related Products

Connector/Net version	ADO.NET version supported	.NET Framework version required	MySQL Server version supported	Currently supported
6.9	2.x+	2.x+ for VS 2008, 4.x+ for VS 2010 / 2012 / 2013, .NET RT for VS 2012 / 2013	5.7, 5.6, 5.5, 5.1, 5.0	Yes
6.8	2.x+	2.x+ for VS 2008, 4.x+ for VS 2010 / 2012 / 2013, .NET RT for VS 2012/2013	5.7, 5.6, 5.5, 5.1, 5.0	Yes
6.7	2.x+	2.x+ for VS 2008, 4.x+ for VS 2010 / 2012 / 2013, .NET RT for VS 2012/2013	5.7, 5.6, 5.5, 5.1, 5.0	Yes
6.6	2.x+	2.x+ for VS 2008, 4.x+ for VS 2010 / 2012 / 2013	5.7, 5.6, 5.5, 5.1, 5.0	Yes
6.5	2.x+	2.x+ for VS 2008, 4.x+ for VS 2010	5.7, 5.6, 5.5, 5.1, 5.0	No
6.4	2.x+	2.x+, 4.x+ for VS 2010	5.6, 5.5, 5.1, 5.0	No
6.3	2.x+	2.x+, 4.x+ for VS 2010	5.6, 5.5, 5.1, 5.0	No
6.2	2.x+	2.x+	5.6, 5.5, 5.1, 5.0, 4.1	No
6.1	2.x+	2.x+	5.6, 5.5, 5.1, 5.0, 4.1	No
6.0	2.x+	2.x+	5.5, 5.1, 5.0, 4.1	No
5.2	2.x+	2.x+	5.5, 5.1, 5.0, 4.1	No
5.1	2.x+	2.x+	5.5, 5.1, 5.0, 4.1, 4.0	No
5.0	2.x+	2.x+	5.0, 4.1, 4.0	No
1.0	1.x	1.x	5.0, 4.1, 4.0	No

Note

Version numbers for MySQL products are formatted as X.Y.Z, where Z=0 indicates alpha, Z=1 indicates beta, and Z>=2 indicates GA. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.YY.ZZ. For example, the official MySQL formatted version number 5.0.9

may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display format is different.

4.3 Connector/Net Installation

MySQL Connector/Net runs on any platform that supports the .NET Framework. The .NET Framework is primarily supported on recent versions of Microsoft Windows, and is supported on Linux through the [Open Source Mono platform](#).

Connector/Net is available for download from the [MySQL Installer](#), as a [standalone .msi](#), or from the [MySQL NuGet repository](#).

4.3.1 Installing Connector/Net on Windows

On Microsoft Windows, you can install either through a binary installation process using a Connector/Net MSI, choose the MySQL Connector/Net product from the MySQL Installer, using Nuget, or by downloading and using the Source.

Before installing, ensure that your system is up to date, including installing the latest version of the .NET Framework.

4.3.1.1 Installing Connector/Net Using MySQL Installer

MySQL Installer provides an easy to use, wizard-based installation experience for all MySQL software on Windows. It can be used to install and upgrade your MySQL Connector/Net installation.

To use, download and install [MySQL Installer](#).

After executing MySQL Installer, choose and install the MySQL Connector/Net product.

4.3.1.2 Installing Connector/Net Using the Standalone Installer

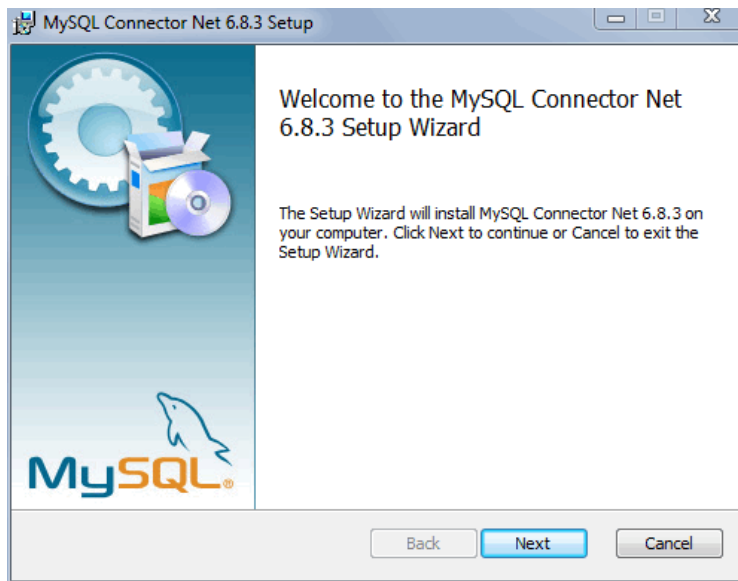
You install MySQL Connector/Net through a Windows Installer ([.msi](#)) installation package, which can install Connector/Net on supported Windows operating systems. The MSI package is a file named [mysql-connector-net-version.msi](#), where [version](#) indicates the Connector/Net version.

Note

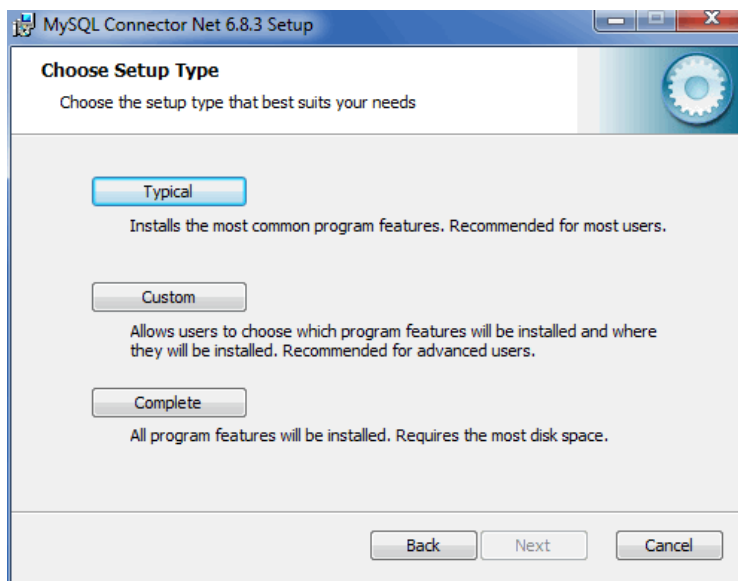
Using the central MySQL Installer is recommended, instead of the standalone package that is documented in this section. The MySQL Installer is available for download at [MySQL Installer](#).

To install Connector/Net:

1. Double-click the MSI installer file, and click **Next** to start the installation.

Figure 4.1 Connector/Net Installation: Welcome

2. You must choose the type of installation to perform.

Figure 4.2 Connector/Net Installation: Choose Setup Type

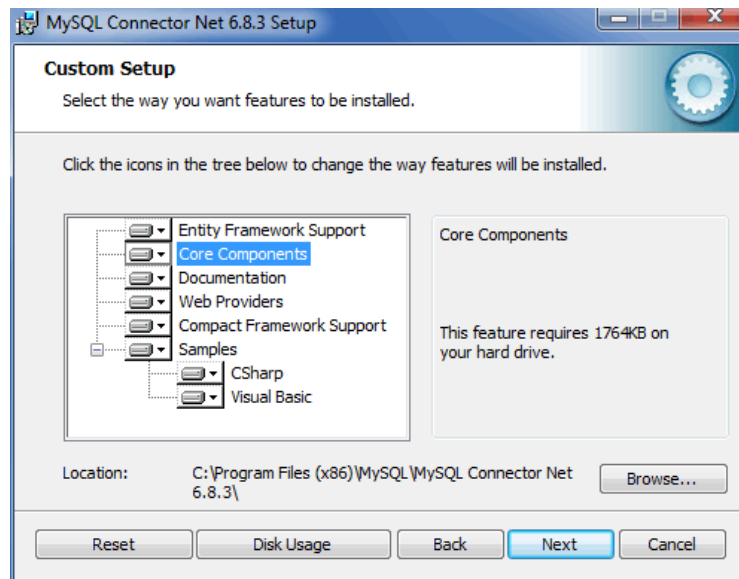
For most situations, the Typical installation is suitable. Click the **Typical** button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the **Complete** button and proceed to step 5. To customize your installation, including choosing the components to install and some installation options, click the **Custom** button and proceed to Step 3.

The Connector/Net installer will register the connector within the Global Assembly Cache (GAC) - this will make the Connector/Net component available to all applications, not just those where you explicitly reference the Connector/Net component. The installer will also create the necessary links in the Start menu to the documentation and release notes.

3. If you have chosen a custom installation, you can select the individual components to install, including the core interface component, supporting documentation (a CHM file) samples and

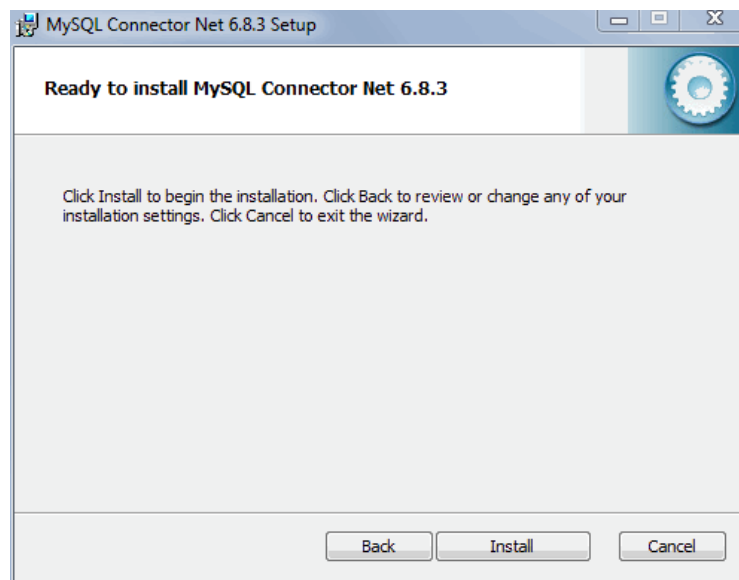
examples, and the source code. Select the items, and their installation level, and then click **Next** to continue the installation.

Figure 4.3 Connector/Net Installation: Custom Setup

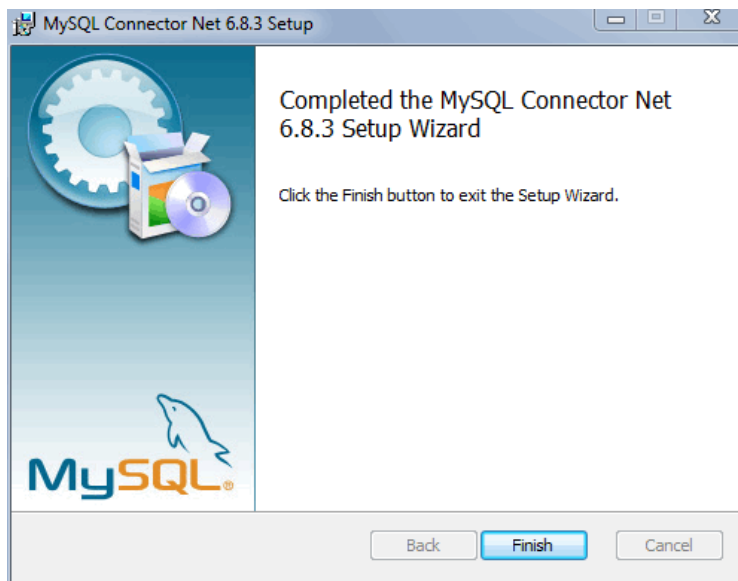


4. You will be given a final opportunity to confirm the installation. Click **Install** to copy and install the files onto your machine.

Figure 4.4 Connector/Net Installation: Ready To Begin



5. Once the installation has been completed, click **Finish** to exit the installer.

Figure 4.5 Connector/Net Installation: Finish Installation

Unless you choose otherwise, Connector/Net is installed in `C:\Program Files (x86)\MySQL\MySQL Connector Net X.X.X`, where `X.X.X` is replaced with the version of Connector/Net you are installing. New installations do not overwrite existing versions of Connector/Net.

Depending on your installation type, the installed components will include some or all of the following components:

- **Assemblies:** Connector/Net MySQL libraries for different versions of the .NET environment.
- **Documentation:** Connector/Net documentation in CHM format.
- **Samples:** Sample code and applications that use the Connector/Net component.

You may also use the `/quiet` or `/q` command-line option with the `msiexec` tool to install the Connector/Net package automatically (using the default options) with no notification to the user. Using this method the user cannot select options. Additionally, no prompts, messages or dialog boxes will be displayed.

```
C:\> msiexec /package connector-net.msi /quiet
```

To provide a progress bar to the user during automatic installation, use the `/passive` option.

4.3.2 Installing Connector/Net on Unix with Mono

There is no installer available for installing the Connector/Net component on your Unix installation. Before installing, ensure that you have a working Mono project installation. To test whether your system has Mono installed, enter:

```
shell> mono --version
```

The version of the Mono JIT compiler is displayed.

To compile C# source code, make sure a Mono C# compiler is installed.

Note

There are three Mono C# compilers available: `mcs`, which accesses the 1.0-profile libraries, `gmcs`, which accesses the 2.0-profile libraries, and `dmcs`, which accesses the 4.0-profile libraries.

To install Connector/Net on Unix/Mono:

1. Download the [mysql-connector-net-version-noinstall.zip](#) and extract the contents to a directory of your choice, for example: `~/connector-net/`.
2. In the directory where you unzipped the connector to, change into the `bin` subdirectory. Ensure the file `MySQL.Data.dll` is present. This filename is case-sensitive.
3. You must register the Connector/Net component, `MySQL.Data`, in the Global Assembly Cache (GAC). In the current directory enter the `gacutil` command:

```
root-shell> gacutil /i MySQL.Data.dll
```

This will register `MySQL.Data` into the GAC. You can check this by listing the contents of `/usr/lib/mono/gac`, where you will find `MySQL.Data` if the registration has been successful.

You are now ready to compile your application. You must ensure that when you compile your application you include the Connector/Net component using the `-r:` command-line option. For example:

```
shell> gmcs -r:System.dll -r:System.Data.dll -r:MySQL.Data.dll HelloWorld.cs
```

The referenced assemblies depend on the requirements of the application, but applications using MySQL Connector/Net must provide `-r:MySQL.Data` at a minimum.

You can further check your installation by running the compiled program, for example:

```
shell> mono HelloWorld.exe
```

4.3.3 Installing Connector/Net from the Source Code

Source packages of Connector/Net are available for download from <http://dev.mysql.com/downloads/connector/net/>.

The file contains the following directories:

- **Documentation**: Source files to build the documentation into the compiled HTML (CHM) format, code examples, collection files, and licenses for third-party components.
- **Samples**: Source files for several example applications.
- **Source\MySQL.ConnectorInstaller**: Source files to build the Connector/Net installer program.
- **Source\MySQL.Data**: Source files for the core data provider.
- **Source\MySQL.Data.Entity**: Source files for the Entity Framework.
- **Source\MySQL.Web**: Source files for the web providers, including the membership/role/profile providers that are used in ASP.NET web sites.
- **Tests**: Test cases for Connector/Net.

Building the Source Code on Microsoft Windows

The following procedure can be used to build the connector on Microsoft Windows.

- Navigate to the root of the source code tree.
- A Microsoft Visual Studio solution file named `MySQLClient.sln` is available to build the connector. Click this file to load the solution into Visual Studio.

`MySqlClient.sln` must be compiled with VS 2008, VS 2010, or VS 2012. Also, depending on the version, the dependencies to build it include Visual Studio SDK, NUnit, Entity Framework, and ANTLR Integration for Visual Studio.

- Select **Build, Build Solution** from the main menu to build the solution.

Building the Source Code on Unix

Support for building Connector/Net on Mono/Unix is not available.

4.4 Connector/Net Tutorials

The following tutorials illustrate how to develop MySQL programs using technologies such as Visual Studio, C#, ASP.NET, and the .NET and Mono frameworks. Work through the first tutorial to verify that you have the right software components installed and configured, then choose other tutorials to try depending on the features you intend to use in your applications.

4.4.1 Tutorial: An Introduction to Connector/Net Programming

This section provides a gentle introduction to programming with Connector/Net. The example code is written in C#, and is designed to work on both Microsoft .NET Framework and Mono.

This tutorial is designed to get you up and running with Connector/Net as quickly as possible, it does not go into detail on any particular topic. However, the following sections of this manual describe each of the topics introduced in this tutorial in more detail. In this tutorial you are encouraged to type in and run the code, modifying it as required for your setup.

This tutorial assumes you have MySQL and Connector/Net already installed. It also assumes that you have installed the `world` database sample, which can be downloaded from the [MySQL Documentation page](#). You can also find details on how to install the database on the same page.

Note

Before compiling the example code, make sure that you have added References to your project as required. The References required are `System`, `System.Data` and `MySql.Data`.

4.4.1.1 The MySqlConnection Object

For your Connector/Net application to connect to a MySQL database, it must establish a connection by using a `MySqlConnection` object.

The `MySqlConnection` constructor takes a connection string as one of its parameters. The connection string provides necessary information to make the connection to the MySQL database. The connection string is discussed more fully in [Section 4.5.1, "Connecting to MySQL Using Connector/Net"](#). For a list of supported connection string options, see [Section 4.6, "Connector/Net Connection String Options Reference"](#).

The following code shows how to create a connection object:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial1
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
```

```
    try
    {
        Console.WriteLine("Connecting to MySQL...");
        conn.Open();
        // Perform database operations
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
    conn.Close();
    Console.WriteLine("Done.");
}
```

When the [MySQLConnection](#) constructor is invoked, it returns a connection object, which is used for subsequent database operations. Open the connection before any other operations take place. Before the application exits, close the connection to the database by calling [Close](#) on the connection object.

Sometimes an attempt to perform an [Open](#) on a connection object can fail, generating an exception that can be handled using standard exception handling code.

In this section you have learned how to create a connection to a MySQL database, and open and close the corresponding connection object.

4.4.1.2 The MySqlCommand Object

Once a connection has been established with the MySQL database, the next step is to carry out the desired database operations. This can be achieved through the use of the [MySqlCommand](#) object.

You will see how to create a [MySqlCommand](#) object. Once it has been created, there are three main methods of interest that you can call:

- **ExecuteReader** - used to query the database. Results are usually returned in a [MySqlDataReader](#) object, created by [ExecuteReader](#).
- **ExecuteNonQuery** - used to insert and delete data.
- **ExecuteScalar** - used to return a single value.

Once a [MySqlCommand](#) object has been created, you will call one of the above methods on it to carry out a database operation, such as perform a query. The results are usually returned into a [MySqlDataReader](#) object, and then processed, for example the results might be displayed. The following code demonstrates how this could be done.

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial2
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'";
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            MySqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
```

```

        Console.WriteLine(rdr[0]+" -- "+rdr[1]);
    }
    rdr.Close();
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}

conn.Close();
Console.WriteLine("Done.");
}
}

```

When a connection has been created and opened, the code then creates a [MySQLCommand](#) object. Then the SQL query to be executed is passed to the [MySQLCommand](#) constructor. The [ExecuteReader](#) method is then used to generate a [MySQLReader](#) object. The [MySQLReader](#) object contains the results generated by the SQL executed on the command object. Once the results have been obtained in a [MySQLReader](#) object, the results can be processed. In this case, the information is printed out by a [while](#) loop. Finally, the [MySQLReader](#) object is disposed of by running its [Close](#) method on it.

In the next example, you will see how to use the [ExecuteNonQuery](#) method.

The procedure for performing an [ExecuteNonQuery](#) method call is simpler, as there is no need to create an object to store results. This is because [ExecuteNonQuery](#) is only used for inserting, updating and deleting data. The following example illustrates a simple update to the [Country](#) table:

```

using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial3
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "INSERT INTO Country (Name, HeadOfState, Continent) VALUES ('Disneyland', 'Mickey M";
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}

```

The query is constructed, the command object created and the [ExecuteNonQuery](#) method called on the command object. You can access your MySQL database with the [mysql](#) command interpreter and verify that the update was carried out correctly.

Finally, you will see how the [ExecuteScalar](#) method can be used to return a single value. Again, this is straightforward, as a [MySQLDataReader](#) object is not required to store results, a simple variable will do. The following code illustrates how to use [ExecuteScalar](#):

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial4
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT COUNT(*) FROM Country";
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            object result = cmd.ExecuteScalar();
            if (result != null)
            {
                int r = Convert.ToInt32(result);
                Console.WriteLine("Number of countries in the world database is: " + r);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

This example uses a simple query to count the rows in the [Country](#) table. The result is obtained by calling [ExecuteScalar](#) on the command object.

4.4.1.3 Working with Decoupled Data

Previously, when using [MySqlDataReader](#), the connection to the database was continually maintained, unless explicitly closed. It is also possible to work in a manner where a connection is only established when needed. For example, in this mode, a connection could be established to read a chunk of data, the data could then be modified by the application as required. A connection could then be reestablished only if and when the application writes data back to the database. This decouples the working data set from the database.

This decoupled mode of working with data is supported by Connector/Net. There are several parts involved in allowing this method to work:

- **Data Set** - The Data Set is the area in which data is loaded to read or modify it. A [DataSet](#) object is instantiated, which can store multiple tables of data.
- **Data Adapter** - The Data Adapter is the interface between the Data Set and the database itself. The Data Adapter is responsible for efficiently managing connections to the database, opening and closing them as required. The Data Adapter is created by instantiating an object of the [MySqlDataAdapter](#) class. The [MySqlDataAdapter](#) object has two main methods: [Fill](#) which reads data into the Data Set, and [Update](#), which writes data from the Data Set to the database.
- **Command Builder** - The Command Builder is a support object. The Command Builder works in conjunction with the Data Adapter. When a [MySqlDataAdapter](#) object is created, it is typically given an initial [SELECT](#) statement. From this [SELECT](#) statement the Command Builder can work out the corresponding [INSERT](#), [UPDATE](#) and [DELETE](#) statements that would be required to update the database. To create the Command Builder, an object of the class [MySqlCommandBuilder](#) is created.

Each of these classes will now be discussed in more detail.

Instantiating a DataSet object

A [DataSet](#) object can be created simply, as shown in the following example code snippet:

```
DataSet dsCountry;  
...  
dsCountry = new DataSet();
```

Although this creates the [DataSet](#) object, it has not yet filled it with data. For that, a Data Adapter is required.

Instantiating a MySqlDataAdapter object

The [MySqlDataAdapter](#) can be created as illustrated by the following example:

```
MySqlDataAdapter daCountry;  
...  
string sql = "SELECT Code, Name, HeadOfState FROM Country WHERE Continent='North America'";  
daCountry = new MySqlDataAdapter (sql, conn);
```

Note

The [MySqlDataAdapter](#) is given the SQL specifying the data to work with.

Instantiating a MySqlCommandBuilder object

Once the [MySqlDataAdapter](#) has been created, it is necessary to generate the additional statements required for inserting, updating and deleting data. There are several ways to do this, but in this tutorial you will see how this can most easily be done with [MySqlCommandBuilder](#). The following code snippet illustrates how this is done:

```
MySqlCommandBuilder cb = new MySqlCommandBuilder(daCountry);
```

Note

The [MySqlDataAdapter](#) object is passed as a parameter to the command builder.

Filling the Data Set

To do anything useful with the data from your database, you need to load it into a Data Set. This is one of the jobs of the [MySqlDataAdapter](#) object, and is carried out with its [Fill](#) method. The following example code illustrates this:

```
DataSet dsCountry;  
...  
dsCountry = new DataSet();  
...  
daCountry.Fill(dsCountry, "Country");
```

The [Fill](#) method is a [MySqlDataAdapter](#) method, and the Data Adapter knows how to establish a connection with the database and retrieve the required data, and then populate the Data Set when the [Fill](#) method is called. The second parameter "Country" is the table in the Data Set to update.

Updating the Data Set

The data in the Data Set can now be manipulated by the application as required. At some point, changes to data will need to be written back to the database. This is achieved through a [MySqlDataAdapter](#) method, the [Update](#) method.

```
daCountry.Update(dsCountry, "Country");
```

Again, the Data Set and the table within the Data Set to update are specified.

Working Example

The interactions between the `DataSet`, `MySqlDataAdapter` and `MySqlCommandBuilder` classes can be a little confusing, so their operation can perhaps be best illustrated by working code.

In this example, data from the `world` database is read into a Data Grid View control. Here, the data can be viewed and changed before clicking an update button. The update button then activates code to write changes back to the database. The code uses the principles explained above. The application was built using the Microsoft Visual Studio to place and create the user interface controls, but the main code that uses the key classes described above is shown below, and is portable.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using MySql.Data;
using MySql.Data.MySqlClient;

namespace WindowsFormsApplication5
{
    public partial class Form1 : Form
    {
        MySqlDataAdapter daCountry;
        DataSet dsCountry;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);
            try
            {
                label2.Text = "Connecting to MySQL...";

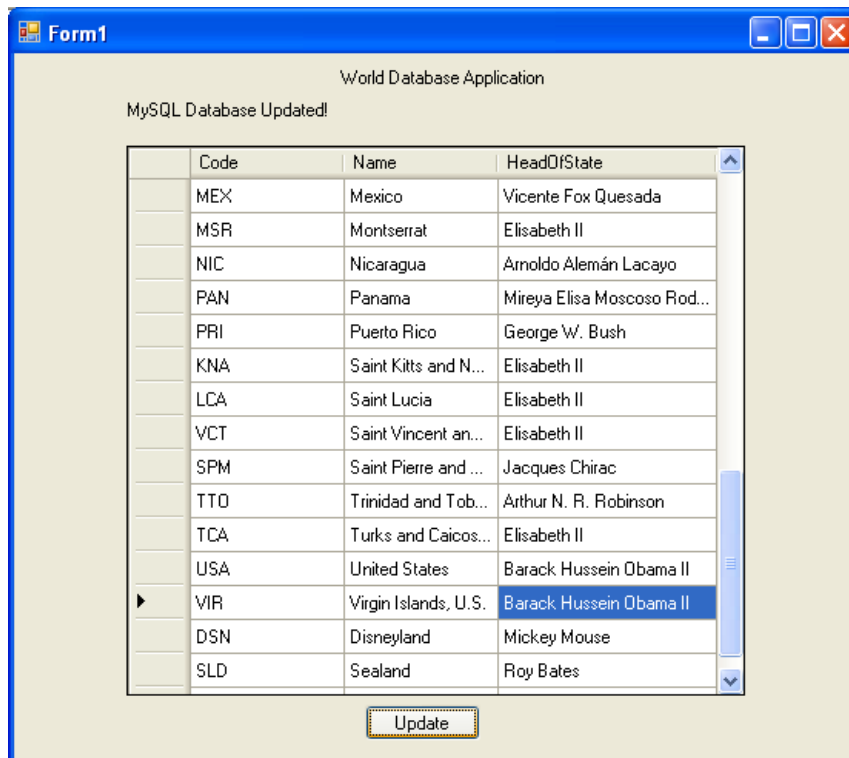
                string sql = "SELECT Code, Name, HeadOfState FROM Country WHERE Continent='North America'";
                daCountry = new MySqlDataAdapter (sql, conn);
                MySqlCommandBuilder cb = new MySqlCommandBuilder(daCountry);

                dsCountry = new DataSet();
                daCountry.Fill(dsCountry, "Country");
                dataGridView1.DataSource = dsCountry;
                dataGridView1.DataMember = "Country";
            }
            catch (Exception ex)
            {
                label2.Text = ex.ToString();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            daCountry.Update(dsCountry, "Country");
            label2.Text = "MySQL Database Updated!";
        }
    }
}
```

The application running is shown below:

Figure 4.6 World Database Application



4.4.1.4 Working with Parameters

This part of the tutorial shows you how to use parameters in your Connector/Net application.

Although it is possible to build SQL query strings directly from user input, this is not advisable as it does not prevent erroneous or malicious information being entered. It is safer to use parameters as they will be processed as field data only. For example, imagine the following query was constructed from user input:

```
string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent = "+user_continent;
```

If the string `user_continent` came from a Text Box control, there would potentially be no control over the string entered by the user. The user could enter a string that generates a runtime error, or in the worst case actually harms the system. When using parameters it is not possible to do this because a parameter is only ever treated as a field parameter, rather than an arbitrary piece of SQL code.

The same query written using a parameter for user input would be:

```
string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent = @Continent";
```

Note

The parameter is preceded by an '@' symbol to indicate it is to be treated as a parameter.

As well as marking the position of the parameter in the query string, it is necessary to add a parameter to the Command object. This is illustrated by the following code snippet:

```
cmd.Parameters.AddWithValue("@Continent", "North America");
```

In this example the string "North America" is supplied as the parameter value statically, but in a more practical example it would come from a user input control.

A further example illustrates the complete process:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial5
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent=@Continent";
            MySqlCommand cmd = new MySqlCommand(sql, conn);

            Console.WriteLine("Enter a continent e.g. 'North America', 'Europe': ");
            string user_input = Console.ReadLine();

            cmd.Parameters.AddWithValue("@Continent", user_input);

            MySqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine(rdr["Name"]+" --- "+rdr["HeadOfState"]);
            }
            rdr.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

In this part of the tutorial you have seen how to use parameters to make your code more secure.

4.4.1.5 Working with Stored Procedures

This section illustrates how to work with stored procedures. Putting database-intensive operations into stored procedures lets you define an API for your database application. You can reuse this API across multiple applications and multiple programming languages. This technique avoids duplicating database code, saving time and effort when you make updates due to schema changes, tune the performance of queries, or add new database operations for logging, security, and so on. Before working through this tutorial, familiarize yourself with the [CREATE PROCEDURE](#) and [CREATE FUNCTION](#) statements that create different kinds of stored routines.

For the purposes of this tutorial, you will create a simple stored procedure to see how it can be called from Connector/Net. In the MySQL Client program, connect to the [world](#) database and enter the following stored procedure:

```
DELIMITER //
CREATE PROCEDURE country_hos
(IN con CHAR(20))
BEGIN
    SELECT Name, HeadOfState FROM Country
    WHERE Continent = con;
END //
```

```
DELIMITER ;
```

Test that the stored procedure works as expected by typing the following into the `mysql` command interpreter:

```
CALL country_hos('Europe');
```

Note

The stored routine takes a single parameter, which is the continent to restrict your search to.

Having confirmed that the stored procedure is present and correct, you can see how to access it from Connector/Net.

Calling a stored procedure from your Connector/Net application is similar to techniques you have seen earlier in this tutorial. A `MySQLCommand` object is created, but rather than taking an SQL query as a parameter, it takes the name of the stored procedure to call. Set the `MySQLCommand` object to the type of stored procedure, as shown by the following code snippet:

```
string rtn = "country_hos";
MySQLCommand cmd = new MySQLCommand(rtn, conn);
cmd.CommandType = CommandType.StoredProcedure;
```

In this case, the stored procedure requires you to pass a parameter. This can be achieved using the techniques seen in the previous section on parameters, [Section 4.4.1.4, "Working with Parameters"](#), as shown in the following code snippet:

```
cmd.Parameters.AddWithValue("@con", "Europe");
```

The value of the parameter `@con` could more realistically have come from a user input control, but for simplicity it is set as a static string in this example.

At this point, everything is set up and you can call the routine using techniques also learned in earlier sections. In this case, the `ExecuteReader` method of the `MySQLCommand` object is used.

Complete working code for the stored procedure example is shown below:

```
using System;
using System.Data;

using MySQL.Data;
using MySQL.Data.MySQLClient;

public class Tutorial6
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string rtn = "country_hos";
            MySQLCommand cmd = new MySQLCommand(rtn, conn);
            cmd.CommandType = CommandType.StoredProcedure;

            cmd.Parameters.AddWithValue("@con", "Europe");

            MySQLDataReader rdr = cmd.ExecuteReader();
            while (rdr.Read())
            {
                Console.WriteLine(rdr[0] + " --- " + rdr[1]);
            }
        }
    }
}
```

```

        }
        rdr.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }

    conn.Close();
    Console.WriteLine("Done.");
}
}

```

In this section, you have seen how to call a stored procedure from Connector/Net. For the moment, this concludes our introductory tutorial on programming with Connector/Net.

4.4.2 Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider

Many web sites feature the facility for the user to create a user account. They can then log into the web site and enjoy a personalized experience. This requires that the developer creates database tables to store user information, along with code to gather and process this data. This represents a burden on the developer, and there is the possibility for security issues to creep into the developed code. However, ASP.NET 2.0 introduced the Membership system. This system is designed around the concept of Membership, Profile and Role Providers, which together provide all of the functionality to implement a user system, that previously would have to have been created by the developer from scratch.

Currently, MySQL Connector/Net provides Membership, Role, Profile and Session State Providers.

This tutorial shows you how to set up your ASP.NET web application to use the MySQL Connector/Net Membership and Role Providers. It assumes that you have MySQL Server installed, along with MySQL Connector/Net and Microsoft Visual Studio. This tutorial was tested with MySQL Connector/Net 6.0.4 and Microsoft Visual Studio 2008 Professional Edition. It is recommended you use 6.0.4 or above for this tutorial.

1. Create a new database in the MySQL Server using the MySQL Command-Line Client program ([mysql](#)), or other suitable tool. It does not matter what name is used for the database, but record it. You specify it in the connection string constructed later in this tutorial. This database contains the tables, automatically created for you later, used to store data about users and roles.
2. Create a new ASP.NET Web Site in Visual Studio. If you are not sure how to do this, refer to [Section 4.4.7, "Tutorial: Databinding in ASP.NET Using LINQ on Entities"](#), which demonstrates how to create a simple ASP.NET web site.
3. Add References to [MySQL.Data](#) and [MySQL.Web](#) to the web site project.
4. Locate the [machine.config](#) file on your system, which is the configuration file for the .NET Framework.
5. Search the [machine.config](#) file to find the membership provider [MySQLMembershipProvider](#).
6. Add the attribute [autogenerateschema="true"](#). The appropriate section should now resemble the following:

Note

For the sake of brevity, some information is excluded.

```

<membership>
  <providers>
    <add name="AspNetSqlMembershipProvider"
        type="System.Web.Security.SqlMembershipProvider"

```

```
...
    connectionStringName="LocalSqlServer"
... />
<add name="MySQLMembershipProvider"
    autogeneratedschema="true"
    type="MySql.Web.Security.MySQLMembershipProvider, MySql.Web, Version=6.0.4.0, Culture=neutral, Pub
    connectionStringName="LocalMySQLServer"
... />
</providers>
</membership>
```

Note

The connection string, [LocalMySQLServer](#), connects to the MySQL server that contains the membership database.

The `autogeneratedschema="true"` attribute will cause MySQL Connector/Net to silently create, or upgrade, the schema on the database server, to contain the required tables for storing membership information.

7. It is now necessary to create the connection string referenced in the previous step. Load the web site's `web.config` file into Visual Studio.
8. Locate the section marked `<connectionStrings>`. Add the following connection string information:

```
<connectionStrings>
  <remove name="LocalMySQLServer"/>
  <add name="LocalMySQLServer"
    connectionString="Datasource=localhost;Database=users;uid=root;pwd=password;"
    providerName="MySql.Data.MySqlClient"/>
</connectionStrings>
```

The database specified is the one created in the first step. You could alternatively have used an existing database.

9. At this point build the solution to ensure no errors are present. This can be done by selecting **Build**, **Build Solution** from the main menu, or pressing **F6**.
10. ASP.NET supports the concept of locally and remotely authenticated users. With local authentication the user is validated using their Windows credentials when they attempt to access the web site. This can be useful in an Intranet environment. With remote authentication, a user is prompted for their login details when accessing the web site, and these credentials are checked against the membership information stored in a database server such as MySQL Server. You will now see how to choose this form of authentication.

Start the ASP.NET Web Site Administration Tool. This can be done quickly by clicking the small hammer/Earth icon in the Solution Explorer. You can also launch this tool by selecting **Website**, **ASP.NET Configuration** from the main menu.

11. In the ASP.NET Web Site Administration Tool click the **Security** tab.
12. Now click the **User Authentication Type** link.
13. Select the **From the internet** radio button. The web site will now need to provide a form to allow the user to enter their login details. These will be checked against membership information stored in the MySQL database.

Figure 4.7 Authentication Type

The screenshot shows the 'ASP.NET Web Site Administration Tool' interface. The 'Security' tab is selected in the top navigation bar. Below the navigation bar, the question 'How will users access your site?' is displayed. There are two radio button options: 'From the internet' (selected) and 'From a local network'. The 'From the internet' option has a description: 'Select this option if users will access your web site from the public internet. Users will be required to log on using a web form. The site will use forms authentication to identify users according to user information that you store in a database.' The 'From a local network' option has a description: 'Select this option if users will access your web site only from a private local network. The site will use built-in Microsoft Windows authentication to identify users. Users with a valid Windows user name and password will be able to access your site.'

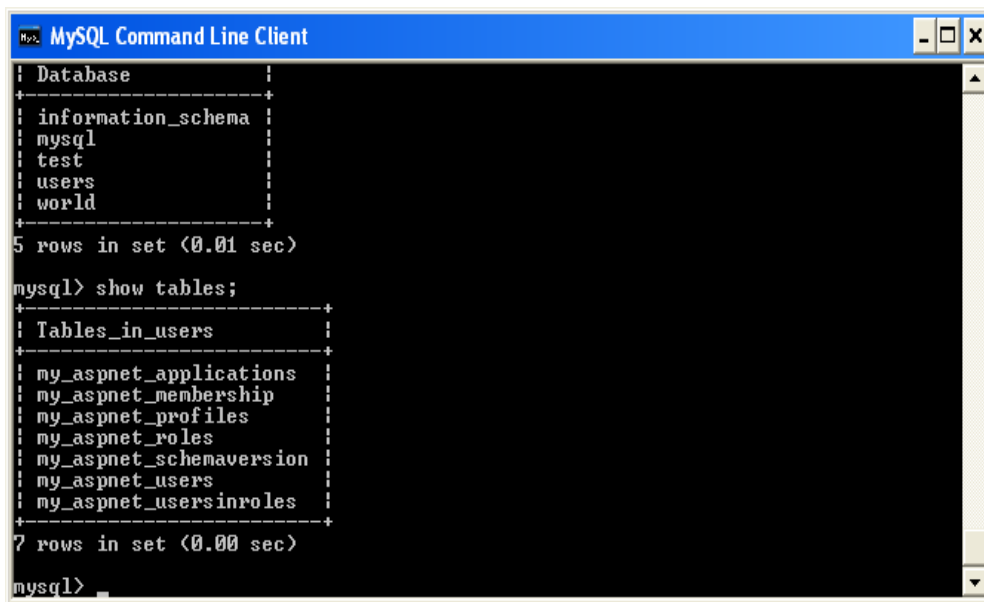
14. You now need to specify the Role and Membership Provider to be used. Click the **Provider** tab.
15. Click the **Select a different provider for each feature (advanced)** link.
16. Now select the **MySQLMembershipProvider** and the **MySQLRoleProvider** radio buttons.

Figure 4.8 Select Membership and Role Provider

The screenshot shows the 'ASP.NET Web Site Administration Tool' interface with the 'Provider' tab selected. The instruction 'Use this page to select a provider for each feature.' is at the top. There are two sections: 'Membership Provider' and 'Role Provider'. Each section contains three radio button options. In the 'Membership Provider' section, 'MySQLMembershipProvider' is selected. In the 'Role Provider' section, 'MySQLRoleProvider' is selected. There are 'Test' links next to the 'AspNetSqlMembershipProvider' and 'AspNetSqlRoleProvider' options.

17. In Visual Studio, rebuild the solution by selecting **Build, Rebuild Solution** from the main menu.
18. Check that the necessary schema has been created. This can be achieved using the `mysql` command interpreter.

Figure 4.9 Membership and Role Provider Tables



```

MySQL Command Line Client
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| test |
| users |
| world |
+-----+
5 rows in set (0.01 sec)

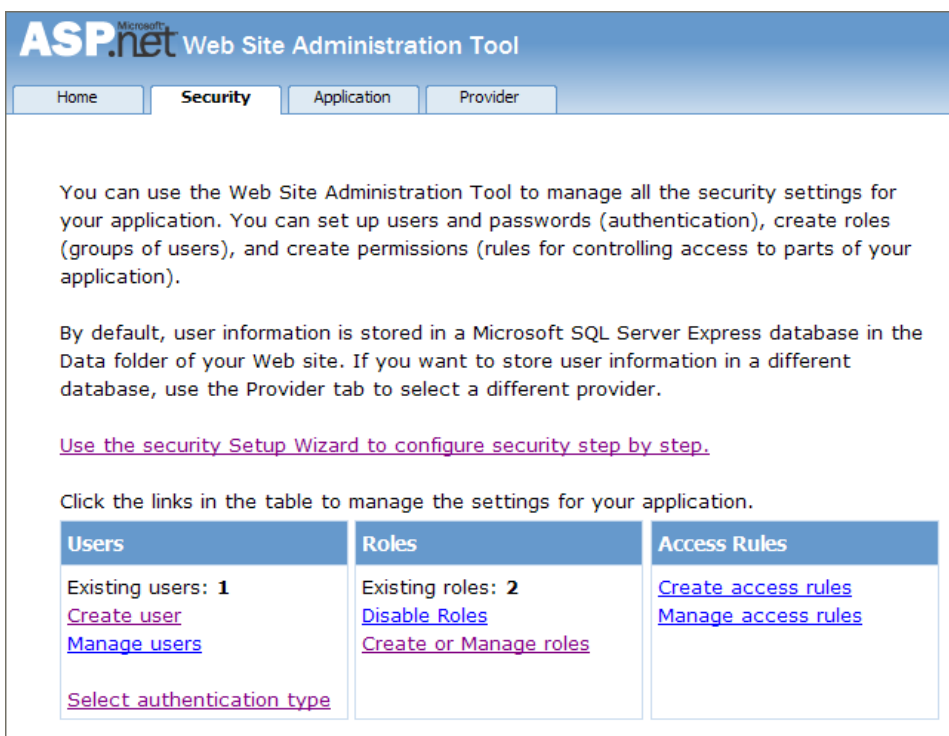
mysql> show tables;
+-----+
| Tables_in_users |
+-----+
| my_aspnet_applications |
| my_aspnet_membership |
| my_aspnet_profiles |
| my_aspnet_roles |
| my_aspnet_schemaversion |
| my_aspnet_users |
| my_aspnet_usersinroles |
+-----+
7 rows in set (0.00 sec)

mysql>

```

19. Assuming all is present and correct, you can now create users and roles for your web application. The easiest way to do this is with the ASP.NET Web Site Administration Tool. However, many web applications contain their own modules for creating roles and users. For simplicity, the ASP.NET Web Site Administration Tool will be used in this tutorial.
20. In the ASP.NET Web Site Administration Tool, click the **Security** tab. Now that both the Membership and Role Provider are enabled, you will see links for creating roles and users. Click the **Create or Manage Roles** link.

Figure 4.10 Security Tab



ASP.NET Web Site Administration Tool

Home **Security** Application Provider

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

[Use the security Setup Wizard to configure security step by step.](#)

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: 1 Create user Manage users Select authentication type	Existing roles: 2 Disable Roles Create or Manage roles	Create access rules Manage access rules

21. You can now enter the name of a new Role and click **Add Role** to create the new Role. Create new Roles as required.

22. Click the **Back** button.
23. Click the **Create User** link. You can now fill in information about the user to be created, and also allocate that user to one or more Roles.

Figure 4.11 Create User

24. Using the `mysql` command interpreter, you can check that your database has been correctly populated with the Membership and Role data.

Figure 4.12 Membership and Roles Table Contents

```

MySQL Command Line Client
mysql> show tables;
+-----+
| my_aspnet_roles |
| my_aspnet_schemaversion |
| my_aspnet_users |
| my_aspnet_usersinroles |
+-----+
7 rows in set (0.00 sec)

mysql> select * from my_aspnet_users;
+----+-----+-----+-----+-----+
| id | applicationId | name | isAnonymous | lastActivityDate |
+----+-----+-----+-----+-----+
| 1 | 1 | johnndoe | 0 | 2009-07-14 17:17:25 |
+----+-----+-----+-----+-----+
1 row in set (0.01 sec)

mysql> select * from my_aspnet_roles;
+----+-----+-----+
| id | applicationId | name |
+----+-----+-----+
| 1 | 1 | Visitors |
| 2 | 1 | Admins |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

In this tutorial, you have seen how to set up the MySQL Connector/Net Membership and Role Providers for use in your ASP.NET web application.

4.4.3 Tutorial: MySQL Connector/Net ASP.NET Profile Provider

This tutorial shows you how to use the MySQL Profile Provider to store user profile information in a MySQL database. The tutorial uses MySQL Connector/Net 6.1.1, MySQL Server 5.1 and Microsoft Visual Studio 2008 Professional Edition.

Many modern web sites allow the user to create a personal profile. This requires a significant amount of code, but ASP.NET reduces this considerably by including the functionality in its Profile classes. The Profile Provider provides an abstraction between these classes and a data source. The MySQL Profile Provider enables profile data to be stored in a MySQL database. This enables the profile properties to be written to a persistent store, and be retrieved when required. The Profile Provider also enables profile data to be managed effectively, for example it enables profiles that have not been accessed since a specific date to be deleted.

The following steps show you how you can select the MySQL Profile Provider.

1. Create a new ASP.NET web project.
2. Select the MySQL Website Configuration tool. Due to a bug in 6.1.1 you may have to first connect to a server in Server Explorer before the tool's icon will display in the toolbar of the Solution Explorer.
3. In the MySQL Website Configuration tool navigate through the tool to the Profiles page.
4. Select the **Use MySQL to manage my profiles** check box.
5. Select the **Autogenerate Schema** check box.
6. Click the **Edit...** button and configure a connection string for the database that will be used to store user profile information.
7. Navigate to the last page of the tool and click **Finish** to save your changes and exit the tool.

At this point you are now ready to start using the MySQL Profile Provider. With the following steps you can carry out a preliminary test of your installation.

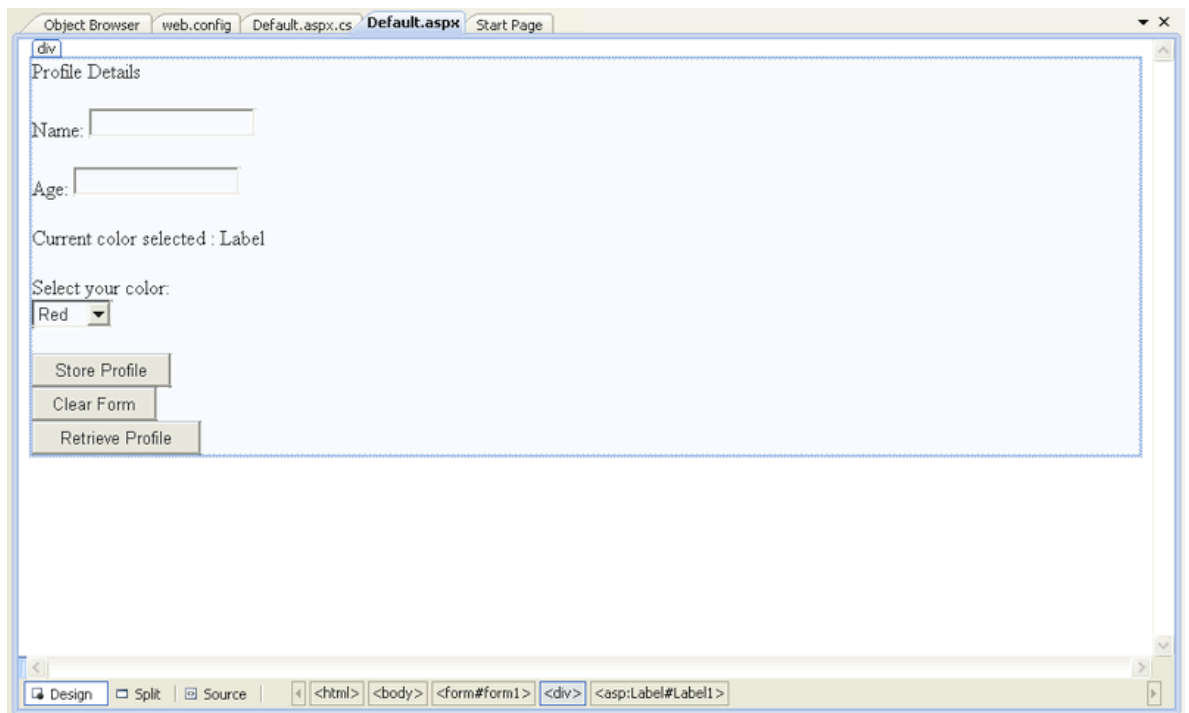
1. Open your [web.config](#) file.
2. Add a simple profile such as the following:

```
<system.web>
  <anonymousIdentification enabled="true"/>
  <profile defaultProvider="MySQLProfileProvider">
    ...
    <properties>
      <add name="Name" allowAnonymous="true"/>
      <add name="Age" allowAnonymous="true" type="System.UInt16"/>
      <group name="UI">
        <add name="Color" allowAnonymous="true" defaultValue="Blue"/>
        <add name="Style" allowAnonymous="true" defaultValue="Plain"/>
      </group>
    </properties>
  </profile>
  ...
```

Setting [anonymousIdentification](#) to true enables unauthenticated users to use profiles. They are identified by a GUID in a cookie rather than by a user name.

Now that the simple profile has been defined in [web.config](#), the next step is to write some code to test the profile.

1. In Design View design a simple page with the following controls:

Figure 4.13 Simple Profile Application

These will allow the user to enter some profile information. The user can also use the buttons to save their profile, clear the page, and restore their profile data.

2. In the Code View add code as follows:

```
...
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        TextBox1.Text = Profile.Name;
        TextBox2.Text = Profile.Age.ToString();
        Label1.Text = Profile.UI.Color;
    }
}

// Store Profile
protected void Button1_Click(object sender, EventArgs e)
{
    Profile.Name = TextBox1.Text;
    Profile.Age = UInt16.Parse(TextBox2.Text);
}

// Clear Form
protected void Button2_Click(object sender, EventArgs e)
{
    TextBox1.Text = "";
    TextBox2.Text = "";
    Label1.Text = "";
}

// Retrieve Profile
protected void Button3_Click(object sender, EventArgs e)
{
    TextBox1.Text = Profile.Name;
    TextBox2.Text = Profile.Age.ToString();
    Label1.Text = Profile.UI.Color;
}

protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{

```

```

    Profile.UI.Color = DropDownList1.SelectedValue;
}
...

```

3. Save all files and build the solution to check that no errors have been introduced.
4. Run the application.
5. Enter your name, age, and select a color from the listbox. Now store this information in your profile by clicking **Store Profile**.

Not selecting a color from the listbox uses the default color, *Blue*, that was specified in the [web.config](#) file.

6. Click **Clear Form** to clear text from the textboxes and the label that displays your chosen color.
7. Now click **Retrieve Profile** to restore your profile data from the MySQL database.
8. Now exit the browser to terminate the application.
9. Run the application again, which also restores your profile information from the MySQL database.

In this tutorial you have seen how to using the MySQL Profile Provider with MySQL Connector/Net.

4.4.4 Tutorial: Web Parts Personalization Provider

MySQL Connector/Net provides a Web Parts Personalization provider that allows you to use a MySQL server to store personalization data.

Note

This feature was added in MySQL Connector/Net 6.9.0.

This tutorial demonstrates how to configure the Web Parts Personalization provider using MySQL Connector/Net.

Requirements

- An ASP.NET web site or web application with a membership provider
- .NET Framework 3.0 or above
- MySQL Server 5.5 or above

Configuring MySQL Web Parts Personalization Provider

Tasks include:

- Add References to *MySql.Data* and *MySql.Web* to the web site or web application project.
- Include a MySQL Connector/Net personalization provider into the *system.web* section in the [web.config](#) file:

```

<webParts>
  <personalization defaultProvider="MySQLPersonalizationProvider">
    <providers>
      <clear/>
      <add name="MySQLPersonalizationProvider"
        type="MySql.Web.Personalization.MySqlPersonalizationProvider,
        MySql.Web, Version=6.9.3.0, Culture=neutral,
        PublicKeyToken=c5687fc88969c44d"
        connectionStringName="LocalMySqlServer"
        applicationName="/" />
    </providers>
  </personalization>
  <authorization>
    <allow verbs="modifyState" users="*" />
  </authorization>
</webParts>

```

```

    <allow verbs="enterSharedScope" users="*" />
  </authorization>
</personalization>
</webParts>

```

Creating Web Part Controls

- Create a Web application using MySQL Connector/Net ASP.NET Membership. For information about doing this, see [Section 4.4.2, "Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider"](#).
- Create a new ASP.NET page.
- Change to the *Design* view.
- From the **Toolbox**, drag a **WebPartManager** control to the page.
- Now define an HTML table with 3 columns and 1 row.
- In the left and middle cells, from the **WebParts Toolbox**, drag and drop a *WebPartZone* control into each cell.
- In the right column, from the **WebParts Toolbox**, drag and drop a *CatalogZone* with *PageCatalogPart* and *EditorZone* controls.
- Add controls to the *WebPartZone*, it will look similar to:

```

<table>
  <tr>
    <td>
      <asp:WebPartZone ID="LeftZone" runat="server" HeaderText="Left Zone">
        <ZoneTemplate>
          <asp:Label ID="Label1" runat="server" title="Left Zone">
            <asp:BulletedList ID="BulletedList1" runat="server">
              <asp:ListItem Text="Item 1"></asp:ListItem>
              <asp:ListItem Text="Item 2"></asp:ListItem>
              <asp:ListItem Text="Item 3"></asp:ListItem>
            </asp:BulletedList>
          </asp:Label>
        </ZoneTemplate>
      </asp:WebPartZone>
    </td>
    <td>
      <asp:WebPartZone ID="MainZone" runat="server" HeaderText="Main Zone">
        <ZoneTemplate>
          <asp:Label ID="Label11" runat="server" title="Main Zone">
            <h2>This is the Main Zone</h2>
          </asp:Label>
        </ZoneTemplate>
      </asp:WebPartZone>
    </td>
    <td>
      <asp:CatalogZone ID="CatalogZone1" runat="server">
        <ZoneTemplate>
          <asp:PageCatalogPart ID="PageCatalogPart1" runat="server" />
        </ZoneTemplate>
      </asp:CatalogZone>
      <asp:EditorZone ID="EditorZone1" runat="server">
        <ZoneTemplate>
          <asp:LayoutEditorPart ID="LayoutEditorPart1" runat="server" />
          <asp:AppearanceEditorPart ID="AppearanceEditorPart1" runat="server" />
        </ZoneTemplate>
      </asp:EditorZone>
    </td>
  </tr>

```

```
</table>
```

- Outside of the HTML table, add a *DropDownList*, 2 *Buttons*, and a *Label* as follows:

```
<asp:DropDownList ID="DisplayModes" runat="server" AutoPostBack="True"
    OnSelectedIndexChanged="DisplayModes_SelectedIndexChanged">
</asp:DropDownList>
<asp:Button ID="ResetButton" runat="server" Text="Reset"
    OnClick="ResetButton_Click" />
<asp:Button ID="ToggleButton" runat="server" OnClick="ToggleButton_Click"
    Text="Toggle Scope" />
<asp:Label ID="ScopeLabel" runat="server"></asp:Label>
```

- The following code will fill the *DropDownList* for the *DisplayModes*, show the current scope, reset the personalization state, toggle the scope (between user and the shared scope), and change the display mode:

```
public partial class WebPart : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            foreach (WebPartDisplayMode mode in WebPartManager1.SupportedDisplayModes)
            {
                if (mode.IsEnabled(WebPartManager1))
                {
                    DisplayModes.Items.Add(mode.Name);
                }
            }
            ScopeLabel.Text = WebPartManager1.Personalization.Scope.ToString();
        }
    }

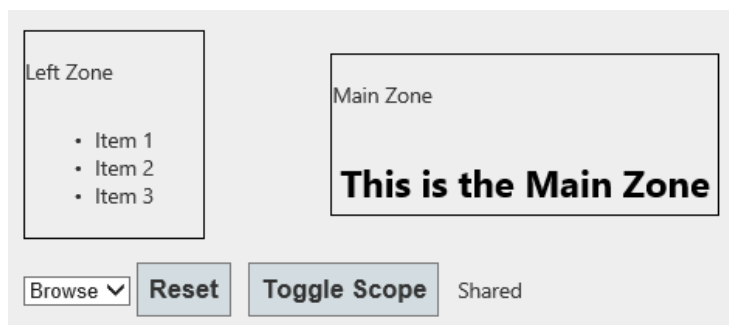
    protected void ResetButton_Click(object sender, EventArgs e)
    {
        if (WebPartManager1.Personalization.IsEnabled &&
            WebPartManager1.Personalization.IsModifiable)
        {
            WebPartManager1.Personalization.ResetPersonalizationState();
        }
    }

    protected void ToggleButton_Click(object sender, EventArgs e)
    {
        WebPartManager1.Personalization.ToggleScope();
    }

    protected void DisplayModes_SelectedIndexChanged(object sender, EventArgs e)
    {
        var mode = WebPartManager1.SupportedDisplayModes[DisplayModes.SelectedValue];
        if (mode != null && mode.IsEnabled(WebPartManager1))
        {
            WebPartManager1.DisplayMode = mode;
        }
    }
}
```

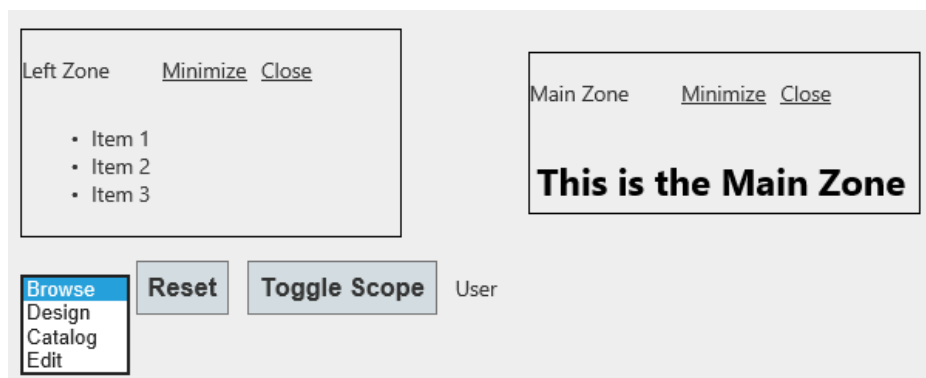
Testing Web Part Changes

Run the application and open the Web Part page, and it should look like this:

Figure 4.14 Web Parts Page

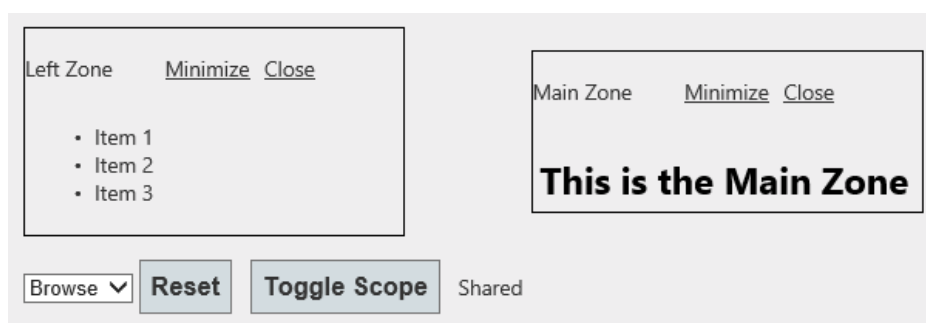
The first time when the user is not authenticated, by default the scope is *Shared*. The user must be authenticated to change settings on the web part controls.

In the following screenshot shows an authenticated user that is able to customize the controls:

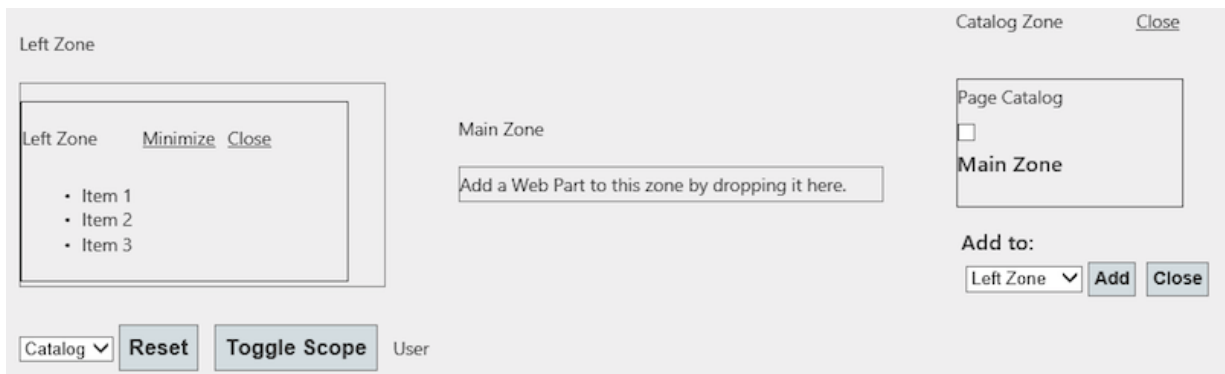
Figure 4.15 Authenticated User Controls

You can see the *DropDownList* and all of the display modes for the current user.

Click *Toggle Scope* to switch the application to the shared scope:

Figure 4.16 Toggle Scope

Now you can personalize the zones using the *Edit* or *Catalog* display modes at a specific user or all users level.

Figure 4.17 Personalize Zones

4.4.5 Tutorial: Simple Membership Web Provider

This section documents the ability to use a Simple Membership Provider on MVC 4 templates. The configuration OAuth compatible for the application to login using external credentials from third party providers like Google, Facebook, Twitter or others.

This tutorial creates an application using a Simple Membership Provider, and then adds third-party (google) OAuth authentication support.

Note

This feature was added in MySQL Connector/Net 6.9.0.

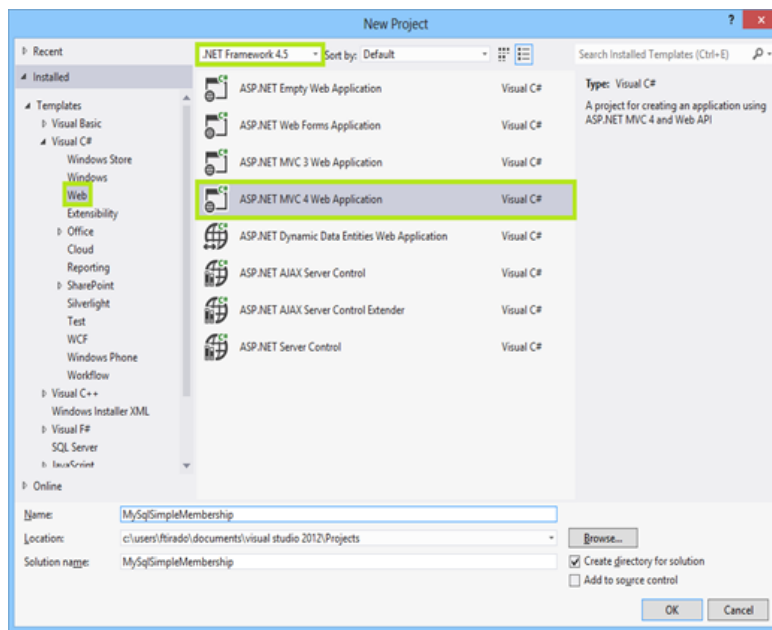
4.4.5.1 Requirements

- Connector/Net 6.9.x, or greater
- .NET Framework 4.0, or greater
- Visual Studio 2012, or greater
- MVC 4

4.4.5.2 Creating and Configuring a New Project

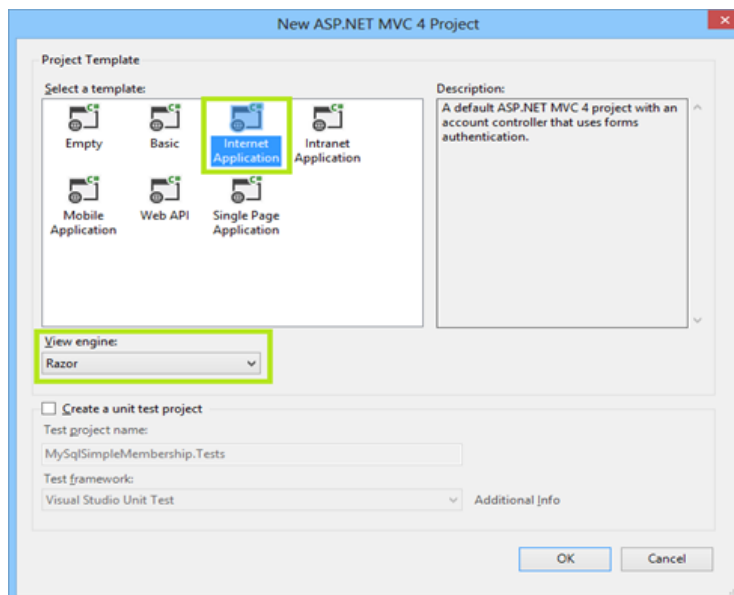
- Open Visual Studio and create a new project, of **MVC 4 Web Application** type, and configure the project to use .NET Framework 4.5.

Figure 4.18 Simple Membership: New Project



- Choose the template and view engine that you like. This tutorial is using the **Internet Application Template** with the Razor view engine. Optionally, here you can add a unit test project by checking **Create a unit test project**.

Figure 4.19 Simple Membership: Choose Template and Engine



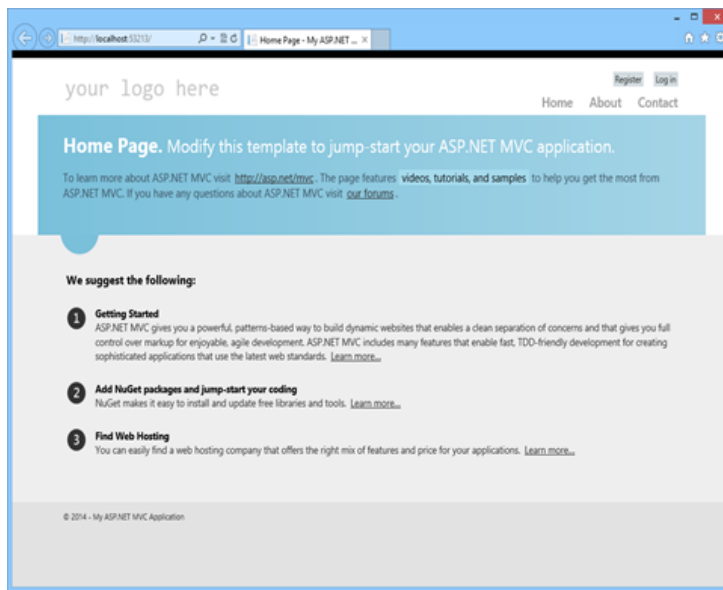
- Add references to the `MySQL.Data`, `MySQL.Data.Entities`, and `MySQL.Web` assemblies. The assemblies chosen must match the .NET Framework and Entity Framework versions added to the project by the template.
- Add a valid MySQL connection string to the `web.config` file, similar to:

```
<add
  name="MyConnection"
  connectionString="server=localhost;UserId=root;password=pass;database=MySQLSimpleMembership;logging
  providerName="MySQL.Data.MySqlClient"/>
```

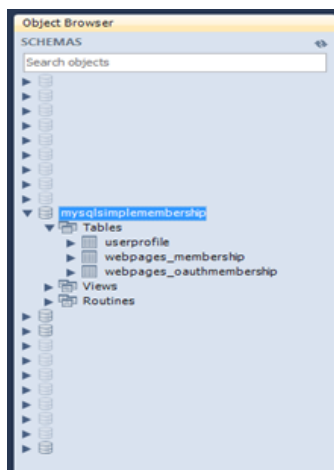
- Under the <system.data> node, add configuration information similar to the following:

```
<membership defaultProvider="MySQLSimpleMembershipProvider">
  <providers>
    <clear/>
    <add
      name="MySQLSimpleMembershipProvider"
      type="MySQL.Web.Security.MySQLSimpleMembershipProvider, MySQL.Web, Version=6.9.2.0, Culture=neutral, PublicKeyToken=..."
      applicationName="MySQLSimpleMembershipTest"
      description="MySQLDefaultApplication"
      connectionStringName="MyConnection"
      userTableName="MyUserTable"
      userIdColumn="MyUserIdColumn"
      userNameColumn="MyUserNameColumn"
      autoGenerateTables="True"/>
    </providers>
  </membership>
```

- Update the above configuration with valid values for the following properties: connectionStringName, userTableName, userIdColumn, userNameColumn, and autoGenerateTables.
 - **userTableName**: Name of the table to store the user information. This table is independent from the schema generated by the provider, and it can be changed in the future.
 - **userId**: Name of the column that stores the ID for the records in the **userTableName**.
 - **userName**: Name of the column that stores the name/user for the records in the **userTableName**.
 - **connectionStringName**: This property must match a connection string defined in [web.config](#) file.
 - **autoGenerateTables**: This must be set to [false](#) if the table to handle the credentials already exists.
- Update your **DBContext** class with the connection string name configured.
- Go to the SimpleMembershipInitializer class contained in the InitializeSimpleMembershipAttribute.cs file that is located in the Filters/ folder. Look for the method call "WebSecurity.InitializeDatabaseConnection", and update the parameters with the configuration for connectionStringName, userTableName, userIdColumn and userNameColumn.
- If the database configured in the connection string does not exist, then create it.
- After running the web application, the following screen is displayed on success:

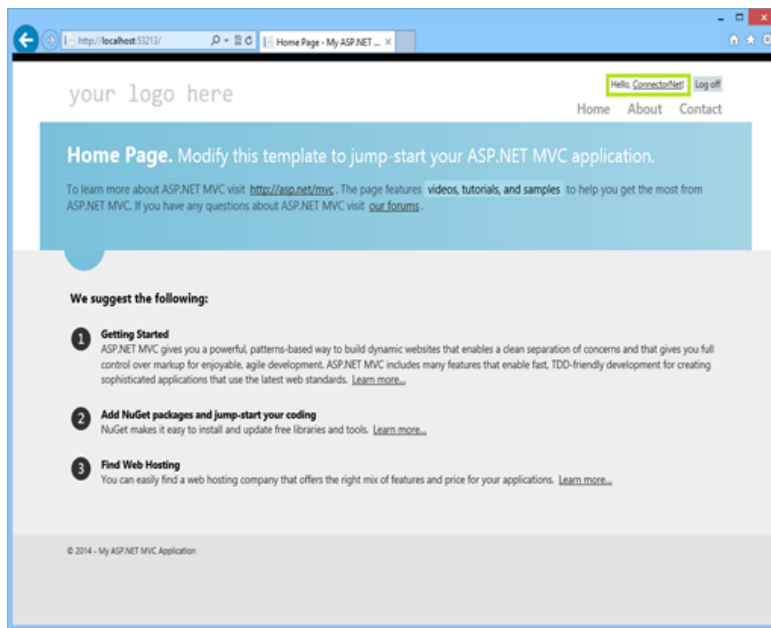
Figure 4.20 Simple Membership: Generated Home Page

- If the application executed with success, then the generated schema will be similar to the following:

Figure 4.21 Simple Membership: Generated Tables

- To create a user, click **Register** on the generated web page. Type the desired user name and password, and then execute the registration form. This redirects you to the home page with the newly created user logged in, as shown near the top-right side of the page:

Figure 4.22 Simple Membership: Logged In



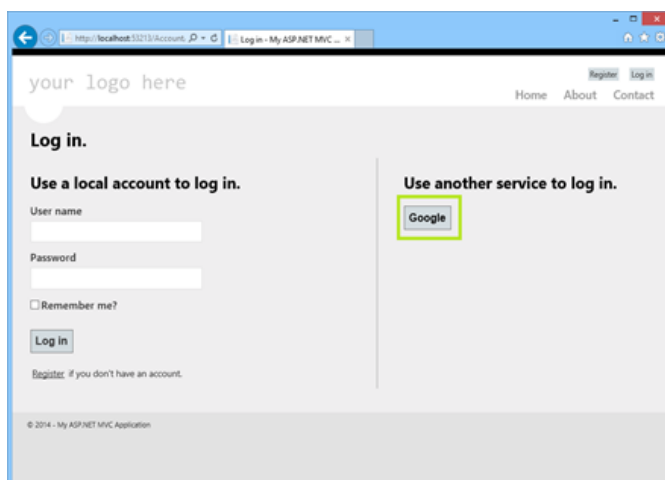
- The data for the newly created user can be located in the [UserProfile](#) and [Webpages_Membership](#) tables.

4.4.5.3 Adding OAuth Authentication to a Project

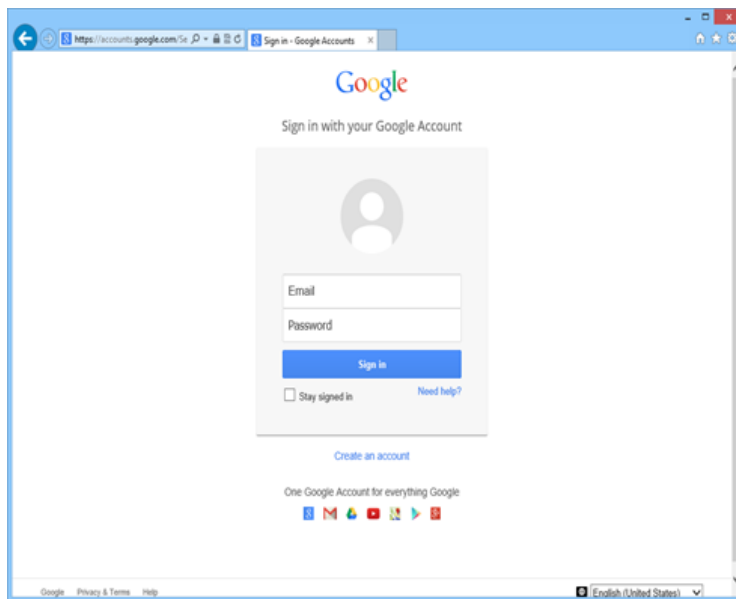
OAuth is another authentication option for websites that use the Simple Membership Provider. A user can be validated using an external account like Facebook, Twitter, Google, and others. The following steps enable authentication using a Google account in the application.

- Go to the class [AuthConfig.cs](#) in the [App_Start](#) folder.
- As we are using google, find the **RegisterAuth** method and uncomment the last line where it calls **OauthWebSecurity.RegisterGoogleClient**.
- Run the application. Once the application is running, click **Log in** to open the log in page. Then, click **Google** under **Use another service to log in**.

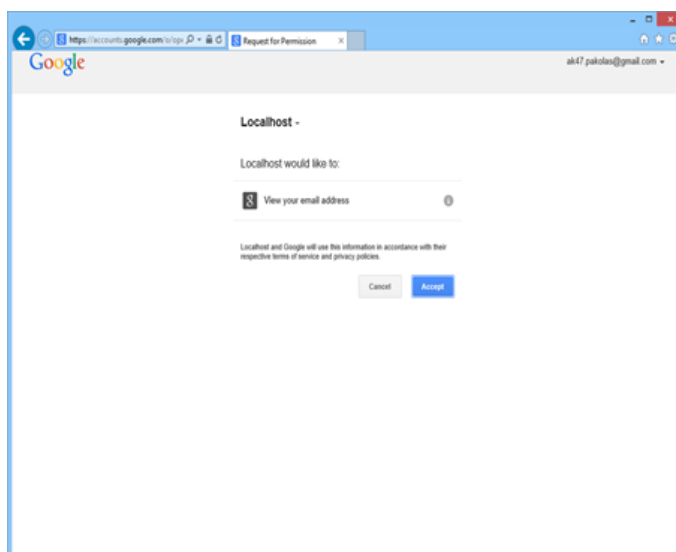
Figure 4.23 Simple Membership with OAuth: Google Service



- This redirects to the Google login page (at google.com), and requests you to sign in with your Google account information.

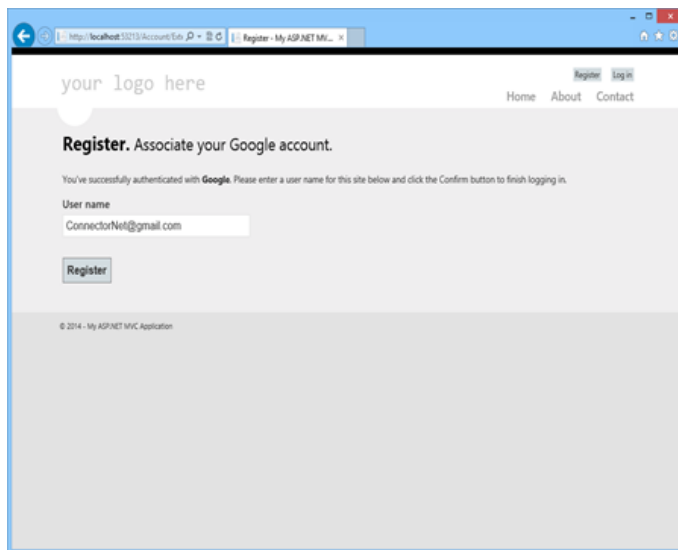
Figure 4.24 Simple Membership with OAuth: Google Service Login

- After submitting the correct credentials, a message requests permission for your application to access the user's information. Read the description and then click **Accept** to allow the quoted actions, and to redirect back to your application's login page.

Figure 4.25 Simple Membership with OAuth: Google Service Approval

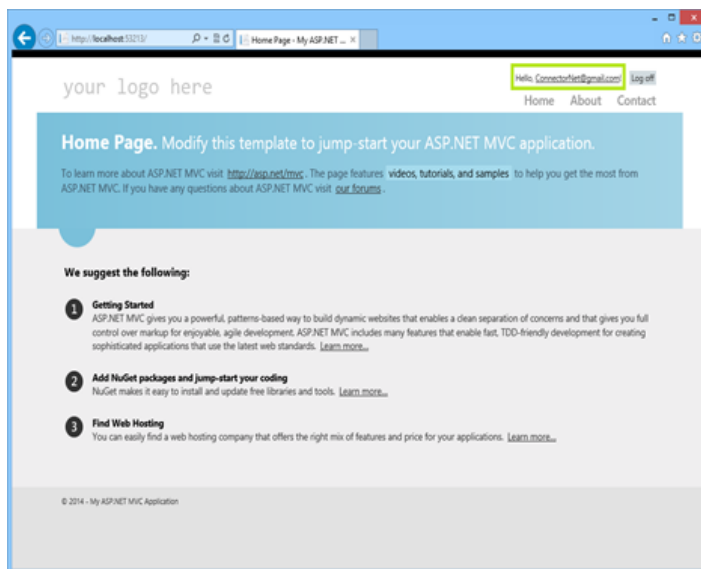
- Now your application can register the account. The **User name** field will be filled in with the appropriate information (in this case, the email address that is associated with the Google account). Click **Register** to register the user with your application.

Figure 4.26 Simple Membership with OAuth: Google Service Registration



- Now the new user is logged into the application from an external source using OAuth.

Figure 4.27 Simple Membership with OAuth: Google Service Logged In



- Information about the new user is stored in the `UserProfile` and `Webpages_OauthMembership` tables.

To use another external option to authenticate users, you must enable the client in the same class where we enabled the Google provider in this tutorial. Typically, providers require you to register your application before allowing OAuth authentication, and once registered they typically provide a token/key and an ID that must be used when registering the provider in the application.

4.4.6 Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source

In this tutorial you will learn how to create a Windows Forms Data Source from an Entity in an Entity Data Model. This tutorial assumes that you have installed the `world` database sample, which can be downloaded from the [MySQL Documentation page](#). You can also find details on how to install the

database on the same page. It will also be convenient for you to create a connection to the [world](#) database after it is installed. For instructions on how to do this, see C.

Creating a new Windows Forms application

The first step is to create a new Windows Forms application.

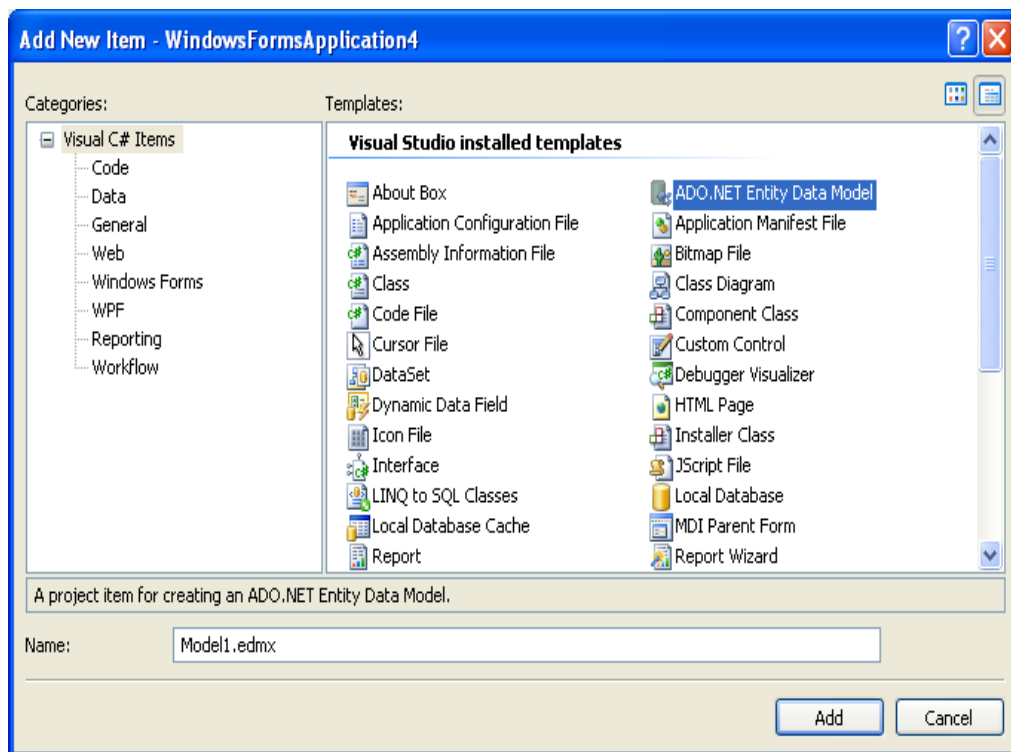
1. In Visual Studio, select **File, New, Project** from the main menu.
2. Choose the **Windows Forms Application** installed template. Click **OK**. The solution is created.

Adding an Entity Data Model

You will now add an Entity Data Model to your solution.

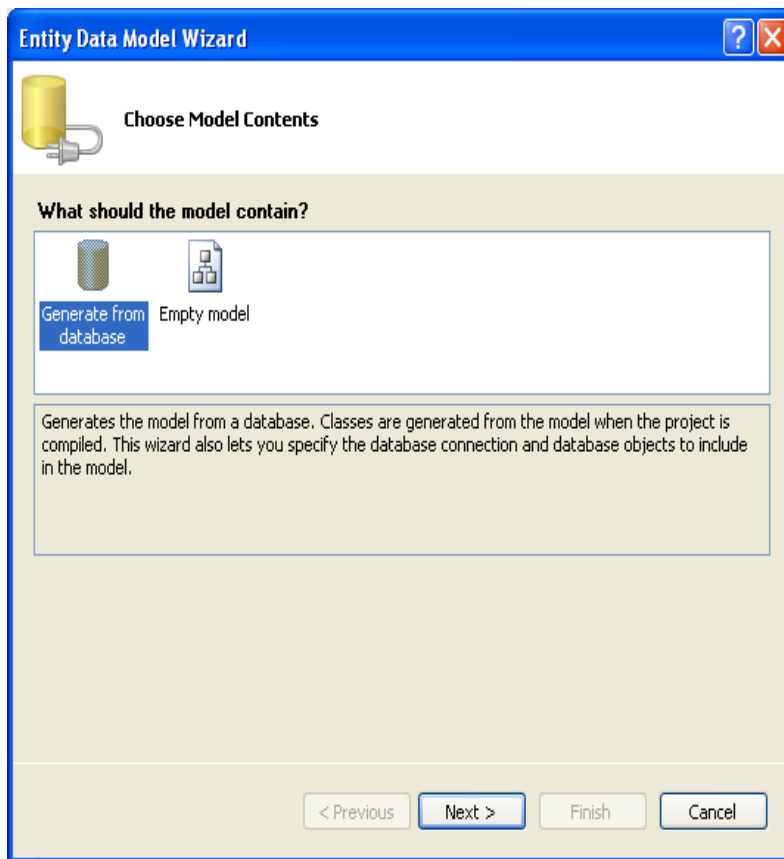
1. In the Solution Explorer, right-click your application and select **Add, New Item....** From **Visual Studio installed templates** select **ADO.NET Entity Data Model**. Click **Add**.

Figure 4.28 Add Entity Data Model

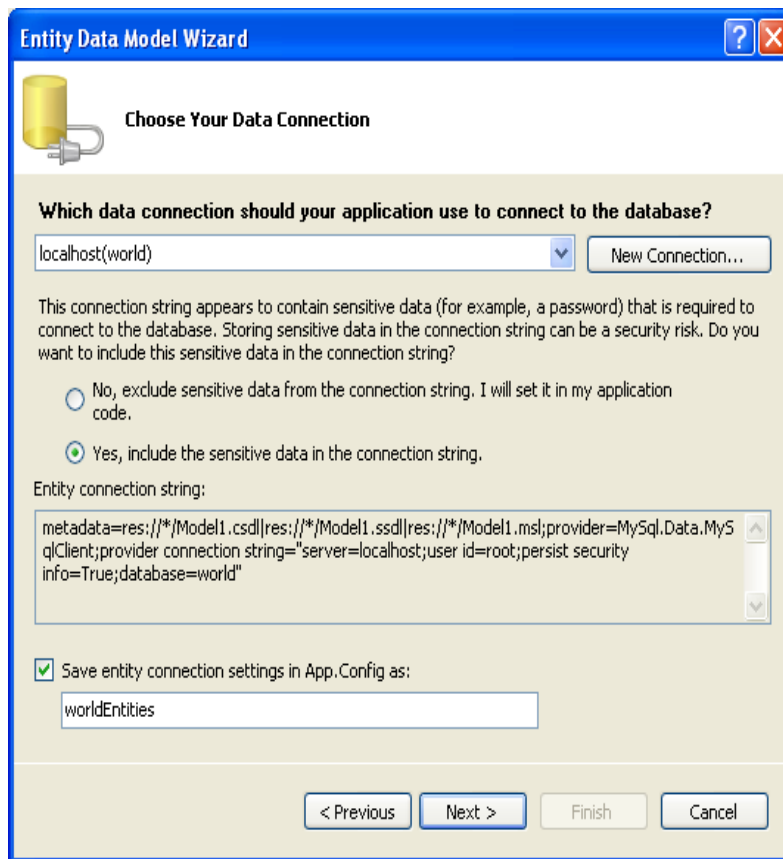


2. You will now see the Entity Data Model Wizard. You will use the wizard to generate the Entity Data Model from the [world](#) database sample. Select the icon **Generate from database**. Click **Next**.

Figure 4.29 Entity Data Model Wizard Screen 1



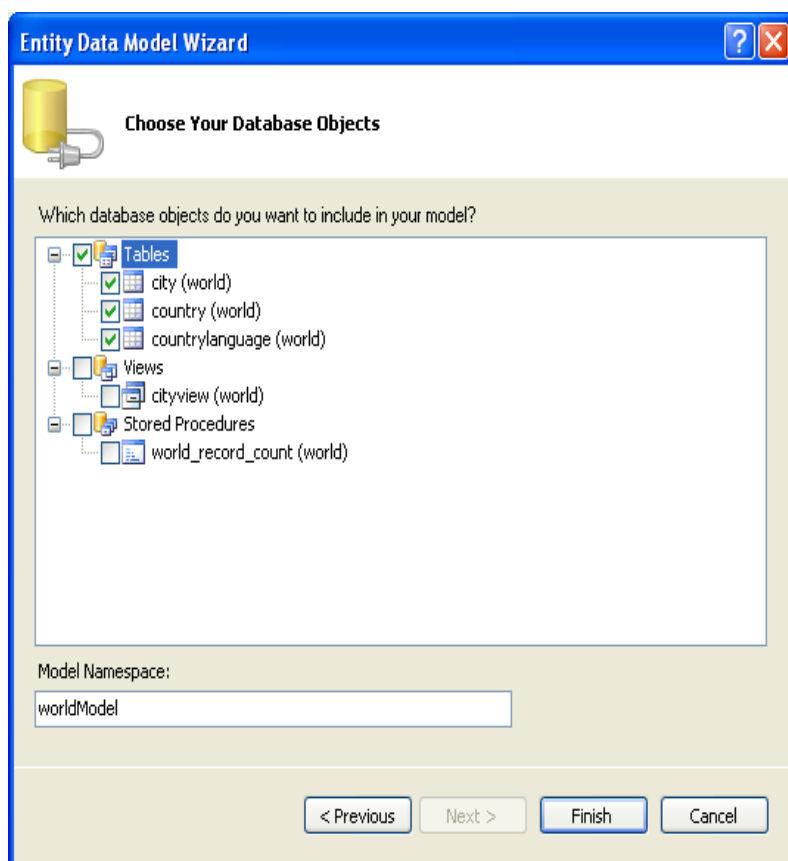
3. You can now select the connection you made earlier to the [world](#) database. If you have not already done so, you can create the new connection at this time by clicking **New Connection....** For further instructions on creating a connection to a database see [Making a Connection](#).

Figure 4.30 Entity Data Model Wizard Screen 2

The screenshot shows the 'Entity Data Model Wizard' window, specifically the 'Choose Your Data Connection' screen. The window has a blue title bar with the text 'Entity Data Model Wizard' and standard Windows window controls. Below the title bar is a yellow cylinder icon with a plug. The main area is titled 'Choose Your Data Connection'. It asks 'Which data connection should your application use to connect to the database?'. A dropdown menu shows 'localhost(world)' and a 'New Connection...' button is next to it. Below this, a warning message states: 'This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?'. There are two radio buttons: 'No, exclude sensitive data from the connection string. I will set it in my application code.' (which is selected) and 'Yes, include the sensitive data in the connection string.'. Below the radio buttons is a text box labeled 'Entity connection string:' containing the string: 'metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider=MySql.Data.MySqlClient;provider connection string="server=localhost;user id=root;persist security info=True;database=world"'. At the bottom, there is a checkbox 'Save entity connection settings in App.Config as:' which is checked, followed by a text box containing 'worldEntities'. At the very bottom are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'.

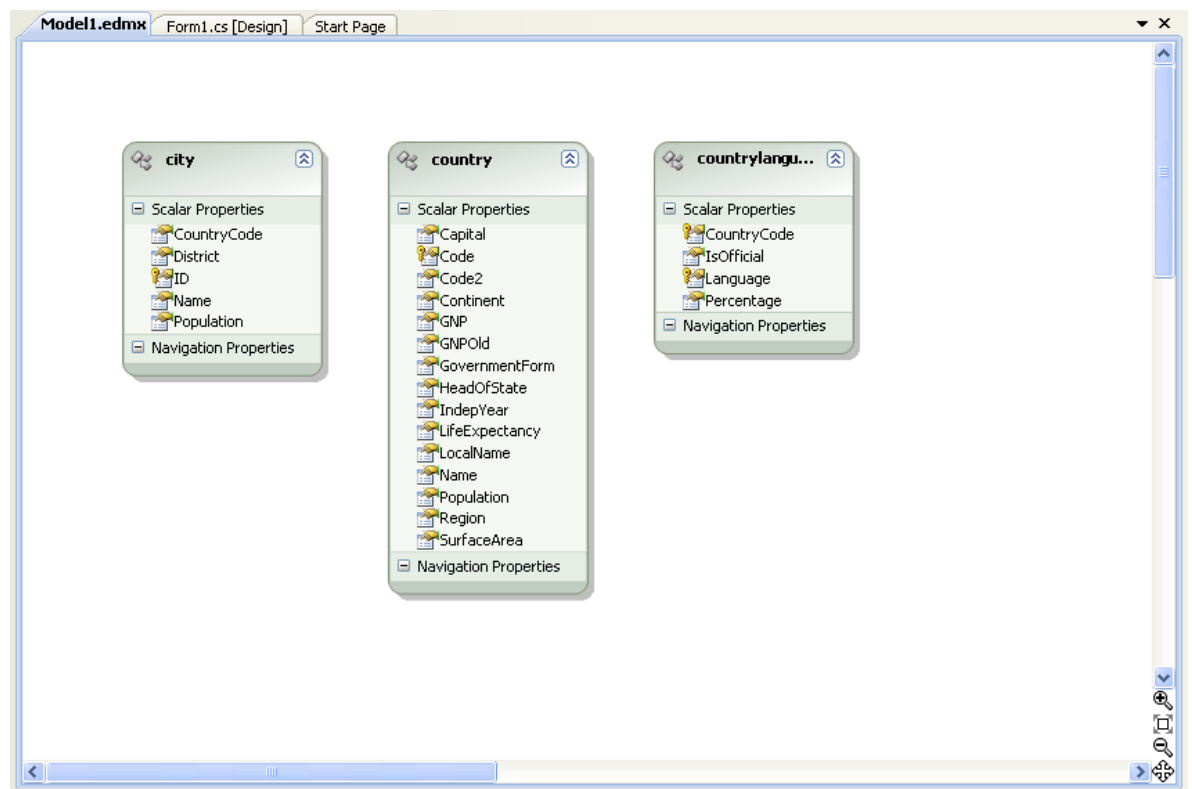
4. Make a note of the entity connection settings to be used in App.Config, as these will be used later to write the necessary control code.
5. Click **Next**.
6. The Entity Data Model Wizard connects to the database. You are then presented with a tree structure of the database. From this you can select the object you would like to include in your model. If you had created Views and Stored Routines these will be displayed along with any tables. In this example you just need to select the tables. Click **Finish** to create the model and exit the wizard.

Figure 4.31 Entity Data Model Wizard Screen 3



7. Visual Studio will generate the model and then display it.

Figure 4.32 Entity Data Model Diagram



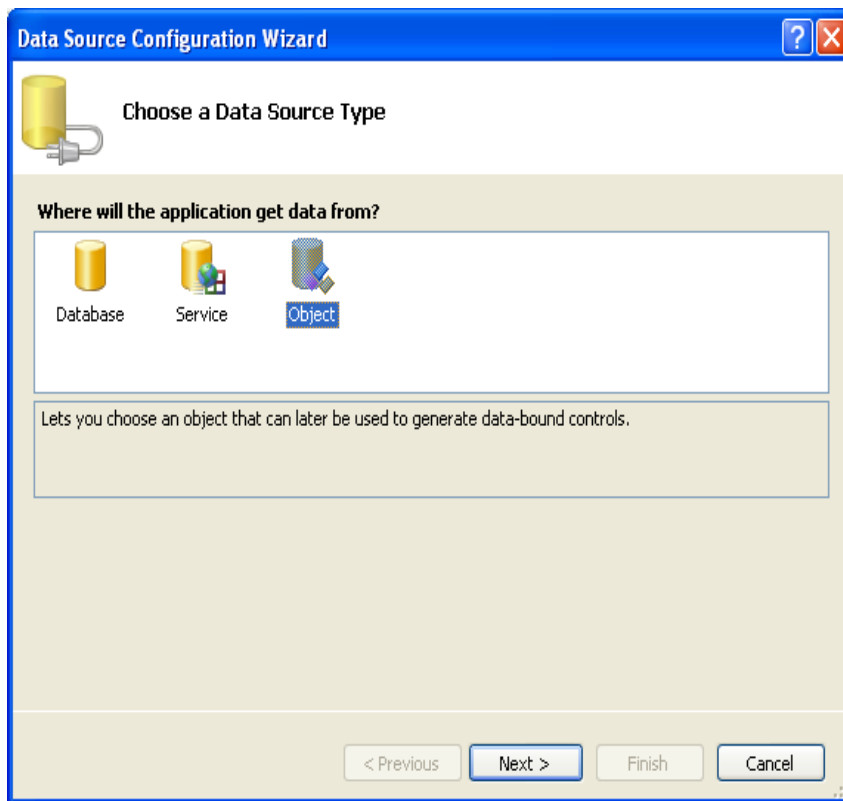
8. From the Visual Studio main menu select **Build, Build Solution**, to ensure that everything compiles correctly so far.

Adding a new Data Source

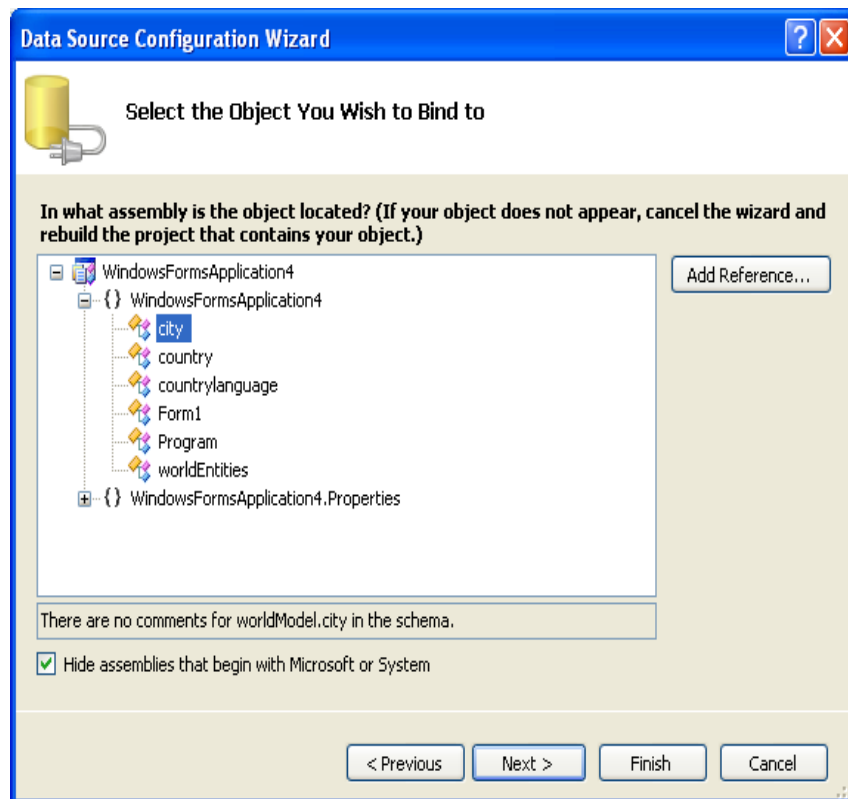
You will now add a new Data Source to your project and see how it can be used to read and write to the database.

1. From the Visual Studio main menu select **Data, Add New Data Source....** You will be presented with the Data Source Configuration Wizard.

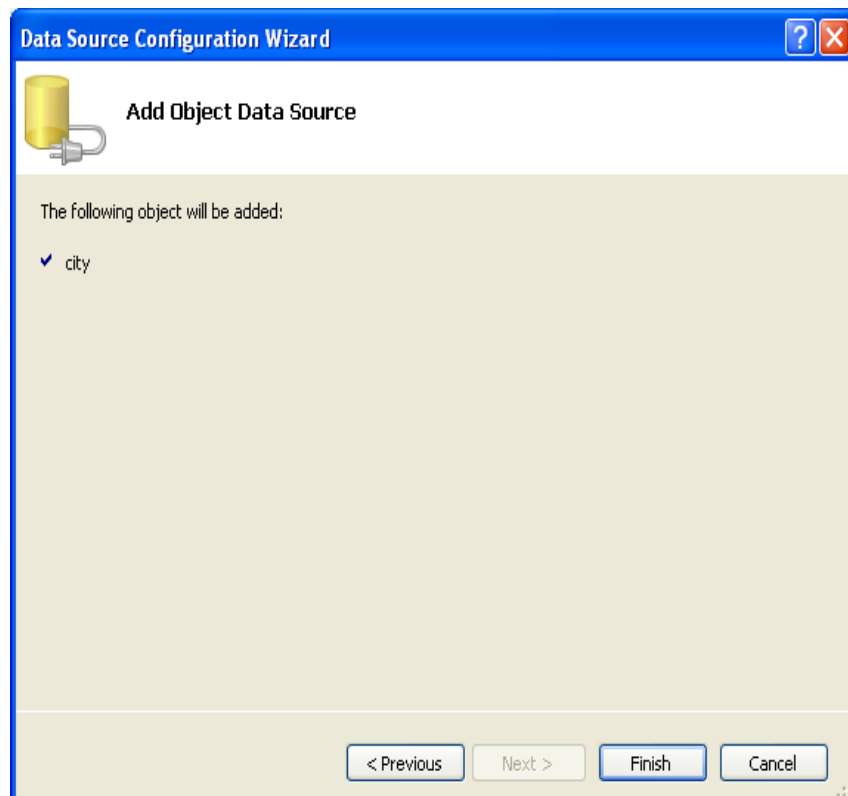
Figure 4.33 Entity Data Source Configuration Wizard Screen 1



2. Select the **Object** icon. Click **Next**.
3. You will now select the Object to bind to. Expand the tree. In this tutorial, you will select the city table. Once the city table has been selected click **Next**.

Figure 4.34 Entity Data Source Configuration Wizard Screen 2

4. The wizard will confirm that the city object is to be added. Click **Finish**.

Figure 4.35 Entity Data Source Configuration Wizard Screen 3

5. The city object will be display in the Data Sources panel. If the Data Sources panel is not displayed, select **Data, Show Data Sources** from the Visual Studio main menu. The docked panel will then be displayed.

Figure 4.36 Data Sources



Using the Data Source in a Windows Form

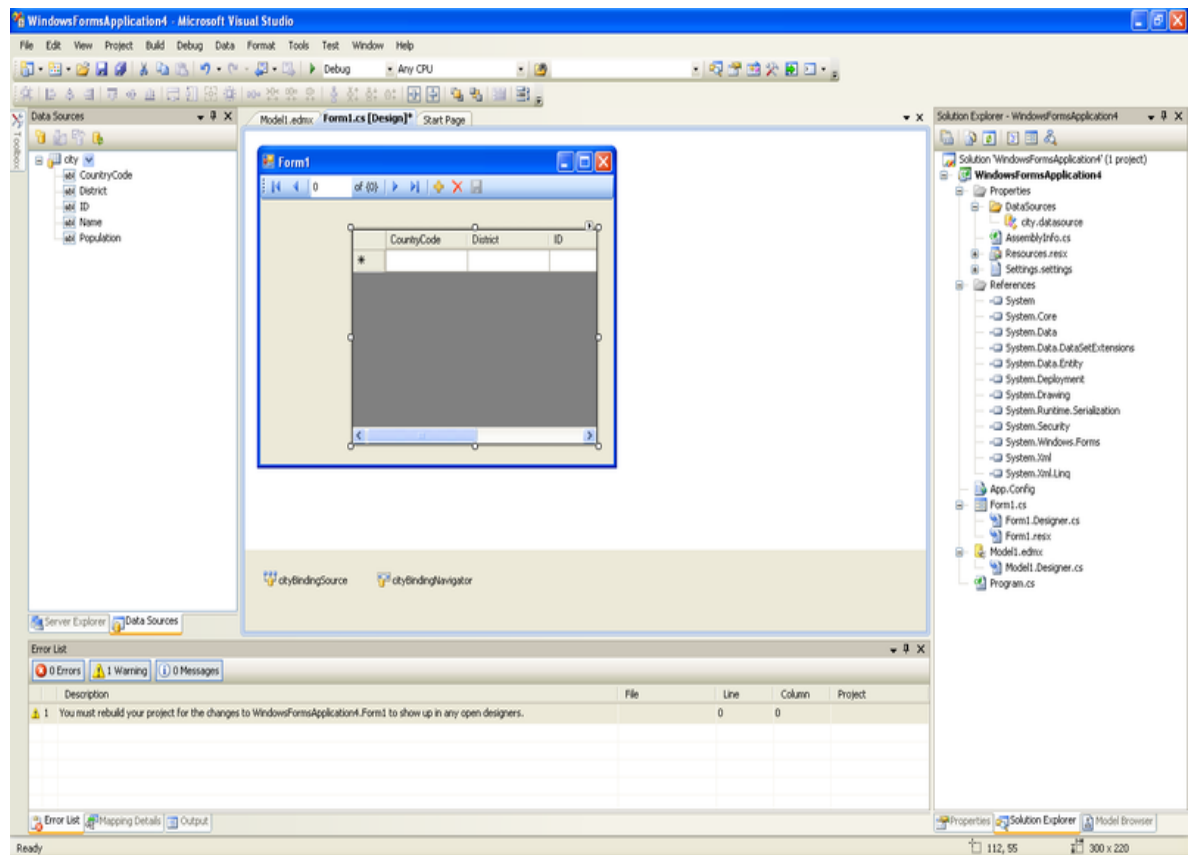
You will now learn how to use the Data Source in a Windows Form.

1. In the Data Sources panel select the Data Source you just created and drag and drop it onto the Form Designer. By default the Data Source object will be added as a Data Grid View control.

Note

The Data Grid View control is bound to `cityBindingSource`, and the Navigator control is bound to `cityBindingNavigator`.

Figure 4.37 Data Form Designer

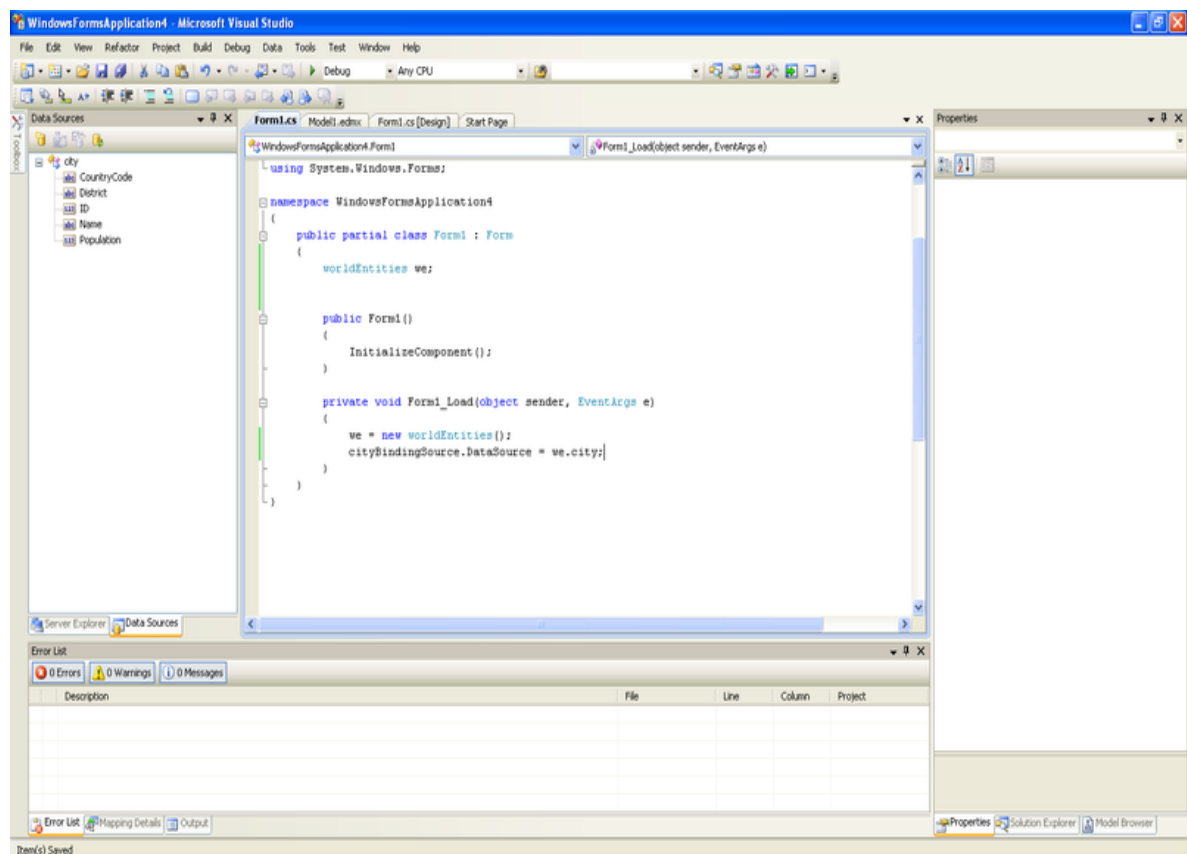


2. Save and rebuild the solution before continuing.

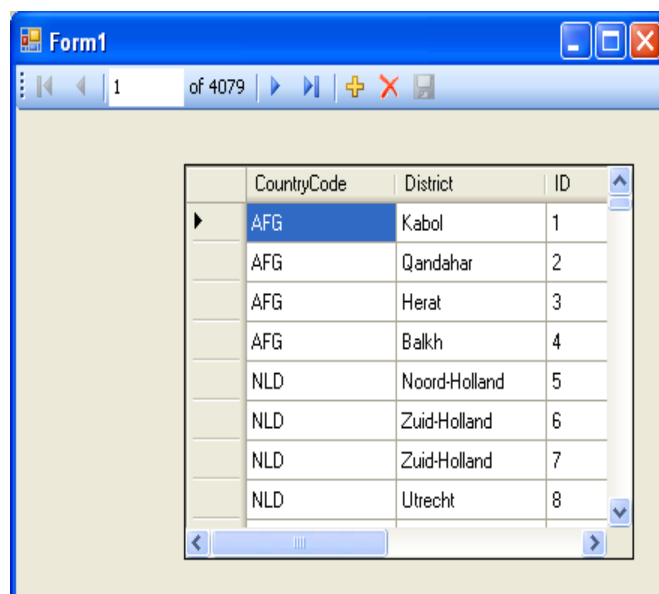
Adding Code to Populate the Data Grid View

You are now ready to add code to ensure that the Data Grid View control will be populated with data from the City database table.

1. Double-click the form to access its code.
2. Add code to instantiate the Entity Data Model's EntityContainer object and retrieve data from the database to populate the control.

Figure 4.38 Adding Code to the Form

3. Save and rebuild the solution.
4. Run the solution. Ensure the grid is populated and you can navigate the database.

Figure 4.39 The Populated Grid Control

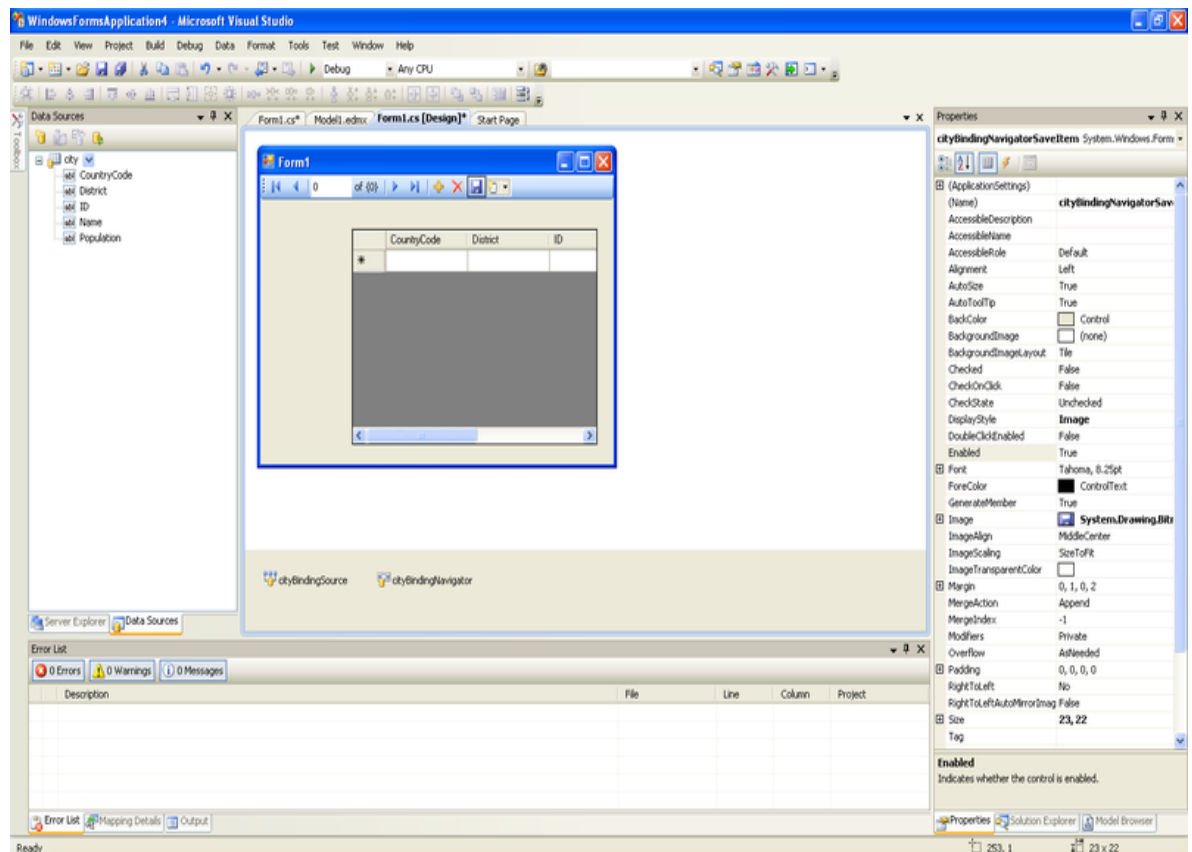
Adding Code to Save Changes to the Database

You will now add code to enable you to save changes to the database.

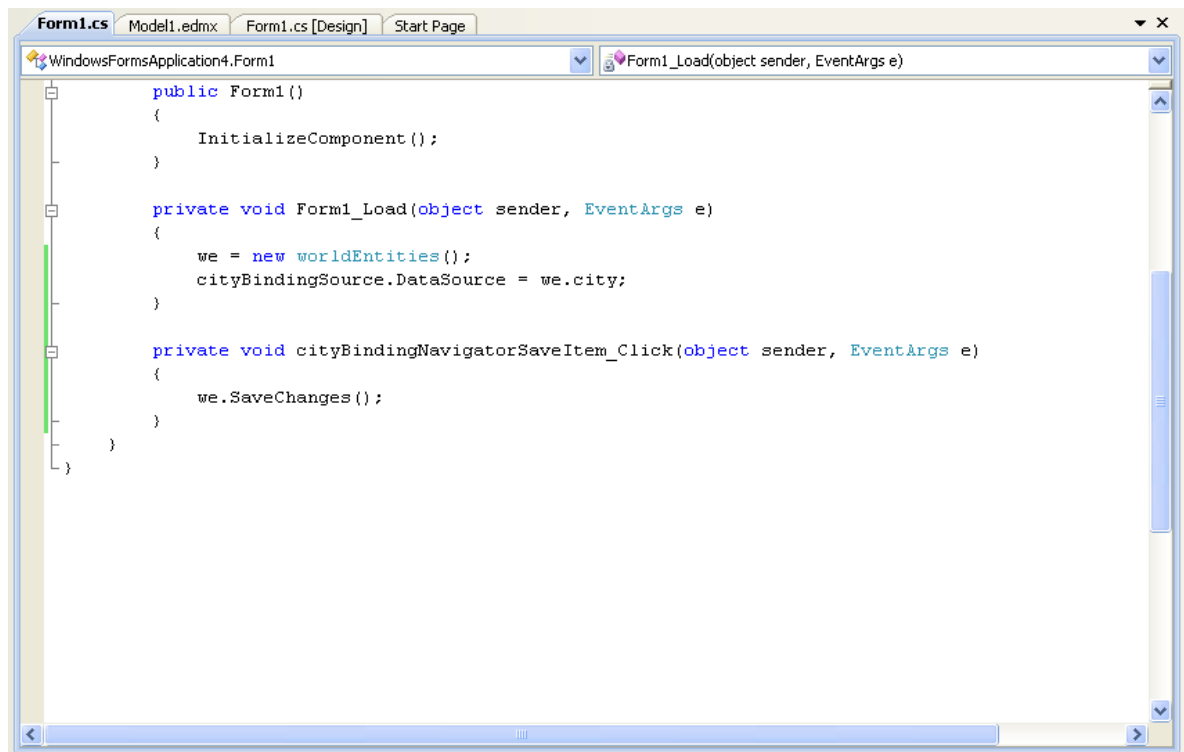
The Binding source component ensures that changes made in the Data Grid View control are also made to the Entity classes bound to it. However, that data needs to be saved back from the entities to the database itself. This can be achieved by the enabling of the Save button in the Navigator control, and the addition of some code.

1. In the Form Designer, click the Save icon in the Form toolbar and ensure that its Enabled property is set to True.

Figure 4.40 Save Button Enabled



2. Double-click the Save icon in the Form toolbar to display its code.
3. You now need to add code to ensure that data is saved to the database when the save button is clicked in the application.

Figure 4.41 Adding Save Code to the Form

4. Once the code has been added, save the solution and rebuild it. Run the application and verify that changes made in the grid are saved.

4.4.7 Tutorial: Databinding in ASP.NET Using LINQ on Entities

In this tutorial you create an ASP.NET web page that binds LINQ queries to entities using the Entity Framework mapping.

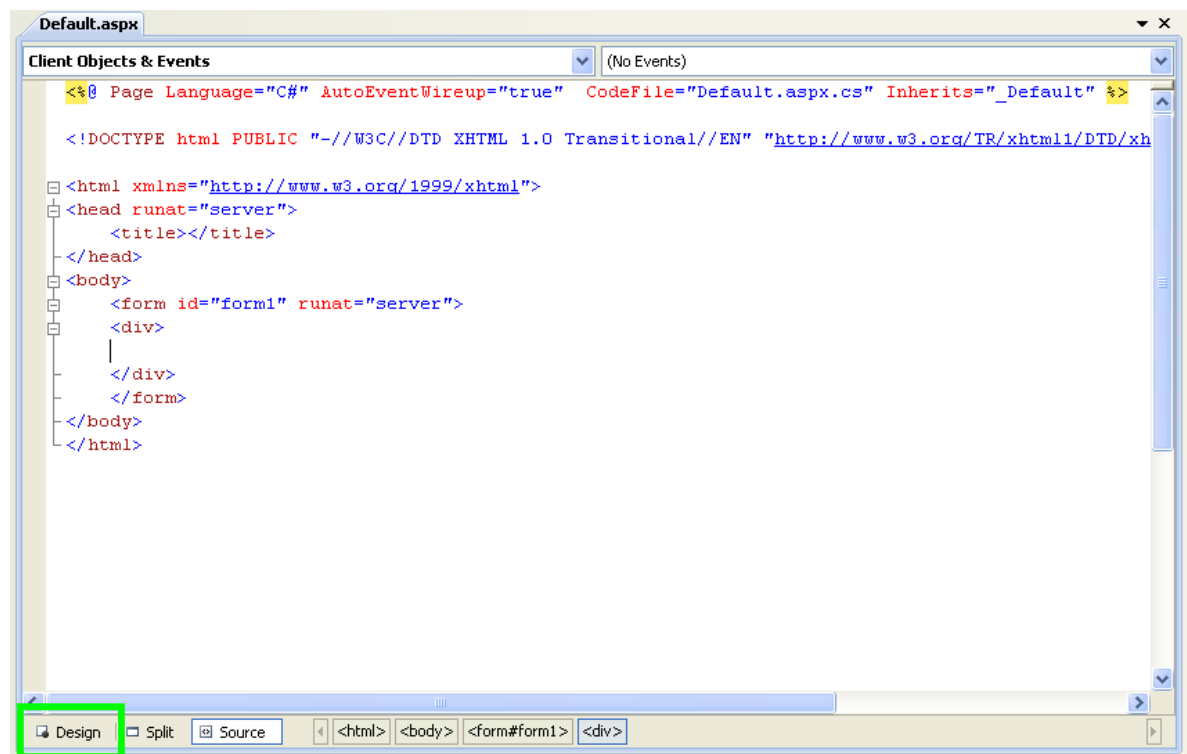
If you have not already done so, install the [world](#) database sample prior to attempting this tutorial. See the tutorial [Section 4.4.6, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source"](#) for instructions on downloading and installing this database.

Creating an ASP.NET web site

In this part of the tutorial, you create an ASP.NET web site. The web site uses the [world](#) database. The main web page features a drop down list from which you can select a country. Data about that country's cities is then displayed in a grid view control.

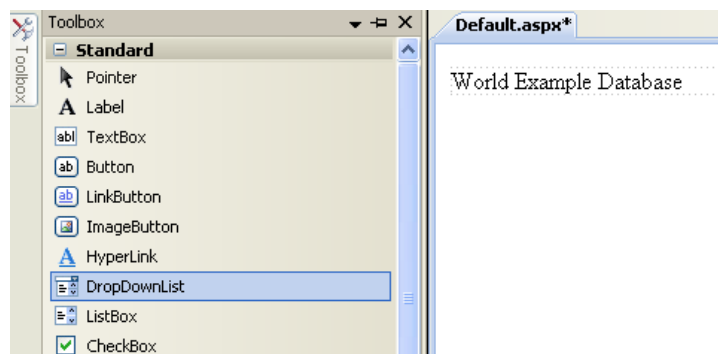
1. From the Visual Studio main menu select **File, New, Web Site...**
2. From the Visual Studio installed templates select **ASP.NET Web Site**. Click **OK**. You will be presented with the Source view of your web page by default.
3. Click the Design view tab situated underneath the Source view panel.

Figure 4.42 The Design Tab



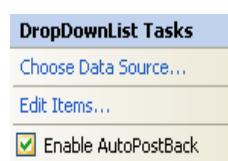
4. In the Design view panel, enter some text to decorate the blank web page.
5. Click Toolbox. From the list of controls select **DropDownList**. Drag and drop the control to a location beneath the text on your web page.

Figure 4.43 Drop Down List

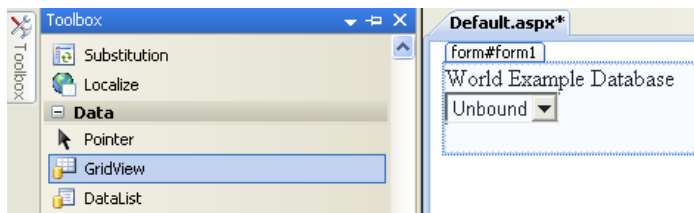


6. From the **DropDownList** control's context menu, ensure that the **Enable AutoPostBack** check box is enabled. This will ensure the control's event handler is called when an item is selected. The user's choice will in turn be used to populate the **GridView** control.

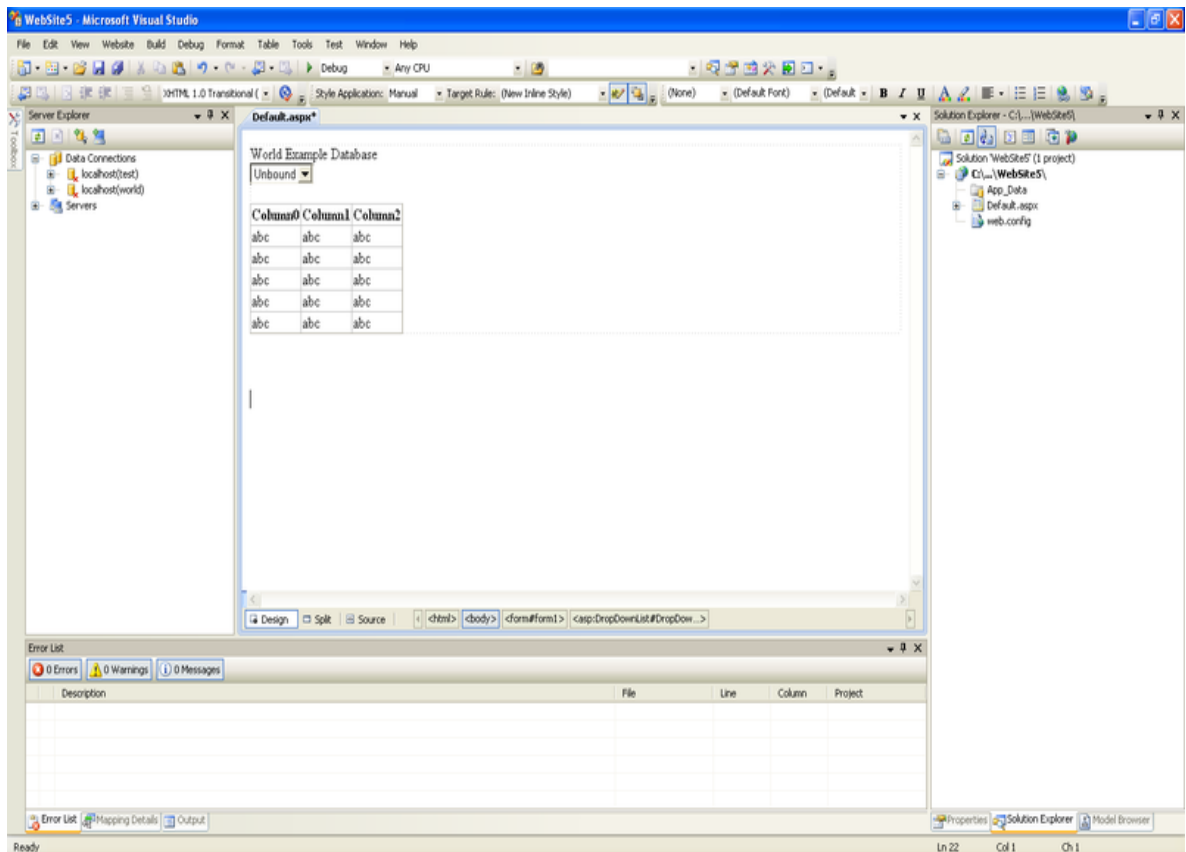
Figure 4.44 Enable AutoPostBack



7. From the Toolbox select the **GridView** control.

Figure 4.45 Grid View Control

Drag and drop the Grid View control to a location just below the Drop Down List you already placed.

Figure 4.46 Placed Grid View Control

8. At this point it is recommended that you save your solution, and build the solution to ensure that there are no errors.
9. If you run the solution you will see that the text and drop down list are displayed, but the list is empty. Also, the grid view does not appear at all. Adding this functionality is described in the following sections.

At this stage you have a web site that will build, but further functionality is required. The next step will be to use the Entity Framework to create a mapping from the `world` database into entities that you can control programmatically.

Creating an ADO.NET Entity Data Model

In this stage of the tutorial you will add an ADO.NET Entity Data Model to your project, using the `world` database at the storage level. The procedure for doing this is described in the tutorial [Section 4.4.6, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source"](#), and so will not be repeated here.

Populating a Drop Data List Box with using the results of a entity LINQ query

In this part of the tutorial you will write code to populate the DropDownList control. When the web page loads the data to populate the list will be achieved by using the results of a LINQ query on the model created previously.

1. In the Design view panel, double-click any blank area. This brings up the [Page_Load](#) method.
2. Modify the relevant section of code according to the following listing:

```
...
public partial class _Default : System.Web.UI.Page
{
    worldModel.worldEntities we;
    protected void Page_Load(object sender, EventArgs e)
    {
        we = new worldModel.worldEntities();
        if (!IsPostBack)
        {
            var countryQuery = from c in we.country
                               orderby c.Name
                               select new { c.Code, c.Name };
            DropDownList1.DataValueField = "Code";
            DropDownList1.DataTextField = "Name";
            DropDownList1.DataSource = countryQuery;
            DataBind();
        }
    }
    ...
}
```

The list control only needs to be populated when the page first loads. The conditional code ensures that if the page is subsequently reloaded, the list control is not repopulated, which would cause the user selection to be lost.

3. Save the solution, build it and run it. You should see the list control has been populated. You can select an item, but as yet the grid view control does not appear.

At this point you have a working Drop Down List control, populated by a LINQ query on your entity data model.

Populating a Grid View control using an entity LINQ query

In the last part of this tutorial you will populate the Grid View Control using a LINQ query on your entity data model.

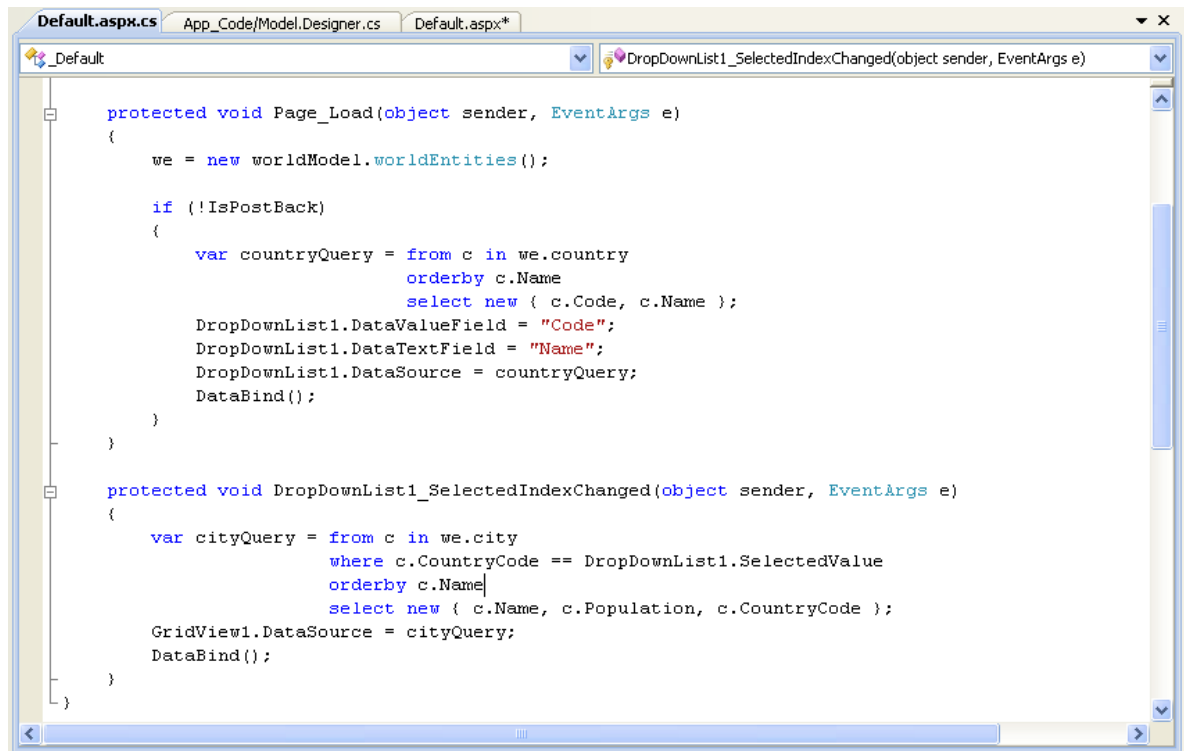
1. In the Design view, double-click the **DropDownList** control. This causes its [SelectedIndexChanged](#) code to be displayed. This method is called when a user selects an item in the list control and thus fires an AutoPostBack event.
2. Modify the relevant section of code accordingly to the following listing:

```
...
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    var cityQuery = from c in we.city
                    where c.CountryCode == DropDownList1.SelectedValue
                    orderby c.Name
                    select new { c.Name, c.Population, c.CountryCode };
    GridView1.DataSource = cityQuery;
    DataBind();
}
...
```

The grid view control is populated from the result of the LINQ query on the entity data model.

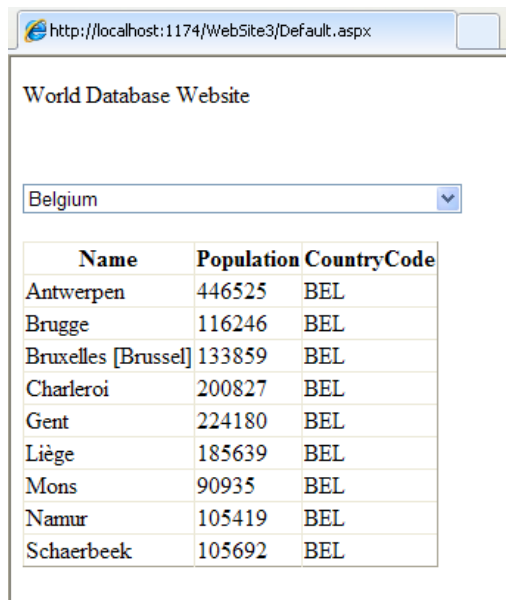
3. As a check compare your code to that shown in the following screenshot:

Figure 4.47 Source Code



4. Save, build and run the solution. As you select a country you will see its cities are displayed in the grid view control.

Figure 4.48 The Working Web Site



In this tutorial you have seen how to create an ASP.NET web site, you have also seen how you can access a MySQL database using LINQ queries on an entity data model.

4.4.8 Tutorial: Using SSL with MySQL Connector/Net

In this tutorial you will learn how you can use MySQL Connector/Net to connect to a MySQL server configured to use SSL. Support for SSL client certificates was added with MySQL Connector/Net 6.2.

MySQL Server uses the PEM format for certificates and private keys. This tutorial will use the test certificates from the server test suite by way of example. You can obtain the MySQL Server source code from [MySQL Downloads](#). The certificates can be found in the directory `./mysql-test/std_data`.

To carry out the steps in this tutorial, you must have Open SSL installed. This can be downloaded for Microsoft Windows at no charge from [Shining Light Productions](#).

Further details on the connection string options used in this tutorial can be found at [Section 4.6, "Connector/Net Connection String Options Reference"](#).

Configuring the MySQL Server to use SSL

1. In the MySQL Server configuration file, set the SSL parameters as follows:

```
ssl-ca=path/to/repo/mysql-test/std_data/cacert.pem
ssl-cert=path/to/repo/mysql-test/std_data/server-cert.pem
ssl-key=path/to/repo/mysql-test/std_data/server-key.pem
```

Adjust the directories according to the location in which you installed the MySQL source code.

2. In this step you create a test user and set the user to require SSL.

Using the MySQL Command-Line Client, connect as `root` and create the user `sslclient`.

3. To set privileges and requirements, issue the following command:

```
GRANT ALL PRIVILEGES ON *.* TO sslclient@'%' REQUIRE SSL;
```

Creating a certificate file to use with the .NET client

1. The .NET client does not use the PEM file format, as .NET does not support this format natively. You will be using test client certificates from the same server repository, for the purposes of this example. Convert these to PFX format first. This format is also known as PKCS#12. An article describing this procedure can be found at the [Citrix website](#). From the directory `server-repository-root/mysql-test/std_data`, issue the following command:

```
openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem -certfile cacert.pem -out client.pfx
```

2. When asked for an export password, enter the password "pass". The file `client.pfx` will be generated. This file is used in the remainder of the tutorial.

Connecting to the server using a file-based certificate

1. You will use PFX file, `client.pfx` you created in the previous step to authenticate the client. The following example demonstrates how to connect using the `SSL Mode`, `CertificateFile` and `CertificatePassword` connection string options:

```
using (MySQLConnection connection = new MySQLConnection(
    "database=test;user=sslclient;" +
    "CertificateFile=H:\\bzt\\mysql-trunk\\mysql-test\\std_data\\client.pfx" +
    "CertificatePassword=pass;" +
    "SSL Mode=Required "))
{
    connection.Open();
}
```

The path to the certificate file will need to be changed to reflect your individual installation.

Connecting to the server using a store-based certificate

1. The first step is to import the PFX file, `client.pfx`, into the Personal Store. Double-click the file in Windows explorer. This launches the Certificate Import Wizard.

2. Follow the steps dictated by the wizard, and when prompted for the password for the PFX file, enter "pass".
3. Click **Finish** to close the wizard and import the certificate into the personal store.

Examine certificates in the Personal Store

1. Start the Microsoft Management Console by entering `mmc.exe` at a command prompt.
2. Select **File, Add/Remove snap-in**. Click **Add**. Select **Certificates** from the list of available snap-ins in the dialog.
3. Click **Add** button in the dialog, and select the **My user account** radio button. This is used for personal certificates.
4. Click the **Finish** button.
5. Click **OK** to close the Add/Remove Snap-in dialog.
6. You will now have **Certificates – Current User** displayed in the left panel of the Microsoft Management Console. Expand the Certificates - Current User tree item and select **Personal, Certificates**. The right-hand panel will display a certificate issued to MySQL. This is the certificate that was previously imported. Double-click the certificate to display its details.
7. After you have imported the certificate to the Personal Store, you can use a more succinct connection string to connect to the database, as illustrated by the following code:

```
using (MySqlConnection connection = new MySqlConnection(
    "database=test;user=sslclient;" +
    "Certificate Store Location=CurrentUser;" +
    "SSL Mode=Required"))
{
    connection.Open();
}
```

Certificate Thumbprint Parameter

If you have a large number of certificates in your store, and many have the same Issuer, this can be a source of confusion and result in the wrong certificate being used. To alleviate this situation, there is an optional Certificate Thumbprint parameter that can additionally be specified as part of the connection string. As mentioned before, you can double-click a certificate in the Microsoft Management Console to display the certificate's details. When the Certificate dialog is displayed click the **Details** tab and scroll down to see the thumbprint. The thumbprint will typically be a number such as `#47 94 36 00 9a 40 f3 01 7a 14 5c f8 47 9e 76 94 d7 aa de f0`. This thumbprint can be used in the connection string, as the following code illustrates:

```
using (MySqlConnection connection = new MySqlConnection(
    "database=test;user=sslclient;" +
    "Certificate Store Location=CurrentUser;" +
    "Certificate Thumbprint=479436009a40f3017a145cf8479e7694d7aadeef0;" +
    "SSL Mode=Required"))
{
    connection.Open();
}
```

Spaces in the thumbprint parameter are optional and the value is case-insensitive.

4.4.9 Tutorial: Using MySqlScript

This tutorial teaches you how to use the `MySqlScript` class. This class enables you to execute a series of statements. Depending on the circumstances, this can be more convenient than using the `MySqlCommand` approach.

Further details of the [MySqlScript](#) class can be found in the reference documentation supplied with MySQL Connector/Net.

To run the example programs in this tutorial, set up a simple test database and table using the [mysql](#) Command-Line Client or MySQL Workbench. Commands for the [mysql](#) Command-Line Client are given here:

```
CREATE DATABASE TestDB;
USE TestDB;
CREATE TABLE TestTable (id INT NOT NULL PRIMARY KEY
    AUTO_INCREMENT, name VARCHAR(100));
```

The main method of the [MySqlScript](#) class is the [Execute](#) method. This method causes the script (sequence of statements) assigned to the [Query](#) property of the [MySqlScript](#) object to be executed. The [Query](#) property can be set through the [MySqlScript](#) constructor or by using the [Query](#) property. [Execute](#) returns the number of statements executed.

The [MySqlScript](#) object will execute the specified script on the connection set using the [Connection](#) property. Again, this property can be set directly or through the [MySqlScript](#) constructor. The following code snippets illustrate this:

```
string sql = "SELECT * FROM TestTable";
...
MySqlScript script = new MySqlScript(conn, sql);
...
MySqlScript script = new MySqlScript();
script.Query = sql;
script.Connection = conn;
...
script.Execute();
```

The [MySqlScript](#) class has several events associated with it. There are:

1. [Error](#) - generated if an error occurs.
2. [ScriptCompleted](#) - generated when the script successfully completes execution.
3. [StatementExecuted](#) - generated after each statement is executed.

It is possible to assign event handlers to each of these events. These user-provided routines are called back when the connected event occurs. The following code shows how the event handlers are set up.

```
script.Error += new MySqlScriptErrorHandler(script_Error);
script.ScriptCompleted += new EventHandler(script_ScriptCompleted);
script.StatementExecuted += new MySqlStatementExecutedEventHandler(script_StatementExecuted);
```

In VisualStudio, you can save typing by using tab completion to fill out stub routines. Start by typing, for example, “`script.Error +=`”. Then press **TAB**, and then press **TAB** again. The assignment is completed, and a stub event handler created. A complete working example is shown below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace MySqlScriptTest
{
    class Program
    {
```

```

static void Main(string[] args)
{
    string connStr = "server=localhost;user=root;database=TestDB;port=3306;password=*****;";
    MySqlConnection conn = new MySqlConnection(connStr);

    try
    {
        Console.WriteLine("Connecting to MySQL...");
        conn.Open();

        string sql = "INSERT INTO TestTable(name) VALUES ('Superman');" +
                     "INSERT INTO TestTable(name) VALUES ('Batman');" +
                     "INSERT INTO TestTable(name) VALUES ('Wolverine');" +
                     "INSERT INTO TestTable(name) VALUES ('Storm');"

        MySqlScript script = new MySqlScript(conn, sql);

        script.Error += new MySqlScriptErrorHandler(script_Error);
        script.ScriptCompleted += new EventHandler(script_ScriptCompleted);
        script.StatementExecuted += new MySqlStatementExecutedEventHandler(script_StatementExecuted);

        int count = script.Execute();

        Console.WriteLine("Executed " + count + " statement(s).");
        Console.WriteLine("Delimiter: " + script.Delimiter);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }

    conn.Close();
    Console.WriteLine("Done.");
}

static void script_StatementExecuted(object sender, MySqlScriptEventArgs args)
{
    Console.WriteLine("script_StatementExecuted");
}

static void script_ScriptCompleted(object sender, EventArgs e)
{
    /// EventArgs e will be EventArgs.Empty for this method
    Console.WriteLine("script_ScriptCompleted!");
}

static void script_Error(Object sender, MySqlScriptErrorEventArgs args)
{
    Console.WriteLine("script_Error: " + args.Exception.ToString());
}
}

```

In the `script_ScriptCompleted` event handler, the `EventArgs` parameter `e` will be `EventArgs.Empty`. In the case of the `ScriptCompleted` event there is no additional data to be obtained, which is why the event object is `EventArgs.Empty`.

4.4.9.1 Using Delimiters with MySqlScript

Depending on the nature of the script, you may need control of the delimiter used to separate the statements that will make up a script. The most common example of this is where you have a multi-statement stored routine as part of your script. In this case if the default delimiter of “;” is used you will get an error when you attempt to execute the script. For example, consider the following stored routine:

```

CREATE PROCEDURE test_routine()
BEGIN
    SELECT name FROM TestTable ORDER BY name;
    SELECT COUNT(name) FROM TestTable;
END

```


This routine actually needs to be executed on the MySQL Server as a single statement. However, with the default delimiter of “;”, the [MySqlScript](#) class would interpret the above as two statements, the first being:

```
CREATE PROCEDURE test_routine()
BEGIN
    SELECT name FROM TestTable ORDER BY name;
```

Executing this as a statement would generate an error. To solve this problem [MySqlScript](#) supports the ability to set a different delimiter. This is achieved through the `Delimiter` property. For example, you could set the delimiter to “??”, in which case the above stored routine would no longer generate an error when executed. Multiple statements can be delimited in the script, so for example, you could have a three statement script such as:

```
string sql = "DROP PROCEDURE IF EXISTS test_routine??" +
    "CREATE PROCEDURE test_routine() " +
    "BEGIN " +
    "SELECT name FROM TestTable ORDER BY name;" +
    "SELECT COUNT(name) FROM TestTable;" +
    "END??" +
    "CALL test_routine()";
```

You can change the delimiter back at any point by setting the `Delimiter` property. The following code shows a complete working example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MySql.Data;
using MySql.Data.MySqlClient;

namespace ConsoleApplication8
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=TestDB;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                string sql = "DROP PROCEDURE IF EXISTS test_routine??" +
                    "CREATE PROCEDURE test_routine() " +
                    "BEGIN " +
                    "SELECT name FROM TestTable ORDER BY name;" +
                    "SELECT COUNT(name) FROM TestTable;" +
                    "END??" +
                    "CALL test_routine()";

                MySqlScript script = new MySqlScript(conn);

                script.Query = sql;
                script.Delimiter = "??";
                int count = script.Execute();
                Console.WriteLine("Executed " + count + " statement(s)");
                script.Delimiter = ";";
                Console.WriteLine("Delimiter: " + script.Delimiter);
                Console.WriteLine("Query: " + script.Query);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
        }
    }
}
```

```
    }  
    conn.Close();  
    Console.WriteLine("Done.");  
  }  
}
```

4.4.10 Tutorial: Generating MySQL DDL from an Entity Framework Model

In this tutorial, you will learn how to create MySQL [DDL](#) from an Entity Framework model. Use Visual Studio 2010 and MySQL Connector/Net 6.3 to carry out this tutorial.

1. Create a new console application in Visual Studio 2010.
2. Using the **Solution Explorer**, add a reference to [MySQL.Data.Entity](#).
3. From the **Solution Explorer** select **Add, New Item**. In the **Add New Item** dialog select **Online Templates**. Select **ADO.NET Entity Data Model** and click **Add**. The **Entity Data Model** dialog will be displayed.
4. In the **Entity Data Model** dialog select **Empty Model**. Click **Finish**. A blank model will be created.
5. Create a simple model. A single Entity will do for the purposes of this tutorial.
6. In the **Properties** panel select **ConceptualEntityModel** from the drop-down listbox.
7. In the **Properties** panel, locate the **DDL Generation Template** in the category **Database Script Generation**.
8. For the **DDL Generation** property select **SSDLToMySQL.tt(VS)** from the drop-down listbox.
9. Save the solution.
10. Right-click an empty space in the model design area. The context-sensitive menu will be displayed.
11. From the context-sensitive menu select **Generate Database from Model**. The **Generate Database Wizard** dialog will be displayed.
12. In the **Generate Database Wizard** dialog select an existing connection, or create a new connection to a server. Select an appropriate radio button to show or hide sensitive data. For the purposes of this tutorial you can select **Yes** (although you might skip this for commercial applications).
13. Click **Next**. MySQL compatible DDL code will be generated. Click **Finish** to exit the wizard.

You have seen how to create MySQL DDL code from an Entity Framework model.

4.5 Connector/Net Programming

Connector/Net comprises several classes that are used to connect to the database, execute queries and statements, and manage query results.

The following are the major classes of Connector/Net:

- [MySQLCommand](#): Represents an SQL statement to execute against a MySQL database.
- [MySQLCommandBuilder](#): Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated MySQL database.
- [MySQLConnection](#): Represents an open connection to a MySQL Server database.
- [MySQLDataAdapter](#): Represents a set of data commands and a database connection that are used to fill a data set and update a MySQL database.
- [MySQLDataReader](#): Provides a means of reading a forward-only stream of rows from a MySQL database.

- [MySqlException](#): The exception that is thrown when MySQL returns an error.
- [MySqlHelper](#): Helper class that makes it easier to work with the provider.
- [MySqlTransaction](#): Represents an SQL [transaction](#) to be made in a MySQL database.

In the following sections, you will learn about some common use cases for Connector/Net, including BLOB handling, date handling, and using Connector/Net with common tools such as Crystal Reports.

4.5.1 Connecting to MySQL Using Connector/Net

All interaction between a .NET application and the MySQL server is routed through a [MySqlConnection](#) object. Before your application can interact with the server, it must instantiate, configure, and open a [MySqlConnection](#) object.

Even when using the [MySqlHelper](#) class, a [MySqlConnection](#) object is created by the helper class.

This section describes how to connect to MySQL using the [MySqlConnection](#) object.

4.5.2 Creating a Connector/Net Connection String

The [MySqlConnection](#) object is configured using a connection string. A connection string contains several key/value pairs, separated by semicolons. In each key/value pair, the option name and its corresponding value are joined by an equal sign. For the list of option names to use in the connection string, see [Section 4.6, “Connector/Net Connection String Options Reference”](#).

The following is a sample connection string:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

In this example, the [MySqlConnection](#) object is configured to connect to a MySQL server at [127.0.0.1](#), with a user name of [root](#) and a password of [12345](#). The default database for all statements will be the [test](#) database.

Note

Using the '@' symbol for parameters is now the preferred approach, although the old pattern of using '?' is still supported. To avoid conflicts when using the '@' symbol in combination with user variables, see the [Allow User Variables](#) connection string option in [Section 4.6, “Connector/Net Connection String Options Reference”](#). The [Old Syntax](#) connection string option has now been deprecated.

4.5.2.1 Opening a Connection

Once you have created a connection string it can be used to open a connection to the MySQL server.

The following code is used to create a [MySqlConnection](#) object, assign the connection string, and open the connection.

Connector/Net can also connect using the native Windows authentication plugin. See [Section 4.5.5, “Using the Windows Native Authentication Plugin”](#) for details.

You can further extend the authentication mechanism by writing your own authentication plugin. See [Section 4.5.6, “Writing a Custom Authentication Plugin”](#) for details.

Visual Basic Example

```
Dim conn As New MySql.Data.MySqlClient.MySqlConnection
Dim myConnectionString as String
myConnectionString = "server=127.0.0.1;" _
```

```

        & "uid=root;" _
        & "pwd=12345;" _
        & "database=test;"
Try
    conn.ConnectionString = myConnectionString
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    MessageBox.Show(ex.Message)
End Try

```

C# Example

```

MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;
myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection();
    conn.ConnectionString = myConnectionString;
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}

```

You can also pass the connection string to the constructor of the [MySqlConnection](#) class:

Visual Basic Example

```

Dim myConnectionString as String
myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"
Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    MessageBox.Show(ex.Message)
End Try

```

C# Example

```

MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;
myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}

```

Once the connection is open it can be used by the other Connector/Net classes to communicate with the MySQL server.

4.5.2.2 Handling Connection Errors

Because connecting to an external server is unpredictable, it is important to add error handling to your .NET application. When there is an error connecting, the [MySqlConnection](#) class will return a [MySqlException](#) object. This object has two properties that are of interest when handling errors:

- **Message**: A message that describes the current exception.
- **Number**: The MySQL error number.

When handling errors, you can your application's response based on the error number. The two most common error numbers when connecting are as follows:

- **0**: Cannot connect to server.
- **1045**: Invalid user name and/or password.

The following code shows how to adapt the application's response based on the actual error:

Visual Basic Example

```
Dim myConnectionString as String
myConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test;"
Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    Select Case ex.Number
        Case 0
            MessageBox.Show("Cannot connect to server. Contact administrator")
        Case 1045
            MessageBox.Show("Invalid username/password, please try again")
    End Select
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;
myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    switch (ex.Number)
    {
        case 0:
            MessageBox.Show("Cannot connect to server. Contact administrator");
            break;
        case 1045:
            MessageBox.Show("Invalid username/password, please try again");
            break;
    }
}
```

Important

If you are using multilanguage databases then you must specify the character set in the connection string. If you do not specify the character set, the connection defaults to the **latin1** charset. You can specify the character set as part of the connection string, for example:

```
MySqlConnection myConnection = new MySqlConnection("server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;Charset=latin1;");
```

4.5.2.3 Using GetSchema on a Connection

The `GetSchema()` method of the connection object can be used to retrieve schema information about the database currently connected to. The schema information is returned in the form of a `DataTable`. The schema information is organized into a number of collections. Different forms of the `GetSchema()` method can be used depending on the information required. There are three forms of the `GetSchema()` method:

- `GetSchema()` - This call will return a list of available collections.
- `GetSchema(String)` - This call returns information about the collection named in the string parameter. If the string "MetaDataCollections" is used then a list of all available collections is returned. This is the same as calling `GetSchema()` without any parameters.
- `GetSchema(String, String[])` - In this call the first string parameter represents the collection name, and the second parameter represents a string array of restriction values. Restriction values limit the amount of data that will be returned. Restriction values are explained in more detail in the [Microsoft .NET documentation](#).

Collections

The collections can be broadly grouped into two types: collections that are common to all data providers, and collections specific to a particular provider.

Common

The following collections are common to all data providers:

- MetaDataCollections
- DataSourceInformation
- DataTypes
- Restrictions
- ReservedWords

Provider-specific

The following are the collections currently provided by MySQL Connector/Net, in addition to the common collections above:

- Databases
- Tables
- Columns
- Users
- Foreign Keys
- IndexColumns
- Indexes
- Foreign Key Columns
- UDF
- Views

- ViewColumns
- Procedure Parameters
- Procedures
- Triggers

Example Code

A list of available collections can be obtained using the following code:

```
using System;
using System.Data;
using System.Text;
using MySql.Data;
using MySql.Data.MySqlClient;
namespace ConsoleApplication2
{
    class Program
    {
        private static void DisplayData(System.Data.DataTable table)
        {
            foreach (System.Data.DataRow row in table.Rows)
            {
                foreach (System.Data.DataColumn col in table.Columns)
                {
                    Console.WriteLine("{0} = {1}", col.ColumnName, row[col]);
                }
                Console.WriteLine("=====");
            }
        }
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);
            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
                DataTable table = conn.GetSchema("MetaDataCollections");
                //DataTable table = conn.GetSchema("UDF");
                DisplayData(table);
                conn.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            Console.WriteLine("Done.");
        }
    }
}
```

Further information on the [GetSchema\(\)](#) method and schema collections can be found in the [Microsoft .NET documentation](#).

4.5.3 Using MySqlCommand

A `MySqlCommand` has the `CommandText` and `CommandType` properties associated with it. The `CommandText` will be handled differently depending on the setting of `CommandType`. `CommandType` can be one of:

1. Text - An SQL text command (default)
2. StoredProcedure - The name of a Stored Procedure
3. TableDirect - The name of a table (new in Connector/Net 6.2)

The default `CommandType`, `Text`, is used for executing queries and other SQL commands. Some example of this can be found in the following section [Section 4.4.1.2, “The MySqlCommand Object”](#).

If `CommandType` is set to `StoredProcedure`, set `CommandText` to the name of the Stored Procedure to access.

If `CommandType` is set to `TableDirect`, all rows and columns of the named table will be returned when you call one of the Execute methods. In effect, this command performs a `SELECT *` on the table specified. The `CommandText` property is set to the name of the table to query. This is illustrated by the following code snippet:

```
...
MySqlCommand cmd = new MySqlCommand();
cmd.CommandText = "mytable";
cmd.Connection = someConnection;
cmd.CommandType = CommandType.TableDirect;
MySqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    Console.WriteLine(reader[0], reader[1]...);
}
...
```

Examples of using the `CommandType` of `StoredProcedure` can be found in the section [Section 4.5.10, “Accessing Stored Procedures with Connector/Net”](#).

Commands can have a timeout associated with them. This is useful as you may not want a situation where a command takes up an excessive amount of time. A timeout can be set using the `CommandTimeout` property. The following code snippet sets a timeout of one minute:

```
MySqlCommand cmd = new MySqlCommand();
cmd.CommandTimeout = 60;
```

The default value is 30 seconds. Avoid a value of 0, which indicates an indefinite wait. To change the default command timeout, use the connection string option `Default Command Timeout`.

Prior to MySQL Connector/Net 6.2, `MySqlCommand.CommandTimeout` included user processing time, that is processing time not related to direct use of the connector. Timeout was implemented through a .NET Timer, that triggered after `CommandTimeout` seconds. This timer consumed a thread.

MySQL Connector/Net 6.2 introduced timeouts that are aligned with how Microsoft handles `SqlCommand.CommandTimeout`. This property is the cumulative timeout for all network reads and writes during command execution or processing of the results. A timeout can still occur in the `MySqlReader.Read` method after the first row is returned, and does not include user processing time, only IO operations. The 6.2 implementation uses the underlying stream timeout facility, so is more efficient in that it does not require the additional timer thread as was the case with the previous implementation.

Further details on this can be found in the relevant [Microsoft documentation](#).

4.5.4 Using Connector/Net with Connection Pooling

The Connector/Net supports connection pooling for better performance and scalability with database-intensive applications. This is enabled by default. You can turn it off or adjust its performance characteristics using the connection string options `Pooling`, `Connection Reset`, `Connection Lifetime`, `Cache Server Properties`, `Max Pool Size` and `Min Pool Size`. See [Section 4.5.2, “Creating a Connector/Net Connection String”](#) for further information.

Connection pooling works by keeping the native connection to the server live when the client disposes of a `MySqlConnection`. Subsequently, if a new `MySqlConnection` object is opened, it will be created from the connection pool, rather than creating a new native connection. This improves performance.

Guidelines

To work as designed, it is best to let the connection pooling system manage all connections. Do not create a globally accessible instance of `MySqlConnection` and then manually open and close it. This interferes with the way the pooling works and can lead to unpredictable results or even exceptions.

One approach that simplifies things is to avoid manually creating a `MySqlConnection` object. Instead use the overloaded methods that take a connection string as an argument. Using this approach, Connector/Net will automatically create, open, close and destroy connections, using the connection pooling system for best performance.

Typed Datasets and the `MembershipProvider` and `RoleProvider` classes use this approach. Most classes that have methods that take a `MySqlConnection` as an argument, also have methods that take a connection string as an argument. This includes `MySqlDataAdapter`.

Instead of manually creating `MySqlCommand` objects, you can use the static methods of the `MySqlHelper` class. These take a connection string as an argument, and they fully support connection pooling.

Resource Usage

Starting with MySQL Connector/Net 6.2, there is a background job that runs every three minutes and removes connections from pool that have been idle (unused) for more than three minutes. The pool cleanup frees resources on both client and server side. This is because on the client side every connection uses a socket, and on the server side every connection uses a socket and a thread.

Prior to this change, connections were never removed from the pool, and the pool always contained the peak number of open connections. For example, a web application that peaked at 1000 concurrent database connections would consume 1000 threads and 1000 open sockets at the server, without ever freeing up those resources from the connection pool. Connections, no matter how old, will not be closed if the number of connections in the pool is less than or equal to the value set by the `Min Pool Size` connection string parameter.

4.5.5 Using the Windows Native Authentication Plugin

Connector/Net applications can authenticate to a MySQL server using the Windows Native Authentication Plugin as of Connector/Net 6.4.4 and MySQL 5.5.16/5.6.10. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. For background and usage information about the authentication plugin, see, [The Windows Native Authentication Plugin](#).

The interface matches the `MySql.Data.MySqlClient` object. To enable, pass in `Integrated Security` to the connection string with a value of `yes` or `sspi`.

Passing in a user ID is optional. When Windows authentication is set up, a MySQL user is created and configured to be used by Windows authentication. By default, this user ID is named `auth_windows`, but can be defined using a different name. If the default name is used, then passing the user ID to the connection string from Connector/Net is optional, because it will use the `auth_windows` user. Otherwise, the name must be passed to the `connection string` using the standard user ID element.

4.5.6 Writing a Custom Authentication Plugin

Advanced users with special security requirements can create their own authentication plugins for Connector/Net applications. You can extend the handshake protocol, adding custom logic. This capability requires Connector/Net 6.6.3 or higher, and MySQL 5.5.16 or higher. For background and usage information about MySQL authentication plugins, see, [Authentication Plugins](#) and [Writing Authentication Plugins](#).

To write a custom authentication plugin, you will need a reference to the assembly `MySql.Data.dll`. The classes relevant for writing authentication plugins are available at the namespace `MySql.Data.MySqlClient.Authentication`.

How the Custom Authentication Plugin Works

At some point during handshake, the internal method

```
void Authenticate(bool reset)
```

of `MySQLAuthenticationPlugin` is called. This method in turns calls several overridable methods of the current plugin.

Creating the Authentication Plugin Class

You put the authentication plugin logic inside a new class derived from `MySQL.Data.MySqlClient.Authentication.MySQLAuthenticationPlugin`. The following methods are available to be overridden:

```
protected virtual void CheckConstraints()
protected virtual void AuthenticationFailed(Exception ex)
protected virtual void AuthenticationSuccessful()
protected virtual byte[] MoreData(byte[] data)
protected virtual void AuthenticationChange()
public abstract string PluginName { get; }
public virtual string GetUsername()
public virtual object GetPassword()
protected byte[] AuthData;
```

The following is a brief explanation of each one:

```
/// <summary>
/// This method must check authentication method specific constraints in the
environment and throw an Exception
/// if the conditions are not met. The default implementation does nothing.
/// </summary>
protected virtual void CheckConstraints()
/// <summary>
/// This method, called when the authentication failed, provides a chance to
plugins to manage the error
/// the way they consider decide (either showing a message, logging it, etc.).
/// The default implementation wraps the original exception in a MySQLException
with an standard message and rethrows it.
/// </summary>
/// <param name="ex">The exception with extra information on the error.</param>
protected virtual void AuthenticationFailed(Exception ex)
/// <summary>
/// This method is invoked when the authentication phase was successful accepted
by the server.
/// Derived classes must override this if they want to be notified of such
condition.
/// </summary>
/// <remarks>The default implementation does nothing.</remarks>
protected virtual void AuthenticationSuccessful()
/// <summary>
/// This method provides a chance for the plugin to send more data when the
server requests so during the
/// authentication phase. This method will be called at least once, and more
than one depending upon whether the
/// server response packets have the 0x01 prefix.
/// </summary>
/// <param name="data">The response data from the server, during the
authentication phase the first time is called is null, in
subsequent calls contains the server response.</param>
/// <returns>The data generated by the plugin for server consumption.</returns>
/// <remarks>The default implementation always returns null.</remarks>
protected virtual byte[] MoreData(byte[] data)
/// <summary>
/// The plugin name.
/// </summary>
public abstract string PluginName { get; }
```

```

/// <summary>
/// Gets the user name to send to the server in the authentication phase.
/// </summary>
/// <returns>An string with the user name</returns>
/// <remarks>Default implementation returns the UserId passed from the
connection string.</remarks>
public virtual string GetUsername()
/// <summary>
/// Gets the password to send to the server in the authentication phase. This
can be a string or a
/// </summary>
/// <returns>An object, can be byte[], string or null, with the password.
</returns>
/// <remarks>Default implementation returns null.</remarks>
public virtual object GetPassword()
/// <summary>
/// The authentication data passed when creating the plugin.
/// For example in mysql_native_password this is the seed to encrypt the
password.
/// </summary>
protected byte[] AuthData;

```

Sample Authentication Plugin

Here is an example showing how to create the authentication plugin, then enable it by means of a configuration file. Follow these steps:

1. Create a console app, adding a reference to [MySQL.Data.dll](#).
2. Design the main program as follows:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MySql.Data.MySqlClient;
namespace AuthPluginTest
{
    class Program
    {
        static void Main(string[] args)
        {
            // Customize the connection string as necessary.
            MySqlConnection con = new MySqlConnection("server=localhost;
database=test; user id=myuser; password=mypass");
            con.Open();
            con.Close();
        }
    }
}

```

3. Create your plugin class. In this example, we add an “alternative” implementation of the Native password plugin by just using the same code from the original plugin. We name our class [MySqlNativePasswordPlugin2](#):

```

using System.IO;
using System;
using System.Text;
using System.Security.Cryptography;
using MySql.Data.MySqlClient.Authentication;
using System.Diagnostics;
namespace AuthPluginTest
{
    public class MySqlNativePasswordPlugin2 : MySqlAuthenticationPlugin

```

```

{
    public override string PluginName
    {
        get { return "mysql_native_password"; }
    }
    public override object GetPassword()
    {
        Debug.WriteLine("Calling MySqlNativePasswordPlugin2.GetPassword");
        return Get411Password(Settings.Password, AuthData);
    }
    /// <summary>
    /// Returns a byte array containing the proper encryption of the
    /// given password/seed according to the new 4.1.1 authentication scheme.
    /// </summary>
    /// <param name="password"></param>
    /// <param name="seed"></param>
    /// <returns></returns>
    private byte[] Get411Password(string password, byte[] seedBytes)
    {
        // if we have no password, then we just return 1 zero byte
        if (password.Length == 0) return new byte[1];
        SHA1 sha = new SHA1CryptoServiceProvider();
        byte[] firstHash = sha.ComputeHash(Encoding.Default.GetBytes(password));
        byte[] secondHash = sha.ComputeHash(firstHash);
        byte[] input = new byte[seedBytes.Length + secondHash.Length];
        Array.Copy(seedBytes, 0, input, 0, seedBytes.Length);
        Array.Copy(secondHash, 0, input, seedBytes.Length, secondHash.Length);
        byte[] thirdHash = sha.ComputeHash(input);
        byte[] finalHash = new byte[thirdHash.Length + 1];
        finalHash[0] = 0x14;
        Array.Copy(thirdHash, 0, finalHash, 1, thirdHash.Length);
        for (int i = 1; i < finalHash.Length; i++)
            finalHash[i] = (byte)(finalHash[i] ^ firstHash[i - 1]);
        return finalHash;
    }
}
}

```

4. Notice that the plugin implementation just overrides `GetPassword`, and provides an implementation to encrypt the password using the 4.1 protocol. We also put the following line in the `GetPassword` body:

```
Debug.WriteLine("Calling MySqlNativePasswordPlugin2.GetPassword");
```

to provide confirmation that the plugin was effectively used. (You could also put a breakpoint on that method.)

5. Enable the new plugin in the configuration file:

```

<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration,
MySql.Data"/>
  </configSections>
  <MySQL>
    <AuthenticationPlugins>
      <add name="mysql_native_password"
type="AuthPluginTest.MySqlNativePasswordPlugin2, AuthPluginTest"></add>
    </AuthenticationPlugins>
  </MySQL>
</startup><supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
</configuration>

```

6. Run the application. In Visual Studio, you will see the message `Calling MySqlNativePasswordPlugin2.GetPassword` in the debug window.

7. Continue enhancing the authentication logic, overriding more methods if you required.

4.5.7 MySQL Fabric Support

Connector/Net supports MySQL Fabric as a Replication/Load balancing plugin.

Note

This feature was added in MySQL Connector/Net 6.9.4.

The following steps are required to use MySQL Fabric with Connector/Net:

System Requirements

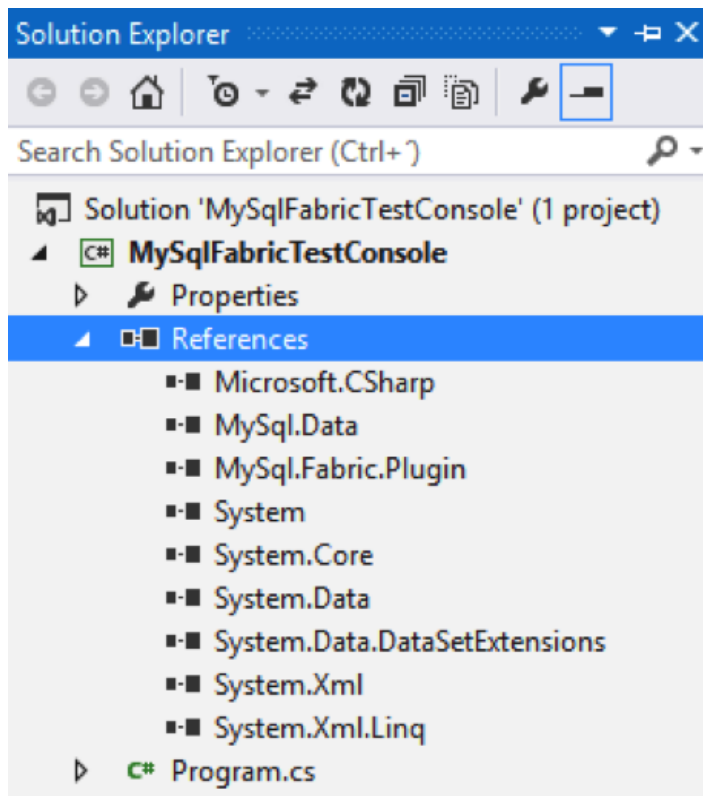
Confirm that you have the required Connector/Net and Fabric versions installed:

- Connector/Net 6.9.4 or newer
- MySQL Fabric 1.5.0 or newer

Set up the MySQL Fabric plugin

First, add **MySql.Data** and **MySql.Fabric.Plugin** to the project references:

Figure 4.49 MySQL Fabric Project References



Second, add a configuration section with the Fabric connection to the [App.config](#) configuration file. For example:

```
<configuration>
  <configSections>
    <section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration, MySql.Data,
      Version=6.9.4.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
    </configSections>
  <MySQL>
    <Replication>
```

```

<ServerGroups>
  <Group name="Fabric" groupType="MySQL.Fabric.FabricServerGroup, MySQL.Fabric.Plugin">
    <Servers>
      <Server name="fabric" connectionString="server=localhost;port=32275;uid=admin;password=adminpass;" />
    </Servers>
  </Group>
</ServerGroups>
</Replication>
</MySQL>
</configuration>

```

Notice that the Fabric connection is set in the Server node:

```

<Server name="fabric" connectionString="server=localhost;port=32275;uid=admin;password=adminpass;" />

```

Connector/Net only supports the MySQL protocol for connecting to MySQL Fabric, so the correct port must be used.

Using MySQL Fabric Groups

The MySQL Fabric group is used with a **MySQLConnection** that contains the server name specified in the [App.config](#) file, and a username and password for connecting to the servers defined in the group.

A Fabric extension method is used to specify the group and mode:

```

MySQLConnection conn = new MySQLConnection(connectionString);
conn.SetFabricProperties(groupId: "my_group", mode: FabricServerModeEnum.Read_Write);

```

The following example shows how to store and retrieve information in a specific Fabric group:

Note

The initial MySQL Fabric configuration for this example is defined in the MySQL Fabric documentation at [Example: Fabric and Replication](#).

```

using System;
using MySql.Data.MySqlClient;
using MySQL.Fabric;

namespace FabricTest
{
    class Program
    {
        public const string connectionString = "server=fabric;uid=appuser;password=pass;";

        static void Main(string[] args)
        {
            RunFabricTest();
        }

        static string AddEmployee(MySQLConnection conn, int emp_no, string first_name, string last_name)
        {
            conn.SetFabricProperties(groupId: "my_group", mode: FabricServerModeEnum.Read_Write);

            MySqlCommand cmd = new MySqlCommand("USE employees", conn);
            cmd.ExecuteNonQuery();

            cmd.CommandText = "INSERT INTO employees VALUES (@emp_no, @first_name, @last_name)";
            cmd.Parameters.Add("emp_no", emp_no);
            cmd.Parameters.Add("first_name", first_name);
            cmd.Parameters.Add("last_name", last_name);
        }
    }
}

```

```

cmd.ExecuteNonQuery();

cmd.CommandText = "SELECT @@global.gtid_executed";
cmd.Parameters.Clear();
using (MySqlDataReader reader = cmd.ExecuteReader())
{
    while (reader.Read())
    {
        Console.WriteLine("Transactions executed on the master " + reader.GetValue(0));
    }
    return reader.GetString(0);
}
}

static void FindEmployee(MySqlConnection conn, int emp_no, string gtid_executed)
{
    conn.SetFabricProperties(groupId: "my_group", mode: FabricServerModeEnum.Read_only);

    MySqlCommand cmd = new MySqlCommand("", conn);
    cmd.CommandText = "SELECT WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(@gtid_executed, 0)";
    cmd.Parameters.Add("gtid_executed", gtid_executed);
    using (MySqlDataReader reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            Console.WriteLine("Had to synchronize " + reader.GetValue(0) + " transactions.");
        }
    }

    cmd.CommandText = "USE employees";
    cmd.Parameters.Clear();
    cmd.ExecuteNonQuery();

    cmd.CommandText = "SELECT first_name, last_name FROM employees ";
    cmd.CommandText += " WHERE emp_no = @emp_no";
    cmd.Parameters.Clear();
    cmd.Parameters.Add("emp_no", emp_no);
    using (MySqlDataReader reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            object[] values = new object[reader.FieldCount];
            reader.GetValues(values);
            Console.WriteLine("Retrieved {0}", string.Join(", ", values));
        }
    }
}

static void RunFabricTest()
{
    using (MySqlConnection conn = new MySqlConnection(connectionString))
    {
        string gtid_executed;
        conn.SetFabricProperties(groupId: "my_group", mode: FabricServerModeEnum.Read_Write);
        conn.Open();

        MySqlCommand cmd = new MySqlCommand("", conn);
        cmd.CommandText = "CREATE DATABASE IF NOT EXISTS employees;";
        cmd.ExecuteNonQuery();

        cmd.CommandText = "USE employees;";
        cmd.ExecuteNonQuery();

        cmd.CommandText = "DROP TABLE IF EXISTS employees;";
        cmd.ExecuteNonQuery();

        cmd.CommandText = "CREATE TABLE employees(";
        cmd.CommandText += "    emp_no INT, ";
        cmd.CommandText += "    first_name CHAR(40), ";
        cmd.CommandText += "    last_name CHAR(40)";
        cmd.CommandText += ")";
        cmd.ExecuteNonQuery();
    }
}

```

```

        gtid_executed = AddEmployee(conn, 12, "John", "Doe");
        FindEmployee(conn, 12, gtid_executed);
    }
}
}
}

```

Using Ranged Sharding

Sharding with Connector/Net requires you to specify the table, key, and scope for each executed query.

```

MySQLConnection con = new MySQLConnection(connectionString);

con.SetFabricProperties(table: "employees.employees", key: empId.ToString(),
    mode: FabricServerModeEnum.Read_Write, scope: FabricScopeEnum.Local);

MySQLCommand cmd = new MySQLCommand(
    string.Format("insert into employees(emp_no, first_name, last_name) values ({0}, '{1}', '{2}']",
        empId, firstName, lastName), con);
cmd.ExecuteScalar();

```

You can use the following MySQL Fabric configuration to execute the code example:

Note

For related MySQL Fabric documentation, see [Sharding Scenario](#).

```

using System;
using System.Collections.Generic;
using MySql.Data.MySqlClient;
using MySql.Fabric;

namespace FabricTest
{
    class ShardTest
    {
        public const string connectionString = "server=fabric;uid=appuser;password=pass;";

        public static void test_shard_range()
        {
            using (MySQLConnection con = new MySQLConnection(connectionString))
            {
                con.SetFabricProperties(groupId: "group_id-global", mode: FabricServerModeEnum.Read_Write,
                    scope: FabricScopeEnum.Global);
                con.Open();

                MySQLCommand cmd = new MySQLCommand("create database if not exists employees", con);
                cmd.ExecuteScalar();

                cmd.CommandText = "use employees";
                cmd.ExecuteScalar();

                cmd.CommandText = "drop table if exists employees";
                cmd.ExecuteScalar();

                cmd.CommandText =
@"create table employees (
    emp_no int,
    first_name char( 40 ),
    last_name char( 40 )
)";
                cmd.ExecuteScalar();

                string gtid = prepare_synchronization(con);

                string[] first_names = { "John", "Buffalo", "Michael", "Kate", "Deep", "Genesis" };
            }
        }
    }
}

```



```

        string[] last_names = { "Doe", "Bill", "Jackson", "Bush", "Purple" };
        List<int> list_emp_no = new List<int>();

        con.SetFabricProperties(scope: FabricScopeEnum.Local);

        for (int i = 0; i < 10; i++)
        {
            int empId = pick_shard_key();
            list_emp_no.Add(empId);
            add_employee(con, empId, first_names[empId % first_names.Length], last_names[empId % last_names.Length]);
        }

        for (int i = 0; i < list_emp_no.Count; i++)
        {
            int empId = list_emp_no[ i ];
            find_employee(con, empId, gtid);
        }
    }

    public static int pick_shard_key()
    {
        Random r = new Random();
        int shard = r.Next(0, 2);
        int shard_range = shard * 10000;
        shard_range = (shard != 0) ? shard_range : shard_range + 1;
        int shift_within_shard = r.Next(0, 99999);
        return shard_range + shift_within_shard;
    }

    public static void add_employee(MySqlConnection con, int empId, string firstName, string lastName,
    {
        con.SetFabricProperties( table: "employees.employees", key: empId.ToString(), mode: FabricServerModeEnum.Read_only);
        synchronize(con, gtid);
        MySqlCommand cmd = new MySqlCommand(
            string.Format("insert into employees( emp_no, first_name, last_name ) values ( {0}, '{1}', '{2}' )",
                empId, firstName, lastName), con);
        cmd.ExecuteScalar();
    }

    public static void find_employee(MySqlConnection con, int empId, string gtid)
    {
        con.SetFabricProperties(table: "employees.employees", key: empId.ToString(),
            mode: FabricServerModeEnum.Read_only);
        synchronize(con, gtid);
        MySqlCommand cmd = new MySqlCommand(string.Format("
            select first_name, last_name from employees where emp_no = {0}", empId), con);
        using (MySqlDataReader r = cmd.ExecuteReader())
        {
            while (r.Read())
            {
                Console.WriteLine("( {0}, {1} )", r.GetString(0), r.GetString(1));
            }
        }
    }

    public static string prepare_synchronization(MySqlConnection con)
    {
        string gtid_executed = "";
        MySqlCommand cmd = new MySqlCommand("select @@global.gtid_executed", con);
        gtid_executed = ( string )cmd.ExecuteScalar();
        return gtid_executed;
    }

    public static void synchronize(MySqlConnection con, string gtid_executed)
    {
        MySqlCommand cmd = new MySqlCommand( string.Format( "SELECT WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS('{0}',
            gtid_executed ), con);
        cmd.ExecuteScalar();
    }
}

```

4.5.8 Using Connector/Net with Table Caching

This feature exists with Connector/Net versions 6.4 and above.

Table caching is a feature that can be used to cache slow-changing datasets on the client side. This is useful for applications that are designed to use readers, but still want to minimize trips to the server for slow-changing tables.

This feature is transparent to the application, and is disabled by default.

Configuration

- To enable table caching, add `'table cache = true'` to the connection string.
- Optionally, specify the `'Default Table Cache Age'` connection string option, which represents the number of seconds a table is cached before the cached data is discarded. The default value is 60.
- You can turn caching on and off and set caching options at runtime, on a per-command basis.

4.5.9 Using the Connector/Net with Prepared Statements

Use of prepared statements can provide significant performance improvements on queries that are executed more than once.

Note

Prepared statement support is available with MySQL 4.1 and above.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Another advantage of prepared statements is that, with server-side prepared statements enabled, it uses a binary protocol that makes data transfer between client and server more efficient.

Note

Enable server-side prepared statements (they are disabled by default) by passing in `"IgnorePrepare=false"` to your connection string.

4.5.9.1 Preparing Statements in Connector/Net

To prepare a statement, create a command object and set the `.CommandText` property to your query.

After entering your statement, call the `.Prepare` method of the `MySQLCommand` object. After the statement is prepared, add parameters for each of the dynamic elements in the query.

After you enter your query and enter parameters, execute the statement using the `.ExecuteNonQuery()`, `.ExecuteScalar()`, or `.ExecuteReader` methods.

For subsequent executions, you need only modify the values of the parameters and call the execute method again, there is no need to set the `.CommandText` property or redefine the parameters.

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
conn.ConnectionString = strConnection
Try
```

```

conn.Open()
cmd.Connection = conn
cmd.CommandText = "INSERT INTO myTable VALUES(NULL, @number, @text)"
cmd.Prepare()
cmd.Parameters.AddWithValue("@number", 1)
cmd.Parameters.AddWithValue("@text", "One")
For i = 1 To 1000
    cmd.Parameters("@number").Value = i
    cmd.Parameters("@text").Value = "A string value"
    cmd.ExecuteNonQuery()
Next
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;
conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();
conn.ConnectionString = strConnection;
try
{
    conn.Open();
    cmd.Connection = conn;
    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, @number, @text)";
    cmd.Prepare();
    cmd.Parameters.AddWithValue("@number", 1);
    cmd.Parameters.AddWithValue("@text", "One");
    for (int i=1; i <= 1000; i++)
    {
        cmd.Parameters["@number"].Value = i;
        cmd.Parameters["@text"].Value = "A string value";
        cmd.ExecuteNonQuery();
    }
}
catch (MySQL.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

4.5.10 Accessing Stored Procedures with Connector/Net

MySQL server version 5 and up supports stored procedures with the SQL 2003 stored procedure syntax.

A stored procedure is a set of SQL statements that is stored in the server. Clients make a single call to the stored procedure, passing parameters that can influence the procedure logic and query conditions, rather than issuing individual hardcoded SQL statements.

Stored procedures can be particularly useful in situations such as the following:

- Stored procedures can act as an API or abstraction layer, allowing multiple client applications to perform the same database operations. The applications can be written in different languages and run on different platforms. The applications do not need to hardcode table and column names, complicated queries, and so on. When you extend and optimize the queries in a stored procedure, all the applications that call the procedure automatically receive the benefits.
- When security is paramount, stored procedures keep applications from directly manipulating tables, or even knowing details such as table and column names. Banks, for example, use stored procedures for all common operations. This provides a consistent and secure environment, and procedures can ensure that each operation is properly logged. In such a setup, applications and users would not get any access to the database tables directly, but can only execute specific stored procedures.

Connector/Net supports the calling of stored procedures through the `MySQLCommand` object. Data can be passed in and out of a MySQL stored procedure through use of the `MySQLCommand.Parameters` collection.

Note

When you call a stored procedure, the command object makes an additional `SELECT` call to determine the parameters of the stored procedure. You must ensure that the user calling the procedure has the `SELECT` privilege on the `mysql.proc` table to enable them to verify the parameters. Failure to do this will result in an error when calling the procedure.

This section will not provide in-depth information on creating Stored Procedures. For such information, please refer to <http://dev.mysql.com/doc/mysql/en/stored-routines.html>.

A sample application demonstrating how to use stored procedures with Connector/Net can be found in the `Samples` directory of your Connector/Net installation.

4.5.10.1 Using Stored Routines from Connector/Net

Stored procedures in MySQL can be created using a variety of tools. First, stored procedures can be created using the `mysql` command-line client. Second, stored procedures can be created using MySQL Workbench. Finally, stored procedures can be created using the `.ExecuteNonQuery` method of the `MySQLCommand` object.

Unlike the command-line and GUI clients, you are not required to specify a special delimiter when creating stored procedures in Connector/Net.

To call a stored procedure using Connector/Net, you create a `MySQLCommand` object and pass the stored procedure name as the `.CommandText` property. You then set the `.CommandType` property to `CommandType.StoredProcedure`.

After the stored procedure is named, you create one `MySQLCommand` parameter for every parameter in the stored procedure. `IN` parameters are defined with the parameter name and the object containing the value, `OUT` parameters are defined with the parameter name and the data type that is expected to be returned. All parameters need the parameter direction defined.

After defining the parameters, you call the stored procedure by using the `MySQLCommand.ExecuteNonQuery()` method.

Once the stored procedure is called, the values of the output parameters can be retrieved by using the `.Value` property of the `MySQLConnector.Parameters` collection.

Note

When a stored procedure is called using `MySQLCommand.ExecuteReader`, and the stored procedure has output parameters, the output parameters are only set after the `MySQLDataReader` returned by `ExecuteReader` is closed.

The following C# example code demonstrates the use of stored procedures. It assumes the database 'employees' has already been created:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
namespace UsingStoredRoutines
{
    class Program
    {
```

```

static void Main(string[] args)
{
    MySqlConnection conn = new MySqlConnection();
    conn.ConnectionString = "server=localhost;user=root;database=employees;port=3306;password=";
    MySqlCommand cmd = new MySqlCommand();
    try
    {
        Console.WriteLine("Connecting to MySQL...");
        conn.Open();
        cmd.Connection = conn;
        cmd.CommandText = "DROP PROCEDURE IF EXISTS add_emp";
        cmd.ExecuteNonQuery();
        cmd.CommandText = "DROP TABLE IF EXISTS emp";
        cmd.ExecuteNonQuery();
        cmd.CommandText = "CREATE TABLE emp (empno INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
        cmd.ExecuteNonQuery();
        cmd.CommandText = "CREATE PROCEDURE add_emp(" +
            "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT)
        BEGIN INSERT INTO emp(first_name, last_name, birthdate) " +
            "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END";
        cmd.ExecuteNonQuery();
    }
    catch (MySqlException ex)
    {
        Console.WriteLine ("Error " + ex.Number + " has occurred: " + ex.Message);
    }
    conn.Close();
    Console.WriteLine("Connection closed.");
    try
    {
        Console.WriteLine("Connecting to MySQL...");
        conn.Open();
        cmd.Connection = conn;
        cmd.CommandText = "add_emp";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Parameters.AddWithValue("@lname", "Jones");
        cmd.Parameters["@lname"].Direction = ParameterDirection.Input;
        cmd.Parameters.AddWithValue("@fname", "Tom");
        cmd.Parameters["@fname"].Direction = ParameterDirection.Input;
        cmd.Parameters.AddWithValue("@bday", "1940-06-07");
        cmd.Parameters["@bday"].Direction = ParameterDirection.Input;
        cmd.Parameters.AddWithValue("@empno", MySqlDbType.Int32);
        cmd.Parameters["@empno"].Direction = ParameterDirection.Output;
        cmd.ExecuteNonQuery();
        Console.WriteLine("Employee number: "+cmd.Parameters["@empno"].Value);
        Console.WriteLine("Birthday: " + cmd.Parameters["@bday"].Value);
    }
    catch (MySql.Data.MySqlClient.MySqlException ex)
    {
        Console.WriteLine("Error " + ex.Number + " has occurred: " + ex.Message);
    }
    conn.Close();
    Console.WriteLine("Done.");
}
}
}

```

The following code shows the same application in Visual Basic:

```

Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text
Imports System.Data
Imports MySql.Data
Imports MySql.Data.MySqlClient
Module Module1
    Sub Main()
        Dim conn As New MySqlConnection()
        conn.ConnectionString = "server=localhost;user=root;database=world;port=3306;password=*****;"
    End Sub
End Module

```

```

Dim cmd As New MySqlCommand()
Try
    Console.WriteLine("Connecting to MySQL...")
    conn.Open()
    cmd.Connection = conn
    cmd.CommandText = "DROP PROCEDURE IF EXISTS add_emp"
    cmd.ExecuteNonQuery()
    cmd.CommandText = "DROP TABLE IF EXISTS emp"
    cmd.ExecuteNonQuery()
    cmd.CommandText = "CREATE TABLE emp (empno INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, fname VARCHAR(20), lname VARCHAR(20), IN
    cmd.ExecuteNonQuery()
    cmd.CommandText = "CREATE PROCEDURE add_emp(" & "IN fname VARCHAR(20), IN lname VARCHAR(20), IN
    cmd.ExecuteNonQuery()
Catch ex As MySqlException
    Console.WriteLine(("Error " & ex.Number & " has occurred: ") + ex.Message)
End Try
conn.Close()
Console.WriteLine("Connection closed.")
Try
    Console.WriteLine("Connecting to MySQL...")
    conn.Open()
    cmd.Connection = conn
    cmd.CommandText = "add_emp"
    cmd.CommandType = CommandType.StoredProcedure
    cmd.Parameters.AddWithValue("@lname", "Jones")
    cmd.Parameters("@lname").Direction = ParameterDirection.Input
    cmd.Parameters.AddWithValue("@fname", "Tom")
    cmd.Parameters("@fname").Direction = ParameterDirection.Input
    cmd.Parameters.AddWithValue("@bday", "1940-06-07")
    cmd.Parameters("@bday").Direction = ParameterDirection.Input
    cmd.Parameters.AddWithValue("@empno", MySqlDbType.Int32)
    cmd.Parameters("@empno").Direction = ParameterDirection.Output
    cmd.ExecuteNonQuery()
    Console.WriteLine("Employee number: " & cmd.Parameters("@empno").Value)
    Console.WriteLine("Birthday: " & cmd.Parameters("@bday").Value)
Catch ex As MySql.Data.MySqlClient.MySqlException
    Console.WriteLine(("Error " & ex.Number & " has occurred: ") + ex.Message)
End Try
conn.Close()
Console.WriteLine("Done.")
End Sub
End Module

```

4.5.11 Handling BLOB Data With Connector/Net

One common use for MySQL is the storage of binary data in **BLOB** columns. MySQL supports four different BLOB data types: **TINYBLOB**, **BLOB**, **MEDIUMBLOB**, and **LONGBLOB**, all described in [The BLOB and TEXT Types](#) and [Data Type Storage Requirements](#).

Data stored in a **BLOB** column can be accessed using Connector/Net and manipulated using client-side code. There are no special requirements for using Connector/Net with **BLOB** data.

Simple code examples will be presented within this section, and a full sample application can be found in the [Samples](#) directory of the Connector/Net installation.

4.5.11.1 Preparing the MySQL Server

The first step is using MySQL with **BLOB** data is to configure the server. Let's start by creating a table to be accessed. In my file tables, I usually have four columns: an **AUTO_INCREMENT** column of appropriate size (**UNSIGNED SMALLINT**) to serve as a primary key to identify the file, a **VARCHAR** column that stores the file name, an **UNSIGNED MEDIUMINT** column that stores the size of the file, and a **MEDIUMBLOB** column that stores the file itself. For this example, I will use the following table definition:

```

CREATE TABLE file(
file_id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
file_name VARCHAR(64) NOT NULL,

```

```
file_size MEDIUMINT UNSIGNED NOT NULL,  
file MEDIUMBLOB NOT NULL);
```

After creating a table, you might need to modify the `max_allowed_packet` system variable. This variable determines how large of a packet (that is, a single row) can be sent to the MySQL server. By default, the server only accepts a maximum size of 1MB from the client application. If you intend to exceed 1MB in your file transfers, increase this number.

The `max_allowed_packet` option can be modified using the MySQL Workbench **Server Administration** screen. Adjust the Maximum permitted option in the **Data / Memory size** section of the Networking tab to an appropriate setting. After adjusting the value, click the **Apply** button and restart the server using the **Startup / Shutdown** screen of MySQL Workbench. You can also adjust this value directly in the `my.cnf` file (add a line that reads `max_allowed_packet=xxM`), or use the `SET max_allowed_packet=xxM;` syntax from within MySQL.

Try to be conservative when setting `max_allowed_packet`, as transfers of BLOB data can take some time to complete. Try to set a value that will be adequate for your intended use and increase the value if necessary.

4.5.11.2 Writing a File to the Database

To write a file to a database, we need to convert the file to a byte array, then use the byte array as a parameter to an `INSERT` query.

The following code opens a file using a `FileStream` object, reads it into a byte array, and inserts it into the `file` table:

Visual Basic Example

```
Dim conn As New MySqlConnection  
Dim cmd As New MySqlCommand  
Dim SQL As String  
Dim FileSize As UInt32  
Dim rawData() As Byte  
Dim fs As FileStream  
conn.ConnectionString = "server=127.0.0.1;" _  
    & "uid=root;" _  
    & "pwd=12345;" _  
    & "database=test"  
Try  
    fs = New FileStream("c:\image.png", FileMode.Open, FileAccess.Read)  
    FileSize = fs.Length  
    rawData = New Byte(FileSize) {}  
    fs.Read(rawData, 0, FileSize)  
    fs.Close()  
    conn.Open()  
    SQL = "INSERT INTO file VALUES(NULL, @FileName, @FileSize, @File)"  
    cmd.Connection = conn  
    cmd.CommandText = SQL  
    cmd.Parameters.AddWithValue("@FileName", strFileName)  
    cmd.Parameters.AddWithValue("@FileSize", FileSize)  
    cmd.Parameters.AddWithValue("@File", rawData)  
    cmd.ExecuteNonQuery()  
    MessageBox.Show("File Inserted into database successfully!", _  
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)  
    conn.Close()  
Catch ex As Exception  
    MessageBox.Show("There was an error: " & ex.Message, "Error", _  
        MessageBoxButtons.OK, MessageBoxIcon.Error)  
End Try
```

C# Example

```
MySQL.Data.MySqlClient.MySqlConnection conn;  
MySQL.Data.MySqlClient.MySqlCommand cmd;  
conn = new MySQL.Data.MySqlClient.MySqlConnection();
```

```

cmd = new MySql.Data.MySqlClient.MySqlCommand();
string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;
conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
try
{
    fs = new FileStream(@"c:\image.png", FileMode.Open, FileAccess.Read);
    FileSize = fs.Length;
    rawData = new byte[FileSize];
    fs.Read(rawData, 0, FileSize);
    fs.Close();
    conn.Open();
    SQL = "INSERT INTO file VALUES(NULL, @FileName, @FileSize, @File)";
    cmd.Connection = conn;
    cmd.CommandText = SQL;
    cmd.Parameters.AddWithValue("@FileName", strFileName);
    cmd.Parameters.AddWithValue("@FileSize", FileSize);
    cmd.Parameters.AddWithValue("@File", rawData);
    cmd.ExecuteNonQuery();
    MessageBox.Show("File Inserted into database successfully!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

The [Read](#) method of the [FileStream](#) object is used to load the file into a byte array which is sized according to the [Length](#) property of the [FileStream](#) object.

After assigning the byte array as a parameter of the [MySqlCommand](#) object, the [ExecuteNonQuery](#) method is called and the [BLOB](#) is inserted into the [file](#) table.

4.5.11.3 Reading a BLOB from the Database to a File on Disk

Once a file is loaded into the [file](#) table, we can use the [MySqlDataReader](#) class to retrieve it.

The following code retrieves a row from the [file](#) table, then loads the data into a [FileStream](#) object to be written to disk:

Visual Basic Example

```

Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myData As MySqlDataReader
Dim SQL As String
Dim rawData() As Byte
Dim FileSize As UInt32
Dim fs As FileStream
conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"
SQL = "SELECT file_name, file_size, file FROM file"
Try
    conn.Open()
    cmd.Connection = conn
    cmd.CommandText = SQL
    myData = cmd.ExecuteReader
    If Not myData.HasRows Then Throw New Exception("There are no BLOBs to save")
    myData.Read()
    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"))
    rawData = New Byte(FileSize) {}
    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize)

```



```

        fs = New FileStream("C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write)
        fs.Write(rawData, 0, FileSize)
        fs.Close()
        MessageBox.Show("File successfully written to disk!", "Success!", MessageBoxButtons.OK, MessageBoxIcon.Information)
        myData.Close()
        conn.Close()
    Catch ex As Exception
        MessageBox.Show("There was an error: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try

```

C# Example

```

MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;
MySQL.Data.MySqlClient.MySqlDataReader myData;
conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();
string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;
conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
SQL = "SELECT file_name, file_size, file FROM file";
try
{
    conn.Open();
    cmd.Connection = conn;
    cmd.CommandText = SQL;
    myData = cmd.ExecuteReader();
    if (! myData.HasRows)
        throw new Exception("There are no BLOBs to save");
    myData.Read();
    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"));
    rawData = new byte[FileSize];
    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, (int)FileSize);
    fs = new FileStream(@"C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write);
    fs.Write(rawData, 0, (int)FileSize);
    fs.Close();
    MessageBox.Show("File successfully written to disk!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    myData.Close();
    conn.Close();
}
catch (MySQL.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

After connecting, the contents of the `file` table are loaded into a `MySqlDataReader` object. The `GetBytes` method of the `MySqlDataReader` is used to load the `BLOB` into a byte array, which is then written to disk using a `FileStream` object.

The `GetOrdinal` method of the `MySqlDataReader` can be used to determine the integer index of a named column. Use of the `GetOrdinal` method prevents errors if the column order of the `SELECT` query is changed.

4.5.12 Asynchronous methods

The Task-based Asynchronous Pattern (TAP) is a pattern for asynchrony in the .NET Framework. It is based on the `Task` and `Task<TResult>` types in the `System.Threading.Tasks` namespace, which are used to represent arbitrary asynchronous operations.

Async-Await are new keywords introduced to work with the TAP. The **Async modifier** is used to specify that a method, lambda expression, or anonymous method is asynchronous. The **Await** operator is applied to a task in an asynchronous method to suspend the execution of the method until the awaited task completes.

Requirements

- **Async-Await** support requires .NET Framework 4.5 or greater
- **TAP** support requires .NET Framework 4.0 or greater
- Connector/Net 6.9 or greater

Methods

These methods can be used with either TAP or Async-Await.

- Namespace MySql.Data.Entity
 - Class EFMySQLCommand
 - Task PrepareAsync()
 - Task PrepareAsync(CancellationTokens)
- Namespace MySql.Data
 - Class MySQLBulkLoader
 - Task<int> LoadAsync()
 - Task<int> LoadAsync(CancellationTokens)
 - Class MySQLConnection
 - Task<MySQLTransaction> BeginTransactionAsync()
 - Task<MySQLTransaction> BeginTransactionAsync (CancellationTokens)
 - Task<MySQLTransaction> BeginTransactionAsync(IsolationLevel)
 - Task<MySQLTransaction> BeginTransactionAsync (IsolationLevel , CancellationTokens)
 - Task ChangeDatabaseAsync(string)
 - Task ChangeDatabaseAsync(string, CancellationTokens)
 - Task CloseAsync()
 - Task CloseAsync(CancellationTokens)
 - Task ClearPoolAsync(MySqlConnection)
 - Task ClearPoolAsync(MySqlConnection, CancellationTokens)
 - Task ClearAllPoolsAsync()
 - Task ClearAllPoolsAsync(CancellationTokens)
 - Task<MySQLSchemaCollection> GetSchemaCollection(string, string[])
 - Task<MySQLSchemaCollection> GetSchemaCollection(string, string[], CancellationTokens)
 - Class MySQLDataAdapter
 - Task<int> FillAsync(DataSet)
 - Task<int> FillAsync(DataSet, CancellationTokens)

- `Task<int> FillAsync(DataTable)`
- `Task<int> FillAsync(DataTable, CancellationToken)`
- `Task<int> FillAsync(DataSet, string)`
- `Task<int> FillAsync(DataSet, string, CancellationToken)`
- `Task<int> FillAsync(DataTable, IDataReader)`
- `Task<int> FillAsync(DataTable, IDataReader, CancellationToken)`
- `Task<int> FillAsync(DataTable, IDbCommand, CommandBehavior)`
- `Task<int> FillAsync(DataTable, IDbCommand, CommandBehavior, CancellationToken)`
- `Task<int> FillAsync(int, int, params DataTable[])`
- `Task<int> FillAsync(int, int, params DataTable[], CancellationToken)`
- `Task<int> FillAsync(DataSet, int, int, string)`
- `Task<int> FillAsync(DataSet, int, int, string, CancellationToken)`
- `Task<int> FillAsync(DataSet, string, IDataReader, int, int)`
- `Task<int> FillAsync(DataSet, string, IDataReader, int, int, CancellationToken)`
- `Task<int> FillAsync(DataTable[], int, int, IDbCommand, CommandBehavior)`
- `Task<int> FillAsync(DataTable[], int, int, IDbCommand, CommandBehavior, CancellationToken)`
- `Task<int> FillAsync(DataSet, int, int, string, IDbCommand, CommandBehavior)`
- `Task<int> FillAsync(DataSet, int, int, string, IDbCommand, CommandBehavior, CancellationToken)`
- `Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType)`
- `Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, CancellationToken)`
- `Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, string)`
- `Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, string, CancellationToken)`
- `Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, string, IDataReader)`
- `Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, string, IDataReader, CancellationToken)`
- `Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, IDbCommand, string, CommandBehavior)`
- `Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, IDbCommand, string, CommandBehavior, CancellationToken)`
- `Task<DataTable> FillSchemaAsync(DataTable, SchemaType)`
- `Task<DataTable> FillSchemaAsync(DataTable, SchemaType, CancellationToken)`
- `Task<DataTable> FillSchemaAsync(DataTable, SchemaType, IDataReader)`

- Task<DataTable> FillSchemaAsync(DataTable, SchemaType, IDataReader, CancellationToken)
- Task<DataTable> FillSchemaAsync(DataTable, SchemaType, IDbCommand, CommandBehavior)
- Task<DataTable> FillSchemaAsync(DataTable, SchemaType, IDbCommand, CommandBehavior, CancellationToken)
- Task<int> UpdateAsync(DataRow[])
- Task<int> UpdateAsync(DataRow[], CancellationToken)
- Task<int> UpdateAsync(DataSet)
- Task<int> UpdateAsync(DataSet, CancellationToken)
- Task<int> UpdateAsync(DataTable)
- Task<int> UpdateAsync(DataTable, CancellationToken)
- Task<int> UpdateAsync(DataRow[], DataTableMapping, CancellationToken)
- Task<int> UpdateAsync(DataSet, string)
- Task<int> UpdateAsync(DataSet, string, CancellationToken)
- Class MySqlHelper
 - Task<DataRow> ExecuteDataRowAsync(string, string, params MySqlParameter[])
 - Task<DataRow> ExecuteDataRowAsync(string, string, CancellationToken, params MySqlParameter[])
 - Task<int> ExecuteNonQueryAsync(MySqlConnection, string, params MySqlParameter[])
 - Task<int> ExecuteNonQueryAsync(MySqlConnection, string, CancellationToken, params MySqlParameter[])
 - Task<int> ExecuteNonQueryAsync(string, string, params MySqlParameter[])
 - Task<int> ExecuteNonQueryAsync(string, string, CancellationToken, params MySqlParameter[])
 - Task<DataSet> ExecuteDatasetAsync(string, string)
 - Task<DataSet> ExecuteDatasetAsync(string, string, CancellationToken)
 - Task<DataSet> ExecuteDatasetAsync(string, string, CancellationToken, params MySqlParameter[])
 - Task<DataSet> ExecuteDatasetAsync(MySqlConnection, string)
 - Task<DataSet> ExecuteDatasetAsync(MySqlConnection, string, CancellationToken)
 - Task<DataSet> ExecuteDatasetAsync(MySqlConnection, string, params MySqlParameter[])
 - Task<DataSet> ExecuteDatasetAsync(MySqlConnection, string, CancellationToken, params MySqlParameter[])
 - Task UpdateDataSetAsync(string, string, DataSet, string)
 - Task UpdateDataSetAsync(string, string, DataSet, string, CancellationToken)

- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, MySqlTransaction, string, MySqlParameter[], bool)
- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, MySqlTransaction, string, MySqlParameter[], bool, CancellationToken)
- Task<MySqlDataReader> ExecuteReaderAsync(string, string)
- Task<MySqlDataReader> ExecuteReaderAsync(string, string, CancellationToken)
- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, string)
- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, string, CancellationToken)
- Task<MySqlDataReader> ExecuteReaderAsync(string, string, params MySqlParameter[])
- Task<MySqlDataReader> ExecuteReaderAsync(string, string, CancellationToken, params MySqlParameter[])
- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, string, params MySqlParameter[])
- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, string, CancellationToken, params MySqlParameter[])
- Task<object> ExecuteScalarAsync(string, string)
- Task<object> ExecuteScalarAsync(string, string, CancellationToken)
- Task<object> ExecuteScalarAsync(string, string, params MySqlParameter[])
- Task<object> ExecuteScalarAsync(string, string, CancellationToken, params MySqlParameter[])
- Task<object> ExecuteScalarAsync(MySqlConnection, string)
- Task<object> ExecuteScalarAsync(MySqlConnection, string, CancellationToken)
- Task<object> ExecuteScalarAsync(MySqlConnection, string, params MySqlParameter[])
- Task<object> ExecuteScalarAsync(MySqlConnection, string, CancellationToken, params MySqlParameter[])
- Class MySqlCommand
 - Task<int> ExecuteAsync()
 - Task<int> ExecuteAsync(CancellationToken)

In addition to the methods listed above, the following are methods inherited from the .NET Framework:

- Namespace MySql.Data.Entity
 - Class EFMySqlCommand
 - Task<DbDataReader> ExecuteDbDataReaderAsync(CommandBehaviour, CancellationToken)
 - Task<int> ExecuteNonQueryAsync()
 - Task<int> ExecuteNonQueryAsync(CancellationToken)
 - Task<DbDataReader> ExecuteReaderAsync()

- Task<DbDataReader> ExecuteReaderAsync(CancellationTokens)
- Task<DbDataReader> ExecuteReaderAsync(CommandBehaviour)
- Task<DbDataReader> ExecuteReaderAsync(CommandBehaviour, CancellationTokens)
- Task<object> ExecuteScalarAsync()
- Task<object> ExecuteScalarAsync(CancellationTokens)
- Namespace MySql.Data
 - Class MySqlCommand
 - Task<DbDataReader> ExecuteDbDataReaderAsync(CommandBehaviour, CancellationTokens)
 - Task<int> ExecuteNonQueryAsync()
 - Task<int> ExecuteNonQueryAsync(CancellationTokens)
 - Task<DbDataReader> ExecuteReaderAsync()
 - Task<DbDataReader> ExecuteReaderAsync(CancellationTokens)
 - Task<DbDataReader> ExecuteReaderAsync(CommandBehaviour)
 - Task<DbDataReader> ExecuteReaderAsync(CommandBehaviour, CancellationTokens)
 - Task<object> ExecuteScalarAsync()
 - Task<object> ExecuteScalarAsync(CancellationTokens)
 - Class MySqlConnection
 - Task OpenAsync()
 - Task OpenAsync(CancellationTokens)
 - Class MySqlDataReader
 - Task<T> GetFieldValueAsync<T>(int)
 - Task<T> GetFieldValueAsync<T>(int, CancellationTokens)
 - Task<bool> IsDBNullAsync(int)
 - Task<bool> IsDBNullAsync(int, CancellationTokens)
 - Task<bool> NextResultAsync()
 - Task<bool> NextResultAsync(CancellationTokens)
 - Task<bool> ReadAsync()
 - Task<bool> ReadAsync(CancellationTokens)

Examples

The following examples demonstrate how to use the Asynchronous methods:

In this example, a method has the "async" modifier because the method "await" call made applies to the method LoadAsync. The method returns a Task which contains information about the result of the

awaited method. Returning Task is like having a void method, but you should not use "async void" if your method is not a top level access method, like an event.

```
public async Task BulkLoadAsync()
{
    MySqlConnection myConn = new MySqlConnection("MyConnectionString");
    MySqlBulkLoader loader = new MySqlBulkLoader(myConn);

    loader.TableName      = "BulkLoadTest";
    loader.FileName        = @"c:\MyPath\MyFile.txt";
    loader.Timeout         = 0;

    var result             = await loader.LoadAsync();
}
```

In this example, an "async void" method is used with "await" for the ExecuteNonQueryAsync method, to correspond to the onclick event of a button. This is why the method does not return a Task.

```
private async void myButton_Click()
{
    MySqlConnection myConn = new MySqlConnection("MyConnectionString");
    MySqlCommand proc      = new MySqlCommand("MyAsyncSpTest", myConn);

    proc.CommandType       = CommandType.StoredProcedure;

    int result              = await proc.ExecuteNonQueryAsync();
}
```

4.5.13 Using the Connector/Net Interceptor Classes

An interceptor is a software design pattern that provides a transparent way to extend or modify some aspect of a program, similar to a user exit. No recompiling is required. With Connector/Net, the interceptors are enabled and disabled by updating the connection string to refer to different sets of interceptor classes that you instantiate.

Connector/Net includes the following interceptor classes:

- The [BaseCommandInterceptor](#) lets you perform additional operations when a program issues a SQL command. For example, you can examine the SQL statement for logging or debugging purposes, substitute your own result set to implement a caching mechanism, and so on. Depending on the use case, your code can supplement the SQL command or replace it entirely.

The [BaseCommandInterceptor](#) class has these methods that you can override:

```
public virtual bool ExecuteScalar(string sql, ref object returnValue);
public virtual bool ExecuteNonQuery(string sql, ref int returnValue);
public virtual bool ExecuteReader(string sql, CommandBehavior behavior, ref MySqlDataReader returnValue);
public virtual void Init(MySqlConnection connection);
```

If your interceptor overrides one of the [Execute...](#) methods, set the [returnValue](#) output parameter and return [true](#) if you handled the event, or [false](#) if you did not handle the event. The SQL command is processed normally only when all command interceptors return [false](#).

The connection passed to the [Init](#) method is the connection that is attached to this interceptor.

- The [BaseExceptionInterceptor](#) lets you perform additional operations when a program encounters an SQL exception. The exception interception mechanism is modeled after the Connector/J model. You can code an interceptor class and connect it to an existing program without

recompiling, and intercept exceptions when they are created. You can then change the exception type and optionally attach information to it. This capability lets you turn on and off logging and debugging code without hardcoding anything in the application. This technique applies to exceptions raised at the SQL level, not to lower-level system or I/O errors.

You develop an exception interceptor first by creating a subclass of the `BaseExceptionInterceptor` class. You must override the `InterceptException()` method. You can also override the `Init()` method to do some one-time initialization.

Each exception interceptor has 2 methods:

```
public abstract Exception InterceptException(Exception exception,
    MySqlConnection connection);
public virtual void Init(MySqlConnection connection);
```

The connection passed to `Init()` is the connection that is attached to this interceptor.

Each interceptor is required to override `InterceptException` and return an exception. It can return the exception it is given, or it can wrap it in a new exception. We currently do not offer the ability to suppress the exception.

Here are examples of using the FQN (fully qualified name) on the connection string:

```
MySqlConnection c1 = new MySqlConnection(@"server=localhost;pooling=false;
commandinterceptors=CommandApp.MyCommandInterceptor,CommandApp");
MySqlConnection c2 = new MySqlConnection(@"server=localhost;pooling=false;
exceptioninterceptors=ExceptionStackTraceTest.MyExceptionInterceptor,ExceptionStackTraceTest");
```

In this example, the command interceptor is called `CommandApp.MyCommandInterceptor` and exists in the `CommandApp` assembly. The exception interceptor is called `ExceptionStackTraceTest.MyExceptionInterceptor` and exists in the `ExceptionStackTraceTest` assembly.

To shorten the connection string, you can register your exception interceptors in your `app.config` or `web.config` file like this:

```
<configSections>
<section name="MySQL" type="MySQL.Data.MySqlClient.MySqlConfiguration,
MySQL.Data"/>
</configSections>
<MySQL>
<CommandInterceptors>
  <add name="myC" type="CommandApp.MyCommandInterceptor,CommandApp" />
</CommandInterceptors>
</MySQL>
<configSections>
<section name="MySQL" type="MySQL.Data.MySqlClient.MySqlConfiguration,
MySQL.Data"/>
</configSections>
<MySQL>
<ExceptionInterceptors>
  <add name="myE"
type="ExceptionStackTraceTest.MyExceptionInterceptor,ExceptionStackTraceTest" />
</ExceptionInterceptors>
</MySQL>
```

Once you have done that, your connection strings can look like these:

```
MySqlConnection c1 = new MySqlConnection(@"server=localhost;pooling=false;
commandinterceptors=myC");
MySqlConnection c2 = new MySqlConnection(@"server=localhost;pooling=false;
exceptioninterceptors=myE");
```


4.5.14 Handling Date and Time Information in Connector/Net

MySQL and the .NET languages handle date and time information differently, with MySQL allowing dates that cannot be represented by a .NET data type, such as '0000-00-00 00:00:00'. These differences can cause problems if not properly handled.

The following sections demonstrate how to properly handle date and time information when using Connector/Net.

4.5.14.1 Fractional Seconds

Connector/Net 6.5 and higher support the fractional seconds feature introduced in MySQL 5.6.4. Fractional seconds could always be specified in a date literal or passed back and forth as parameters and return values, but the fractional part was always stripped off when stored in a table column. In MySQL 5.6.4 and higher, the fractional part is now preserved in data stored and retrieved through SQL. For fractional second handling in MySQL 5.6.4 and higher, see [Fractional Seconds in Time Values](#). For the behavior of fractional seconds prior to MySQL 5.6.4, see [Fractional Seconds in Time Values](#).

To use the more precise date and time types, specify a value from 1 to 6 when creating the table column, for example `TIME(3)` or `DATETIME(6)`, representing the number of digits of precision after the decimal point. Specifying a precision of 0 leaves the fractional part out entirely. In your C# or Visual Basic code, refer to the `Millisecond` member to retrieve the fractional second value from the `MySqlDateTime` object returned by the `GetMySqlDateTime` function. The `DateTime` object returned by the `GetDateTime` function also contains the fractional value, but only the first 3 digits.

For related code examples, see the following blog post: https://blogs.oracle.com/MySqlOnWindows/entry/milliseconds_value_support_on_datetime

4.5.14.2 Problems when Using Invalid Dates

The differences in date handling can cause problems for developers who use invalid dates. Invalid MySQL dates cannot be loaded into native .NET `DateTime` objects, including `NULL` dates.

Because of this issue, .NET `DataSet` objects cannot be populated by the `Fill` method of the `MySqlDataAdapter` class as invalid dates will cause a `System.ArgumentOutOfRangeException` exception to occur.

4.5.14.3 Restricting Invalid Dates

The best solution to the date problem is to restrict users from entering invalid dates. This can be done on either the client or the server side.

Restricting invalid dates on the client side is as simple as always using the .NET `DateTime` class to handle dates. The `DateTime` class will only allow valid dates, ensuring that the values in your database are also valid. The disadvantage of this is that it is not useful in a mixed environment where .NET and non .NET code are used to manipulate the database, as each application must perform its own date validation.

Users of MySQL 5.0.2 and higher can use the new `traditional` SQL mode to restrict invalid date values. For information on using the `traditional` SQL mode, see [Server SQL Modes](#).

4.5.14.4 Handling Invalid Dates

Although it is strongly recommended that you avoid the use of invalid dates within your .NET application, it is possible to use invalid dates by means of the `MySqlDateTime` data type.

The `MySqlDateTime` data type supports the same date values that are supported by the MySQL server. The default behavior of Connector/Net is to return a .NET `DateTime` object for valid date values, and return an error for invalid dates. This default can be modified to cause Connector/Net to return `MySqlDateTime` objects for invalid dates.

To instruct Connector/Net to return a `MySqlDateTime` object for invalid dates, add the following line to your connection string:

```
Allow Zero Datetime=True
```

The `MySqlDateTime` class can still be problematic. The following are some known issues:

1. Data binding for invalid dates can still cause errors (zero dates like 0000-00-00 do not seem to have this problem).
2. The `Tostring` method return a date formatted in the standard MySQL format (for example, `2005-02-23 08:50:25`). This differs from the `Tostring` behavior of the .NET `DateTime` class.
3. The `MySqlDateTime` class supports NULL dates, while the .NET `DateTime` class does not. This can cause errors when trying to convert a `MySQLDateTime` to a `DateTime` if you do not check for NULL first.

Because of the known issues, the best recommendation is still to use only valid dates in your application.

4.5.14.5 Handling NULL Dates

The .NET `DateTime` data type cannot handle `NULL` values. As such, when assigning values from a query to a `DateTime` variable, you must first check whether the value is in fact `NULL`.

When using a `MySqlDataReader`, use the `.IsDBNull` method to check whether a value is `NULL` before making the assignment:

Visual Basic Example

```
If Not myReader.IsDBNull(myReader.GetOrdinal("mytime")) Then
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"))
Else
    myTime = DateTime.MinValue
End If
```

C# Example

```
if (! myReader.IsDBNull(myReader.GetOrdinal("mytime")))
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"));
else
    myTime = DateTime.MinValue;
```

`NULL` values will work in a data set and can be bound to form controls without special handling.

4.5.15 Using the MySqlBulkLoader Class

MySQL Connector/Net features a bulk loader class that wraps the MySQL statement `LOAD DATA INFILE`. This gives MySQL Connector/Net the ability to load a data file from a local or remote host to the server. The class concerned is `MySqlBulkLoader`. This class has various methods, the main one being `load` to cause the specified file to be loaded to the server. Various parameters can be set to control how the data file is processed. This is achieved through setting various properties of the class. For example, the field separator used, such as comma or tab, can be specified, along with the record terminator, such as newline.

The following code shows a simple example of using the `MySqlBulkLoader` class. First an empty table needs to be created, in this case in the `test` database:

```
CREATE TABLE Career (
    Name VARCHAR(100) NOT NULL,
    Age INTEGER,
    Profession VARCHAR(200)
);
```

A simple tab-delimited data file is also created (it could use any other field delimiter such as comma):

```
Table Career in Test Database
Name Age Profession
Tony 47 Technical Writer
Ana 43 Nurse
Fred 21 IT Specialist
Simon 45 Hairy Biker
```

The first three lines need to be ignored with this test file, as they do not contain table data. This can be achieved using the [NumberOfLinesToSkip](#) property. This file can then be loaded and used to populate the [Career](#) table in the [test](#) database:

```
using System;
using System.Text;
using MySql.Data;
using MySql.Data.MySqlClient;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=test;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);
            MySqlBulkLoader bl = new MySqlBulkLoader(conn);
            bl.TableName = "Career";
            bl.FieldTerminator = "\t";
            bl.LineTerminator = "\n";
            bl.FileName = "c:/career_data.txt";
            bl.NumberOfLinesToSkip = 3;
            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
                // Upload data from file
                int count = bl.Load();
                Console.WriteLine(count + " lines uploaded.");
                string sql = "SELECT Name, Age, Profession FROM Career";
                MySqlCommand cmd = new MySqlCommand(sql, conn);
                MySqlDataReader rdr = cmd.ExecuteReader();
                while (rdr.Read())
                {
                    Console.WriteLine(rdr[0] + " -- " + rdr[1] + " -- " + rdr[2]);
                }
                rdr.Close();
                conn.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            Console.WriteLine("Done.");
        }
    }
}
```

Further information on [LOAD DATA INFILE](#) can be found in [LOAD DATA INFILE Syntax](#). Further information on [MySqlBulkLoader](#) can be found in the reference documentation that was included with your connector.

4.5.16 Using the MySQL Connector/Net Trace Source Object

MySQL Connector/Net 6.2 introduced support for .NET 2.0 compatible tracing, using [TraceSource](#) objects.

The .NET 2.0 tracing architecture consists of four main parts:

- *Source* - This is the originator of the trace information. The source is used to send trace messages. The name of the source provided by MySQL Connector/Net is `mysql`.
- *Switch* - This defines the level of trace information to emit. Typically, this is specified in the `app.config` file, so that it is not necessary to recompile an application to change the trace level.
- *Listener* - Trace listeners define where the trace information will be written to. Supported listeners include, for example, the Visual Studio Output window, the Windows Event Log, and the console.
- *Filter* - Filters can be attached to listeners. Filters determine the level of trace information that will be written. While a switch defines the level of information that will be written to all listeners, a filter can be applied on a per-listener basis, giving finer grained control of trace information.

To use tracing `MySql.Data.MySqlClient.MySqlTrace` can be used as a `TraceSource` for MySQL Connector/Net and the connection string must include `"Logging=True"`.

To enable trace messages, configure a trace switch. Trace switches have associated with them a trace level enumeration, these are **Off**, **Error**, **Warning**, **Info**, and **Verbose**.

```
MySqlTrace.Switch.Level = SourceLevels.Verbose;
```

This sets the trace level to **Verbose**, meaning that all trace messages will be written.

It is convenient to be able to change the trace level without having to recompile the code. This is achieved by specifying the trace level in application configuration file, `app.config`. You then simply need to specify the desired trace level in the configuration file and restart the application. The trace source is configured within the `system.diagnostics` section of the file. The following XML snippet illustrates this:

```
<configuration>
...
<system.diagnostics>
  <sources>
    <source name="mysql" switchName="MySwitch"
           switchType="System.Diagnostics.SourceSwitch" />
    ...
  </sources>
  <switches>
    <add name="MySwitch" value="Verbose"/>
    ...
  </switches>
</system.diagnostics>
...
</configuration>
```

By default, trace information is written to the Output window of Microsoft Visual Studio. There are a wide range of listeners that can be attached to the trace source, so that trace messages can be written out to various destinations. You can also create custom listeners to allow trace messages to be written to other destinations as mobile devices and web services. A commonly used example of a listener is `ConsoleTraceListener`, which writes trace messages to the console.

To add a listener at runtime, use code such as the following:

```
ts.Listeners.Add(new ConsoleTraceListener());
```

Then, call methods on the trace source object to generate trace information. For example, the `TraceInformation()`, `TraceEvent()`, or `TraceData()` methods can be used.

4.5.16.1 Viewing MySQL Trace Information

This section describes how to set up your application to view MySQL trace information.

The first thing you need to do is create a suitable `app.config` file for your application. An example is shown in the following code:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <source name="mysql" switchName="SourceSwitch"
        switchType="System.Diagnostics.SourceSwitch" >
        <listeners>
          <add name="console" />
          <remove name="Default" />
        </listeners>
      </source>
    </sources>
    <switches>

      <add name="SourceSwitch" value="Verbose" />

    </switches>
    <sharedListeners>
      <add name="console"
        type="System.Diagnostics.ConsoleTraceListener"
        initializeData="false"/>
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

This ensures a suitable trace source is created, along with a switch. The switch level in this case is set to `Verbose` to display the maximum amount of information.

In the application the only other step required is to add `logging=true` to the connection string. An example application could be:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using MySql.Data;
using MySql.Data.MySqlClient;
using MySql.Web;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      string connStr = "server=localhost;user=root;database=world;port=3306;password=*****;logging=true";
      MySqlConnection conn = new MySqlConnection(connStr);
      try
      {
        Console.WriteLine("Connecting to MySQL...");
        conn.Open();
        string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'";
        MySqlCommand cmd = new MySqlCommand(sql, conn);
        MySqlDataReader rdr = cmd.ExecuteReader();
        while (rdr.Read())
        {
          Console.WriteLine(rdr[0] + " -- " + rdr[1]);
        }
        rdr.Close();
        conn.Close();
      }
      catch (Exception ex)
      {
      }
    }
  }
}
```

```

        Console.WriteLine(ex.ToString());
    }
    Console.WriteLine("Done.");
}
}
}

```

This simple application will then generate the following output:

```

Connecting to MySQL...
mysql Information: 1 : 1: Connection Opened: connection string = 'server=localhost;User Id=root;database=world;password=*****;logging=True'
mysql Information: 3 : 1: Query Opened: SHOW VARIABLES
mysql Information: 4 : 1: Resultset Opened: field(s) = 2, affected rows = -1, inserted id = -1
mysql Information: 5 : 1: Resultset Closed. Total rows=272, skipped rows=0, size (bytes)=7058
mysql Information: 6 : 1: Query Closed
mysql Information: 3 : 1: Query Opened: SHOW COLLATION
mysql Information: 4 : 1: Resultset Opened: field(s) = 6, affected rows = -1, inserted id = -1
mysql Information: 5 : 1: Resultset Closed. Total rows=127, skipped rows=0, size (bytes)=4102
mysql Information: 6 : 1: Query Closed
mysql Information: 3 : 1: Query Opened: SET character_set_results=NULL
mysql Information: 4 : 1: Resultset Opened: field(s) = 0, affected rows = 0, inserted id = 0
mysql Information: 5 : 1: Resultset Closed. Total rows=0, skipped rows=0, size (bytes)=0
mysql Information: 6 : 1: Query Closed
mysql Information: 10 : 1: Set Database: world
mysql Information: 3 : 1: Query Opened: SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'
mysql Information: 4 : 1: Resultset Opened: field(s) = 2, affected rows = -1, inserted id = -1
American Samoa -- George W. Bush
Australia -- Elisabeth II
...
Wallis and Futuna -- Jacques Chirac
Vanuatu -- John Bani
United States Minor Outlying Islands -- George W. Bush
mysql Information: 5 : 1: Resultset Closed. Total rows=28, skipped rows=0, size (bytes)=788
mysql Information: 6 : 1: Query Closed
Done.
mysql Information: 2 : 1: Connection Closed

```

The first number displayed in the trace message corresponds to the MySQL event type:

Event	Description
1	ConnectionOpened: connection string
2	ConnectionClosed:
3	QueryOpened: mysql server thread id, query text
4	ResultOpened: field count, affected rows (-1 if select), inserted id (-1 if select)
5	ResultClosed: total rows read, rows skipped, size of resultset in bytes
6	QueryClosed:
7	StatementPrepared: prepared sql, statement id
8	StatementExecuted: statement id, mysql server thread id
9	StatementClosed: statement id
10	NonQuery: [varies]
11	UsageAdvisorWarning: usage advisor flag. NoIndex = 1, BadIndex = 2, SkippedRows = 3, SkippedColumns = 4, FieldConversion = 5.
12	Warning: level, code, message
13	Error: error number, error message

The second number displayed in the trace message is the connection count.

Although this example uses the `ConsoleTraceListener`, any of the other standard listeners could have been used. Another possibility is to create a custom listener that uses the information passed using the `TraceEvent` method. For example, a custom trace listener could be created to perform active monitoring of the MySQL event messages, rather than simply writing these to an output device.

It is also possible to add listeners to the MySQL Trace Source at runtime. This can be done with the following code:

```
MySQLTrace.Listeners.Add(new ConsoleTraceListener());
```

MySQL Connector/Net 6.3.2 introduced the ability to switch tracing on and off at runtime. This can be achieved using the calls `MySQLTrace.EnableQueryAnalyzer(string host, int postInterval)` and `MySQLTrace.DisableQueryAnalyzer()`. The parameter `host` is the URL of the MySQL Enterprise Monitor server to monitor. The parameter `postInterval` is how often to post the data to MySQL Enterprise Monitor, in seconds.

4.5.16.2 Building Custom Listeners

To build custom listeners that work with the MySQL Connector/Net Trace Source, it is necessary to understand the key methods used, and the event data formats used.

The main method involved in passing trace messages is the `TraceSource.TraceEvent` method. This has the prototype:

```
public void TraceEvent(
    TraceEventType eventType,
    int id,
    string format,
    params Object[] args
)
```

This trace source method will process the list of attached listeners and call the listener's `TraceListener.TraceEvent` method. The prototype for the `TraceListener.TraceEvent` method is as follows:

```
public virtual void TraceEvent(
    TraceEventCache eventCache,
    string source,
    TraceEventType eventType,
    int id,
    string format,
    params Object[] args
)
```

The first three parameters are used in the standard as [defined by Microsoft](#). The last three parameters contain MySQL-specific trace information. Each of these parameters is now discussed in more detail.

`int id`

This is a MySQL-specific identifier. It identifies the MySQL event type that has occurred, resulting in a trace message being generated. This value is defined by the `MySQLTraceEventType` public enum contained in the MySQL Connector/Net code:

```
public enum MySQLTraceEventType : int
{
    ConnectionOpened = 1,
    ConnectionClosed,
    QueryOpened,
    ResultOpened,
    ResultClosed,
    QueryClosed,
    StatementPrepared,
    StatementExecuted,
    StatementClosed,
```

```

NonQuery,
UsageAdvisorWarning,
Warning,
Error
}

```

The MySQL event type also determines the contents passed using the parameter `params Object[] args`. The nature of the `args` parameters are described in further detail in the following material.

string format

This is the format string that contains zero or more format items, which correspond to objects in the `args` array. This would be used by a listener such as `ConsoleTraceListener` to write a message to the output device.

params Object[] args

This is a list of objects that depends on the MySQL event type, `id`. However, the first parameter passed using this list is always the driver id. The driver id is a unique number that is incremented each time the connector is opened. This enables groups of queries on the same connection to be identified. The parameters that follow driver id depend on the MySQL event id, and are as follows:

MySQL-specific event type	Arguments (params Object[] args)
ConnectionOpened	Connection string
ConnectionClosed	No additional parameters
QueryOpened	mysql server thread id, query text
ResultOpened	field count, affected rows (-1 if select), inserted id (-1 if select)
ResultClosed	total rows read, rows skipped, size of resultset in bytes
QueryClosed	No additional parameters
StatementPrepared	prepared sql, statement id
StatementExecuted	statement id, mysql server thread id
StatementClosed	statement id
NonQuery	Varies
UsageAdvisorWarning	usage advisor flag. NoIndex = 1, BadIndex = 2, SkippedRows = 3, SkippedColumns = 4, FieldConversion = 5.
Warning	level, code, message
Error	error number, error message

This information will allow you to create custom trace listeners that can actively monitor the MySQL-specific events.

4.5.17 Binary/Nonbinary Issues

There are certain situations where MySQL will return incorrect metadata about one or more columns. More specifically, the server will sometimes report that a column is binary when it is not and vice versa. In these situations, it becomes practically impossible for the connector to be able to correctly identify the correct metadata.

Some examples of situations that may return incorrect metadata are:

- Execution of `SHOW PROCESSLIST`. Some of the columns will be returned as binary even though they only hold string data.
- When a temporary table is used to process a resultset, some columns may be returned with incorrect binary flags.
- Some server functions such `DATE_FORMAT` will incorrectly return the column as binary.

With the availability of [BINARY](#) and [VARBINARY](#) data types, it is important that we respect the metadata returned by the server. However, we are aware that some existing applications may break with this change, so we are creating a connection string option to enable or disable it. By default, Connector/Net 5.1 respects the binary flags returned by the server. You might need to make small changes to your application to accommodate this change.

In the event that the changes required to your application would be too large, adding `'respect binary flags=false'` to your connection string causes the connector to use the prior behavior: any column that is marked as string, regardless of binary flags, will be returned as string. Only columns that are specifically marked as a [BLOB](#) will be returned as [BLOB](#).

4.5.18 Character Set Considerations for Connector/Net

Treating Binary Blobs As UTF8

Before the introduction of [The utf8mb4 Character Set \(4-Byte UTF-8 Unicode Encoding\)](#) (in MySQL 5.5.3), MySQL did not support 4-byte UTF8 sequences. This makes it difficult to represent some multibyte languages such as Japanese. To try and alleviate this, Connector/Net supports a mode where binary blobs can be treated as strings.

To do this, you set the `'Treat Blobs As UTF8'` connection string keyword to [yes](#). This is all that needs to be done to enable conversion of all binary blobs to UTF8 strings. To convert only some of your BLOB columns, you can make use of the `'BlobAsUTF8IncludePattern'` and `'BlobAsUTF8ExcludePattern'` keywords. Set these to a regular expression pattern that matches the column names to include or exclude respectively.

When the regular expression patterns both match a single column, the include pattern is applied before the exclude pattern. The result, in this case, would be that the column would be excluded. Also, be aware that this mode does not apply to columns of type [BINARY](#) or [VARBINARY](#) and also do not apply to nonbinary [BLOB](#) columns.

This mode only applies to reading strings out of MySQL. To insert 4-byte UTF8 strings into blob columns, use the `.NET Encoding.GetBytes` function to convert your string to a series of bytes. You can then set this byte array as a parameter for a [BLOB](#) column.

4.5.19 Using Connector/Net with Crystal Reports

Crystal Reports is a common tool used by Windows application developers to perform reporting and document generation. In this section we will show how to use Crystal Reports XI with MySQL and Connector/Net.

4.5.19.1 Creating a Data Source

When creating a report in Crystal Reports there are two options for accessing the MySQL data while designing your report.

The first option is to use Connector/ODBC as an ADO data source when designing your report. You will be able to browse your database and choose tables and fields using drag and drop to build your report. The disadvantage of this approach is that additional work must be performed within your application to produce a data set that matches the one expected by your report.

The second option is to create a data set in VB.NET and save it as XML. This XML file can then be used to design a report. This works quite well when displaying the report in your application, but is less versatile at design time because you must choose all relevant columns when creating the data set. If you forget a column you must re-create the data set before the column can be added to the report.

The following code can be used to create a data set from a query and write it to disk:

Visual Basic Example

```

Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter
conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"
Try
    conn.Open()
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn
    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)
    myData.WriteXml("C:\dataset.xml", XmlWriteMode.WriteSchema)
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;
conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();
conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;
    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);
    myData.WriteXml(@"C:\dataset.xml", XmlWriteMode.WriteSchema);
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

The resulting XML file can be used as an ADO.NET XML datasource when designing your report.

If you choose to design your reports using Connector/ODBC, it can be downloaded from dev.mysql.com.

4.5.19.2 Creating the Report

For most purposes, the Standard Report wizard helps with the initial creation of a report. To start the wizard, open Crystal Reports and choose the **New > Standard Report** option from the File menu.

The wizard first prompts you for a data source. If you use Connector/ODBC as your data source, use the OLEDB provider for ODBC option from the OLE DB (ADO) tree instead of the ODBC (RDO) tree when choosing a data source. If using a saved data set, choose the ADO.NET (XML) option and browse to your saved data set.

The remainder of the report creation process is done automatically by the wizard.

After the report is created, choose the Report Options... entry of the File menu. Un-check the Save Data With Report option. This prevents saved data from interfering with the loading of data within our application.

4.5.19.3 Displaying the Report

To display a report we first populate a data set with the data needed for the report, then load the report and bind it to the data set. Finally we pass the report to the crViewer control for display to the user.

The following references are needed in a project that displays a report:

- CrystalDecisions.CrystalReports.Engine
- CrystalDecisions.ReportSource
- CrystalDecisions.Shared
- CrystalDecisions.Windows.Forms

The following code assumes that you created your report using a data set saved using the code shown in [Section 4.5.19.1, "Creating a Data Source"](#), and have a crViewer control on your form named `myViewer`.

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient
Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter
conn.ConnectionString = _
    "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"
Try
    conn.Open()
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn
    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)
    myReport.Load(".\world_report.rpt")
    myReport.SetDataSource(myData)
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;
ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;
conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();
conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
```

```

        "FROM country, city ORDER BY country.continent, country.name";
        cmd.Connection = conn;
        myAdapter.SelectCommand = cmd;
        myAdapter.Fill(myData);
        myReport.Load(@".\world_report.rpt");
        myReport.SetDataSource(myData);
        myViewer.ReportSource = myReport;
    }
    catch (MySQL.Data.MySqlClient.MySqlException ex)
    {
        MessageBox.Show(ex.Message, "Report could not be created",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

A new data set is generated using the same query used to generate the previously saved data set. Once the data set is filled, a ReportDocument is used to load the report file and bind it to the data set. The ReportDocument is then passed as the ReportSource of the crViewer.

This same approach is taken when a report is created from a single table using Connector/ODBC. The data set replaces the table used in the report and the report is displayed properly.

When a report is created from multiple tables using Connector/ODBC, a data set with multiple tables must be created in our application. This enables each table in the report data source to be replaced with a report in the data set.

We populate a data set with multiple tables by providing multiple **SELECT** statements in our MySqlCommand object. These **SELECT** statements are based on the SQL query shown in Crystal Reports in the Database menu's Show SQL Query option. Assume the following query:

```

SELECT `country`.`Name`, `country`.`Continent`, `country`.`Population`, `city`.`Name`, `city`.`Population`
FROM `world`.`country` `country` LEFT OUTER JOIN `world`.`city` `city` ON `country`.`Code`=`city`.`CountryCode`
ORDER BY `country`.`Continent`, `country`.`Name`, `city`.`Name`

```

This query is converted to two **SELECT** queries and displayed with the following code:

Visual Basic Example

```

Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySQL.Data.MySqlClient
Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter
conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"
Try
    conn.Open()
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER BY countrycode, name;" _
        & "SELECT name, population, code, continent FROM country ORDER BY continent, name"
    cmd.Connection = conn
    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)
    myReport.Load(@".\world_report.rpt")
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0))
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1))
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

```

C# Example

```

using CrystalDecisions.CrystalReports.Engine;

```

```

using System.Data;
using MySql.Data.MySqlClient;
ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;
conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();
conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
try
{
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER " +
        "BY countrycode, name; SELECT name, population, code, continent FROM " +
        "country ORDER BY continent, name";
    cmd.Connection = conn;
    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);
    myReport.Load(@"..\world_report.rpt");
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0));
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1));
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

It is important to order the **SELECT** queries in alphabetic order, as this is the order the report will expect its source tables to be in. One SetDataSource statement is needed for each table in the report.

This approach can cause performance problems because Crystal Reports must bind the tables together on the client-side, which will be slower than using a pre-saved data set.

4.5.20 ASP.NET Provider Model

MySQL Connector/Net provides support for the ASP.NET 2.0 provider model. This model enables application developers to focus on the business logic of their application instead of having to recreate such boilerplate items as membership and roles support.

MySQL Connector/Net supplies the following providers:

- Membership Provider
- Role Provider
- Profile Provider
- Session State Provider (MySQL Connector/Net 6.1 and later)

The following tables show the supported providers, their default provider and the corresponding MySQL provider.

Membership Provider

Default Provider	MySQL Provider
System.Web.Security.SqlMembershipProvider	MySql.Web.Security.MySQLMembershipProvider

Role Provider

Default Provider	MySQL Provider
System.Web.Security.SqlRoleProvider	MySql.Web.Security.MySQLRoleProvider

Profile Provider

Default Provider	MySQL Provider
System.Web.Profile.SqlProfileProvider	MySql.Web.Profile.MySQLProfileProvider

SessionState Provider

Default Provider	MySQL Provider
System.Web.SessionState.InProcSessionStateStore	MySql.Web.SessionState.MySqlSessionStateStore

Note

The MySQL Session State provider uses slightly different capitalization on the class name compared to the other MySQL providers.

Installing The Providers

The installation of Connector/Net 5.1 or later will install the providers and register them in your machine's .NET configuration file, [machine.config](#). The additional entries created will result in the [system.web](#) section appearing similar to the following code:

```
<system.web>
  <processModel autoConfig="true" />
  <httpHandlers />
  <membership>
    <providers>
      <add name="AspNetSqlMembershipProvider" type="System.Web.Security.SqlMembershipProvider, System.Web, Version=6.1.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
      <add name="MySQLMembershipProvider" type="MySql.Web.Security.MySQLMembershipProvider, MySql.Web, Version=6.1.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
    </providers>
  </membership>
  <profile>
    <providers>
      <add name="AspNetSqlProfileProvider" connectionStringName="LocalSqlServer" applicationName="/" type="System.Web.Profile.SqlProfileProvider, System.Web, Version=6.1.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
      <add name="MySQLProfileProvider" type="MySql.Web.Profile.MySQLProfileProvider, MySql.Web, Version=6.1.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
    </providers>
  </profile>
  <roleManager>
    <providers>
      <add name="AspNetSqlRoleProvider" connectionStringName="LocalSqlServer" applicationName="/" type="System.Web.Security.SqlRoleProvider, System.Web, Version=6.1.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
      <add name="AspNetWindowsTokenRoleProvider" applicationName="/" type="System.Web.Security.WindowsTokenRoleProvider, System.Web, Version=6.1.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
      <add name="MySQLRoleProvider" type="MySql.Web.Security.MySQLRoleProvider, MySql.Web, Version=6.1.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
    </providers>
  </roleManager>
</system.web>
```

Each provider type can have multiple provider implementations. The default provider can also be set here using the [defaultProvider](#) attribute, but usually this is set in the [web.config](#) file either manually or by using the ASP.NET configuration tool.

At time of writing, the [MySqlSessionStateStore](#) is not added to [machine.config](#) at install time, and so add the following:

```
<sessionState>
  <providers>
    <add name="MySqlSessionStateStore" type="MySql.Web.SessionState.MySqlSessionStateStore, MySql.Web, Version=6.1.1.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
  </providers>
</sessionState>
```

The SessionState Provider uses the [customProvider](#) attribute, rather than [defaultProvider](#), to set the provider as the default. A typical [web.config](#) file might contain:

```
<system.web>
  <membership defaultProvider="MySQLMembershipProvider" />
  <roleManager defaultProvider="MySQLRoleProvider" />
  <profile defaultProvider="MySQLProfileProvider" />
  <sessionState customProvider="MySqlSessionStateStore" />
  <compilation debug="false">
```

...

This sets the MySQL Providers as the defaults to be used in this web application.

The providers are implemented in the file `mysql.web.dll` and this file can be found in your MySQL Connector/Net installation folder. There is no need to run any type of SQL script to set up the database schema, as the providers create and maintain the proper schema automatically.

Using The Providers

The easiest way to start using the providers is to use the ASP.NET configuration tool that is available on the Solution Explorer toolbar when you have a website project loaded.

In the web pages that open, you can select the MySQL membership and roles providers by picking a custom provider for each area.

When the provider is installed, it creates a dummy connection string named `LocalMySqlServer`. Although this has to be done so that the provider will work in the ASP.NET configuration tool, you override this connection string in your `web.config` file. You do this by first removing the dummy connection string and then adding in the proper one, as shown in the following example:

```
<connectionStrings>
  <remove name="LocalMySqlServer"/>
  <add name="LocalMySqlServer" connectionString="server=xxx;uid=xxx;pwd=xxx;database=xxx;" />
</connectionStrings>
```

Note

You must specify the database in this connection.

Rather than manually editing configuration files, consider using the MySQL Website Configuration tool in MySQL for Visual Studio to configure your desired provider setup. The tool modifies your `website.config` file to the desired configuration. A tutorial on doing this is available in the following section [MySQL Website Configuration Tool](#).

A tutorial demonstrating how to use the Membership and Role Providers can be found in the following section [Section 4.4.2, "Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider"](#).

Deployment

To use the providers on a production server, distribute the `MySql.Data` and the `MySql.Web` assemblies, and either register them in the remote systems Global Assembly Cache or keep them in your application's `bin/` directory.

4.5.21 Working with Partial Trust / Medium Trust

.NET applications operate under a given trust level. Normal desktop applications operate under full trust, while web applications that are hosted in shared environments are normally run under the partial trust level (also known as "medium trust"). Some hosting providers host shared applications in their own app pools and allow the application to run under full trust, but this configuration is relatively rare. The Connector/Net support for partial trust has improved over time to simplify the configuration and deployment process for hosting providers.

4.5.21.1 Evolution of Partial Trust Support Across Connector/Net Versions

The partial trust support for Connector/Net has improved rapidly throughout the 6.5.x and 6.6.x versions. The latest enhancements do require some configuration changes in existing deployments. Here is a summary of the changes for each version.

6.6.4 and Above: Library Can Be Inside or Outside GAC

Now you can install the `MySql.Data.dll` library in the Global Assembly Cache (GAC) as explained in [Section 4.5.21.2, "Configuring Partial Trust with Connector/Net Library Installed in GAC"](#), or in a `bin` or

`lib` folder inside the project or solution as explained in [Section 4.5.21.3, “Configuring Partial Trust with Connector/Net Library Not Installed in GAC”](#). If the library is not in the GAC, the only protocol supported is TCP/IP.

6.5.1 and Above: Partial Trust Requires Library in the GAC

Connector/Net 6.5 fully enables our provider to run in a partial trust environment when the library is installed in the Global Assembly Cache (GAC). The new `MySQLClientPermission` class, derived from the .NET `DBDataPermission` class, helps to simplify the permission setup.

5.0.8 / 5.1.3 and Above: Partial Trust Requires Socket Permissions

Starting with these versions, Connector/Net can be used under partial trust hosting that has been modified to allow the use of sockets for communication. By default, partial trust does not include `SocketPermission`. Connector/Net uses sockets to talk with the MySQL server, so the hosting provider must create a new trust level that is an exact clone of partial trust but that has the following permissions added:

- `System.Net.SocketPermission`
- `System.Security.Permissions.ReflectionPermission`
- `System.Net.DnsPermission`
- `System.Security.Permissions.SecurityPermission`

Prior to 5.0.8 / 5.1.3: Partial Trust Not Supported

Connector/Net versions prior to 5.0.8 and 5.1.3 were not compatible with partial trust hosting.

4.5.21.2 Configuring Partial Trust with Connector/Net Library Installed in GAC

If the library is installed in the GAC, you must include the connection option `includesecurityasserts=true` in your connection string. This is a new requirement as of Connector/Net 6.6.4.

The following list shows steps and code fragments needed to run a Connector/Net application in a partial trust environment. For illustration purposes, we use the Pipe Connections protocol in this example.

1. Install Connector/Net: version 6.6.1 or higher, or 6.5.4 or higher.
2. After installing the library, make the following configuration changes:

In the `SecurityClasses` section, add a definition for the `MySQLClientPermission` class, including the version to use.

```
<configuration>
  <mscorlib>
    <security>
      <policy>
        <PolicyLevel version="1">
          <SecurityClasses>
            ....
            <SecurityClass Name="MySQLClientPermission" Description="MySQL.Data.MySQLClient.MySQLClientT
```

Scroll down to the `ASP.Net` section:

```
<PermissionSet class="NamedPermissionSet" version="1" Name="ASP.Net">
```


Add a new entry for the detailed configuration of the `MySQLClientPermission` class:

```
<IPermission class="MySQLClientPermission" version="1" Unrestricted="true"/>
```

Note

This configuration is the most generalized way that includes all keywords.

3. Configure the MySQL server to accept pipe connections, by adding the `--enable-named-pipe` option on the command line. If you need more information about this, see [Installing MySQL on Microsoft Windows](#).
4. Confirm that the hosting provider has installed the Connector/Net library (`MySQL.Data.dll`) in the GAC.
5. Optionally, the hosting provider can avoid granting permissions globally by using the new `MySQLClientPermission` class in the trust policies. (The alternative is to globally enable the permissions `System.Net.SocketPermission`, `System.Security.Permissions.ReflectionPermission`, `System.Net.DnsPermission`, and `System.Security.Permissions.SecurityPermission`.)
6. Create a simple web application using Visual Studio 2010.
7. Add the reference in your application for the `MySQL.Data.MySQLClient` library.
8. Edit your `web.config` file so that your application runs using a Medium trust level:

```
<system.web>
  <trust level="Medium"/>
</system.web>
```

9. Add the `MySQL.Data.MySQLClient` namespace to your server-code page.
10. Define the connection string, in slightly different ways depending on the Connector/Net version.

Only for 6.6.4 or later: To use the connections inside any web application that will run in Medium trust, add the new `includesecurityasserts` option to the connection string. `includesecurityasserts=true` that makes the library request the following permissions when required: `SocketPermissions`, `ReflectionPermissions`, `DnsPermissions`, `SecurityPermissions` among others that are not granted in Medium trust levels.

For Connector/Net 6.6.3 or earlier: No special setting for security is needed within the connection string.

```
MySQLConnectionStringBuilder myconnString = new MySQLConnectionStringBuilder("server=localhost;User=
myconnString.PipeName = "MySQL55";
myconnString.ConnectionProtocol = MySQLConnectionProtocol.Pipe;
// Following attribute is a new requirement when the library is in the GAC.
// Could also be done by adding includesecurityasserts=true; to the string literal
// in the constructor above.
// Not needed with Connector/Net 6.6.3 and earlier.
myconnString.IncludeSecurityAsserts = true;
```

11. Define the `MySQLConnection` to use:

```
MySQLConnection myconn = new MySQLConnection(myconnString.ConnectionString);
myconn.Open();
```

- Retrieve some data from your tables:

```
MySQLCommand cmd = new MySQLCommand("Select * from products", myconn);
MySQLDataAdapter da = new MySQLDataAdapter(cmd);
DataSet1 tds = new DataSet1();
da.Fill(tds, tds.Tables[0].TableName);
GridView1.DataSource = tds;
GridView1.DataBind();
myconn.Close()
```

- Run the program. It should execute successfully, without requiring any special code or encountering any security problems.

4.5.21.3 Configuring Partial Trust with Connector/Net Library Not Installed in GAC

When deploying a web application to a Shared Hosted environment, where this environment is configured to run all their .NET applications under a partial or medium trust level, you might not be able to install the Connector/Net library in the GAC. Instead, you put a reference to the library in the [bin](#) or [lib](#) folder inside the project or solution. In this case, you configure the security in a different way than when the library is in the GAC.

Connector/Net is commonly used by applications that run in Windows environments where the default communication for the protocol is used via sockets or by TCP/IP. For this protocol to operate is necessary have the required socket permissions in the web configuration file as follows:

- Open the medium trust policy web configuration file, which should be under this folder:

```
%windir%\Microsoft.NET\Framework\{version}\CONFIG\web_mediumtrust.config
```

Use [Framework64](#) in the path instead of [Framework](#) if you are using a 64-bit installation of the framework.

- Locate the [SecurityClasses](#) tag:

```
<SecurityClass Name="SocketPermission"
Description="System.Net.SocketPermission, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
```

- Scroll down and look for the following [PermissionSet](#):

```
<PermissionSet version="1" Name="ASP.Net">
```

- Add the following inside this [PermissionSet](#):

```
<IPermission class="SocketPermission" version="1" Unrestricted="true" />
```

This configuration lets you use the driver with the default Windows protocol TCP/IP without having any security issues. This approach only supports the TCP/IP protocol, so you cannot use any other type of connection.

Also, since the [MySQLClientPermissions](#) class is not added to the medium trust policy, you cannot use it. This configuration is the minimum required in order to work with Connector/Net without the GAC.

4.6 Connector/Net Connection String Options Reference

For usage information about connection strings, see [Section 4.5.2, “Creating a Connector/Net Connection String”](#). The first table lists options that apply generally to all server configurations. The options related to systems using a connection pool are split into a separate table.

General Options

Table 4.2 Connector/Net Connection String Options - General

Name	Default	Description
Allow Batch , AllowBatch	true	When true, multiple SQL statements can be sent with one command execution. Note: starting with MySQL 4.1.1, batch statements should be separated by the server-defined separator character. Statements sent to earlier versions of MySQL should be separated by ';'.
Allow User Variables , AllowUserVariables	false	Setting this to true indicates that the provider expects user variables in the SQL. This option was added in Connector/Net version 5.2.2.
Allow Zero Datetime , AllowZeroDateTime	false	If set to True , MySqlDataReader.GetValue() returns a MySqlDateTime object for date or datetime columns that have disallowed values, such as zero datetime values, and a System.DateTime object for valid values. If set to False (the default setting) it causes a System.DateTime object to be returned for all valid values and an exception to be thrown for disallowed values, such as zero datetime values.
Auto Enlist , AutoEnlist	true	If AutoEnlist is set to true , which is the default, a connection opened using TransactionScope participates in this scope, it commits when the scope commits and rolls back if TransactionScope does not commit. However, this feature is considered security sensitive and therefore cannot be used in a medium trust environment.
BlobAsUTF8ExcludePattern	null	A POSIX-style regular expression that matches the names of BLOB columns that do not contain UTF-8 character data. See Section 4.5.18, “Character Set Considerations for Connector/Net” for usage details.
BlobAsUTF8IncludePattern	null	A POSIX-style regular expression that matches the names of BLOB columns containing UTF-8 character data. See Section 4.5.18, “Character Set Considerations for Connector/Net” for usage details.
Certificate File , CertificateFile	null	This option specifies the path to a certificate file in PKCS #12 format (.pfx). For an example of usage, see Section 4.4.8, “Tutorial: Using SSL with MySQL Connector/Net” . Was introduced with 6.2.1.
Certificate Password , CertificatePassword	null	Specifies a password that is used in conjunction with a certificate specified using the option CertificateFile . For an example of usage, see Section 4.4.8, “Tutorial: Using SSL with MySQL Connector/Net” . Was introduced with 6.2.1.
Certificate Store Location , CertificateStoreLocation	null	Enables you to access a certificate held in a personal store, rather than use a certificate file and password combination. For an example of usage, see Section 4.4.8, “Tutorial: Using SSL with MySQL Connector/Net” . Was introduced with 6.2.1.

Name	Default	Description
Certificate Thumbprint , CertificateThumbprint	null	Specifies a certificate thumbprint to ensure correct identification of a certificate contained within a personal store. For an example of usage, see Section 4.4.8 , “ Tutorial: Using SSL with MySQL Connector/Net ”. Was introduced with 6.2.1.
CharSet , Character Set , CharacterSet		Specifies the character set that should be used to encode all queries sent to the server. Resultsets are still returned in the character set of the result data.
Check Parameters , CheckParameters	true	Indicates if stored routine parameters should be checked against the server.
Command Interceptors , CommandInterceptors		The list of interceptors that can intercept SQL command operations.
Connect Timeout , Connection Timeout , ConnectionTimeout	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
Connect_Attrs		Passes a predefined set of key/value pairs containing connection-specific data items to the server, for application-defined purposes. connector/Net automatically transmits the following attributes: _pid , _client_version , _platform , _program_name , _os , _os_details , and _thread . See Performance Schema Connection Attribute Tables for the Performance Schema tables to query on the server side to make use of these attributes.
Convert Zero Datetime , ConvertZeroDateTime	false	True to have MySqlDataReader.GetValue() and MySqlDataReader.GetDateTime() return DateTime.MinValue for date or datetime columns that have disallowed values.
Default Command Timeout , DefaultCommandTimeout	30	Sets the default value of the command timeout to be used. This does not supersede the individual command timeout property on an individual command object. If you set the command timeout property, that will be used. This option was added in Connector/Net 5.1.4
Default Table Cache Age , DefaultTableCacheAge	60	Specifies how long a TableDirect result should be cached, in seconds. For usage information about table caching, see Section 4.5.8 , “ Using Connector/Net with Table Caching ”. This option was added in Connector/Net 6.4.
enableSessionExpireCallback	false	When set to true , causes the session-expiry scanner to raise the session_end event before deleting the session data stored in the my_aspnet_sessions table, when a session times out. Enable this option to write additional application-specific cleanup code to handle the session_end event of the global.asax class, before the stored data of the session gets deleted. Within the session_end method, any other required cleanup can be done. This option was added in Connector/Net 6.4.5; it is not yet available in Connector/Net 6.5.x releases.
Encrypt , UseSSL	false	For Connector/Net 5.0.3 and later, when true , SSL encryption is used for all data sent between the client and server if the server has a certificate installed.

Name	Default	Description
		Recognized values are true , false , yes , and no . In versions before 5.0.3, this option had no effect. From version 6.2.1, this option is deprecated and is replaced by SSL Mode . The option still works if used. If this option is set to true, it is equivalent to SSL Mode = Preferred .
Exception Interceptors , ExceptionInterceptors		The list of interceptors that can triage thrown MySQLException exceptions.
Functions Return String , FunctionsReturnString	false	Causes the connector to return binary/varbinary values as strings, if they do not have a tablename in the metadata.
Host , Server , Data Source , DataSource , Address , Addr , Network Address	localhost	The name or network address of the instance of MySQL to which to connect. Multiple hosts can be specified separated by commas. This can be useful where multiple MySQL servers are configured for replication and you are not concerned about the precise server you are connecting to. No attempt is made by the provider to synchronize writes to the database, so take care when using this option. In Unix environment with Mono, this can be a fully qualified path to a MySQL socket file. With this configuration, the Unix socket is used instead of the TCP/IP socket. Currently, only a single socket name can be given, so accessing MySQL in a replicated environment using Unix sockets is not currently supported.
Ignore Prepare , IgnorePrepare	true	When true, instructs the provider to ignore any calls to MySQLCommand.Prepare() . This option is provided to prevent issues with corruption of the statements when used with server-side prepared statements. If you use server-side prepare statements, set this option to false. This option was added in Connector/Net 5.0.3 and Connector/Net 1.0.9.
includesecurityasserts , include security asserts	false	Must be set to true when using the MySQLClientPermissions class in a partial trust environment, with the library installed in the GAC of the hosting environment. This requirement is new for partial-trust applications in Connector/Net 6.6.4 and higher. See Section 4.5.21, "Working with Partial Trust / Medium Trust" for details.
Initial Catalog , Database	mysql	The case-sensitive name of the database to use initially.
Interactive , Interactive Session , InteractiveSession	false	If set to true, the client is interactive. An interactive client is one where the server variable CLIENT_INTERACTIVE is set. If an interactive client is set, the wait_timeout variable is set to the value of interactive_timeout . The client will then time out after this period of inactivity. For more details, see Server System Variables in the MySQL Reference Manual.
Integrated Security , IntegratedSecurity	no	Use Windows authentication when connecting to server. By default, it is turned off. To enable, specify a value of yes . (You can also use the value sspi as an alternative to yes .) For details, see Section 4.5.5 ,

Name	Default	Description
		"Using the Windows Native Authentication Plugin" . This option was introduced in Connector/Net 6.4.4.
Keep Alive , Keepalive	0	For TCP connections, idle connection time measured in seconds, before the first keepalive packet is sent. A value of 0 indicates that keepalive is not used. Before Connector/Net 6.6.7/6.7.5/6.8.4, this value was measured in milliseconds.
Logging	false	When true, various pieces of information is output to any configured TraceListeners. See Section 4.5.16 , "Using the MySQL Connector/Net Trace Source Object" for further details.
Old Guids , OldGuids	false	This option was introduced in Connector/Net 6.1.1. The backend representation of a GUID type was changed from BINARY(16) to CHAR(36) . This was done to allow developers to use the server function UUID() to populate a GUID table - UUID() generates a 36-character string. Developers of older applications can add ' Old Guids=true ' to the connection string to use a GUID of data type BINARY(16) .
Old Syntax , OldSyntax , Use Old Syntax , UseOldSyntax	false	This option was deprecated in Connector/Net 5.2.2. All code should now be written using the '@' symbol as the parameter marker.
Password , pwd		The password for the MySQL account being used.
Persist Security Info , PersistSecurityInfo	false	When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values, including the password. Recognized values are true , false , yes , and no .
Pipe Name , Pipe , PipeName	mysql	When set to the name of a named pipe, the MySQLConnection attempts to connect to MySQL on that named pipe. This setting only applies to the Windows platform.
Port	3306	The port MySQL is using to listen for connections. This value is ignored if Unix socket is used.
Procedure Cache Size , ProcedureCacheSize , procedure cache , procedurecache	25	Sets the size of the stored procedure cache. By default, Connector/Net stores the metadata (input/output data types) about the last 25 stored procedures used. To disable the stored procedure cache, set the value to zero (0). This option was added in Connector/Net 5.0.2 and Connector/Net 1.0.9.
Protocol , Connection Protocol , ConnectionProtocol	socket	Specifies the type of connection to make to the server. Values can be: socket or tcp for a socket connection, pipe for a named pipe connection, unix for a Unix socket connection, memory to use MySQL shared memory.
Replication	false	Indicates if this connection is to use replicated servers.
Respect Binary Flags , RespectBinaryFlags	true	Setting this option to false means that Connector/Net ignores a column's binary flags as set by the server. This option was added in Connector/Net version 5.1.3.

Name	Default	Description
Shared Memory Name, SharedMemoryName	MYSQL	The name of the shared memory object to use for communication if the connection protocol is set to memory .
Sql Server Mode, sqlservermode	false	Allow SQL Server syntax. When set to true , enables Connector/Net to support square brackets around symbols instead of backticks. This enables Visual Studio wizards that bracket symbols with [] to work with Connector/Net. This option incurs a performance hit, so should only be used if necessary. This option was added in version 6.3.1.
SSL Mode, SslMode	None	This option has the following values: • None - do not use SSL. • Preferred - use SSL if the server supports it, but allow connection in all cases. • Required - Always use SSL. Deny connection if server does not support SSL. • VerifyCA - Always use SSL. Validate the CA but tolerate name mismatch. • VerifyFull - Always use SSL. Fail if the host name is not correct. This option was introduced in MySQL Connector/Net 6.2.1.
Table Cache, tablecache, tablecaching	false	Enables or disables caching of TableDirect commands. A value of true enables the cache while false disables it. For usage information about table caching, see Section 4.5.8, "Using Connector/Net with Table Caching" . This option was added in Connector/Net 6.4.
Treat BLOBs as UTF8, TreatBlobsAsUTF8	false	
Treat Tiny As Boolean, TreatTinyAsBoolean	true	Setting this value to false causes TINYINT(1) to be treated as an INT . See Numeric Type Overview for a further explanation of the TINYINT and BOOL data types.
Use Affected Rows, UseAffectedRows	false	When true , the connection reports changed rows instead of found rows. This option was added in Connector/Net version 5.2.6.
Use Procedure Bodies, UseProcedureBodies, procedure bodies	true	When set to true , the default value, MySQL Connector/Net expects the body of the procedure to be viewable. This enables it to determine the parameter types and order. Set the option to false when the user connecting to the database does not have the SELECT privileges for the mysql.proc (stored procedures) table, or cannot view INFORMATION_SCHEMA.ROUTINES . In this case, MySQL Connector/Net cannot determine the types and order of the parameters, and must be alerted to this fact by setting this option to false . When set to false ,

Name	Default	Description
		MySQL Connector/Net does not rely on this information being available when the procedure is called. Because MySQL Connector/Net will not be able to determine this information, explicitly set the types of all the parameters before the call and add the parameters to the command in the same order as they appear in the procedure definition. This option was added in MySQL Connector/Net 5.0.4 and MySQL Connector/Net 1.0.10.
User Id, UserID, Username, Uid, User name, User		The MySQL login account being used.
Compress, Use Compression, UseCompression	false	Setting this option to <code>true</code> enables compression of packets exchanged between the client and the server. This exchange is defined by the MySQL client/server protocol. Compression is used if both client and server support ZLIB compression, and the client has requested compression using this option. A compressed packet header is: packet length (3 bytes), packet number (1 byte), and Uncompressed Packet Length (3 bytes). The Uncompressed Packet Length is the number of bytes in the original, uncompressed packet. If this is zero, the data in this packet has not been compressed. When the compression protocol is in use, either the client or the server may compress packets. However, compression will not occur if the compressed length is greater than the original length. Thus, some packets will contain compressed data while other packets will not.
Use Usage Advisor, Usage Advisor, UseUsageAdvisor	false	Logs inefficient database operations.
Use Performance Monitor, UsePerformanceMonitor, userperfmon, perfmon	false	Indicates that performance counters should be updated during execution.

Connection Pooling Options

The following table lists the valid names for options related to connection pooling within the `ConnectionString`. For more information about connection pooling, see [Section 4.5.4, "Using Connector/Net with Connection Pooling"](#).

Table 4.3 Connector/Net Connection String Options - Connection Pooling

Name	Default	Description
Cache Server Properties, CacheServerProperties	false	Specifies whether server variable settings are updated by a <code>SHOW VARIABLES</code> command each time a pooled connection is returned. Enabling this setting speeds up connections in a connection pool environment. Your application is not informed of any changes to configuration variables made by other connections. This option was added in Connector/Net 6.3.
Connection Lifetime, ConnectionLifeTime	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in

Name	Default	Description
		seconds) exceeds the value specified by Connection Lifetime . This is useful in clustered configurations to force load balancing between a running server and a server just brought online. A value of zero (0) causes pooled connections to have the maximum connection timeout.
Connection Reset , ConnectionReset	false	If true, the connection state is reset when it is retrieved from the pool. The default value of false avoids making an additional server round trip when obtaining a connection, but the connection state is not reset.
Maximum Pool Size , Max Pool Size , MaximumPoolsize , maxpoolsize (only for Connector/Net 6.7 and later)	100	The maximum number of connections allowed in the pool.
Minimum Pool Size , Min Pool Size , MinimumPoolSize , minpoolsize (only for Connector/Net 6.7 and later)	0	The minimum number of connections allowed in the pool.
Pooling	true	When true , the MySQLConnection object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are true , false , yes , and no .

4.7 Connector/Net Support for Windows Store

Starting with 6.7, Connector/Net fully supports building Windows Store apps. Windows Store apps are based on .NET but use a very restrictive subset of that functionality. The main difference is the complete lack of the ADO.Net data subsystem. Below is a list of the differences between Connector/Net 6.7 and Connector/Net RT 6.7. Please note that we are looking at ways of bringing these functionalities back in future versions.

- No support for [MySQLDataAdapter](#). [MySQLCommand](#) and [MySQLDataReader](#) are supported.
- Connector/Net RT does not support SSL connections or Windows authentication. Also, SHA256 is not currently supported.
- Connector/Net RT only supports TCP connections. Named pipe and shared memory connections are not supported.
- Connector/Net RT doesn't support tracing.

This version of Connector/Net is no longer supported.

- Connector/Net RT doesn't support load balancing. Command and Exception interceptors are supported however.

This version of Connector/Net is no longer supported.

- [MySQLConnection.GetSchema](#) methods do not return [DataTable](#) types. Instead they return a new object called [MySQLSchemaCollection](#). You can query this object for the schema information. Normal Connector/Net will include support for returning schema information in both [DataTable](#) and [MySQLSchemaCollection](#) format.

This version of Connector/Net is no longer supported.

- Some constructors and other APIs on supported classes may have been removed or altered. Any API that used [DataTable](#), [DataSet](#), or [DataRow](#) has been altered or removed.

This version of Connector/Net is no longer supported.

Using Connector/Net RT is easy. Simply create a Windows Store application using Visual Studio and then reference the `MySQL.Data.RT.dll` assembly in your project. The code you write should be exactly the same as for normal Connector/Net (including using same namespace `MySQL.Data.MySqlClient`) except for the differences listed above.

4.8 EF5 Support

Support for Entity Framework 5 (EF5) is based on the introduction of .NET Framework 4.5, which is the default target framework in Microsoft Visual Studio 2012, or higher. Using this framework version is a requirement for applying any of the features in EF5 to your code.

This chapter provides a detailed description of each one of the features.

EF5 Features

Spatial Data Type (Geometry type support). Spatial data types support is backed up by the server capabilities which are documented at: [Extensions for Spatial Data](#) There are different types for spatial data, and the instantiable types that are supported at MySQL are:

Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiLineString, and MultiPolygon.

In MySQL Connector/Net, these different types can be managed with the new Geometry type. Before Connector/Net 6.7 the Geometry types did not have a MySQL type inside the driver. Common practice was to use a binary type; however, this is no longer needed. Now, all the specific operations for any Geometry type can be performed by using this new class.

An example of how to write a point data is shown here using Connector/Net 6.7 and MySQL Server 5.6.7 or higher.

```
//Storing a geometry point
MySQLConnection conn = new MySQLConnection("server=localhost;userid=root;database=testgeo;");
conn.Open();
MySQLCommand cmd = new MySQLCommand("CREATE TABLE Test (v Geometry NOT NULL)");
cmd.Connection = conn;
cmd.ExecuteNonQuery();
cmd = new MySQLCommand("INSERT INTO Test VALUES(GeomFromText(?v))", conn);
cmd.Parameters.Add("?v", MySqlDbType.String);
cmd.Parameters[0].Value = "POINT(47.37-122.21)";
cmd.ExecuteNonQuery();
cmd.CommandText = "SELECT AsText(v) FROM Test";
using(MySqlDataReader reader = cmd.ExecuteReader())
{
    reader.Read();
    varval = reader.GetValue(0);
    Console.WriteLine(val);
}
conn.Close();
```

Output:

POINT(47.37, -122.21)

To read a set of rows that contains a Geometry column can be achieved by using an appropriate `MySqlDataReader`. Example:

```
cmd.CommandText = "SELECT v FROM Test";
using(MySqlDataReader reader = cmd.ExecuteReader())
```

```

{
    reader.Read();
    var val = reader.GetMySQLGeometry(0);
    var valWithName = reader.GetMySQLGeometry("v");
    Console.WriteLine(val.ToString());
    // output : ("POINT(47.37 -122.21)"
    Console.WriteLine(valWithName.ToString());
    // output("POINT(47.37 -122.21)"
}

```

A geometry point also can contain a Spatial Reference System Identifier (SRID) value. This value can also be stored using a geometry type.

Example:

```

    MySQLGeometry v = new MySQLGeometry(47.37, -122.21, 101);
    var par = new MySQLParameter("?v", MySQLDbType.Geometry);
    par.Value = v;
    MySqlCommand cmd = new MySqlCommand("INSERT INTO Test VALUES(?v)", conn);
    cmd.Parameters.Add(par);
    cmd.ExecuteNonQuery();
    cmd.CommandText = "SELECT SRID(v) FROM Test";
    using (MySQLDataReader reader = cmd.ExecuteReader())
    {
        reader.Read(); var val = reader.GetString(0);
        Console.WriteLine(val); // output "101"
    }

```

Other types of values besides point can be stored at the Geometry type:

- LineString
- MultiLineString
- Polygon
- MultiPolygon
- GeometryCollection

Spatial data is supported in Model, Database and Code First.

Using spatial types with an Entity Framework Model is one of the new features of Connector/Net 6.7.

Spatial types can be used with any of the strategies used to create the data layer of any application: Database First, Code First or Model First. Entity Framework support two main types for spatial data: DbGeometry and DBGeography. The second one is not supported by Connector/Net, because MySQL server does not have any equivalent type to map to this type. So, all the examples will use the DbGeometry type.

Example of usage in Code First

An entity that contains a Geometry column can be defined as follows:

```

public class Distributor { public int DistributorId { get; set; }
    public string Name { get; set; } public DbGeometry point { get; set; }
}
public class DistributorsContext : DbContext { public
    DbSet<Distributor> Distributors { get; set; } }

```

Creating the db:

```
using (DistributorsContext context = new DistributorsContext()) {
    context.Database.Delete(); context.Database.Create();
    context.Distributors.Add(new Distributor() { Name = "Graphic
Design Institute", point =
    DbGeometry.FromText("POINT(-122.336106 47.605049)", 101),
}); context.SaveChanges();
var result = (from u in context.Distributors select
u.point.CoordinateSystemId).ToList();
foreach (var item in result)
    Console.WriteLine("CoordinateSystemId " + item);
}
```

Output:

CoordinateSystemId 101

Spatial-Supported Functions

There are some useful functions that takes Geometry values:

- SpatialDimension
- SpatialEnvelope
- IsSimpleGeometry
- SpatialTypeName
- CoordinateSystemId
- Point
- XCoordinate
- YCoordinate
- GeometryFromText
- SpatialContains
- AsText
- SpatialBuffer
- SpatialDifference
- SpatialIntersection

Enumeration Support

- Enumeration types are used to define a set of named constants that may be assigned to a numeric value.
- By default the underlying type of each element in the enum is int. But it can be specified another numeric type by using a colon. Example: `enum Months : byte { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };`
- When a value is not specified in the defined list, the values are automatically incremented by 1. In the previous example `Months.Jan` has a default value of 0.

In order to use enum types inside an Entity Framework defined class it should follow the next steps:

In Visual Studio 2012

Using Database First approach

- Create a new project (make sure .NET Framework is target in the New Project dialog.).
- Add a new Entity Model.
- Select the Existing Database option.
- Select the tables to be imported in the model.
- In order to create an Enum type an integer column should exists inside of a table.
- In the Model designer select the table to use.
- Right click on the column of integer type and select Create enum type.
- At this point all linq to entities query can use the Enum type directly.

Example:

```
using (var db = new DistributorsEntities()) {
    db.Database.CreateIfNotExists();
    db.distributors.Add(new distributor { Type = DistributorType.Regional });
    db.distributors.Add(new distributor { Type = DistributorType.Reseller });
    db.distributors.Add(new distributor { Type = DistributorType.Zone });
    db.SaveChanges();
    var testQ = (from d in db.distributors select d).FirstOrDefault();
    foreach (var item in testQ)
        Console.WriteLine(item.Type);
}
```

Output:

Regional Reseller Zone

Other important features and improvements included in Entity Framework are:

- All LINQ queries are now automatically compiled and cached to improve query performance.
- Multiple Diagrams for a model is supported.
- Enhancement Table per type in SQL Generation.
- Performance Enhancements.
- The StoreGeneratedPattern for key columns can now be set on an entity Properties window and this value will propagate from the entity model down to the stored definition. The stored Generated Pattern attribute allows you to control how the Entity Framework synchronizes database column values and entity property values.

4.9 EF6 Support

MySQL Connector/Net 6.8 integrates support for Entity Framework 6.0 (EF6), but also offers support for Entity Framework 5 (EF 5). This chapter explains the new features in Entity Framework 6 implemented in MySQL Connector/Net 6.8.

Requirements for Entity Framework 6.0 Support

- MySQL Connector/Net 6.8.x
- MySQL Server 5.1 or higher

- Entity Framework 6 assemblies
- .NET Framework 4.0 or higher

Configuration

Configure Connector/Net to support EF6 by the following steps:

1. An important first step is editing the configuration sections in the `App.config` file to add the connection string and the MySQL Connector/Net provider for EF6:

```
<connectionStrings>
  <add name="MyContext" providerName="MySQL.Data.MySqlClient"
    connectionString="server=localhost;port=3306;database=mycontext;uid=root;password=*****"/>
</connectionStrings>
<entityFramework>
  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />
  <providers>
    <provider invariantName="MySQL.Data.MySqlClient"
      type="MySQL.Data.MySqlClient.MySqlProviderServices, MySQL.Data.Entity.EF6" />
    <provider invariantName="System.Data.SqlClient"
      type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
  </providers>
</entityFramework>
```

2. Add the reference for `MySQL.Data.Entity.EF6` assembly into the project. Depending on the .NET Framework used, the assembly is to be taken from either the `v4.0` or the `v4.5` folder).
3. Set the new `DbConfiguration` class for `MySQL`. This step is optional but highly recommended, since it adds all the dependency resolvers for `MySQL` classes. This can be done in three ways:

- Adding the `DbConfigurationTypeAttribute` on the context class:

```
[DbConfigurationType(typeof(MySqlEFConfiguration))]
```

- Calling `DbConfiguration.SetConfiguration(new MySqlEFConfiguration())` at the application startup.
- Set the `DbConfiguration` type in the configuration file:

```
<entityFramework codeConfigurationType="MySQL.Data.Entity.MySqlEFConfiguration, MySQL.Data.Entity.EF6" />
```

It is also possible to create a custom `DbConfiguration` class and add the dependency resolvers needed.

EF6 Features

Following are the new features in Entity Framework 6 implemented in MySQL Connector/Net 6.8:

- *Async Query and Save* adds support for the task-based asynchronous patterns that have been introduced since .NET 4.5. The new asynchronous methods supported by Connector/Net are:
 - `ExecuteNonQueryAsync`
 - `ExecuteScalarAsync`
 - `PrepareAsync`
- *Connection Resiliency / Retry Logic* enables automatic recovery from transient connection failures. To use this feature, add to the `OnCreateModel` method:

```
SetExecutionStrategy(MySqlProviderInvariantName.ProviderName, () => new MySqlExecutionStrategy());
```

- *Code-Based Configuration* gives you the option of performing configuration in code, instead of performing it in a configuration file, as it has been done traditionally.
- *Dependency Resolution* introduces support for the Service Locator. Some pieces of functionality that can be replaced with custom implementations have been factored out. To add a dependency resolver, use:

```
AddDependencyResolver(new MySqlDependencyResolver());
```

The following resolvers can be added:

- *DbProviderFactory* -> *MySqlClientFactory*
- *IDbConnectionFactory* -> *MySqlConnectionFactory*
- *MigrationSqlGenerator* -> *MySqlMigrationSqlGenerator*
- *DbProviderServices* -> *MySqlProviderServices*
- *IProviderInvariantName* -> *MySqlProviderInvariantName*
- *IDbProviderFactoryResolver* -> *MySqlProviderFactoryResolver*
- *IManifestTokenResolver* -> *MySqlManifestTokenResolver*
- *IDbModelCacheKey* -> *MySqlModelCacheKeyFactory*
- *IDbExecutionStrategy* -> *MySqlExecutionStrategy*
- *Interception/SQL logging* provides low-level building blocks for interception of Entity Framework operations with simple SQL logging built on top:

```
myContext.Database.Log = delegate(string message) { Console.Write(message); };
```

- *DbContext* can now be created with a *DbConnection* that is already opened, which enables scenarios where it would be helpful if the connection could be open when creating the context (such as sharing a connection between components when you cannot guarantee the state of the connection)

```
[DbConfigurationType(typeof(MySqlEFConfiguration))]
class JourneyContext : DbContext
{
    public DbSet<MyPlace> MyPlaces { get; set; }

    public JourneyContext()
        : base()
    {
    }

    public JourneyContext(DbConnection existingConnection, bool contextOwnsConnection)
        : base(existingConnection, contextOwnsConnection)
    {
    }
}
using (MySqlConnection conn = new MySqlConnection("<connectionString>"))
{
    conn.Open();
    ...
    using (var context = new JourneyContext(conn, false))
    {
        ...
    }
}
```

```
}
```

- *Improved Transaction Support* provides support for a transaction external to the framework as well as improved ways of creating a transaction within the Entity Framework. Starting with Entity Framework 6, `Database.ExecuteSqlCommand()` will wrap by default the command in a transaction if one was not already present. There are overloads of this method that allow users to override this behavior if wished. Execution of stored procedures included in the model through APIs such as `ObjectContext.ExecuteFunction()` does the same. It is also possible to pass an existing transaction to the context.
- *DbSet.AddRange/RemoveRange* provides an optimized way to add or remove multiple entities from a set.

Code First Features

Following are new Code First features supported by Connector/Net:

- *Code First Mapping to Insert/Update/Delete Stored Procedures* supported:

```
modelBuilder.Entity<EntityType>().MapToStoredProcedures();
```

- *Idempotent migrations scripts* allow you to generate an SQL script that can upgrade a database at any version up to the latest version. To do so, run the *Update-Database -Script -SourceMigration: \$InitialDatabase* command in Package Manager Console.
- *Configurable Migrations History Table* allows you to customize the definition of the migrations history table.

Example for Using EF6

Model:

```
using MySql.Data.Entity;
using System.Data.Common;
using System.Data.Entity;

namespace EF6
{
    // Code-Based Configuration and Dependency resolution
    [DbConfigurationType(typeof(MySqlEFConfiguration))]
    public class Parking : DbContext
    {
        public DbSet<Car> Cars { get; set; }

        public Parking()
            : base()
        {
        }

        // Constructor to use on a DbConnection that is already opened
        public Parking(DbConnection existingConnection, bool contextOwnsConnection)
            : base(existingConnection, contextOwnsConnection)
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Car>().MapToStoredProcedures();
        }
    }

    public class Car
```



```
{
    public int CarId { get; set; }

    public string Model { get; set; }

    public int Year { get; set; }

    public string Manufacturer { get; set; }
}
```

Program:

```
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;

namespace EF6
{
    class Example
    {
        public static void ExecuteExample()
        {
            string connectionString = "server=localhost;port=3305;database=parking;uid=root;";

            using (MySqlConnection connection = new MySqlConnection(connectionString))
            {
                // Create database if not exists
                using (Parking contextDB = new Parking(connection, false))
                {
                    contextDB.Database.CreateIfNotExists();
                }

                connection.Open();
                MySqlTransaction transaction = connection.BeginTransaction();

                try
                {
                    // DbConnection that is already opened
                    using (Parking context = new Parking(connection, false))
                    {
                        // Interception/SQL logging
                        context.Database.Log = (string message) => { Console.WriteLine(message); };

                        // Passing an existing transaction to the context
                        context.Database.UseTransaction(transaction);

                        // DbSet.AddRange
                        List<Car> cars = new List<Car>();

                        cars.Add(new Car { Manufacturer = "Nissan", Model = "370Z", Year = 2012 });
                        cars.Add(new Car { Manufacturer = "Ford", Model = "Mustang", Year = 2013 });
                        cars.Add(new Car { Manufacturer = "Chevrolet", Model = "Camaro", Year = 2012 });
                        cars.Add(new Car { Manufacturer = "Dodge", Model = "Charger", Year = 2013 });

                        context.Cars.AddRange(cars);

                        context.SaveChanges();
                    }

                    transaction.Commit();
                }
                catch
                {
                    transaction.Rollback();
                    throw;
                }
            }
        }
    }
}
```

4.10 Connector/Net API Reference

This section of the manual contains a complete reference to the Connector/Net ADO.NET component, automatically generated from the embedded documentation.

4.10.1 MySql.Data.MySqlClient Namespace

[Namespace hierarchy](#)

Classes

Class	Description
BaseCommandInterceptor	Provides a means of enhancing or replacing SQL commands through the connection string rather than recompiling.
BaseExceptionInterceptor	Provides a means of enabling and disabling exception handling through the connection string rather than recompiling.
MySqlCommandPermission	Derived from the .NET DBDataPermission class. For usage information, see Section 4.5.21 , "Working with Partial Trust / Medium Trust".
MySqlCommand	
MySqlCommandBuilder	
MySqlConnection	
MySqlDataAdapter	
MySqlDataReader	Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.
MySqlErrorCode	Collection of error codes that can be returned by the server
MySqlException	The exception that is thrown when MySQL returns an error. This class cannot be inherited.
MySqlHelper	Helper class that makes it easier to work with the provider.
MySqlInfoMessageEventArgs	Provides data for the InfoMessage event. This class cannot be inherited.
MySqlParameter	Represents a parameter to a MySqlCommand , and optionally, its mapping to DataSet columns. This class cannot be inherited.
MySqlParameterCollection	Represents a collection of parameters relevant to a MySqlCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.
MySqlRowUpdatedEventArgs	Provides data for the RowUpdated event. This class cannot be inherited.
MySqlRowUpdatingEventArgs	Provides data for the RowUpdating event. This class cannot be inherited.
MySqlTransaction	

Delegates

Delegate	Description
MySqlInfoMessageEventHandler	Represents the method that will handle the InfoMessage event of a MySqlConnection .
MySqlRowUpdatedEventHandler	Represents the method that will handle the RowUpdatedevent of a MySqlDataAdapter .
MySqlRowUpdatingEventHandler	Represents the method that will handle the RowUpdatingevent of a MySqlDataAdapter .

Enumerations

Enumeration	Description
MySqlDbType	Specifies MySQL-specific data type of a field, property, for use in a MySqlParameter .
MySqlErrorCode	

4.10.1.1 MySql.Data.MySqlClientHierarchy

See Also

[MySql.Data.MySqlClient Namespace](#)

4.10.1.2 BaseCommandInterceptor

The [BaseCommandInterceptor](#) class has these methods that you can override:

```
public virtual bool ExecuteScalar(string sql, ref object returnValue);
public virtual bool ExecuteNonQuery(string sql, ref int returnValue);
public virtual bool ExecuteReader(string sql, CommandBehavior behavior, ref MySqlDataReader returnValue);
public virtual void Init(MySqlConnection connection);
```

If your interceptor overrides one of the [Execute...](#) methods, set the [returnValue](#) output parameter and return [true](#) if you handled the event, or [false](#) if you did not handle the event. The SQL command is processed normally only when all command interceptors return [false](#).

The connection passed to the [Init](#) method is the connection that is attached to this interceptor.

For full usage and examples, see [Section 4.5.13, "Using the Connector/Net Interceptor Classes"](#).

4.10.1.3 BaseExceptionInterceptor

You develop an exception interceptor first by creating a subclass of the [BaseExceptionInterceptor](#) class. You must override the [InterceptException\(\)](#) method. You can also override the [Init\(\)](#) method to do some one-time initialization.

Each exception interceptor has 2 methods:

```
public abstract Exception InterceptException(Exception exception,
    MySqlConnection connection);
public virtual void Init(MySqlConnection connection);
```

The connection passed to [Init\(\)](#) is the connection that is attached to this interceptor.

Each interceptor is required to override [InterceptException](#) and return an exception. It can return the exception it is given, or it can wrap it in a new exception. We currently do not offer the ability to suppress the exception.

For full usage and examples, see [Section 4.5.13, "Using the Connector/Net Interceptor Classes"](#).

4.10.1.4 MySqlCommand Class

For a list of all members of this type, see [MySqlCommand Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommand_
    Inherits Component_
    Implements IDbCommand, ICloneable
```

Syntax: C#

```
public sealed class MySqlCommand : Component, IDbCommand, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlCommand Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand Members

[MySqlCommand overview](#)

Public Instance Constructors

MySqlCommand	Overloaded. Initializes a new instance of the MySqlCommand class.

Public Instance Properties

CommandText	
CommandTimeout	
CommandType	
Connection	
Container (inherited from Component)	Gets the IContainer that contains the Component.
IsPrepared	
Parameters	
Site (inherited from Component)	Gets or sets the ISite of the Component.
Transaction	
UpdatedRowSource	

Public Instance Methods

Cancel	Attempts to cancel the execution of a MySqlCommand. This operation is not supported.

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
CreateParameter	Creates a new instance of a MySqlParameter object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExecuteNonQuery	
ExecuteReader	Overloaded.
ExecuteScalar	
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Prepare	
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand Constructor

Initializes a new instance of the [MySqlCommand](#) class.

Overload List

Initializes a new instance of the [MySqlCommand](#) class.

- [public MySqlCommand\(\);](#)
- [public MySqlCommand\(string\);](#)
- [public MySqlCommand\(string, MySqlConnection\);](#)
- [public MySqlCommand\(string, MySqlConnection, MySqlTransaction\);](#)

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand Constructor ()

Initializes a new instance of the [MySqlCommand](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommand();
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand Constructor Overload List](#)

MySqlCommand Constructor (String)

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String _  
)
```

Syntax: C#

```
public MySqlCommand(  
    stringcmdText  
);
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand Constructor Overload List](#)

MySqlCommand Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection _  
)
```

Syntax: C#

```
public MySqlCommand(  
    stringcmdText,  
    MySqlConnectionconnection  
);
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand Constructor Overload List](#)

MySqlConnection Class

For a list of all members of this type, see [MySqlConnection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_  
    Inherits Component_
```

Implements [IDbConnection](#), [ICloneable](#)

Syntax: C#

```
public sealed class MySqlConnection : Component, IDbConnection, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlConnection Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlConnection Members

[MySqlConnection overview](#)

Public Instance Constructors

MySqlConnection	Overloaded. Initializes a new instance of the MySqlConnection class.

Public Instance Properties

ConnectionString	
ConnectionTimeout	
Container (inherited from Component)	Gets the IContainer that contains the Component.
Database	
DataSource	Gets the name of the MySQL server to which to connect.
ServerThread	Returns the ID of the server thread this connection is executing on.
ServerVersion	
Site (inherited from Component)	Gets or sets the ISite of the Component.
State	
UseCompression	Indicates if this connection should use compression when communicating with the server.

Public Instance Methods

BeginTransaction	Overloaded.
ChangeDatabase	
Close	
CreateCommand	
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.

Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Open	
Ping	Ping
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
InfoMessage	
StateChange	

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlConnection Constructor

Initializes a new instance of the [MySqlConnection](#) class.

Overload List

Initializes a new instance of the [MySqlConnection](#) class.

- [public MySqlConnection\(\);](#)
- [public MySqlConnection\(string\);](#)

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlConnection Constructor

Initializes a new instance of the [MySqlConnection](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlConnection();
```


See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLConnection Constructor Overload List](#)

MySQLConnection Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal connectionString As String _
)
```

Syntax: C#

```
public MySqlConnection(
    string connectionString
);
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLConnection Constructor Overload List](#)

ConnectionString Property

Syntax: Visual Basic

```
NotOverridable Public Property ConnectionString As String _
- Implements IDbConnection.ConnectionString
```

Syntax: C#

```
public string ConnectionString {get; set;}
```

Implements

IDbConnection.ConnectionString

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

ConnectionTimeout Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property ConnectionTimeout As Integer _
- Implements IDbConnection.ConnectionTimeout
```

Syntax: C#

```
public int ConnectionTimeout {get;}
```

Implements

IDbConnection.ConnectionTimeout

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

Database Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Database As String _  
    Implements IDbConnection.Database
```

Syntax: C#

```
public string Database {get;}
```

Implements

IDbConnection.Database

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

DataSource Property

Gets the name of the MySQL server to which to connect.

Syntax: Visual Basic

```
Public ReadOnly Property DataSource As String
```

Syntax: C#

```
public string DataSource {get;}
```

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

ServerThread Property

Returns the ID of the server thread this connection is executing on

Syntax: Visual Basic

```
Public ReadOnly Property ServerThread As Integer
```

Syntax: C#

```
public int ServerThread {get;}
```

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

ServerVersion Property

Syntax: Visual Basic

```
Public ReadOnly Property ServerVersion As String
```

Syntax: C#

```
public string ServerVersion {get;}
```

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

State Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property State As ConnectionState _  
    Implements IDbConnection.State
```

Syntax: C#

```
public System.Data.ConnectionState State {get;}
```

Implements

IDbConnection.State

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

UseCompression Property

Indicates if this connection should use compression when communicating with the server.

Syntax: Visual Basic

```
Public ReadOnly Property UseCompression As Boolean
```

Syntax: C#

```
public bool UseCompression {get;}
```

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

BeginTransaction Method

Overload List

- [public MySqlTransaction BeginTransaction\(\);](#)
- [public MySqlTransaction BeginTransaction\(IsolationLevel\);](#)

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlConnection.BeginTransaction Method

Syntax: Visual Basic

Overloads Public Function BeginTransaction() As MySqlTransaction

Syntax: C#

```
public MySqlTransaction BeginTransaction();
```

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlConnection.BeginTransaction Overload List](#)

MySqlTransaction Class

For a list of all members of this type, see [MySqlTransaction Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlTransaction_  
    Implements IDbTransaction, IDisposable
```

Syntax: C#

```
public sealed class MySqlTransaction : IDbTransaction, IDisposable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlTransaction Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlTransaction Members

[MySqlTransaction overview](#)

Public Instance Properties

Connection	Gets the MySqlConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
IsolationLevel	Specifies the IsolationLevel for this transaction.

Public Instance Methods

Commit	
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.

GetType (inherited from Object)	Gets the Type of the current instance.
Rollback	
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

Connection Property

Gets the [MySqlConnection](#) object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Syntax: Visual Basic

```
Public ReadOnly Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get;}
```

Property Value

The [MySqlConnection](#) object associated with this transaction.

Remarks

A single application may have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by [BeginTransaction](#).

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

IsolationLevel Property

Specifies the IsolationLevel for this transaction.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsolationLevel As IsolationLevel _  
    Implements IDbTransaction.IsolationLevel
```

Syntax: C#

```
public System.Data.IsolationLevel IsolationLevel {get;}
```

Property Value

The IsolationLevel for this transaction. The default is ReadCommitted.

Implements

IDbTransaction.IsolationLevel

Remarks

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.

See Also

[MySQLTransaction Class](#), [MySql.Data.MySqlClient Namespace](#)

MySQLTransaction.Commit Method

Syntax: Visual Basic

```
NotOverridable Public Sub Commit() _
    Implements IDbTransaction.Commit
```

Syntax: C#

```
public void Commit();
```

Implements

IDbTransaction.Commit

See Also

[MySQLTransaction Class](#), [MySql.Data.MySqlClient Namespace](#)

MySQLTransaction.Rollback Method

Syntax: Visual Basic

```
NotOverridable Public Sub Rollback() _
    Implements IDbTransaction.Rollback
```

Syntax: C#

```
public void Rollback();
```

Implements

IDbTransaction.Rollback

See Also

[MySQLTransaction Class](#), [MySql.Data.MySqlClient Namespace](#)

MySQLConnection.BeginTransaction Method

Syntax: Visual Basic

```
Overloads Public Function BeginTransaction( _
    ByVal iso As IsolationLevel _
) As MySQLTransaction
```

Syntax: C#

```
public MySQLTransaction BeginTransaction(
    IsolationLevel iso
);
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLConnection.BeginTransaction Overload List](#)

MySQLConnection.ChangeDatabase Method

Syntax: Visual Basic

```
NotOverridable Public Sub ChangeDatabase( _
    ByVal databaseName As String _
) _
    Implements IDbConnection.ChangeDatabase
```

Syntax: C#

```
public void ChangeDatabase(
    string databaseName
);
```

Implements

IDbConnection.ChangeDatabase

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySQLConnection.Close Method

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _
    Implements IDbConnection.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDbConnection.Close

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySQLConnection.CreateCommand Method

Syntax: Visual Basic

```
Public Function CreateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand CreateCommand();
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlConnection.Open Method

Syntax: Visual Basic

```
NotOverridable Public Sub Open() _  
    Implements IDbConnection.Open
```

Syntax: C#

```
public void Open();
```

Implements

IDbConnection.Open

See Also

[MySqlConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlConnection.Ping Method

Ping

Syntax: Visual Basic

```
Public Function Ping() As Boolean
```

Syntax: C#

```
public bool Ping();
```

Return Value

See Also

[MySqlConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlConnection.InfoMessage Event

Syntax: Visual Basic

```
Public Event InfoMessage As MySqlInfoMessageEventHandler
```

Syntax: C#

```
public event MySqlInfoMessageEventHandler InfoMessage;
```

See Also

[MySqlConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlInfoMessageEventHandler Delegate

Represents the method that will handle the [InfoMessage](#) event of a [MySqlConnection](#).

Syntax: Visual Basic

```
Public Delegate Sub MySqlInfoMessageEventHandler( _  
    ByVal sender As Object, _
```



```
ByVal args As MySqlInfoMessageEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlInfoMessageEventHandler(
    object sender,
    MySqlInfoMessageEventArgs args
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

MySqlInfoMessageEventArgs Class

Provides data for the InfoMessage event. This class cannot be inherited.

For a list of all members of this type, see [MySqlInfoMessageEventArgs Members](#) .

Syntax: Visual Basic

```
Public Class MySqlInfoMessageEventArgs_
    Inherits EventArgs
```

Syntax: C#

```
public class MySqlInfoMessageEventArgs : EventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlInfoMessageEventArgs Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlInfoMessageEventArgs Members

[MySqlInfoMessageEventArgs overview](#)

Public Instance Constructors

MySqlInfoMessageEventArgs Constructor	Initializes a new instance of the MySqlInfoMessageEventArgs class.

Public Instance Fields

errors	

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlInfoMessageEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlInfoMessageEventArgs Constructor

Initializes a new instance of the [MySqlInfoMessageEventArgs](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlInfoMessageEventArgs();
```

See Also

[MySqlInfoMessageEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlInfoMessageEventArgs.errors Field

Syntax: Visual Basic

```
Public errors As MySqlError()
```

Syntax: C#

```
public MySqlError[] errors;
```

See Also

[MySqlInfoMessageEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlError Class

Collection of error codes that can be returned by the server

For a list of all members of this type, see [MySqlError Members](#) .

Syntax: Visual Basic

```
Public Class MySqlError
```

Syntax: C#

```
public class MySqlError
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlError Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlError Members

[MySqlError overview](#)

Public Instance Constructors

MySqlError Constructor	

Public Instance Properties

Code	Error code
Level	Error level
Message	Error message

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlError Class, MySql.Data.MySqlClient Namespace](#)

MySqlError Constructor

Syntax: Visual Basic

```
Public Sub New( _
    ByVal level As String, _
    ByVal code As Integer, _
    ByVal message As String _
)
```

Syntax: C#

```
public MySqlError(
    string level,
    int code,
    string message
);
```

Parameters

- **level**:
- **code**:
- **message**:

See Also

[MySqlError Class, MySql.Data.MySqlClient Namespace](#)

Code Property

Error code

Syntax: Visual Basic

```
Public ReadOnly Property Code As Integer
```

Syntax: C#

```
public int Code {get;}
```

See Also

[MySqlError Class, MySql.Data.MySqlClient Namespace](#)

Level Property

Error level

Syntax: Visual Basic

```
Public ReadOnly Property Level As String
```

Syntax: C#

```
public string Level {get;}
```

See Also

[MySqlError Class, MySql.Data.MySqlClient Namespace](#)

Message Property

Error message

Syntax: Visual Basic

```
Public ReadOnly Property Message As String
```

Syntax: C#

```
public string Message {get;}
```

See Also

[MySqlError Class, MySql.Data.MySqlClient Namespace](#)

MySqlConnection.StateChange Event

Syntax: Visual Basic

```
Public Event StateChange As StateChangeEventHandler
```

Syntax: C#

```
public event StateChangeEventHandler StateChange;
```

See Also

[MySqlConnection Class, MySql.Data.MySqlClient Namespace](#)

MySqlCommand Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection, _  
    ByVal transaction As MySqlTransaction _  
)
```

Syntax: C#

```
public MySqlCommand(  
    stringcmdText,  
    MySqlConnectionconnection,  
    MySqlTransactiontransaction  
);
```

See Also

[MySqlCommand Class, MySql.Data.MySqlClient Namespace, MySqlCommand Constructor Overload List](#)

CommandText Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandText As String _
```

```

- Implements IDbCommand.CommandText

```

Syntax: C#

```

public string CommandText {get; set;}

```

Implements

IDbCommand.CommandText

See Also

[MySQLCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

CommandTimeout Property

Syntax: Visual Basic

```

NotOverridable Public Property CommandTimeout As Integer _
- Implements IDbCommand.CommandTimeout

```

Syntax: C#

```

public int CommandTimeout {get; set;}

```

Implements

IDbCommand.CommandTimeout

See Also

[MySQLCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

CommandType Property

Syntax: Visual Basic

```

NotOverridable Public Property CommandType As CommandType _
- Implements IDbCommand.CommandType

```

Syntax: C#

```

public System.Data.CommandType CommandType {get; set;}

```

Implements

IDbCommand.CommandType

See Also

[MySQLCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

Connection Property

Syntax: Visual Basic

```

Public Property Connection As MySqlConnection

```

Syntax: C#

```
public MySqlConnection Connection {get; set;}
```

See Also

[MySqlCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

IsPrepared Property
Syntax: Visual Basic

```
Public ReadOnly Property IsPrepared As Boolean
```

Syntax: C#

```
public bool IsPrepared {get;}
```

See Also

[MySqlCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

Parameters Property
Syntax: Visual Basic

```
Public ReadOnly Property Parameters As MySqlParameterCollection
```

Syntax: C#

```
public MySqlParameterCollection Parameters {get;}
```

See Also

[MySqlCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlParameterCollection Class

Represents a collection of parameters relevant to a [MySqlCommand](#) as well as their respective mappings to columns in a DataSet. This class cannot be inherited.

For a list of all members of this type, see [MySQLParameterCollection Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameterCollection_  
    Inherits MarshalByRefObject_  
    Implements IDataParameterCollection, IList, ICollection, IEnumerable
```

Syntax: C#

```
public sealed class MySqlParameterCollection : MarshalByRefObject, IDataParameterCollection, IList, ICollection, IEnumerable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlParameterCollection Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlParameterCollection Members

[MySqlParameterCollection overview](#)

Public Instance Constructors

MySqlParameterCollection Constructor	Initializes a new instance of the MySqlParameterCollection class.

Public Instance Properties

Count	Gets the number of MySqlParameter objects in the collection.
Item	Overloaded. Gets the MySqlParameter with a specified attribute. In C#, this property is the indexer for the MySqlParameterCollection class.

Public Instance Methods

Add	Overloaded. Adds the specified MySqlParameter object to the MySqlParameterCollection .
Clear	Removes all items from the collection.
Contains	Overloaded. Gets a value indicating whether a MySqlParameter exists in the collection.
CopyTo	Copies MySqlParameter objects from the MySqlParameterCollection to the specified array.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IndexOf	Overloaded. Gets the location of a MySqlParameter in the collection.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Insert	Inserts a MySqlParameter into the collection at the specified index.
Remove	Removes the specified MySqlParameter from the collection.

RemoveAt	Overloaded. Removes the specified MySqlParameter from the collection.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlParameterCollection Constructor

Initializes a new instance of the [MySqlParameterCollection](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlParameterCollection();
```

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

Count Property

Gets the number of [MySqlParameter](#) objects in the collection.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Count As Integer _  
    Implements ICollection.Count
```

Syntax: C#

```
public int Count {get;}
```

Implements

ICollection.Count

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

Item Property

Gets the [MySqlParameter](#) with a specified attribute. In C#, this property is the indexer for the [MySqlParameterCollection](#) class.

Overload List

Gets the [MySqlParameter](#) at the specified index.

- [public MySqlParameter this\[int\] {get; set;}](#)

Gets the [MySqlParameter](#) with the specified name.

- [public MySqlParameter this\[string\] {get; set;}](#)

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlParameter Class

Represents a parameter to a [MySqlCommand](#), and optionally, its mapping to DataSetcolumns. This class cannot be inherited.

For a list of all members of this type, see [MySqlParameter Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameter_
    Inherits MarshalByRefObject_
    Implements IDataParameter, IDbDataParameter, ICloneable
```

Syntax: C#

```
public sealed class MySqlParameter : MarshalByRefObject, IDataParameter, IDbDataParameter, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlParameter Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlParameter Members

[MySqlParameter overview](#)

Public Instance Constructors

MySqlParameter	Overloaded. Initializes a new instance of the MySqlParameter class.

Public Instance Properties

DbType	Gets or sets the DbType of the parameter.
Direction	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySQL version 4.1 and earlier, input-only is the only valid choice.
IsNullable	Gets or sets a value indicating whether the parameter accepts null values.
IsUnsigned	

MySqlDbType	Gets or sets the MySqlDbType of the parameter.
ParameterName	Gets or sets the name of the MySqlParameter.
Precision	Gets or sets the maximum number of digits used to represent the Value property.
Scale	Gets or sets the number of decimal places to which Value is resolved.
Size	Gets or sets the maximum size, in bytes, of the data within the column.
SourceColumn	Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the Value .
SourceVersion	Gets or sets the DataRowVersion to use when loading Value .
Value	Gets or sets the value of the parameter.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ToString	Overridden. Gets a string containing the ParameterName .

See Also

[MySqlParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySQLParameter Constructor

Initializes a new instance of the MySQLParameter class.

Overload List

Initializes a new instance of the MySQLParameter class.

- [public MySQLParameter\(\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and the data type.

- [public MySQLParameter\(string,MySqlDbType\);](#)

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySqlDbType](#), and the size.

- [public MySQLParameter\(string,MySqlDbType,int\);](#)

Initializes a new instance of the [MySqlParameter](#) class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersion to use, and the value of the parameter.

- [public](#)
[MySqlParameter\(string, MySqlDbType, int, ParameterDirection, bool, byte, byte, string, DataRowVersion, object\);](#)

Initializes a new instance of the [MySqlParameter](#) class with the parameter name, the [MySqlDbType](#), the size, and the source column name.

- [public MySqlParameter\(string, MySqlDbType, int, string\);](#)

Initializes a new instance of the [MySqlParameter](#) class with the parameter name and a value of the new [MySqlParameter](#).

- [public MySqlParameter\(string, object\);](#)

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlParameter Constructor ()

Initializes a new instance of the [MySqlParameter](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlParameter();
```

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

MySqlParameter Constructor

Initializes a new instance of the [MySqlParameter](#) class with the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal parameterName As String, _  
    ByVal dbType As MySqlDbType _  
)
```

Syntax: C#

```
public MySqlParameter(  
    stringparameterName,  
    MySqlDbTypedbType  
);
```

Parameters

- [parameterName](#): The name of the parameter to map.
- [dbType](#): One of the [MySqlDbType](#) values.

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

MySqlDbType Enumeration

Specifies MySQL-specific data type of a field, property, for use in a [MySqlParameter](#).

Syntax: Visual Basic

```
Public Enum MySqlDbType
```

Syntax: C#

```
public enum MySqlDbType
```

Members

Member Name	Description
Newdate	Obsolete. Use Datetime or Date type.
Timestamp	A timestamp. The range is '1970-01-01 00:00:01.000000' to '2038-01-19 03:14:07.999999'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Time	The range is '-838:59:59.000000' to '838:59:59.000000'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Date	The supported range is '1000-01-01' to '9999-12-31'.
Datetime	The supported range is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Year	A year in 2- or 4-digit format (default is 4-digit). The allowable values are 1901 to 2155, 0000 in the 4-digit year format, and 1970-2069 if you use the 2-digit format (70-69).
TinyBlob	A BLOB column with a maximum length of 255 (2 ⁸ - 1) characters.
Blob	A BLOB column with a maximum length of 65535 (2 ¹⁶ - 1) characters.
MediumBlob	A BLOB column with a maximum length of 16777215 (2 ²⁴ - 1) characters.
LongBlob	A BLOB column with a maximum length of 4294967295 or 4G (2 ³² - 1) characters.
Int16	A 16-bit signed integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.
Int24	Specifies a 24 (3 byte) signed or unsigned value.

Int32	A 32-bit signed integer.
Int64	A 64-bit signed integer.
Byte	The signed range is -128 to 127. The unsigned range is 0 to 255.
Float	A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.
Double	A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.
UByte	An 8-bit unsigned value.
UInt16	A 16-bit unsigned value.
UInt24	A 24-bit unsigned value.
UInt32	A 32-bit unsigned value.
UInt64	A 64-bit unsigned value.
Decimal	A fixed precision and scale numeric value between -10 ³⁸ -1 and 10 ³⁸ -1.
NewDecimal	New Decimal
Set	A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', ... A SET can have a maximum of 64 members.
String	Obsolete. Use VarChar type.
VarChar	A variable-length string containing 0 to 255 characters.
VarString	A variable-length string containing 0 to 65535 characters.
Enum	An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', ..., NULL or the special "" error value. An ENUM can have a maximum of 65535 distinct values.
Geometry	
Bit	Bit-field data type.
TinyText	A nonbinary string column supporting a maximum length of 255 (2 ⁸ - 1) characters.
Text	A nonbinary string column supporting a maximum length of 65535 (2 ¹⁶ - 1) characters.
MediumText	A nonbinary string column supporting a maximum length of 16777215 (2 ²⁴ - 1) characters.
LongText	A nonbinary string column supporting a maximum length of 4294967295 (2 ³² - 1) characters.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

MySqlParameter Constructor (String, MySqlDbType, Int32)

Initializes a new instance of the [MySqlParameter](#) class with the parameter name, the [MySqlDbType](#), and the size.

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer _
)
```

Syntax: C#

```
public MySqlParameter(
    string parameterName,
    MySqlDbType dbType,
    int size
);
```

Parameters

- [parameterName](#): The name of the parameter to map.
- [dbType](#): One of the [MySqlDbType](#) values.
- [size](#): The length of the parameter.

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

MySqlParameter Constructor

Initializes a new instance of the [MySqlParameter](#) class with the parameter name, the type of the parameter, the size of the parameter, a [ParameterDirection](#), the precision of the parameter, the scale of the parameter, the source column, a [DataRowVersion](#) to use, and the value of the parameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer, _
    ByVal direction As ParameterDirection, _
    ByVal isNullable As Boolean, _
    ByVal precision As Byte, _
    ByVal scale As Byte, _
    ByVal sourceColumn As String, _
    ByVal sourceVersion As DataRowVersion, _
    ByVal value As Object _
)
```

Syntax: C#

```
public MySqlParameter(
    string parameterName,
    MySqlDbType dbType,
    int size,
    ParameterDirection direction,
    bool isNullable,
    byte precision,
    byte scale,
    string sourceColumn,
    DataRowVersion sourceVersion,
    object value
);
```

Parameters

- **parameterName**: The name of the parameter to map.
- **dbType**: One of the [MySqlDbType](#) values.
- **size**: The length of the parameter.
- **direction**: One of the [ParameterDirection](#) values.
- **isNullable**: true if the value of the field can be null, otherwise false.
- **precision**: The total number of digits to the left and right of the decimal point to which [Value](#) is resolved.
- **scale**: The total number of decimal places to which [Value](#) is resolved.
- **sourceColumn**: The name of the source column.
- **sourceVersion**: One of the [DataRowVersion](#) values.
- **value**: An Object that is the value of the [MySqlParameter](#).

Exceptions

Exception Type	Condition
ArgumentException	

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

Value Property

Gets or sets the value of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property Value As Object _
    Implements IDataParameter.Value
```

Syntax: C#

```
public object Value {get; set;}
```

Implements

[IDataParameter.Value](#)

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySQLParameter Constructor

Initializes a new instance of the [MySQLParameter](#) class with the parameter name, the [MySQLDbType](#), the size, and the source column name.

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal parameterName As String, _
    ByVal dbType As MySQLDbType, _
    ByVal size As Integer, _
    ByVal sourceColumn As String _
)
```

Syntax: C#

```
public MySQLParameter(
    stringparameterName,
    MySQLDbTypeDbType,
    intsize,
    stringsourceColumn
);
```

Parameters

- [parameterName](#): The name of the parameter to map.
- [dbType](#): One of the [MySQLDbType](#) values.
- [size](#): The length of the parameter.
- [sourceColumn](#): The name of the source column.

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLParameter Constructor Overload List](#)

MySQLParameter Constructor

Initializes a new instance of the [MySQLParameter](#) class with the parameter name and a value of the new MySQLParameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal parameterName As String, _
    ByVal value As Object _
)
```

Syntax: C#

```
public MySQLParameter(
    stringparameterName,
    objectvalue
);
```

Parameters

- **parameterName**: The name of the parameter to map.
- **value**: An Object that is the value of the [MySqlParameter](#).

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

DbType Property

Gets or sets the DbType of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property DbType As DbType _
    Implements IDataParameter.DbType
```

Syntax: C#

```
public System.Data.DbType DbType {get; set;}
```

Implements

IDataParameter.DbType

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

Direction Property

Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySQL version 4.1 and earlier, input-only is the only valid choice.

Syntax: Visual Basic

```
NotOverridable Public Property Direction As ParameterDirection _
    Implements IDataParameter.Direction
```

Syntax: C#

```
public System.Data.ParameterDirection Direction {get; set;}
```

Implements

IDataParameter.Direction

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

IsNullable Property

Gets or sets a value indicating whether the parameter accepts null values.

Syntax: Visual Basic

```
NotOverridable Public Property IsNullable As Boolean _
    Implements IDataParameter.IsNullable
```

Syntax: C#

```
public bool IsNullable {get; set;}
```

Implements

IDataParameter.IsNullable

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

IsUnsigned Property

Syntax: Visual Basic

```
Public Property IsUnsigned As Boolean
```

Syntax: C#

```
public bool IsUnsigned {get; set;}
```

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySQLDbType Property

Gets or sets the MySQLDbType of the parameter.

Syntax: Visual Basic

```
Public Property MySQLDbType As MySQLDbType
```

Syntax: C#

```
public MySQLDbType MySQLDbType {get; set;}
```

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

ParameterName Property

Gets or sets the name of the MySQLParameter.

Syntax: Visual Basic

```
NotOverridable Public Property ParameterName As String _
    Implements IDataParameter.ParameterName
```

Syntax: C#

```
public string ParameterName {get; set;}
```

Implements

IDataParameter.ParameterName

See Also

[MySQLParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

Precision Property

Gets or sets the maximum number of digits used to represent the [Value](#) property.

Syntax: Visual Basic

```
NotOverridable Public Property Precision As Byte _  
- Implements IDbDataParameter.Precision
```

Syntax: C#

```
public byte Precision {get; set;}
```

Implements

IDbDataParameter.Precision

See Also

[MySQLParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

Scale Property

Gets or sets the number of decimal places to which [Value](#) is resolved.

Syntax: Visual Basic

```
NotOverridable Public Property Scale As Byte _  
- Implements IDbDataParameter.Scale
```

Syntax: C#

```
public byte Scale {get; set;}
```

Implements

IDbDataParameter.Scale

See Also

[MySQLParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

Size Property

Gets or sets the maximum size, in bytes, of the data within the column.

Syntax: Visual Basic

```
NotOverridable Public Property Size As Integer _
- Implements IDbDataParameter.Size
```

Syntax: C#

```
public int Size {get; set;}
```

Implements

IDbDataParameter.Size

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

SourceColumn Property

Gets or sets the name of the source column that is mapped to the DataSet and used for loading or returning the [Value](#).

Syntax: Visual Basic

```
NotOverridable Public Property SourceColumn As String _
- Implements IDataParameter.SourceColumn
```

Syntax: C#

```
public string SourceColumn {get; set;}
```

Implements

IDataParameter.SourceColumn

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

SourceVersion Property

Gets or sets the DataRowVersion to use when loading [Value](#).

Syntax: Visual Basic

```
NotOverridable Public Property SourceVersion As DataRowVersion _
- Implements IDataParameter.SourceVersion
```

Syntax: C#

```
public System.Data.DataRowVersion SourceVersion {get; set;}
```

Implements

IDataParameter.SourceVersion

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlParameter.ToString Method

Overridden. Gets a string containing the [ParameterName](#).

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

Return Value

See Also

[MySqlParameter Class](#), [MySql.Data.MySqlClient Namespace](#)

Item Property (Int32)

Gets the [MySqlParameter](#) at the specified index.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _  
    ByVal index As Integer _  
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter this[  
    int index  
] {get; set;}
```

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Item Overload List](#)

Item Property (String)

Gets the [MySqlParameter](#) with the specified name.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _  
    ByVal name As String _  
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter this[  
    string name  
] {get; set;}
```

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Item Overload List](#)

Add Method

Adds the specified [MySqlParameter](#) object to the [MySqlParameterCollection](#).

Overload List

Adds the specified [MySqlParameter](#) object to the [MySqlParameterCollection](#).

- [public MySqlParameter Add\(MySqlParameter\);](#)

Adds the specified [MySqlParameter](#) object to the [MySqlParameterCollection](#).

- [public int Add\(object\);](#)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) given the parameter name and the data type.

- [public MySqlParameter Add\(string,MySqlDbType\);](#)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, and the column length.

- [public MySqlParameter Add\(string,MySqlDbType,int\);](#)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, the column length, and the source column name.

- [public MySqlParameter Add\(string,MySqlDbType,int,string\);](#)

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) given the specified parameter name and value.

- [public MySqlParameter Add\(string,object\);](#)

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlParameterCollection.Add Method

Adds the specified [MySqlParameter](#) object to the [MySqlParameterCollection](#).

Syntax: Visual Basic

```
Overloads Public Function Add( _  
    ByVal value As MySqlParameter _  
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(  
    MySqlParametervalue  
);
```

Parameters

- [value](#): The [MySqlParameter](#) to add to the collection.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.Add Overload List](#)

MySQLParameterCollection.Add Method

Adds the specified [MySQLParameter](#) object to the [MySQLParameterCollection](#).

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Add( _
    ByVal value As Object _
) As Integer _
    Implements IList.Add
```

Syntax: C#

```
public int Add(
    objectvalue
);
```

Parameters

- **value**: The [MySQLParameter](#) to add to the collection.

Return Value

The index of the new [MySQLParameter](#) object.

Implements

IList.Add

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.Add Overload List](#)

MySQLParameterCollection.Add Method

Adds a [MySQLParameter](#) to the [MySQLParameterCollection](#) given the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Function Add( _
    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType _
) As MySQLParameter
```

Syntax: C#

```
public MySQLParameter Add(
    stringparameterName,
    MySqlDbTypedbType
);
```

Parameters

- **parameterName**: The name of the parameter.
- **dbType**: One of the [MySQLDbType](#) values.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Add Overload List](#)

MySqlParameterCollection.Add Method

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, and the column length.

Syntax: Visual Basic

```
Overloads Public Function Add( _
    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
    stringparameterName,
    MySqlDbTypedbType,
    intsize
);
```

Parameters

- [parameterName](#): The name of the parameter.
- [dbType](#): One of the [MySqlDbType](#) values.
- [size](#): The length of the column.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Add Overload List](#)

MySqlParameterCollection.Add Method

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) with the parameter name, the data type, the column length, and the source column name.

Syntax: Visual Basic

```
Overloads Public Function Add( _
    ByVal parameterName As String, _
    ByVal dbType As MySqlDbType, _
    ByVal size As Integer, _
    ByVal sourceColumn As String _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
    string parameterName,
    MySqlDbType dbType,
    int size,
    string sourceColumn
);
```

Parameters

- **parameterName**: The name of the parameter.
- **dbType**: One of the [MySqlDbType](#) values.
- **size**: The length of the column.
- **sourceColumn**: The name of the source column.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Add Overload List](#)

MySqlParameterCollection.Add Method

Adds a [MySqlParameter](#) to the [MySqlParameterCollection](#) given the specified parameter name and value.

Syntax: Visual Basic

```
Overloads Public Function Add( _
    ByVal parameterName As String, _
    ByVal value As Object _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
    string parameterName,
    object value
);
```

Parameters

- **parameterName**: The name of the parameter.
- **value**: The [Value](#) of the [MySqlParameter](#) to add to the collection.

Return Value

The newly added [MySqlParameter](#) object.

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Add Overload List](#)

MySqlParameterCollection.Clear Method

Removes all items from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Clear() _
    Implements IList.Clear
```

Syntax: C#

```
public void Clear();
```

Implements

IList.Clear

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

Contains Method

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

Overload List

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

- [public bool Contains\(object\);](#)

Gets a value indicating whether a [MySQLParameter](#) with the specified parameter name exists in the collection.

- [public bool Contains\(string\);](#)

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySQLParameterCollection.Contains Method

Gets a value indicating whether a [MySQLParameter](#) exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _
    ByVal value As Object _
) As Boolean _
    Implements IList.Contains
```

Syntax: C#

```
public bool Contains(
    objectvalue
);
```

Parameters

- [value](#): The value of the [MySQLParameter](#) object to find.

Return Value

true if the collection contains the [MySQLParameter](#) object; otherwise, false.

Implements

ICollection.Contains

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Contains Overload List](#)

MySqlParameterCollection.Contains Method

Gets a value indicating whether a [MySqlParameter](#) with the specified parameter name exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _
    ByVal name As String _
) As Boolean _
- Implements IDataParameterCollection.Contains
```

Syntax: C#

```
public bool Contains(
    string name
);
```

Parameters

- name**: The name of the [MySqlParameter](#) object to find.

Return Value

true if the collection contains the parameter; otherwise, false.

Implements

IDataParameterCollection.Contains

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Contains Overload List](#)

MySqlParameterCollection.CopyTo Method

Copies MySqlParameter objects from the MySqlParameterCollection to the specified array.

Syntax: Visual Basic

```
NotOverridable Public Sub CopyTo( _
    ByVal array As Array, _
    ByVal index As Integer _
) _
- Implements ICollection.CopyTo
```

Syntax: C#

```
public void CopyTo(
```

```
Arrayarray,
intindex
);
```

Parameters

- `array`:
- `index`:

Implements

ICollection.CopyTo

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

IndexOf Method

Gets the location of a [MySQLParameter](#) in the collection.

Overload List

Gets the location of a [MySQLParameter](#) in the collection.

- `public int IndexOf(object);`

Gets the location of the [MySQLParameter](#) in the collection with a specific parameter name.

- `public int IndexOf(string);`

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySQLParameterCollection.IndexOf Method

Gets the location of a [MySQLParameter](#) in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _
    ByVal value As Object _
) As Integer _
-
    Implements IList.IndexOf
```

Syntax: C#

```
public int IndexOf(
    objectvalue
);
```

Parameters

- `value`: The [MySQLParameter](#) object to locate.

Return Value

The zero-based location of the [MySQLParameter](#) in the collection.

Implements

IList.IndexOf

See Also

[MySQLParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.IndexOf Overload List](#)

MySQLParameterCollection.IndexOf Method

Gets the location of the [MySQLParameter](#) in the collection with a specific parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _
    ByVal parameterName As String _
) As Integer _
- Implements IDataParameterCollection.IndexOf
```

Syntax: C#

```
public int IndexOf(
    string parameterName
);
```

Parameters

- **parameterName**: The name of the [MySQLParameter](#) object to retrieve.

Return Value

The zero-based location of the [MySQLParameter](#) in the collection.

Implements

IDataParameterCollection.IndexOf

See Also

[MySQLParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.IndexOf Overload List](#)

MySQLParameterCollection.Insert Method

Inserts a [MySQLParameter](#) into the collection at the specified index.

Syntax: Visual Basic

```
NotOverridable Public Sub Insert( _
    ByVal index As Integer, _
    ByVal value As Object _
) _
- Implements IList.Insert
```

Syntax: C#

```
public void Insert(
    int index,
    object value
);
```

Parameters

- [index:](#)
- [value:](#)

Implements

ICollection.Insert

See Also

[MySQLParameterCollection Class, MySQL.Data.MySqlClient Namespace](#)

MySQLParameterCollection.Remove Method

Removes the specified [MySQLParameter](#) from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Remove( _
    ByVal value As Object _
) _
    Implements ICollection.Remove
```

Syntax: C#

```
public void Remove(
    object value
);
```

Parameters

- [value:](#)

Implements

ICollection.Remove

See Also

[MySQLParameterCollection Class, MySQL.Data.MySqlClient Namespace](#)

RemoveAt Method

Removes the specified [MySQLParameter](#) from the collection.

Overload List

Removes the specified [MySQLParameter](#) from the collection using a specific index.

- [public void RemoveAt\(int\);](#)

Removes the specified [MySQLParameter](#) from the collection using the parameter name.

- [public void RemoveAt\(string\);](#)

See Also

[MySQLParameterCollection Class, MySQL.Data.MySqlClient Namespace](#)

MySQLParameterCollection.RemoveAt Method

Removes the specified [MySQLParameter](#) from the collection using a specific index.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _
    ByVal index As Integer _
) _
- Implements IList.RemoveAt
```

Syntax: C#

```
public void RemoveAt(
    int index
);
```

Parameters

- **index**: The zero-based index of the parameter.

Implements

IList.RemoveAt

See Also

[MySQLParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.RemoveAt Overload List](#)

MySQLParameterCollection.RemoveAt Method

Removes the specified [MySQLParameter](#) from the collection using the parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _
    ByVal name As String _
) _
- Implements IDataParameterCollection.RemoveAt
```

Syntax: C#

```
public void RemoveAt(
    string name
);
```

Parameters

- **name**: The name of the [MySQLParameter](#) object to retrieve.

Implements

IDataParameterCollection.RemoveAt

See Also

[MySQLParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.RemoveAt Overload List](#)

Transaction Property

Syntax: Visual Basic


```
Public Property Transaction As MySqlConnectionTransaction
```

Syntax: C#

```
public MySqlConnection Transaction {get; set;}
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

UpdatedRowSource Property

Syntax: Visual Basic

```
NotOverridable Public Property UpdatedRowSource As UpdateRowSource _  
    Implements IDbCommand.UpdatedRowSource
```

Syntax: C#

```
public System.Data.UpdateRowSource UpdatedRowSource {get; set;}
```

Implements

IDbCommand.UpdatedRowSource

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand.Cancel Method

Attempts to cancel the execution of a MySqlCommand. This operation is not supported.

Syntax: Visual Basic

```
NotOverridable Public Sub Cancel() _  
    Implements IDbCommand.Cancel
```

Syntax: C#

```
public void Cancel();
```

Implements

IDbCommand.Cancel

Remarks

Cancelling an executing command is currently not supported on any version of MySQL.

Exceptions

Exception Type	Condition
NotSupportedException	This operation is not supported.

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand.CreateParameter Method

Creates a new instance of a [MySqlParameter](#) object.

Syntax: Visual Basic

```
Public Function CreateParameter() As MySqlParameter
```

Syntax: C#

```
public MySqlParameter CreateParameter();
```

Return Value

A [MySqlParameter](#) object.

Remarks

This method is a strongly-typed version of CreateParameter.

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand.ExecuteNonQuery Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteNonQuery() As Integer _  
    Implements IDbCommand.ExecuteNonQuery
```

Syntax: C#

```
public int ExecuteNonQuery();
```

Implements

IDbCommand.ExecuteNonQuery

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

ExecuteReader Method

Overload List

- [public MySqlDataReader ExecuteReader\(\);](#)
- [public MySqlDataReader ExecuteReader\(CommandBehavior\);](#)

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand.ExecuteReader Method

Syntax: Visual Basic

Overloads Public Function ExecuteReader() As MySqlDataReader

Syntax: C#

```
public MySqlDataReader ExecuteReader();
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand.ExecuteReader Overload List](#)

MySqlDataReader Class

Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.

For a list of all members of this type, see [MySqlDataReader Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataReader_  
    Inherits MarshalByRefObject_  
    Implements IEnumerable, IDataReader, IDisposable, IDataRecord
```

Syntax: C#

```
public sealed class MySqlDataReader : MarshalByRefObject, IEnumerable, IDataReader, IDisposable, IDataRecord
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDataReader Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader Members

[MySqlDataReader overview](#)

Public Instance Properties

Depth	Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.
FieldCount	Gets the number of columns in the current row.
HasRows	Gets a value indicating whether the MySqlDataReader contains one or more rows.
IsClosed	Gets a value indicating whether the data reader is closed.

Item	Overloaded. Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Public Instance Methods

Close	Closes the MySqlDataReader object.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBoolean	Gets the value of the specified column as a Boolean.
GetByte	Gets the value of the specified column as a byte.
GetBytes	Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.
GetChar	Gets the value of the specified column as a single character.
GetChars	Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.
GetDataTypeName	Gets the name of the source data type.
GetDateTime	
GetDecimal	
GetDouble	
GetFieldType	Gets the Type that is the data type of the object.
GetFloat	
GetGuid	
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetInt16	
GetInt32	
GetInt64	
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetMySqlDateTime	
GetName	Gets the name of the specified column.
GetOrdinal	Gets the column ordinal, given the name of the column.
GetSchemaTable	Returns a DataTable that describes the column metadata of the MySqlDataReader.
GetString	
GetTimeSpan	

GetType (inherited from Object)	Gets the Type of the current instance.
GetUInt16	
GetUInt32	
GetUInt64	
GetValue	Gets the value of the specified column in its native format.
GetValues	Gets all attribute columns in the collection for the current row.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
IsDBNull	Gets a value indicating whether the column contains non-existent or missing values.
NextResult	Advances the data reader to the next result, when reading the results of batch SQL statements.
Read	Advances the MySqlDataReader to the next record.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

Depth Property

Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Depth As Integer _
    Implements IDataReader.Depth
```

Syntax: C#

```
public int Depth {get;}
```

Implements

IDataReader.Depth

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

FieldCount Property

Gets the number of columns in the current row.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property FieldCount As Integer _
    Implements IDataRecord.FieldCount
```

Syntax: C#

```
public int FieldCount {get;}
```

Implements

IDataRecord.FieldCount

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

HasRows Property

Gets a value indicating whether the MySqlDataReader contains one or more rows.

Syntax: Visual Basic

```
Public ReadOnly Property HasRows As Boolean
```

Syntax: C#

```
public bool HasRows {get;}
```

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

IsClosed Property

Gets a value indicating whether the data reader is closed.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsClosed As Boolean _  
    Implements IDataReader.IsClosed
```

Syntax: C#

```
public bool IsClosed {get;}
```

Implements

IDataReader.IsClosed

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

Item Property

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

Overload List

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

- [public object this\[int\] {get;}](#)

Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

- `public object this[string] {get;}`

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

Item Property (Int32)

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _
    ByVal i As Integer _
) _
- Implements IDataRecord.Item As Object _
- Implements IDataRecord.Item
```

Syntax: C#

```
public object this[
    inti
] {get;}
```

Implements

`IDataRecord.Item`

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlDataReader.Item Overload List](#)

Item Property (String)

Gets the value of a column in its native format. In C#, this property is the indexer for the `MySqlDataReader` class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _
    ByVal name As String _
) _
- Implements IDataRecord.Item As Object _
- Implements IDataRecord.Item
```

Syntax: C#

```
public object this[
    stringname
] {get;}
```

Implements

`IDataRecord.Item`

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlDataReader.Item Overload List](#)

RecordsAffected Property

Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property RecordsAffected As Integer _
- Implements IDataReader.RecordsAffected
```

Syntax: C#

```
public int RecordsAffected {get;}
```

Implements

IDataReader.RecordsAffected

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.Close Method

Closes the MySqlDataReader object.

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _
- Implements IDataReader.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDataReader.Close

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetBoolean Method

Gets the value of the specified column as a Boolean.

Syntax: Visual Basic

```
NotOverridable Public Function GetBoolean( _
    ByVal i As Integer _
) As Boolean _
- Implements IDataRecord.GetBoolean
```

Syntax: C#


```
public bool GetBoolean(
    int i
);
```

Parameters

- [i](#):

Return Value

Implements

IDataRecord.GetBoolean

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetByte Method

Gets the value of the specified column as a byte.

Syntax: Visual Basic

```
NotOverridable Public Function GetByte( _
    ByVal i As Integer _
) As Byte _
- Implements IDataRecord.GetByte
```

Syntax: C#

```
public byte GetByte(
    int i
);
```

Parameters

- [i](#):

Return Value

Implements

IDataRecord.GetByte

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetBytes Method

Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetBytes( _
    ByVal i As Integer, _
    ByVal dataIndex As Long, _
    ByVal buffer As Byte(), _
    ByVal bufferIndex As Integer, _
```

```

    ByVal length As Integer _
) As Long _
- Implements IDataRecord.GetBytes

```

Syntax: C#

```

public long GetBytes(
    inti,
    long dataIndex,
    byte[] buffer,
    int bufferIndex,
    int length
);

```

Parameters

- **i**: The zero-based column ordinal.
- **dataIndex**: The index within the field from which to begin the read operation.
- **buffer**: The buffer into which to read the stream of bytes.
- **bufferIndex**: The index for buffer to begin the read operation.
- **length**: The maximum length to copy into the buffer.

Return Value

The actual number of bytes read.

Implements

IDataRecord.GetBytes

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetChar Method

Gets the value of the specified column as a single character.

Syntax: Visual Basic

```

NotOverridable Public Function GetChar( _
    ByVal i As Integer _
) As Char _
- Implements IDataRecord.GetChar

```

Syntax: C#

```

public char GetChar(
    inti
);

```

Parameters

- **i**:

Return Value

Implements

IDataRecord.GetChar

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetChars Method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetChars( _
    ByVal i As Integer, _
    ByVal fieldOffset As Long, _
    ByVal buffer As Char(), _
    ByVal bufferoffset As Integer, _
    ByVal length As Integer _
) As Long _
    Implements IDataRecord.GetChars
```

Syntax: C#

```
public long GetChars(
    inti,
    longfieldOffset,
    char[]buffer,
    intbufferoffset,
    intlength
);
```

Parameters

- **i**:
- **fieldOffset**:
- **buffer**:
- **bufferoffset**:
- **length**:

Return Value

Implements

IDataRecord.GetChars

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetDataTypeName Method

Gets the name of the source data type.

Syntax: Visual Basic

```
NotOverridable Public Function GetDataTypeName( _
    ByVal i As Integer _
```

```
) As String _  
- Implements IDataRecord.GetDataTypeName
```

Syntax: C#

```
public string GetDataTypeName(  
    inti  
);
```

Parameters

- **i**:

Return Value

Implements

IDataRecord.GetDataTypeName

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetDateTime Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDateTime( _  
    ByVal index As Integer _  
) As Date _  
- Implements IDataRecord.GetDateTime
```

Syntax: C#

```
public DateTime GetDateTime(  
    int index  
);
```

Implements

IDataRecord.GetDateTime

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetDecimal Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDecimal( _  
    ByVal index As Integer _  
) As Decimal _  
- Implements IDataRecord.GetDecimal
```

Syntax: C#

```
public decimal GetDecimal(  
    int index
```

```
);
```

Implements

IDataRecord.GetDecimal

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetDouble Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDouble( _  
    ByVal index As Integer _  
) As Double _  
- Implements IDataRecord.GetDouble
```

Syntax: C#

```
public double GetDouble(  
    int index  
);
```

Implements

IDataRecord.GetDouble

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetFieldType Method

Gets the Type that is the data type of the object.

Syntax: Visual Basic

```
NotOverridable Public Function GetFieldType( _  
    ByVal i As Integer _  
) As Type _  
- Implements IDataRecord.GetFieldType
```

Syntax: C#

```
public Type GetFieldType(  
    int i  
);
```

Parameters

- **i**:

Return Value

Implements

IDataRecord.GetFieldType

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetFloat Method

Syntax: Visual Basic

```
NotOverridable Public Function GetFloat( _  
    ByVal index As Integer _  
) As Single _  
- Implements IDataRecord.GetFloat
```

Syntax: C#

```
public float GetFloat(  
int index  
);
```

Implements

IDataRecord.GetFloat

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetGuid Method

Syntax: Visual Basic

```
NotOverridable Public Function GetGuid( _  
    ByVal index As Integer _  
) As Guid _  
- Implements IDataRecord.GetGuid
```

Syntax: C#

```
public Guid GetGuid(  
int index  
);
```

Implements

IDataRecord.GetGuid

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetInt16 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt16( _  
    ByVal index As Integer _  
) As Short _  
- Implements IDataRecord.GetInt16
```

Syntax: C#

```
public short GetInt16(
    int index
);
```

Implements

IDataRecord.GetInt16

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetInt32 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt32( _
    ByVal index As Integer _
) As Integer _
-
    Implements IDataRecord.GetInt32
```

Syntax: C#

```
public int GetInt32(
    int index
);
```

Implements

IDataRecord.GetInt32

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetInt64 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt64( _
    ByVal index As Integer _
) As Long _
-
    Implements IDataRecord.GetInt64
```

Syntax: C#

```
public long GetInt64(
    int index
);
```

Implements

IDataRecord.GetInt64

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetMySqlDateTime Method

Syntax: Visual Basic

```
Public Function GetMySqlDateTime( _
    ByVal index As Integer _
) As MySqlDateTime
```

Syntax: C#

```
public MySqlDateTime GetMySqlDateTime(
    int index
);
```

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetName Method

Gets the name of the specified column.

Syntax: Visual Basic

```
NotOverridable Public Function GetName( _
    ByVal i As Integer _
) As String _
    Implements IDataRecord.GetName
```

Syntax: C#

```
public string GetName(
    int i
);
```

Parameters

- **i**:

Return Value

Implements

IDataRecord.GetName

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetOrdinal Method

Gets the column ordinal, given the name of the column.

Syntax: Visual Basic

```
NotOverridable Public Function GetOrdinal( _
    ByVal name As String _
) As Integer _
    Implements IDataRecord.GetOrdinal
```

Syntax: C#

```
public int GetOrdinal(
```



```
stringname
);
```

Parameters

- **name:**

Return Value

Implements

IDataRecord.GetOrdinal

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetSchemaTable Method

Returns a DataTable that describes the column metadata of the MySqlDataReader.

Syntax: Visual Basic

```
NotOverridable Public Function GetSchemaTable() As DataTable _
    Implements IDataReader.GetSchemaTable
```

Syntax: C#

```
public DataTable GetSchemaTable();
```

Return Value

Implements

IDataReader.GetSchemaTable

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetString Method

Syntax: Visual Basic

```
NotOverridable Public Function GetString( _
    ByVal index As Integer _
) As String _
    Implements IDataRecord.GetString
```

Syntax: C#

```
public string GetString(
    int index
);
```

Implements

IDataRecord.GetString

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetTimeSpan Method

Syntax: Visual Basic

```
Public Function GetTimeSpan( _
    ByVal index As Integer _
) As TimeSpan
```

Syntax: C#

```
public TimeSpan GetTimeSpan(
    int index
);
```

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetUInt16 Method

Syntax: Visual Basic

```
Public Function GetUInt16( _
    ByVal index As Integer _
) As UInt16
```

Syntax: C#

```
public ushort GetUInt16(
    int index
);
```

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetUInt32 Method

Syntax: Visual Basic

```
Public Function GetUInt32( _
    ByVal index As Integer _
) As UInt32
```

Syntax: C#

```
public uint GetUInt32(
    int index
);
```

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.GetUInt64 Method

Syntax: Visual Basic

```
Public Function GetUInt64( _
```

```

        ByVal index As Integer _
    ) As UInt64

```

Syntax: C#

```

public ulong GetUInt64(
    int index
);

```

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetValue Method

Gets the value of the specified column in its native format.

Syntax: Visual Basic

```

NotOverridable Public Function GetValue( _
    ByVal i As Integer _
) As Object _
    Implements IDataRecord.GetValue

```

Syntax: C#

```

public object GetValue(
    int i
);

```

Parameters

- **i**:

Return Value

Implements

IDataRecord.GetValue

See Also

[MySqlDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlDataReader.GetValues Method

Gets all attribute columns in the collection for the current row.

Syntax: Visual Basic

```

NotOverridable Public Function GetValues( _
    ByVal values As Object() _
) As Integer _
    Implements IDataRecord.GetValues

```

Syntax: C#

```

public int GetValues(
    object[] values
);

```

Parameters

- [values](#):

Return Value

Implements

IDataRecord.GetValues

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.IsDBNull Method

Gets a value indicating whether the column contains non-existent or missing values.

Syntax: Visual Basic

```
NotOverridable Public Function IsDBNull( _
    ByVal i As Integer _
) As Boolean _
- Implements IDataRecord.IsDBNull
```

Syntax: C#

```
public bool IsDBNull(
    inti
);
```

Parameters

- [i](#):

Return Value

Implements

IDataRecord.IsDBNull

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.NextResult Method

Advances the data reader to the next result, when reading the results of batch SQL statements.

Syntax: Visual Basic

```
NotOverridable Public Function NextResult() As Boolean _
- Implements IDataReader.NextResult
```

Syntax: C#

```
public bool NextResult();
```

Return Value

Implements

IDataReader.NextResult

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataReader.Read Method

Advances the MySqlDataReader to the next record.

Syntax: Visual Basic

```
NotOverridable Public Function Read() As Boolean _
    Implements IDataReader.Read
```

Syntax: C#

```
public bool Read();
```

Return Value

Implements

IDataReader.Read

See Also

[MySqlDataReader Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand.ExecuteReader Method

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader( _
    ByVal behavior As CommandBehavior _
) As MySqlDataReader
```

Syntax: C#

```
public MySqlDataReader ExecuteReader(
    CommandBehaviorbehavior
);
```

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommand.ExecuteReader Overload List](#)

MySqlCommand.ExecuteScalar Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteScalar() As Object _
    Implements IDbCommand.ExecuteScalar
```

Syntax: C#

```
public object ExecuteScalar();
```

Implements

IDbCommand.ExecuteScalar

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommand.Prepare Method

Syntax: Visual Basic

```
NotOverridable Public Sub Prepare() _  
    Implements IDbCommand.Prepare
```

Syntax: C#

```
public void Prepare();
```

Implements

IDbCommand.Prepare

See Also

[MySqlCommand Class](#), [MySql.Data.MySqlClient Namespace](#)

4.10.1.5 MySqlCommandBuilder Class

For a list of all members of this type, see [MySqlCommandBuilder Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommandBuilder_  
    Inherits Component
```

Syntax: C#

```
public sealed class MySqlCommandBuilder : Component
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlCommandBuilder Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommandBuilder Members

[MySqlCommandBuilder overview](#)

Public Static (Shared) Methods

DeriveParameters	Overloaded. Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySQL.

Public Instance Constructors

MySqlCommandBuilder	Overloaded. Initializes a new instance of the MySqlCommandBuilder class.

Public Instance Properties

Container (inherited from Component)	Gets the IContainer that contains the Component.
DataAdapter	
QuotePrefix	
QuoteSuffix	
Site (inherited from Component)	Gets or sets the ISite of the Component.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetDeleteCommand	
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetInsertCommand	
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
GetUpdateCommand	
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
RefreshSchema	
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Overload List

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

- [public static void DeriveParameters\(MySqlCommand\);](#)
- [public static void DeriveParameters\(MySqlCommand,bool\);](#)

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommandBuilder.DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _
    ByVal command As MySqlCommand _
)
```

Syntax: C#

```
public static void DeriveParameters(
    MySqlCommand command
);
```

Parameters

- [command](#): The MySqlCommand referencing the stored procedure from which the parameter information is to be derived. The derived parameters are added to the Parameters collection of the MySqlCommand.

Exceptions

Exception Type	Condition
InvalidOperationException	The command text is not a valid stored procedure name.

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlCommandBuilder.DeriveParameters Overload List](#)

MySqlCommandBuilder.DeriveParameters Method

Syntax: Visual Basic


```
Overloads Public Shared Sub DeriveParameters( _
    ByVal command As MySqlCommand, _
    ByVal useProc As Boolean _
)
```

Syntax: C#

```
public static void DeriveParameters(
    MySqlCommand command,
    bool useProc
);
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlCommandBuilder.DeriveParameters Overload List](#)

MySqlCommandBuilder Constructor

Initializes a new instance of the [MySqlCommandBuilder](#) class.

Overload List

Initializes a new instance of the [MySqlCommandBuilder](#) class.

- [public MySqlCommandBuilder\(\);](#)
- [public MySqlCommandBuilder\(MySqlDataAdapter\);](#)
- [public MySqlCommandBuilder\(MySqlDataAdapter,bool\);](#)
- [public MySqlCommandBuilder\(bool\);](#)

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommandBuilder Constructor

Initializes a new instance of the [MySqlCommandBuilder](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommandBuilder();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

MySqlCommandBuilder Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal adapter As MySqlDataAdapter _
)
```

)

Syntax: C#

```
public MySqlCommandBuilder(
    MySqlDataAdapter adapter
);
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

MySqlDataAdapter Class

For a list of all members of this type, see [MySqlDataAdapter Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataAdapter_
    Inherits DbDataAdapter
```

Syntax: C#

```
public sealed class MySqlDataAdapter : DbDataAdapter
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDataAdapter Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataAdapter Members

[MySqlDataAdapter overview](#)

Public Instance Constructors

MySqlDataAdapter	Overloaded. Initializes a new instance of the MySqlDataAdapter class.

Public Instance Properties

AcceptChangesDuringFill (inherited from DataAdapter)	Gets or sets a value indicating whether AcceptChanges is called on a DataRow after it is added to the DataTable during any of the Fill operations.
AcceptChangesDuringUpdate (inherited from DataAdapter)	Gets or sets whether AcceptChanges is called during a Update.
Container (inherited from Component)	Gets the IContainer that contains the Component.

ContinueUpdateOnError (inherited from DataAdapter)	Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.
DeleteCommand	Overloaded.
FillLoadOption (inherited from DataAdapter)	Gets or sets the LoadOption that determines how the adapter fills the DataTable from the DbDataReader.
InsertCommand	Overloaded.
MissingMappingAction (inherited from DataAdapter)	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction (inherited from DataAdapter)	Determines the action to take when existing DataSet schema does not match incoming data.
ReturnProviderSpecificTypes (inherited from DataAdapter)	Gets or sets whether the Fill method should return provider-specific values or common CLS-compliant values.
SelectCommand	Overloaded.
Site (inherited from Component)	Gets or sets the ISite of the Component.
TableMappings (inherited from DataAdapter)	Gets a collection that provides the master mapping between a source table and a DataTable.
UpdateBatchSize (inherited from DbDataAdapter)	Gets or sets a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
UpdateCommand	Overloaded.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
Fill (inherited from DbDataAdapter)	Overloaded. Adds or refreshes rows in the DataSet to match those in the data source using the DataSet name, and creates a DataTable named "Table."
FillSchema (inherited from DbDataAdapter)	Overloaded. Configures the schema of the specified DataTable based on the specified SchemaType.
GetFillParameters (inherited from DbDataAdapter)	Gets the parameters set by the user when executing an SQL SELECT statement.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.

ResetFillLoadOption (inherited from DataAdapter)	Resets FillLoadOption to its default state and causes Fill to honor AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill (inherited from DataAdapter)	Determines whether the AcceptChangesDuringFill property should be persisted.
ShouldSerializeFillLoadOption (inherited from DataAdapter)	Determines whether the FillLoadOption property should be persisted.
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.
Update (inherited from DbDataAdapter)	Overloaded. Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
FillError (inherited from DataAdapter)	Returned when an error occurs during a fill operation.
RowUpdated	Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.
RowUpdating	Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Protected Internal Instance Properties

FillCommandBehavior (inherited from DbDataAdapter)	Gets or sets the behavior of the command used to fill the data adapter.

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataAdapter Constructor

Initializes a new instance of the [MySqlDataAdapter](#) class.

Overload List

Initializes a new instance of the [MySqlDataAdapter](#) class.

- [public MySqlDataAdapter\(\);](#)
- [public MySqlDataAdapter\(MySqlCommand\);](#)
- [public MySqlDataAdapter\(string, MySqlConnection\);](#)
- [public MySqlDataAdapter\(string, string\);](#)

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataAdapter Constructor

Initializes a new instance of the [MySqlDataAdapter](#) class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlDataAdapter();
```

See Also

[MySqlDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlDataAdapter Constructor Overload List](#)

MySqlDataAdapter Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommand As MySqlCommand _  
)
```

Syntax: C#

```
public MySqlDataAdapter(  
    MySqlCommandselectCommand  
);
```

See Also

[MySqlDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlDataAdapter Constructor Overload List](#)

MySqlDataAdapter Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommandText As String, _  
    ByVal connection As MySqlConnection _  
)
```

Syntax: C#

```
public MySqlDataAdapter(  
    stringselectCommandText,  
    MySqlConnectionconnection  
);
```

See Also

[MySqlDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlDataAdapter Constructor Overload List](#)

MySqlDataAdapter Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal selectCommandText As String, _
    ByVal selectConnString As String _
)
```

Syntax: C#

```
public MySqlDataAdapter(
    stringselectCommandText,
    stringselectConnString
);
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlDataAdapter Constructor Overload List](#)

DeleteCommand Property

Syntax: Visual Basic

```
Overloads Public Property DeleteCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand DeleteCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

InsertCommand Property

Syntax: Visual Basic

```
Overloads Public Property InsertCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand InsertCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

SelectCommand Property

Syntax: Visual Basic

```
Overloads Public Property SelectCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand SelectCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

UpdateCommand Property

Syntax: Visual Basic

```
Overloads Public Property UpdateCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand UpdateCommand {get; set;}
```

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataAdapter.RowUpdated Event

Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdated As MySqlRowUpdatedEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatedEventHandler RowUpdated;
```

Event Data

The event handler receives an argument of type [MySqlRowUpdatedEventArgs](#) containing data related to this event. The following [MySqlRowUpdatedEventArgsproperties](#) provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand executed when Update is called.
Errors	Gets any errors generated by the .NET Framework data provider when the Commandwas executed.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row	Gets the DataRow sent through an Update.
RowCount	Gets the number of rows processed in a batch of updated records.
StatementType	Gets the type of SQL statement executed.
Status	Gets the UpdateStatus of the Commandproperty.
TableMapping	Gets the DataTableMapping sent through an Update.

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlRowUpdatedEventHandler Delegate

Represents the method that will handle the RowUpdatedevent of a [MySqlDataAdapter](#).

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatedEventHandler( _
    ByVal sender As Object, _
    ByVal e As MySqlRowUpdatedEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatedEventHandler(
    object sender,
    MySqlRowUpdatedEventArgs
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

MySqlRowUpdatedEventArgs Class

Provides data for the RowUpdated event. This class cannot be inherited.

For a list of all members of this type, see [MySqlRowUpdatedEventArgs Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatedEventArgs_
    Inherits RowUpdatedEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatedEventArgs : RowUpdatedEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlRowUpdatedEventArgs Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlRowUpdatedEventArgs Members

[MySqlRowUpdatedEventArgs overview](#)

Public Instance Constructors

MySqlRowUpdatedEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatedEventArgs class.

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand executed when Update is called.
Errors (inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command was executed.
RecordsAffected (inherited from RowUpdatedEventArgs)	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row (inherited from RowUpdatedEventArgs)	Gets the DataRow sent through an Update.
RowCount (inherited from RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.
StatementType (inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status (inherited from RowUpdatedEventArgs)	Gets the UpdateStatus of the Command property.
TableMapping (inherited from RowUpdatedEventArgs)	Gets the DataTableMapping sent through an Update.

Public Instance Methods

CopyToRows (inherited from RowUpdatedEventArgs)	Overloaded. Copies references to the modified rows into the provided array.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySqlRowUpdatedEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlRowUpdatedEventArgs Constructor

Initializes a new instance of the MySqlRowUpdatedEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
    ByVal row As DataRow, _
    ByVal command As IDbCommand, _
    ByVal statementType As StatementType, _
    ByVal tableMapping As DataTableMapping _
)
```

Syntax: C#

```
public MySqlRowUpdatedEventArgs(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
);
```

Parameters

- **row**: The DataRow sent through an Update.
- **command**: The IDbCommand executed when Update is called.
- **statementType**: One of the StatementType values that specifies the type of query executed.
- **tableMapping**: The DataTableMapping sent through an Update.

See Also

[MySqlRowUpdatedEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

Command Property

Gets or sets the MySqlCommand executed when Update is called.

Syntax: Visual Basic

```
Overloads Public ReadOnly Property Command As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand Command {get;}
```

See Also

[MySqlRowUpdatedEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlDataAdapter.RowUpdating Event

Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdating As MySqlRowUpdatingEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatingEventHandler RowUpdating;
```

Event Data

The event handler receives an argument of type [MySqlRowUpdatingEventArgs](#) containing data related to this event. The following [MySqlRowUpdatingEventArgs](#) properties provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand to execute when performing the Update.
Errors	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType	Gets the type of SQL statement to execute.

Status	Gets or sets the UpdateStatus of the Commandproperty.
TableMapping	Gets the DataTableMapping to send through the Update.

See Also

[MySqlDataAdapter Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlRowUpdatingEventHandler Delegate

Represents the method that will handle the RowUpdatingevent of a [MySqlDataAdapter](#).

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatingEventHandler( _
    ByVal sender As Object, _
    ByVal e As MySqlRowUpdatingEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatingEventHandler(
    object sender,
    MySqlRowUpdatingEventArgs
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

MySqlRowUpdatingEventArgs Class

Provides data for the RowUpdating event. This class cannot be inherited.

For a list of all members of this type, see [MySqlRowUpdatingEventArgs Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatingEventArgs_
    Inherits RowUpdatingEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatingEventArgs : RowUpdatingEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlRowUpdatingEventArgs Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlRowUpdatingEventArgs Members

[MySqlRowUpdatingEventArgs overview](#)

Public Instance Constructors

MySqlRowUpdatingEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatingEventArgs class.

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand to execute when performing the Update.
Errors (inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row (inherited from RowUpdatingEventArgs)	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType (inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status (inherited from RowUpdatingEventArgs)	Gets or sets the UpdateStatus of the Commandproperty.
TableMapping (inherited from RowUpdatingEventArgs)	Gets the DataTableMapping to send through the Update.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySqlRowUpdatingEventArgs Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlRowUpdatingEventArgs Constructor

Initializes a new instance of the MySqlRowUpdatingEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
    ByVal row As DataRow, _
    ByVal command As IDbCommand, _
    ByVal statementType As StatementType, _
    ByVal tableMapping As DataTableMapping _
```

)

Syntax: C#

```
public MySqlRowUpdatingEventArgs(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
);
```

Parameters

- **row**: The DataRow to update.
- **command**: The IDbCommand to execute during update.
- **statementType**: One of the StatementType values that specifies the type of query executed.
- **tableMapping**: The DataTableMapping sent through an update.

See Also

[MySqlRowUpdatingEventArgs Class](#), [MySQL.Data.MySqlClient Namespace](#)

Command Property

Gets or sets the MySqlCommand to execute when performing the update.

Syntax: Visual Basic

```
Overloads Public Property Command As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand Command {get; set;}
```

See Also

[MySqlRowUpdatingEventArgs Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlCommandBuilder Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal adapter As MySqlDataAdapter, _
    ByVal lastOneWins As Boolean _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
    MySqlDataAdapter adapter,
    bool lastOneWins
);
```

See Also

[MySqlCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

MySqlCommandBuilder Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _
    ByVal lastOneWins As Boolean _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
    boollastOneWins
);
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

DataAdapter Property

Syntax: Visual Basic

```
Public Property DataAdapter As MySqlDataAdapter
```

Syntax: C#

```
public MySqlDataAdapter DataAdapter {get; set;}
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

QuotePrefix Property

Syntax: Visual Basic

```
Public Property QuotePrefix As String
```

Syntax: C#

```
public string QuotePrefix {get; set;}
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

QuoteSuffix Property

Syntax: Visual Basic

```
Public Property QuoteSuffix As String
```

Syntax: C#

```
public string QuoteSuffix {get; set;}
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommandBuilder.GetDeleteCommand Method

Syntax: Visual Basic

```
Public Function GetDeleteCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetDeleteCommand();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommandBuilder.GetInsertCommand Method

Syntax: Visual Basic

```
Public Function GetInsertCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetInsertCommand();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommandBuilder.GetUpdateCommand Method

Syntax: Visual Basic

```
Public Function GetUpdateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetUpdateCommand();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlCommandBuilder.RefreshSchema Method

Syntax: Visual Basic

```
Public Sub RefreshSchema()
```

Syntax: C#

```
public void RefreshSchema();
```

See Also

[MySqlCommandBuilder Class](#), [MySql.Data.MySqlClient Namespace](#)

4.10.1.6 MySqlConnectionException Class

The exception that is thrown when MySQL returns an error. This class cannot be inherited.

For a list of all members of this type, see [MySqlConnectionException Members](#).

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnectionException_
    Inherits SystemException
```

Syntax: C#

```
public sealed class MySqlConnectionException : SystemException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlConnectionException Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlConnectionException Members

[MySqlConnectionException overview](#)

Public Instance Properties

Data (inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the ExceptionInstance that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.
Number	Gets a number that identifies the type of error. This number corresponds to the error numbers given in Server Error Codes and Messages .
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Exception)	Gets the runtime type of the current instance.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

See Also

[MySqlException Class](#), [MySQL.Data.MySqlClient Namespace](#)

Number Property

Gets a number that identifies the type of error.

Syntax: Visual Basic

```
Public ReadOnly Property Number As Integer
```

Syntax: C#

```
public int Number {get;}
```

See Also

[MySqlException Class](#), [MySQL.Data.MySqlClient Namespace](#)

4.10.1.7 MySqlHelper Class

Helper class that makes it easier to work with the provider.

For a list of all members of this type, see [MySqlHelper Members](#) .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlHelper
```

Syntax: C#

```
public sealed class MySqlHelper
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlHelper Members](#), [MySql.Data.MySqlClient Namespace](#)

MySqlHelper Members

[MySqlHelper overview](#)

Public Static (Shared) Methods

ExecuteDataRow	Executes a single SQL statement and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteDataset	Overloaded. Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteNonQuery	Overloaded. Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.
ExecuteReader	Overloaded. Executes a single command against a MySQL database.
ExecuteScalar	Overloaded. Execute a single command against a MySQL database.
UpdateDataSet	Updates the given table with data from the given DataSet

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlHelper.ExecuteDataRow Method

Executes a single SQL statement and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Public Shared Function ExecuteDataRow( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ParamArray parms As MySqlParameter() _
```

```
) As DataRow
```

Syntax: C#

```
public static DataRow ExecuteDataRow(
    string connectionString,
    string commandText,
    params MySqlParameter[] params
);
```

Parameters

- **connectionString**: Settings to be used for the connection
- **commandText**: Command to execute
- **params**: Parameters to use for the command

Return Value

DataRow containing the first row of the resultset

See Also

[MySqlHelper Class](#), [MySQL.Data.MySqlClient Namespace](#)

ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Overload List

Executes a single SQL statement and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

- [public static DataSet ExecuteDataset\(MySqlConnection,string\);](#)

Executes a single SQL statement and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

- [public static DataSet ExecuteDataset\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- [public static DataSet ExecuteDataset\(string,string\);](#)

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- [public static DataSet ExecuteDataset\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
    ByVal connection As MySqlConnection, _
    ByVal commandText As String _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
    MySqlConnectionconnection,
    stringcommandText
);
```

Parameters

- **connection**: [MySqlConnection](#) object to use
- **commandText**: Command to execute

Return Value

DataSetcontaining the resultset

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteDataset Overload List](#)

MySqlHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. The state of the [MySqlConnection](#) object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
    ByVal connection As MySqlConnection, _
    ByVal commandText As String, _
    ParamArray commandParameters As MySqlParameter() _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
    MySqlConnectionconnection,
    stringcommandText,
    params MySqlParameter[]commandParameters
);
```

Parameters

- **connection**: [MySqlConnection](#) object to use
- **commandText**: Command to execute
- **commandParameters**: Parameters to use for the command

Return Value

DataSetcontaining the resultset

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteDataset Overload List](#)

MySqlHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
    ByVal connectionString As String, _
    ByVal commandText As String _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
    string connectionString,
    string commandText
);
```

Parameters

- **connectionString**: Settings to be used for the connection
- **commandText**: Command to execute

Return Value

DataSet containing the resultset

See Also

[MySqlHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteDataset Overload List](#)

MySqlHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ParamArray commandParameters As MySqlParameter() _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
    string connectionString,
    string commandText,
    params MySqlParameter[] commandParameters
);
```

Parameters

- **connectionString**: Settings to be used for the connection
- **commandText**: Command to execute
- **commandParameters**: Parameters to use for the command

Return Value

DataSet containing the resultset

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteDataset Overload List](#)

ExecuteNonQuery Method

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

Overload List

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

- [public static int ExecuteNonQuery\(MySqlConnection,string,params MySqlParameter\[\]\)](#);

Executes a single command against a MySQL database. A new [MySqlConnection](#) is created using the [ConnectionString](#) given.

- [public static int ExecuteNonQuery\(string,string,params MySqlParameter\[\]\)](#);

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlHelper.ExecuteNonQuery Method

Executes a single command against a MySQL database. The [MySqlConnection](#) is assumed to be open when the method is called and remains open after the method completes.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _
    ByVal connection As MySqlConnection, _
    ByVal commandText As String, _
    ParamArray commandParameters As MySqlParameter() _
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(
    MySqlConnection connection,
    string commandText,
    params MySqlParameter[] commandParameters
);
```

Parameters

- [connection](#): [MySqlConnection](#) object to use
- [commandText](#): SQL statement to be executed
- [commandParameters](#): Array of [MySqlParameter](#) objects to use with the command.

Return Value

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteNonQuery Overload List](#)

MySqlHelper.ExecuteNonQuery Method

Executes a single command against a MySQL database. A new [MySQLConnection](#) is created using the [ConnectionString](#) given.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ParamArray parms As MySqlParameter() _
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(
    string connectionString,
    string commandText,
    params MySqlParameter[] parms
);
```

Parameters

- **connectionString**: [ConnectionString](#) to use
- **commandText**: SQL statement to be executed
- **parms**: Array of [MySqlParameter](#) objects to use with the command.

Return Value

See Also

[MySqlHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteNonQuery Overload List](#)

ExecuteReader Method

Executes a single command against a MySQL database.

Overload List

Executes a single command against a MySQL database.

- [public static MySqlDataReader ExecuteReader\(string,string\);](#)

Executes a single command against a MySQL database.

- [public static MySqlDataReader ExecuteReader\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#), [MySQL.Data.MySqlClient Namespace](#)

MySqlHelper.ExecuteReader Method

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _
    ByVal connectionString As String, _
```

```
ByVal commandText As String _
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(
string connectionString,
string commandText
);
```

Parameters

- **connectionString**: Settings to use for this command
- **commandText**: Command text to use

Return Value

[MySqlDataReader](#) object ready to read the results of the command

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteReader Overload List](#)

MySqlHelper.ExecuteReader Method

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ParamArray commandParameters As MySqlParameter() _
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(
string connectionString,
string commandText,
params MySqlParameter[] commandParameters
);
```

Parameters

- **connectionString**: Settings to use for this command
- **commandText**: Command text to use
- **commandParameters**: Array of [MySqlParameter](#) objects to use with the command

Return Value

[MySqlDataReader](#) object ready to read the results of the command

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteReader Overload List](#)

ExecuteScalar Method

Execute a single command against a MySQL database.

Overload List

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(MySqlConnection,string\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(MySqlConnection,string,params MySqlParameter\[\]\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(string,string\);](#)

Execute a single command against a MySQL database.

- [public static object ExecuteScalar\(string,string,params MySqlParameter\[\]\);](#)

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

MySqlHelper.ExecuteScalar Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
    ByVal connection As MySqlConnection, _
    ByVal commandText As String _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
    MySqlConnectionconnection,
    stringcommandText
);
```

Parameters

- **connection**: [MySqlConnection](#) object to use
- **commandText**: Command text to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteScalar Overload List](#)

MySqlHelper.ExecuteScalar Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
    ByVal connection As MySqlConnection, _
```

```

    ByVal commandText As String, _
    ParamArray commandParameters As MySqlParameter() _
) As Object

```

Syntax: C#

```

public static object ExecuteScalar(
    MySqlConnection connection,
    string commandText,
    params MySqlParameter[] commandParameters
);

```

Parameters

- **connection**: [MySqlConnection](#) object to use
- **commandText**: Command text to use for the command
- **commandParameters**: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteScalar Overload List](#)

MySqlHelper.ExecuteScalar Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```

Overloads Public Shared Function ExecuteScalar( _
    ByVal connectionString As String, _
    ByVal commandText As String _
) As Object

```

Syntax: C#

```

public static object ExecuteScalar(
    string connectionString,
    string commandText
);

```

Parameters

- **connectionString**: Settings to use for the update
- **commandText**: Command text to use for the update

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteScalar Overload List](#)

MySqlHelper.ExecuteScalar Method

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ParamArray commandParameters As MySqlParameter() _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
    string connectionString,
    string commandText,
    params MySqlParameter[] commandParameters
);
```

Parameters

- [connectionString](#): Settings to use for the command
- [commandText](#): Command text to use for the command
- [commandParameters](#): Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

[MySqlHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteScalar Overload List](#)

MySqlHelper.UpdateDataSet Method

Updates the given table with data from the given DataSet

Syntax: Visual Basic

```
Public Shared Sub UpdateDataSet( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ByVal ds As DataSet, _
    ByVal tablename As String _
)
```

Syntax: C#

```
public static void UpdateDataSet(
    string connectionString,
    string commandText,
    DataSet ds,
    string tablename
);
```

Parameters

- [connectionString](#): Settings to use for the update
- [commandText](#): Command text to use for the update
- [ds](#): DataSet containing the new data to use in the update
- [tablename](#): Tablename in the data set to update

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

4.10.1.8 MySQLErrorCode Enumeration

Syntax: Visual Basic

```
Public Enum MySQLErrorCode
```

Syntax: C#

```
public enum MySQLErrorCode
```

Members

Member Name	Description
PacketTooLarge	
PasswordNotAllowed	
DuplicateKeyEntry	
HostNotPrivileged	
PasswordNoMatch	
AnonymousUser	
DuplicateKey	
KeyNotFound	
DuplicateKeyName	

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

4.10.2 MySql.Data.Types Namespace

[Namespace hierarchy](#)

Classes

Class	Description
MySQLConversionException	Summary description for MySQLConversionException.
MySQLDateTime	Summary description for MySQLDateTime.
MySQLValue	

4.10.2.1 MySql.Data.TypesHierarchy

See Also

[MySql.Data.Types Namespace](#)

4.10.2.2 MySqlConnectionException Class

Summary description for MySqlConnectionException.

For a list of all members of this type, see [MySqlConnectionException Members](#) .

Syntax: Visual Basic

```
Public Class MySqlConnectionException_  
    Inherits ApplicationException
```

Syntax: C#

```
public class MySqlConnectionException : ApplicationException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.Types](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlConnectionException Members](#), [MySql.Data.Types Namespace](#)

MySqlConnectionException Members

[MySqlConnectionException overview](#)

Public Instance Constructors

MySqlConnectionException Constructor	Ctor

Public Instance Properties

Data (inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the Exceptioninstance that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Exception)	Gets the runtime type of the current instance.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets HRESULT , a coded numeric value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySQLConversionException Class](#), [MySql.Data.Types Namespace](#)

MySQLConversionException Constructor

Syntax: Visual Basic

```
Public Sub New( _
    ByVal msg As String _
)
```

Syntax: C#

```
public MySQLConversionException(
    string msg
);
```

See Also

[MySQLConversionException Class](#), [MySql.Data.Types Namespace](#)

4.10.2.3 MySQLDateTime Class

Summary description for MySQLDateTime.

For a list of all members of this type, see [MySQLDateTime Members](#) .

Syntax: Visual Basic

```
Public Class MySqlDateTime_  
    Inherits MySqlValue_  
    Implements IConvertible, IComparable
```

Syntax: C#

```
public class MySqlDateTime : MySqlValue, IConvertible, IComparable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.Types](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySqlDateTime Members](#), [MySql.Data.Types Namespace](#)

MySqlDateTime Members

[MySqlDateTime overview](#)

Public Static (Shared) Type Conversions

Explicit MySqlDateTime to DateTime Conversion	

Public Instance Properties

Day	Returns the day portion of this datetime
Hour	Returns the hour portion of this datetime
IsNull (inherited from MySqlValue)	
IsValidDateTime	Indicates if this object contains a value that can be represented as a DateTime
Minute	Returns the minute portion of this datetime
Month	Returns the month portion of this datetime
Second	Returns the second portion of this datetime
Millisecond	Returns the millisecond portion of this datetime
ValueAsObject (inherited from MySqlValue)	Returns the value of this field as an object
Year	Returns the year portion of this datetime

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetDateTime	Returns this value as a DateTime
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.

GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Returns a MySQL-specific string representation of this value

Protected Instance Fields

classType (inherited from MySqlValue)	The system type represented by this value
dbType (inherited from MySqlValue)	The generic dbtype of this value
isNull (inherited from MySqlValue)	Is this value null
mysqlDbType (inherited from MySqlValue)	The specific MySQL db type
mysqlTypeName (inherited from MySqlValue)	The MySQL-specific typename of this value
objectValue (inherited from MySqlValue)	

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

MySqlDateTime Explicit MySqlDateTime to DateTime Conversion

Syntax: Visual Basic

```
MySqlDateTime.op_Explicit(val)
```

Syntax: C#

```
public static explicit operator DateTime(
    MySqlDateTime val
);
```

Parameters

- **val**:

Return Value

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

Day Property

Returns the day portion of this datetime

Syntax: Visual Basic

```
Public Property Day As Integer
```

Syntax: C#


```
public int Day {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySQL.Data.Types Namespace](#)

Hour Property

Returns the hour portion of this datetime

Syntax: Visual Basic

```
Public Property Hour As Integer
```

Syntax: C#

```
public int Hour {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySQL.Data.Types Namespace](#)

IsNull Property

Syntax: Visual Basic

```
Public Property IsNull As Boolean
```

Syntax: C#

```
public bool IsNull {get; set;}
```

See Also

[MySQLValue Class](#), [MySQL.Data.Types Namespace](#)

MySQLValue Class

For a list of all members of this type, see [MySQLValue Members](#) .

Syntax: Visual Basic

```
MustInherit Public Class MySQLValue
```

Syntax: C#

```
public abstract class MySQLValue
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.Types](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLValue Members](#), [MySQL.Data.Types Namespace](#)

MySQLValue Members

[MySQLValue overview](#)
Protected Static (Shared) Fields

numberFormat	

Public Instance Constructors

MySQLValue Constructor	Initializes a new instance of the MySQLValue class.

Public Instance Properties

IsNull	
ValueAsObject	Returns the value of this field as an object

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Returns a string representation of this value

Protected Instance Fields

classType	The system type represented by this value
dbType	The generic dbtype of this value
isNull	Is this value null
mysqlDbType	The specific MySQL db type
mysqlTypeName	The MySQL-specific typename of this value
objectValue	

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also
[MySQLValue Class](#), [MySQL.Data.Types Namespace](#)

MySQLValue.numberFormat Field

Syntax: Visual Basic

```
Protected Shared numberFormat As NumberFormatInfo
```

Syntax: C#

```
protected static NumberFormatInfo numberFormat;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

MySqlValue Constructor

Initializes a new instance of the [MySqlValue](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlValue();
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

ValueAsObject Property

Returns the value of this field as an object

Syntax: Visual Basic

```
Public ReadOnly Property ValueAsObject As Object
```

Syntax: C#

```
public object ValueAsObject {get;}
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

MySqlValue.ToString Method

Returns a string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

MySqlValue.classType Field

The system type represented by this value

Syntax: Visual Basic

```
Protected classType As Type
```

Syntax: C#

```
protected Type classType;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

MySqlValue.dbType Field

The generic dbtype of this value

Syntax: Visual Basic

```
Protected dbType As DbType
```

Syntax: C#

```
protected DbType dbType;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

MySqlValue.mysqlDbType Field

The specific MySQL db type

Syntax: Visual Basic

```
Protected mysqlDbType As MySqlDbType
```

Syntax: C#

```
protected MySqlDbType mysqlDbType;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

MySqlValue.mysqlTypeName Field

The MySQL-specific typename of this value

Syntax: Visual Basic

```
Protected mysqlTypeName As String
```

Syntax: C#

```
protected string mysqlTypeName;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

MySqlValue.objectValue Field

Syntax: Visual Basic

```
Protected objectValue As Object
```

Syntax: C#

```
protected object objectValue;
```

See Also

[MySqlValue Class](#), [MySQL.Data.Types Namespace](#)

IsValidDateTime Property

Indicates if this object contains a value that can be represented as a DateTime

Syntax: Visual Basic

```
Public ReadOnly Property IsValidDateTime As Boolean
```

Syntax: C#

```
public bool IsValidDateTime {get;}
```

See Also

[MySqlDateTime Class](#), [MySQL.Data.Types Namespace](#)

Millisecond Property

Returns the millisecond portion of this datetime

Syntax: Visual Basic

```
Public Property Millisecond As Integer
```

Syntax: C#

```
public int Millisecond {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySQL.Data.Types Namespace](#)

Minute Property

Returns the minute portion of this datetime

Syntax: Visual Basic

```
Public Property Minute As Integer
```

Syntax: C#

```
public int Minute {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

Month Property

Returns the month portion of this datetime

Syntax: Visual Basic

```
Public Property Month As Integer
```

Syntax: C#

```
public int Month {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

Second Property

Returns the second portion of this datetime

Syntax: Visual Basic

```
Public Property Second As Integer
```

Syntax: C#

```
public int Second {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

Year Property

Returns the year portion of this datetime

Syntax: Visual Basic

```
Public Property Year As Integer
```

Syntax: C#

```
public int Year {get; set;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

MySqlDateTime.GetDateTime Method

Returns this value as a DateTime

Syntax: Visual Basic

```
Public Function GetDateTime() As Date
```

Syntax: C#

```
public DateTime GetDateTime();
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

MySqlDateTime.ToString Method

Returns a MySQL-specific string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

Client Class

For a list of all members of this type, see [Client Members](#) .

Syntax: Visual Basic

```
Public MustInherit Class Client
```

Syntax: C#

```
public abstract class Client
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient.Memcached](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[Client Members](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Client Members

[Client overview](#)

Public Static Methods

GetInstance	Factory method for creating instances of Client that implement a connection with the requested features. The connection object returned must be explicitly opened see method Client.Open

Protected Instance Properties

server	The server DNS or IP address used by the connection.
port	The port used by the connection.
stream	The network stream used by the connection.

Public Instance Methods

Open	Opens the client connection.
Close	Closes the client connection.
Add	Adds a new key/value pair with the given TimeSpan expiration.
Append	Appends the data to the existing data for the associated key..
Cas	Executes the Check-and-set Memcached operation.
Decrement>	Decrements the value associated with a key by the given amount.
Delete	Removes they pair key/value given the specified key.
FlushAll	Removes all entries from the storage, effectively invalidating the whole cache.
Get	Get the key/value pair associated with a given key.
Increment	Increments the value associated with a key by the given amount.
Prepend	Prepends the data to the existing data for the associated key.
Replace	Replaces the value associated with the given key with another value.
Set	Set the value of a given key.

Public Static Methods

GetInstance	Factory method for creating instances of Client that implement a connection with the requested features. The connection object returned must be explicitly opened see method Open.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

server Property

The server DNS or IP address used by the connection.

Syntax: Visual Basic

```
Protected server As String
```

Syntax: C#


```
protected string server
```

Property Value

The server DNS or IP address used by the connection.

Remarks

None

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Port Property

The port used by the connection.

Syntax: Visual Basic

```
Protected port As UInteger
```

Syntax: C#

```
protected uint port
```

Property Value

The TCP port where the Innodb Memcached plugin will be listening for connections.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

stream

The network stream used by the connection.

Syntax: Visual Basic

```
Protected stream As Stream
```

Syntax: C#

```
protected Stream stream;
```

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Client.Open Method

Syntax: Visual Basic

Public Overridable Sub Open

Syntax: C#

```
public virtual void Open();
```

Opens the client connection.

Remarks

A client object can be opened and closed many times during its life cycle.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Client.Close Method

Syntax: Visual Basic

Public Overridable Sub Close

Syntax: C#

```
public virtual void Close();
```

Closes the client connection.

Remarks

A client object can be opened and closed many times during its life cycle.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Client.Add Method

Syntax: Visual Basic

```
Public MustOverride Sub Add (
    key As String,
    data As Object,
    expiration As TimeSpan
)
```

Syntax: C#

```
public abstract void Add(
    string key,
    Object data,
    TimeSpan expiration
);
```

Adds a new key/value pair with the given TimeSpan expiration.

Remarks

None.

See Also

[Client Class, MySQL.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key for identifying the entry.
- [data](#), The data to associate with the key.
- [expiration](#), The interval of timespan, use [TimeSpan.Zero](#) for no expiration.

Client.Append Method

Syntax: Visual Basic

```
Public MustOverride Sub Append (
    key As String,
    data As Object
)
```

Syntax: C#

```
public abstract void Append(
    string key,
    Object data
);
```

Appends the data to the existing data for the associated key.

Remarks

None.

See Also

[Client Class, MySQL.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key for identifying the entry.
- [data](#), The data to append with the data associated with the key.

Client.Cas Method

Syntax: Visual Basic

```
Public MustOverride Sub Cas (
    key As String,
    data As Object,
    expiration As TimeSpan,
    casUnique As ULong
)
```

Syntax: C#

```
public abstract void Cas(
    string key,
    Object data,
    TimeSpan expiration,
    ulong casUnique
);
```

Executes the Check-and-set Memcached operation.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key for identifying the entry.
- [data](#), The data to use in the CAS.
- [expiration](#), The interval of timespan, use [TimeSpan.Zero](#) for no expiration.
- [casUnique](#), The CAS unique value to use.

Client.Decrement Method

Syntax: Visual Basic

```
Public MustOverride Sub Decrement (
    key As String,
    amount As Integer
)
```

Syntax: C#

```
public abstract void Decrement(
    string key,
    int amount
);
```

Decrements the value associated with a key by the given amount.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key associated with the value to decrement.
- [amount](#), The amount to decrement the value.

Client.Delete Method

Syntax: Visual Basic

```
Public MustOverride Sub Delete (
    key As String
)
```

Syntax: C#

```
public abstract void Delete(
    string key
```

```
);
```

Removes the key/value pair given the specified key.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key associated with the value to delete.

Client.FlushAll Method

Syntax: Visual Basic

```
Public MustOverride Sub FlushAll (
    delay As TimeSpan
)
```

Syntax: C#

```
public abstract void FlushAll(
    TimeSpan delay
);
```

Removes all entries from the storage, effectively invalidating the whole cache.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [TimeSpan](#), The interval after which the cache will be cleaned. Can be TimeSpan.Zero for immediately.

Client.Get Method

Syntax: Visual Basic

```
Public MustOverride Function Get (
    key As String
) As KeyValuePair(Of String, Object)
```

Syntax: C#

```
public abstract KeyValuePair<string, Object> Get(
    string key
);
```

Get the key/value pair associated with a given key.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key for which to return the key/value.

Return Value

The key/value associated with the key or a MemcachedException if it does not exists.

Client.Increment Method

Syntax: Visual Basic

```
Public MustOverride Sub Increment (
    key As String,
    amount As Integer
)
```

Syntax: C#

```
public abstract void Increment(
    string key,
    int amount
);
```

Increments the value associated with a key by the given amount.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key associated with the value to increment.
- [amount](#), The amount to increment the value.

Client.Prepend Method

Syntax: Visual Basic

```
Public MustOverride Sub Prepend (
    key As String,
    data As Object
)
```

Syntax: C#

```
public abstract void Prepend(
    string key,
    Object data
);
```

Prepends the data to the existing data for the associated key.

Remarks

None.

See Also

[Client Class](#), [MySQL.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key for identifying the entry.
- [data](#), The data to append with the data associated with the key.

Client.Replace Method

Syntax: Visual Basic

```
Public MustOverride Sub Replace (
    key As String,
    data As Object,
    expiration As TimeSpan
)
```

Syntax: C#

```
public abstract void Replace(
    string key,
    Object data,
    TimeSpan expiration
);
```

Replaces the value associated with the given key with another value.

Remarks

None.

See Also

[Client Class](#), [MySQL.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key for identifying the entry.
- [data](#), The data to replace the value associated with the key.
- [expiration](#), The interval of timespan, use [TimeSpan.Zero](#) for no expiration.

Client.Set Method

Syntax: Visual Basic

```
Public MustOverride Sub Set (
    key As String,
    data As Object,
    expiration As TimeSpan
)
```

Syntax: C#

```
public abstract void Set(
    string key,
    Object data,
    TimeSpan expiration
);
```

Set the value of a given key.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [key](#), The key for identifying the entry.
- [data](#), The data to associate with the given key.
- [expiration](#), The interval of timespan, use [TimeSpan.Zero](#) for no expiration.

Client.GetInstance Method

Syntax: Visual Basic

```
Public Shared Function GetInstance (
    server As String,
    port As UInteger,
    flags As MemcachedFlags
) As Client
```

Syntax: C#

```
public static Client GetInstance(
    string server,
    uint port,
    MemcachedFlags flags
);
```

Factory method for creating instances of Client that implement a connection with the requested features. The connection object returned must be explicitly opened see method `Open()`.

Remarks

None.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

Parameters

- [server](#), The Memcached server DNS or IP address.
- [port](#), The port for the Memcached server.
- [flags](#), A set of flags indicating characteristics requested.

Return Value

An instance of a client connection ready to be used.

BinaryClient Class

For a list of all members of this type, see [BinaryClient Members](#) .

Syntax: Visual Basic

```
Public Class BinaryClient
    Inherits Client
```

Syntax: C#

```
public class BinaryClient : Client
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySQL.Data.MySqlClient.Memcached](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[BinaryClient Members](#), [MySQL.Data.MySqlClient.Memcached Namespace](#)

BinaryClient Members

[BinaryClient overview](#)

Public Static Methods

GetInstance	Inherited from Client

Protected Instance Properties

server	Inherited from Client.
port	Inherited from Client.
stream	Inherited from Client.

Public Instance Methods

Open	Inherited from Client.
Close	Inherited from Client.
Add	Inherited from Client.
Append	Inherited from Client.
Cas	Inherited from Client.
Decrement>	Inherited from Client.
Delete	Inherited from Client.
FlushAll	Inherited from Client.
Get	Inherited from Client.
Increment	Inherited from Client.

Prepend	Inherited from Client.
Replace	Inherited from Client.
Set	Inherited from Client.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

TextClient Class

For a list of all members of this type, see [TextClient Members](#) .

Syntax: Visual Basic

```
Public Class TextClient
    Inherits Client
```

Syntax: C#

```
public class TextClient : Client
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient.Memcached](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[TextClient Members](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

TextClient Members

[TextClient overview](#)

Public Static Methods

GetInstance	Inherited from Client

Protected Instance Properties

server	Inherited from Client.
port	Inherited from Client.
stream	Inherited from Client.

Public Instance Methods

Open	Inherited from Client.
Close	Inherited from Client.
Add	Inherited from Client.
Append	Inherited from Client.

Cas	Inherited from Client.
Decrement>	Inherited from Client.
Delete	Inherited from Client.
FlushAll	Inherited from Client.
Get	Inherited from Client.
Increment	Inherited from Client.
Prepend	Inherited from Client.
Replace	Inherited from Client.
Set	Inherited from Client.

See Also

[Client Class](#), [MySql.Data.MySqlClient.Memcached Namespace](#)

MySql.Data.MySqlClient.Replication Namespace

This section of the manual contains the API reference for Replication and Load balancing components.

Classes

Class	Description
ReplicationManager	Manager for Replication and Load Balancing features.
ReplicationRoundRobinServerGroup	Class that implements Round Robing Load Balancing technique.
ReplicationServer	Represents a server in Replication environment that contains information about
ReplicationServerGroup	Abstract class used to implement a custom load balancing plugin

ReplicationManager Class

Members

Namespace: [MySql.Data.MySqlClient.Replication](#)

Assembly: MySql.Data (in [MySql.Data.dll](#))

Syntax C#

```
public static class ReplicationManager
```

Syntax Visual Basic

```
Public NotInheritable Class ReplicationManager
```

Syntax Visual C++

```
public ref class ReplicationManager abstract sealed
```

See Also

ReplicationManager Members [MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationManager Members

The [ReplicationManager](#) type exposes the following members.

Public Static Methods

Name	Description
AddGroup(String, Int32)	Adds a Default Server Group to the list
AddGroup(String, String, Int32)	Adds a Server Group to the list
GetGroup	Gets a Server Group by name
GetNewConnection	Assigns a new server driver to the connection object
GetServer	Gets the next server from a replication group
IsReplicationGroup	Validates if the replication group name exists

Public Static Properties

Name	Description
Groups	Returns Replication Server Group List

See Also

[ReplicationManager Class \[MySql.Data.MySqlClient.Replication Namespace\]\(#\)](#)

AddGroup Method

Overload List

- [AddGroup\(String, Int32\)](#)
- [AddGroup\(String, String, Int32\)](#)

See Also

[ReplicationManager Class \[ReplicationManager Members \\[MySql.Data.MySqlClient.Replication Namespace\\]\\(#\\)\]\(#\)](#)

ReplicationManager.AddGroup Method (String, Int32)

Adds a Default Server Group to the list

Syntax C#

```
public static ReplicationServerGroup AddGroup(
    string name,
    int retryTime
)
```

Syntax Visual Basic

```
Public Shared Function AddGroup ( _
    name As String, _
    retryTime As Integer _
) As ReplicationServerGroup
```

Syntax Visual C++

```
public:
static ReplicationServerGroup^ AddGroup(
    String^ name,
    int retryTime
)
```

Parameters

name

Type: System.String

Group name

retryTime

Type: System.Int32

Time between reconnections for failed servers

Return Value

A [ReplicationServerGroup](#) object

See Also

[ReplicationManager Class AddGroup Overload MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationManager.AddGroup Method (String, String, Int32)

Adds a Server Group to the list

Syntax C#

```
public static ReplicationServerGroup AddGroup(
    string name,
    string groupType,
    int retryTime
)
```

Syntax Visual Basic

```
Public Shared Function AddGroup ( _
    name As String, _
    groupType As String, _
    retryTime As Integer _
) As ReplicationServerGroup
```

Syntax Visual C++

```
public:
static ReplicationServerGroup^ AddGroup(
    String^ name,
    String^ groupType,
    int retryTime
)
```

Parameters

name

Type: System.String

Group name

groupType

Type: System.String

ServerGroup type reference

retryTime

Type: System.Int32

Time between reconnections for failed servers

Return Value

A [ReplicationServerGroup](#) object

See Also

[ReplicationManager Class AddGroup Overload MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationManager.GetGroup Method

Gets a Server Group by name

Syntax C#

```
public static ReplicationServerGroup GetGroup(
    string groupName
)
```

Syntax Visual Basic

```
Public Shared Function GetGroup ( _
    groupName As String _
) As ReplicationServerGroup
```

Syntax Visual C++

```
public:
    static ReplicationServerGroup^ GetGroup(
        String^ groupName
    )
```

Parameters

groupName

Type: System.String

Group name

Return Value

A [ReplicationServerGroup](#) object

See Also

[ReplicationManager Class MySQL.Data.MySqlClient.Replication Namespace](#)

ReplicationManager.GetNewConnection Method

Assigns a new server driver to the connection object

Syntax C#

```
public static void GetNewConnection(
    string groupName,
    bool master,
    MySqlConnection connection
)
```

Syntax Visual Basic

```
Public Shared Sub GetNewConnection ( _
    groupName As String, _
    master As Boolean, _
    connection As MySqlConnection _
)
```

Syntax Visual C++

```
public:
static void GetNewConnection(
    String^ groupName,
    bool master,
    MySqlConnection^ connection
)
```

Parameters

groupName

Type: System.String

Group name

master

Type: System.Boolean

True if the server connection to assign must be a master

connection

Type: MySQL.Data.MySqlClient.MySqlConnection

MySqlConnection object where the new driver will be assigned

See Also

[ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationManager.GetServer Method

Gets the next server from a replication group

Syntax C#

```
public static ReplicationServer GetServer(
    string groupName,
    bool isMaster
)
```

Syntax Visual Basic

```
Public Shared Function GetServer ( _
    groupName As String, _
    isMaster As Boolean _
) As ReplicationServer
```

Syntax Visual C++

```
public:
static ReplicationServer^ GetServer(
    String^ groupName,
    bool isMaster
)
```

Parameters

groupName

Type: System.String

Group name

isMaster

Type: System.Boolean

True if the server to return must be a master

Return Value

A [ReplicationServer](#) object

See Also

[ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationManager.IsReplicationGroup Method

Validates if the replication group name exists.

Syntax C#

```
public static bool IsReplicationGroup(
    string groupName
```


)

Syntax Visual Basic

```
Public Shared Function IsReplicationGroup ( _
    groupName As String _
) As Boolean
```

Syntax Visual C++

```
public:
static bool IsReplicationGroup(
    String^ groupName
)
```

Parameters

groupName

Type: System.String

Group name

Return Value

True if replication group name is found, otherwise false

See Also

[ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationManager.Groups Property

Returns Replication Server Group List

Syntax C#

```
public static IList<ReplicationServerGroup> Groups { get; private set; }
```

Syntax Visual Basic

```
Public Shared Property Groups As IList(Of ReplicationServerGroup)
    Get
    Private Set
```

Syntax Visual C++

```
public:
static property IList<ReplicationServerGroup^> Groups {
    IList<ReplicationServerGroup^> get ();
    private: void set (IList<ReplicationServerGroup^> value);
}
```

See Also

[ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationRoundRobinServerGroup Class

Members

Namespace: [MySql.Data.MySqlClient.Replication](#)

Assembly: MySql.Data (in MySql.Data.dll)

Syntax C#

```
public class ReplicationRoundRobinServerGroup : ReplicationServerGroup
```

Syntax Visual Basic

```
Public Class ReplicationRoundRobinServerGroup _  
    Inherits ReplicationServerGroup
```

Syntax Visual C++

```
public ref class ReplicationRoundRobinServerGroup : public ReplicationServerGroup
```

See Also

[ReplicationRoundRobinServerGroup Members](#) [MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationRoundRobinServerGroup Members

The [ReplicationRoundRobinServerGroup](#) type exposes the following members.

Public Instance Constructors

Name	Description
ReplicationRoundRobinServerGroup	Initializes a new instance of the ReplicationRoundRobinServerGroup class

Public Instance Methods

Name	Description
AddServer	Adds a server into the group (Inherited from ReplicationServerGroup)
Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetServer(Boolean)	Gets an available server based on Round Robin load balancing (Overrides ReplicationServerGroup.GetServer(Boolean) .)
GetServer(String)	Gets a server by name (Inherited from ReplicationServerGroup .)
GetType	Gets the Type of the current instance. (Inherited from Object.)
RemoveServer	Removes a server from group (Inherited from ReplicationServerGroup .)

Name	Description
ToString	Returns a string that represents the current object. (Inherited from Object.)

Public Instance Properties

Name	Description
Name	(Inherited from ReplicationServerGroup .)
RetryTime	(Inherited from ReplicationServerGroup .)
Servers	(Inherited from ReplicationServerGroup .)

See Also

[ReplicationRoundRobinServerGroup Class](#) [MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationRoundRobinServerGroup Constructor

Initializes a new instance of the [ReplicationRoundRobinServerGroup](#) class

Syntax C#

```
public ReplicationRoundRobinServerGroup(
    string name,
    int retryTime
)
```

Syntax Visual Basic

```
Public Sub New ( _
    name As String, _
    retryTime As Integer _
)
```

Syntax Visual C++

```
public:
    ReplicationRoundRobinServerGroup(
        String^ name,
        int retryTime
    )
```

Parameters

name

Type: System.String

Server name

retryTime

Type: System.Int32

Retry time between connections to failed connections

See Also

[ReplicationRoundRobinServerGroup Class MySql.Data.MySqlClient.Replication Namespace](#)

GetServer Method

Overload List

- [GetServer\(Boolean\)](#)
- [GetServer\(String\)](#)

See Also

[ReplicationRoundRobinServerGroup Class ReplicationRoundRobinServerGroup Members MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationRoundRobinServerGroup.GetServer Method (Boolean)

Gets an available server based on Round Robin load balancing (Overrides [ReplicationServerGroup.GetServer\(Boolean\)](#).)

Syntax C#

```
public override ReplicationServer GetServer(
    bool isMaster
)
```

Syntax Visual Basic

```
Public Overrides Function GetServer ( _
    isMaster As Boolean _
) As ReplicationServer
```

Syntax Visual C++

```
public:
virtual ReplicationServer^ GetServer(
    bool isMaster
) override
```

Parameters

isMaster

Type: System.Boolean

True if the server to return must be a master

Return Value

A [ReplicationServer](#) object

See Also

[ReplicationRoundRobinServerGroup Class GetServer Overload MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServer Class

[Members](#)

Namespace: [MySql.Data.MySqlClient.Replication](#)

Assembly: MySql.Data (in [MySql.Data.dll](#))

Syntax C#

```
public class ReplicationServer
```

Syntax Visual Basic

```
Public Class ReplicationServer
```

Syntax Visual C++

```
public ref class ReplicationServer
```

See Also

[ReplicationServer Members](#) [MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServer Members

The [ReplicationServer](#) type exposes the following members.

Public Instance Constructors

Name	Description
ReplicationServer	Initializes a new instance of the ReplicationServer class

Public Instance Methods

Name	Description
Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetType	Gets the Type of the current instance. (Inherited from Object.)
ToString	Returns a string that represents the current object. (Inherited from Object.)

Public Instance Properties

Name	Description
ConnectionString	Connection string used to connect to the server
IsAvailable	Defines if the server is available to be considered in load balancing
IsMaster	Defines if the server is master (True) or slave
Name	Server name

See Also

[ReplicationServer Class MySQL.Data.MySqlClient.Replication Namespace](#)

ReplicationServer Constructor

Initializes a new instance of the [ReplicationServer](#) class

Syntax C#

```
public ReplicationServer(
    string name,
    bool isMaster,
    string connectionString
)
```

Syntax Visual Basic

```
Public Sub New ( _
    name As String, _
    isMaster As Boolean, _
    connectionString As String _
)
```

Syntax Visual C++

```
public:
    ReplicationServer(
        String^ name,
        bool isMaster,
        String^ connectionString
    )
```

Parameters

name

Type: System.String

Server name

isMaster

Type: System.Boolean

Defines if the server is master (True) or slave

connectionString

Type: System.String

Connection string used to connect to the server

See Also

[ReplicationServer Class MySQL.Data.MySqlClient.Replication Namespace](#)

ReplicationServer.ConnectionString Property

Connection string used to connect to the server

Syntax C#

```
public string ConnectionString { get; private set; }
```

Syntax Visual Basic

```
Public Property ConnectionString As String
    Get
    Private Set
```

Syntax Visual C++

```
public:
property String^ ConnectionString {
    String^ get ();
    private: void set (String^ value);
}
```

See Also

[ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServer.IsAvailable Property

Defines if the server is available to be considered in load balancing

Syntax C#

```
public bool IsAvailable { get; set; }
```

Syntax Visual Basic

```
Public Property IsAvailable As Boolean
    Get
    Set
```

Syntax Visual C++

```
public:
property bool IsAvailable {
    bool get ();
    void set (bool value);
}
```

See Also

[ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServer.IsMaster Property

Defines if the server is master (True) or slave

Syntax C#

```
public bool IsMaster { get; private set; }
```

Syntax Visual Basic

```
Public Property IsMaster As Boolean
    Get
    Private Set
```

Syntax Visual C++

```
public:
property bool IsMaster {
    bool get ();
    private: void set (bool value);
}
```

See Also

[ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServer.Name Property

Server name

Syntax C#

```
public string Name { get; private set; }
```

Syntax Visual Basic

```
Public Property Name As String
    Get
    Private Set
```

Syntax Visual C++

```
public:
property String^ Name {
    String^ get ();
    private: void set (String^ value);
}
```

See Also

[ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup Class

Members

Base class used to implement load balancing features

Namespace: [MySql.Data.MySqlClient.Replication](#)

Assembly: MySql.Data (in MySql.Data.dll)

Syntax C#


```
public abstract class ReplicationServerGroup
```

Syntax Visual Basic

```
Public MustInherit Class ReplicationServerGroup
```

Syntax Visual C++

```
public ref class ReplicationServerGroup abstract
```

See Also

[ReplicationServerGroup Members MySQL.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup Members

The [ReplicationServerGroup](#) type exposes the following members.

Public Instance Constructors

Name	Description
ReplicationServerGroup	Initializes a new instance of the ReplicationServerGroup class

Public Instance Methods

Name	Description
AddServer	Adds a server into the group
Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetServer(Boolean)	Must be implemented. Defines the next server for a custom load balancing implementation.
GetServer(String)	Gets a server by name.
GetType	Gets the Type of the current instance. (Inherited from Object.)
RemoveServer	Removes a server from group.
ToString	Returns a string that represents the current object. (Inherited from Object.)

Public Instance Properties

Name	Description
Name	Group name
RetryTime	Retry time between connections to failed servers
Servers	Servers list in the group

See Also

[ReplicationServerGroup Class MySQL.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup Constructor

Initializes a new instance of the [ReplicationServerGroup](#) class

Syntax C#

```
public ReplicationServerGroup(
    string name,
    int retryTime
)
```

Syntax Visual Basic

```
Public Sub New ( _
    name As String, _
    retryTime As Integer _
)
```

Syntax Visual C++

```
public:
    ReplicationServerGroup(
        String^ name,
        int retryTime
    )
```

Parameters

name

Type: System.String

Group name

retryTime

Type: System.Int32

Retry time between connections to failed servers

See Also

[ReplicationServerGroup Class](#) [MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.AddServer Method

Adds a server into the group.

Syntax C#

```
public ReplicationServer AddServer(
    string name,
    bool isMaster,
    string connectionString
)
```

Syntax Visual Basic

```
Public Function AddServer ( _
    name As String, _
    isMaster As Boolean, _
    connectionString As String _
) As ReplicationServer
```

Syntax Visual C++

```
public:
ReplicationServer^ AddServer(
    String^ name,
    bool isMaster,
    String^ connectionString
)
```

Parameters

name

Type: System.String

Server name

isMaster

Type: System.Boolean

True if the server to add is master, False for slave server

connectionString

Type: System.String

Connection string used by this server

See Also

[ReplicationServerGroup Class MySql.Data.MySqlClient.Replication Namespace](#)

GetServer Method

Overload List

- [GetServer\(Boolean\)](#)
- [GetServer\(String\)](#)

See Also

[ReplicationServerGroup Class ReplicationServerGroup Members MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.GetServer Method (Boolean)

Must be implemented. Defines the next server for a custom load balancing implementation.

Syntax C#

```
public abstract ReplicationServer GetServer(
    bool isMaster
```

```
)
```

Syntax Visual Basic

```
Public MustOverride Function GetServer ( _  
    isMaster As Boolean _  
) As ReplicationServer
```

Syntax Visual C++

```
public:  
virtual ReplicationServer^ GetServer(  
    bool isMaster  
) abstract
```

Parameters

isMaster

Type: System.Boolean

Defines if the server to return is a master or any

Return Value

A [ReplicationServer](#) object

See Also

[ReplicationServerGroup Class GetServer Overload MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.GetServer Method (String)

Gets a server by name.

Syntax C#

```
public ReplicationServer GetServer(  
    string name  
)
```

Syntax Visual Basic

```
Public Function GetServer ( _  
    name As String _  
) As ReplicationServer
```

Syntax Visual C++

```
public:  
ReplicationServer^ GetServer(  
    String^ name  
)
```

Parameters

name

Type: System.String

Server name

Return Value

A [ReplicationServer](#) object

See Also

[ReplicationServerGroup Class GetServer Overload MySQL.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.RemoveServer Method

Removes a server from group.

Syntax C#

```
public void RemoveServer(
    string name
)
```

Syntax Visual Basic

```
Public Sub RemoveServer ( _
    name As String _
)
```

Syntax Visual C++

```
public:
void RemoveServer(
    String^ name
)
```

Parameters

name

Type: System.String

Server name

See Also

[ReplicationServerGroup Class MySQL.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.Name Property

Group name.

Syntax C#

```
public string Name { get; private set; }
```

Syntax Visual Basic

```
Public Property Name As String
    Get
    Private Set
```

Syntax Visual C++

```
public:
property String^ Name {
    String^ get ();
    private: void set (String^ value);
}
```

See Also

[ReplicationServerGroup Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.RetryTime Property

Retry time between connections to failed servers.

Syntax C#

```
public int RetryTime { get; private set; }
```

Syntax Visual Basic

```
Public Property RetryTime As Integer
    Get
    Private Set
```

Syntax Visual C++

```
public:
property int RetryTime {
    int get ();
    private: void set (int value);
}
```

See Also

[ReplicationServerGroup Class MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.Servers Property

Servers list in the group.

Syntax C#

```
public IList<ReplicationServer> Servers { get; private set; }
```

Syntax Visual Basic

```
Public Property Servers As IList(Of ReplicationServer)
    Get
    Private Set
```

Syntax Visual C++

```
public:
property IList<ReplicationServer^>^ Servers {
    IList<ReplicationServer^>^ get ();
    private: void set (IList<ReplicationServer^>^ value);
}
```

See Also

[ReplicationServerGroup Class](#) [MySQL.Data.MySqlClient.Replication Namespace](#)

4.11 Connector/Net Support

The developers of Connector/Net greatly value the input of our users in the software development process. If you find Connector/Net lacking some feature important to you, or if you discover a bug and need to file a bug report, please use the instructions in [How to Report Bugs or Problems](#).

4.11.1 Connector/Net Community Support

- Community support for Connector/Net can be found through the forums at <http://forums.mysql.com>.
- Community support for Connector/Net can also be found through the mailing lists at <http://lists.mysql.com>.
- Paid support is available from Oracle. Additional information is available at <http://dev.mysql.com/support/>.

4.11.2 How to Report Connector/Net Problems or Bugs

If you encounter difficulties or problems with Connector/Net, contact the Connector/Net community, as explained in [Section 4.11.1, “Connector/Net Community Support”](#).

First try to execute the same SQL statements and commands from the [mysql](#) client program or from [admindemo](#). This helps you determine whether the error is in Connector/Net or MySQL.

If reporting a problem, ideally include the following information with the email:

- Operating system and version.
- Connector/Net version.
- MySQL server version.
- Copies of error messages or other unexpected output.
- Simple reproducible sample.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through <http://bugs.mysql.com/>.

Chapter 5 MySQL Connector/Python Developer Guide

Table of Contents

5.1 Introduction to MySQL Connector/Python	455
5.2 Guidelines for Python Developers	456
5.3 Connector/Python Versions	457
5.4 Connector/Python Installation	458
5.4.1 Obtaining Connector/Python	459
5.4.2 Installing Connector/Python from a Binary Distribution	459
5.4.3 Installing Connector/Python from a Source Distribution	460
5.4.4 Verifying Your Connector/Python Installation	462
5.5 Connector/Python Coding Examples	462
5.5.1 Connecting to MySQL Using Connector/Python	462
5.5.2 Creating Tables Using Connector/Python	464
5.5.3 Inserting Data Using Connector/Python	466
5.5.4 Querying Data Using Connector/Python	467
5.6 Connector/Python Tutorials	468
5.6.1 Tutorial: Raise Employee's Salary Using a Buffered Cursor	468
5.7 Connector/Python Connection Establishment	469
5.7.1 Connector/Python Connection Arguments	469
5.7.2 Connector/Python Option-File Support	474
5.8 Connector/Python Other Topics	476
5.8.1 Connector/Python Connection Pooling	476
5.8.2 Connector/Python Fabric Support	478
5.8.3 Connector/Python Django Backend	478
5.9 Connector/Python API Reference	479
5.9.1 mysql.connector Module	479
5.9.2 connection.MySQLConnection Class	480
5.9.3 pooling.MySQLConnectionPool Class	492
5.9.4 pooling.PooledMySQLConnection Class	494
5.9.5 cursor.MySQLCursor Class	495
5.9.6 Subclasses cursor.MySQLCursor	503
5.9.7 constants.ClientFlag Class	506
5.9.8 constants.FieldType Class	507
5.9.9 constants.SQLMode Class	507
5.9.10 constants.CharacterSet Class	507
5.9.11 constants.RefreshOption Class	507
5.9.12 Errors and Exceptions	508

5.1 Introduction to MySQL Connector/Python

MySQL Connector/Python enables Python programs to access MySQL databases, using an API that is compliant with the [Python Database API Specification v2.0 \(PEP 249\)](#). It is written in pure Python and does not have any dependencies except for the [Python Standard Library](#).

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

MySQL Connector/Python includes support for:

- Almost all features provided by MySQL Server up to and including MySQL Server version 5.7.
- Converting parameter values back and forth between Python and MySQL data types, for example Python `datetime` and MySQL `DATETIME`. You can turn automatic conversion on for convenience, or off for optimal performance.
- All MySQL extensions to standard SQL syntax.

- Protocol compression, which enables compressing the data stream between the client and server.
- Connections using TCP/IP sockets and on Unix using Unix sockets.
- Secure TCP/IP connections using SSL.
- Self-contained driver. Connector/Python does not require the MySQL client library or any Python modules outside the standard library.

MySQL Connector/Python 2.0 supports Python 2.6 and 2.7, and Python 3.3 and later. MySQL Connector/Python 1.2 and 1.1 support Python from version 2.6 through 2.7, and Python 3.1 and later. MySQL Connector/Python 1.0 supports Python from version 2.4 through 2.7, and Python 3.1 and later.

Note

Connector/Python does not support the old MySQL Server authentication methods, which means that MySQL versions prior to 4.1 will not work.

5.2 Guidelines for Python Developers

The following guidelines cover aspects of developing MySQL applications that might not be immediately obvious to developers coming from a Python background:

- For security, do not hardcode the values needed to connect and log into the database in your main script. Python has the convention of a `config.py` module, where you can keep such values separate from the rest of your code.
- Python scripts often build up and tear down large data structures in memory, up to the limits of available RAM. Because MySQL often deals with data sets that are many times larger than available memory, techniques that optimize storage space and disk I/O are especially important. For example, in MySQL tables, you typically use numeric IDs rather than string-based dictionary keys, so that the key values are compact and have a predictable length. This is especially important for columns that make up the [primary key](#) for an [InnoDB](#) table, because those column values are duplicated within each [secondary index](#).
- Any application that accepts input must expect to handle bad data.

The bad data might be accidental, such as out-of-range values or misformatted strings. The application can use server-side checks such as [unique constraints](#) and [NOT NULL constraints](#), to keep the bad data from ever reaching the database. On the client side, use techniques such as exception handlers to report any problems and take corrective action.

The bad data might also be deliberate, representing an “SQL injection” attack. For example, input values might contain quotation marks, semicolons, `%` and `_` wildcard characters and other characters significant in SQL statements. Validate input values to make sure they have only the expected characters. Escape any special characters that could change the intended behavior when substituted into an SQL statement. Never concatenate a user input value into an SQL statement without doing validation and escaping first. Even when accepting input generated by some other program, expect that the other program could also have been compromised and be sending you incorrect or malicious data.

- Because the result sets from SQL queries can be very large, use the appropriate method to retrieve items from the result set as you loop through them. `fetchone()` retrieves a single item, when you know the result set contains a single row. `fetchall()` retrieves all the items, when you know the result set contains a limited number of rows that can fit comfortably into memory. `fetchmany()` is the general-purpose method when you cannot predict the size of the result set: you keep calling it and looping through the returned items, until there are no more results to process.
- Since Python already has convenient modules such as [pickle](#) and [cPickle](#) to read and write data structures on disk, data that you choose to store in MySQL instead is likely to have special characteristics:

- **Too large to all fit in memory at one time.** You use [SELECT](#) statements to query only the precise items you need, and [aggregate functions](#) to perform calculations across multiple items. You configure the [innodb_buffer_pool_size](#) option within the MySQL server to dedicate a certain amount of RAM for caching query results.
- **Too complex to be represented by a single data structure.** You divide the data between different SQL tables. You can recombine data from multiple tables by using a [join](#) query. You make sure that related data is kept in sync between different tables by setting up [foreign key](#) relationships.
- **Updated frequently, perhaps by multiple users simultaneously.** The updates might only affect a small portion of the data, making it wasteful to write the whole structure each time. You use the SQL [INSERT](#), [UPDATE](#), and [DELETE](#) statements to update different items concurrently, writing only the changed values to disk. You use [InnoDB](#) tables and [transactions](#) to keep write operations from conflicting with each other, and to return consistent query results even as the underlying data is being updated.
- Using MySQL best practices for performance can help your application to scale without requiring major rewrites and architectural changes. See [Optimization](#) for best practices for MySQL performance. It offers guidelines and tips for SQL tuning, database design, and server configuration.
- You can avoid reinventing the wheel by learning the MySQL SQL statements for common operations: operators to use in queries, techniques for bulk loading data, and so on. Some statements and clauses are extensions to the basic ones defined by the SQL standard. See [Data Manipulation Statements](#), [Data Definition Statements](#), and [SELECT Syntax](#) for the main classes of statements.
- Issuing SQL statements from Python typically involves declaring very long, possibly multi-line string literals. Because string literals within the SQL statements could be enclosed by single quotation, double quotation marks, or contain either of those characters, for simplicity you can use Python's triple-quoting mechanism to enclose the entire statement. For example:

```
'''It doesn't matter if this string contains 'single'
or "double" quotes, as long as there aren't 3 in a
row.'''
```

You can use either of the ' or " characters for triple-quoting multi-line string literals.

- Many of the secrets to a fast, scalable MySQL application involve using the right syntax at the very start of your setup procedure, in the [CREATE TABLE](#) statements. For example, Oracle recommends the [ENGINE=INNODB](#) clause for most tables, and makes [InnoDB](#) the default storage engine in MySQL 5.5 and up. Using [InnoDB](#) tables enables transactional behavior that helps scalability of read-write workloads and offers automatic [crash recovery](#). Another recommendation is to declare a numeric [primary key](#) for each table, which offers the fastest way to look up values and can act as a pointer to associated values in other tables (a [foreign key](#)). Also within the [CREATE TABLE](#) statement, using the most compact column data types that meet your application requirements helps performance and scalability because that enables the database server to move less data back and forth between memory and disk.

5.3 Connector/Python Versions

The following table summarizes the available Connector/Python versions. For series that have reached General Availability (GA) status, development releases in the series prior to the GA version are no longer supported.

Table 5.1 Connector/Python Version Reference

Connector/Python Version	MySQL Server Versions	Python Versions	Connector Status
2.2	5.7, 5.6, 5.5	3.3 and later, 2.7, 2.6	Developer Milestone
2.1	5.7, 5.6, 5.5	3.3 and later, 2.7, 2.6	Recommended version
2.0	5.7, 5.6, 5.5	3.3 and later, 2.7, 2.6	GA, Supported
1.2	5.7, 5.6, 5.5 (5.1, 5.0, 4.1)	3.1 and later, 2.7, 2.6	GA, Supported

Note

MySQL server and Python versions within parentheses are known to work with Connector/Python, but are not officially supported. Bugs might not get fixed for those versions.

Note

Connector/Python does not support the old MySQL Server authentication methods, which means that MySQL versions prior to 4.1 will not work.

5.4 Connector/Python Installation

Connector/Python runs on any platform where Python is installed. Python comes preinstalled on most Unix and Unix-like systems, such as Linux, OS X, and FreeBSD. On Microsoft Windows, a Python installer is available at the [Python Download website](#). If necessary, download and install Python for Windows before attempting to install Connector/Python.

Note

Connector/Python requires [python](#) to be in the system's [PATH](#) and installation fails if [python](#) cannot be located. On Unix and Unix-like systems, [python](#) is normally located in a directory included in the default [PATH](#) setting. On Windows, if you install Python, either enable **Add python.exe to Path** during the installation process, or manually add the directory containing [python.exe](#) yourself.

For more information about installation and configuration of Python on Windows, see [Using Python on Windows](#) in the Python documentation.

Connector/Python implements the MySQL client/server protocol two ways:

- As pure Python. This implementation of the protocol does not require any other MySQL client libraries or other components.
- As a C Extension that interfaces with the MySQL C client library. This implementation of the protocol is dependent on the client library, but can use the library provided by either MySQL Connector/C or MySQL Server packages (see [MySQL C API Implementations](#)). The C Extension is available as of Connector/Python 2.1.1.

Neither implementation of the client/server protocol has any third-party dependencies. However, if you need SSL support, verify that your Python installation has been compiled using the [OpenSSL](#) libraries.

Installation of Connector/Python is similar on every platform and follows the standard [Python Distribution Utilities](#) or [Distutils](#). Distributions are available in native format for some platforms, such as RPM packages for Linux.

Python terminology regarding distributions:

- **Built Distribution:** A package created in the native packaging format intended for a given platform. It contains both sources and platform-independent bytecode. Connector/Python binary distributions are built distributions.
- **Source Distribution:** A distribution that contains only source files and is generally platform independent.

5.4.1 Obtaining Connector/Python

Packages are available at the [Connector/Python download site](#). For some packaging formats, there are different packages for different versions of Python; choose the one appropriate for the version of Python installed on your system.

5.4.2 Installing Connector/Python from a Binary Distribution

Connector/Python installers in native package formats are available for Windows and for Unix and Unix-like systems:

- Windows: MSI installer package
- Linux: Yum repository for EL6 and EL7 and Fedora 20 and 21; RPM packages for Oracle Linux, Red Hat, and SuSE; Debian packages for Debian and Ubuntu
- OS X: Disk image package with PKG installer

You may need [root](#) or administrator privileges to perform the installation operation.

As of Connector/Python 2.1.1, binary distributions are available that include a C Extension that interfaces with the MySQL C client library. Some packaging types have a single distribution file that includes the pure-Python Connector/Python code together with the C Extension. (Windows MSI and OS X Disk Image packages fall into this category.) Other packaging types have two related distribution files: One that includes the pure-Python Connector/Python code, and one that includes only the C Extension. For packaging types that have separate distribution files, install both distributions if you want to use the C Extension. The two files have related names, the difference being that the one that contains the C Extension has “cext” in the distribution file name.

Binary distributions that provide the C Extension are either statically linked to MySQL Connector/C or link to an already installed C client library provided by a Connector/C or MySQL Server installation. For those distributions that are not statically linked, you must install Connector/C or MySQL Server if it is not already present on your system. To obtain either product, visit the [MySQL download site](#).

Installing Connector/Python on Microsoft Windows Using an MSI Package

Connector/Python Windows MSI Installers ([.msi](#) files) are available from the Connector/Python download site (see [Section 5.4.1, “Obtaining Connector/Python”](#)). Choose an installer appropriate for the version of Python installed on your system. As of Connector/Python 2.1.1, MSI Installers include the C Extension; it need not be installed separately.

To use the MSI Installer, launch it and follow the prompts in the screens it presents to install Connector/Python in the location of your choosing.

Alternatively, to run the installer from the command line, use this command in a console window, where [VER](#) and [PYVER](#) are the respective Connector/Python and Python version numbers in the installer file name:

```
shell> msixexec /i mysql-connector-python-VER-pyPYVER.msi
```

Subsequent executions of Connector/Python using the MSI installer permit you to either repair or remove the existing Connector/Python installation.

Installing Connector/Python on Linux Using the MySQL Yum Repository

For EL6 or EL7-based platforms and Fedora 19 or 20, you can install Connector/Python using the MySQL Yum repository (see [Installing Additional MySQL Products and Components with Yum](#)). You must have the MySQL Yum repository on your system's repository list (for details, see [Adding the MySQL Yum Repository](#)). To make sure that your Yum repository is up-to-date, use this command:

```
shell> sudo yum update mysql-community-release
```

Then install Connector/Python as follows:

```
shell> sudo yum install mysql-connector-python
```

Installing Connector/Python on Linux Using an RPM Package

Connector/Python Linux RPM packages (`.rpm` files) are available from the Connector/Python download site (see [Section 5.4.1, "Obtaining Connector/Python"](#)).

To install a Connector/Python RPM package (denoted here as `PACKAGE.rpm`), use this command:

```
shell> rpm -i PACKAGE.rpm
```

To install the C Extension (available as of Connector/Python 2.1.1), install the corresponding package with "cext" in the package name.

RPM provides a feature to verify the integrity and authenticity of packages before installing them. To learn more, see [Verifying Package Integrity Using MD5 Checksums or GnuPG](#).

Installing Connector/Python on Linux Using a Debian Package

Connector/Python Debian packages (`.deb` files) are available for Debian or Debian-like Linux systems from the Connector/Python download site (see [Section 5.4.1, "Obtaining Connector/Python"](#)).

To install a Connector/Python Debian package (denoted here as `PACKAGE.deb`), use this command:

```
shell> dpkg -i PACKAGE.deb
```

To install the C Extension (available as of Connector/Python 2.1.1), install the corresponding package with "cext" in the package name.

Installing Connector/Python on OS X Using a Disk Image

Connector/Python OS X disk images (`.dmg` files) are available from the Connector/Python download site (see [Section 5.4.1, "Obtaining Connector/Python"](#)). As of Connector/Python 2.1.1, OS X disk images include the C Extension; it need not be installed separately.

Download the `.dmg` file and install Connector/Python by opening it and double clicking the resulting `.pkg` file.

5.4.3 Installing Connector/Python from a Source Distribution

Connector/Python source distributions are platform independent and can be used on any platform. Source distributions are packaged in two formats:

- Zip archive format (`.zip` file)
- Compressed tar archive format (`.tar.gz` file)

Either packaging format can be used on any platform, but Zip archives are more commonly used on Windows systems and [tar](#) archives on Unix and Unix-like systems.

Prerequisites for Compiling Connector/Python with the C Extension

As of Connector/Python 2.1.1, source distributions include the C Extension that interfaces with the MySQL C client library. You can build the distribution with or without support for this extension. To build Connector/Python with support for the C Extension, you must satisfy the following prerequisites.

- Prerequisites for Windows systems:
 - Correct version of Visual Studio: VS 2009 for Python 2.7, VS 2010 for Python 3.3
 - Python development files
 - MySQL Connector/C or MySQL Server installed, including development files
- Prerequisites for Unix and Unix-like systems:
 - A C/C++ compiler, such as [gcc](#)
 - Python development files
 - MySQL Connector/C or MySQL Server installed, including development files

You must install Connector/C or MySQL Server if it is not already present on your system. To obtain either product, visit the [MySQL download site](#).

For certain platforms, MySQL development files are provided in separate packages. This is true for RPM and Debian packages, for example.

Installing Connector/Python from Source on Microsoft Windows

A Connector/Python Zip archive ([.zip](#) file) is available from the Connector/Python download site (see [Section 5.4.1, "Obtaining Connector/Python"](#)).

To install Connector/Python from a Zip archive, download the latest version and follow these steps:

1. Unpack the Zip archive in the intended installation directory (for example, `C:\mysql-connector\`) using [WinZip](#) or another tool that can read [.zip](#) files.
2. Start a console window and change location to the folder where you unpacked the Zip archive:

```
shell> cd C:\mysql-connector\
```

3. Inside the Connector/Python folder, perform the installation using this command:

```
shell> python setup.py install
```

To include the C Extension (available as of Connector/Python 2.1.1), use this command instead:

```
shell> python setup.py install --with-mysql-capi="path_name"
```

The argument to `--with-mysql-capi` is the path to the installation directory of either MySQL Connector/C or MySQL Server.

To see all options and commands supported by [setup.py](#), use this command:

```
shell> python setup.py --help
```

Installing Connector/Python from Source on Unix and Unix-Like Systems

For Unix and Unix-like systems such as Linux, Solaris, OS X, and FreeBSD, a Connector/Python `tar` archive (`.tar.gz` file) is available from the Connector/Python download site (see [Section 5.4.1](#), “Obtaining Connector/Python”).

To install Connector/Python from a `tar` archive, download the latest version (denoted here as `VER`), and execute these commands:

```
shell> tar xzf mysql-connector-python-VER.tar.gz
shell> cd mysql-connector-python-VER
shell> sudo python setup.py install
```

To include the C Extension (available as of Connector/Python 2.1.1), use this command instead:

```
shell> sudo python setup.py install --with-mysql-capi=value
```

The argument to `--with-mysql-capi` is the path to the installation directory of either MySQL Connector/C or MySQL Server, or the path to the `mysql_config` command.

To see all options and commands supported by `setup.py`, use this command:

```
shell> python setup.py --help
```

5.4.4 Verifying Your Connector/Python Installation

On Windows, the default Connector/Python installation location is `C:\PythonX.Y\Lib\site-packages\`, where `X.Y` is the Python version you used to install the connector.

On Unix-like systems, the default Connector/Python installation location is `/prefix/pythonX.Y/site-packages/`, where `prefix` is the location where Python is installed and `X.Y` is the Python version. See [How installation works](#) in the Python manual.

The C Extension is installed as `_mysql_connector.so` in the `site-packages` directory, not in the `mysql/connector` directory.

Depending on your platform, the installation path might differ from the default. If you are not sure where Connector/Python is installed, do the following to determine its location. The output here shows installation locations as might be seen on OS X:

```
shell> python
>>> from distutils.sysconfig import get_python_lib
>>> print get_python_lib()           # Python v2.x
/Library/Python/2.7/site-packages
>>> print(get_python_lib())         # Python v3.x
/Library/Frameworks/Python.framework/Versions/3.1/lib/python3.1/site-packages
```

To test that your Connector/Python installation is working and able to connect to MySQL Server, you can run a very simple program where you supply the login credentials and host information required for the connection. For an example, see [Section 5.5.1](#), “Connecting to MySQL Using Connector/Python”.

5.5 Connector/Python Coding Examples

These coding examples illustrate how to develop Python applications and scripts which connect to MySQL Server using MySQL Connector/Python.

5.5.1 Connecting to MySQL Using Connector/Python

The `connect()` constructor creates a connection to the MySQL server and returns a `MySQLConnection` object.

The following example shows how to connect to the MySQL server:


```
import mysql.connector
cnx = mysql.connector.connect(user='scott', password='tiger',
                             host='127.0.0.1',
                             database='employees')
cnx.close()
```

[Section 5.7.1, “Connector/Python Connection Arguments”](#) describes the permitted connection arguments.

It is also possible to create connection objects using the [connection.MySQLConnection\(\)](#) class:

```
from mysql.connector import (connection)
cnx = connection.MySQLConnection(user='scott', password='tiger',
                                  host='127.0.0.1',
                                  database='employees')
cnx.close()
```

Both methods, using the [connect\(\)](#) constructor, or the class directly, are valid and functionally equal, but using [connector\(\)](#) is preferred and is used in most examples in this manual.

To handle connection errors, use the [try](#) statement and catch all errors using the [errors.Error](#) exception:

```
import mysql.connector
from mysql.connector import errorcode
try:
    cnx = mysql.connector.connect(user='scott',
                                  database='testt')
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("Something is wrong with your user name or password")
    elif err.errno == errorcode.ER_BAD_DB_ERROR:
        print("Database does not exist")
    else:
        print(err)
else:
    cnx.close()
```

If you have lots of connection arguments, it's best to keep them in a dictionary and use the [**](#) operator:

```
import mysql.connector
config = {
    'user': 'scott',
    'password': 'tiger',
    'host': '127.0.0.1',
    'database': 'employees',
    'raise_on_warnings': True,
}
cnx = mysql.connector.connect(**config)
cnx.close()
```

Using the Connector/Python C Extension

As of Connector/Python 2.1.1, the [use_pure](#) connection argument determines whether to connect using a pure Python interface to MySQL, or a C Extension that uses the MySQL C client library (see [The Connector/Python C Extension](#)). The default is [True](#) (use the pure Python implementation). Setting [use_pure](#) to [False](#) causes the connection to use the C Extension if your Connector/Python installation includes it. The following examples are similar to others shown previously but with the inclusion of [use_pure=False](#).

Connect by naming arguments in the [connect\(\)](#) call:

```
import mysql.connector
cnx = mysql.connector.connect(user='scott', password='tiger',
                              host='127.0.0.1',
```

```

        database='employees',
        use_pure=False)
cnx.close()

```

Connect using an argument dictionary:

```

import mysql.connector
config = {
    'user': 'scott',
    'password': 'tiger',
    'host': '127.0.0.1',
    'database': 'employees',
    'raise_on_warnings': True,
    'use_pure': False,
}
cnx = mysql.connector.connect(**config)
cnx.close()

```

It is also possible to use the C Extension directly, by importing the `_mysql_connector` module rather than the `mysql.connector` module. For more information, see [The _mysql_connector C Extension Module](#).

5.5.2 Creating Tables Using Connector/Python

All [DDL](#) (Data Definition Language) statements are executed using a handle structure known as a cursor. The following examples show how to create the tables of the [Employee Sample Database](#). You need them for the other examples.

In a MySQL server, tables are very long-lived objects, and are often accessed by multiple applications written in different languages. You might typically work with tables that are already set up, rather than creating them within your own application. Avoid setting up and dropping tables over and over again, as that is an expensive operation. The exception is [temporary tables](#), which can be created and dropped quickly within an application.

```

from __future__ import print_function
import mysql.connector
from mysql.connector import errorcode
DB_NAME = 'employees'
TABLES = {}
TABLES['employees'] = (
    "CREATE TABLE `employees` ("
    "  `emp_no` int(11) NOT NULL AUTO_INCREMENT,"
    "  `birth_date` date NOT NULL,"
    "  `first_name` varchar(14) NOT NULL,"
    "  `last_name` varchar(16) NOT NULL,"
    "  `gender` enum('M','F') NOT NULL,"
    "  `hire_date` date NOT NULL,"
    "  PRIMARY KEY (`emp_no`)"
    ") ENGINE=InnoDB")
TABLES['departments'] = (
    "CREATE TABLE `departments` ("
    "  `dept_no` char(4) NOT NULL,"
    "  `dept_name` varchar(40) NOT NULL,"
    "  PRIMARY KEY (`dept_no`), UNIQUE KEY `dept_name` (`dept_name`)"
    ") ENGINE=InnoDB")
TABLES['salaries'] = (
    "CREATE TABLE `salaries` ("
    "  `emp_no` int(11) NOT NULL,"
    "  `salary` int(11) NOT NULL,"
    "  `from_date` date NOT NULL,"
    "  `to_date` date NOT NULL,"
    "  PRIMARY KEY (`emp_no`,`from_date`), KEY `emp_no` (`emp_no`),"
    "  CONSTRAINT `salaries_ibfk_1` FOREIGN KEY (`emp_no`) "
    "    REFERENCES `employees` (`emp_no`) ON DELETE CASCADE"
    ") ENGINE=InnoDB")
TABLES['dept_emp'] = (
    "CREATE TABLE `dept_emp` ("
    "  `emp_no` int(11) NOT NULL,"

```

```

" `dept_no` char(4) NOT NULL,"
" `from_date` date NOT NULL,"
" `to_date` date NOT NULL,"
" PRIMARY KEY (`emp_no`,`dept_no`), KEY `emp_no` (`emp_no`),"
" KEY `dept_no` (`dept_no`),"
" CONSTRAINT `dept_emp_ibfk_1` FOREIGN KEY (`emp_no`) "
" REFERENCES `employees` (`emp_no`) ON DELETE CASCADE,"
" CONSTRAINT `dept_emp_ibfk_2` FOREIGN KEY (`dept_no`) "
" REFERENCES `departments` (`dept_no`) ON DELETE CASCADE"
") ENGINE=InnoDB")
TABLES['dept_manager'] = (
" CREATE TABLE `dept_manager` ("
" `dept_no` char(4) NOT NULL,"
" `emp_no` int(11) NOT NULL,"
" `from_date` date NOT NULL,"
" `to_date` date NOT NULL,"
" PRIMARY KEY (`emp_no`,`dept_no`),"
" KEY `emp_no` (`emp_no`),"
" KEY `dept_no` (`dept_no`),"
" CONSTRAINT `dept_manager_ibfk_1` FOREIGN KEY (`emp_no`) "
" REFERENCES `employees` (`emp_no`) ON DELETE CASCADE,"
" CONSTRAINT `dept_manager_ibfk_2` FOREIGN KEY (`dept_no`) "
" REFERENCES `departments` (`dept_no`) ON DELETE CASCADE"
") ENGINE=InnoDB")
TABLES['titles'] = (
"CREATE TABLE `titles` ("
" `emp_no` int(11) NOT NULL,"
" `title` varchar(50) NOT NULL,"
" `from_date` date NOT NULL,"
" `to_date` date DEFAULT NULL,"
" PRIMARY KEY (`emp_no`,`title`,`from_date`), KEY `emp_no` (`emp_no`),"
" CONSTRAINT `titles_ibfk_1` FOREIGN KEY (`emp_no`) "
" REFERENCES `employees` (`emp_no`) ON DELETE CASCADE"
") ENGINE=InnoDB")

```

The preceding code shows how we are storing the `CREATE` statements in a Python dictionary called `TABLES`. We also define the database in a global variable called `DB_NAME`, which enables you to easily use a different schema.

```

cnx = mysql.connector.connect(user='scott')
cursor = cnx.cursor()

```

A single MySQL server can manage multiple [databases](#). Typically, you specify the database to switch to when connecting to the MySQL server. This example does not connect to the database upon connection, so that it can make sure the database exists, and create it if not:

```

def create_database(cursor):
    try:
        cursor.execute(
            "CREATE DATABASE {} DEFAULT CHARACTER SET 'utf8'".format(DB_NAME))
    except mysql.connector.Error as err:
        print("Failed creating database: {}".format(err))
        exit(1)
try:
    cnx.database = DB_NAME
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_BAD_DB_ERROR:
        create_database(cursor)
        cnx.database = DB_NAME
    else:
        print(err)
        exit(1)

```

We first try to change to a particular database using the `database` property of the connection object `cnx`. If there is an error, we examine the error number to check if the database does not exist. If so, we call the `create_database` function to create it for us.

On any other error, the application exits and displays the error message.

After we successfully create or change to the target database, we create the tables by iterating over the items of the `TABLES` dictionary:

```
for name, ddl in TABLES.iteritems():
    try:
        print("Creating table {}: ".format(name), end='')
        cursor.execute(ddl)
    except mysql.connector.Error as err:
        if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
            print("already exists.")
        else:
            print(err.msg)
    else:
        print("OK")
cursor.close()
cnx.close()
```

To handle the error when the table already exists, we notify the user that it was already there. Other errors are printed, but we continue creating tables. (The example shows how to handle the “table already exists” condition for illustration purposes. In a real application, we would typically avoid the error condition entirely by using the `IF NOT EXISTS` clause of the `CREATE TABLE` statement.)

The output would be something like this:

```
Creating table employees: already exists.
Creating table salaries: already exists.
Creating table titles: OK
Creating table departments: already exists.
Creating table dept_manager: already exists.
Creating table dept_emp: already exists.
```

To populate the employees tables, use the dump files of the [Employee Sample Database](#). Note that you only need the data dump files that you will find in an archive named like `employees_db-dump-files-1.0.5.tar.bz2`. After downloading the dump files, execute the following commands, adding connection options to the `mysql` commands if necessary:

```
shell> tar xzf employees_db-dump-files-1.0.5.tar.bz2
shell> cd employees_db
shell> mysql employees < load_employees.dump
shell> mysql employees < load_titles.dump
shell> mysql employees < load_departments.dump
shell> mysql employees < load_salaries.dump
shell> mysql employees < load_dept_emp.dump
shell> mysql employees < load_dept_manager.dump
```

5.5.3 Inserting Data Using Connector/Python

Inserting or updating data is also done using the handler structure known as a cursor. When you use a transactional storage engine such as `InnoDB` (the default in MySQL 5.5 and later), you must `commit` the data after a sequence of `INSERT`, `DELETE`, and `UPDATE` statements.

This example shows how to insert new data. The second `INSERT` depends on the value of the newly created `primary key` of the first. The example also demonstrates how to use extended formats. The task is to add a new employee starting to work tomorrow with a salary set to 50000.

Note

The following example uses tables created in the example [Section 5.5.2, “Creating Tables Using Connector/Python”](#). The `AUTO_INCREMENT` column option for the primary key of the `employees` table is important to ensure reliable, easily searchable data.

```
from __future__ import print_function
```

```

from datetime import date, datetime, timedelta
import mysql.connector
cnx = mysql.connector.connect(user='scott', database='employees')
cursor = cnx.cursor()
tomorrow = datetime.now().date() + timedelta(days=1)
add_employee = ("INSERT INTO employees "
                "(first_name, last_name, hire_date, gender, birth_date) "
                "VALUES (%s, %s, %s, %s, %s)")
add_salary = ("INSERT INTO salaries "
              "(emp_no, salary, from_date, to_date) "
              "VALUES (%(emp_no)s, %(salary)s, %(from_date)s, %(to_date)s)")
data_employee = ('Geert', 'Vanderkelen', tomorrow, 'M', date(1977, 6, 14))
# Insert new employee
cursor.execute(add_employee, data_employee)
emp_no = cursor.lastrowid
# Insert salary information
data_salary = {
    'emp_no': emp_no,
    'salary': 50000,
    'from_date': tomorrow,
    'to_date': date(9999, 1, 1),
}
cursor.execute(add_salary, data_salary)
# Make sure data is committed to the database
cnx.commit()
cursor.close()
cnx.close()

```

We first open a connection to the MySQL server and store the [connection object](#) in the variable `cnx`. We then create a new cursor, by default a [MySQLCursor](#) object, using the connection's `cursor()` method.

We could calculate tomorrow by calling a database function, but for clarity we do it in Python using the [datetime](#) module.

Both `INSERT` statements are stored in the variables called `add_employee` and `add_salary`. Note that the second `INSERT` statement uses extended Python format codes.

The information of the new employee is stored in the tuple `data_employee`. The query to insert the new employee is executed and we retrieve the newly inserted value for the `emp_no` column (an [AUTO_INCREMENT](#) column) using the `lastrowid` property of the cursor object.

Next, we insert the new salary for the new employee, using the `emp_no` variable in the dictionary holding the data. This dictionary is passed to the `execute()` method of the cursor object if an error occurred.

Since by default Connector/Python turns [autocommit](#) off, and MySQL 5.5 and later uses transactional [InnoDB](#) tables by default, it is necessary to commit your changes using the connection's `commit()` method. You could also [roll back](#) using the `rollback()` method.

5.5.4 Querying Data Using Connector/Python

The following example shows how to [query](#) data using a cursor created using the connection's `cursor()` method. The data returned is formatted and printed on the console.

The task is to select all employees hired in the year 1999 and print their names and hire dates to the console.

```

import datetime
import mysql.connector
cnx = mysql.connector.connect(user='scott', database='employees')
cursor = cnx.cursor()
query = ("SELECT first_name, last_name, hire_date FROM employees "
        "WHERE hire_date BETWEEN %s AND %s")
hire_start = datetime.date(1999, 1, 1)
hire_end = datetime.date(1999, 12, 31)
cursor.execute(query, (hire_start, hire_end))

```

```
for (first_name, last_name, hire_date) in cursor:
    print("{} {} was hired on {:%d %b %Y}".format(
        last_name, first_name, hire_date))
cursor.close()
cnx.close()
```

We first open a connection to the MySQL server and store the [connection object](#) in the variable `cnx`. We then create a new cursor, by default a [MySQLCursor](#) object, using the connection's `cursor()` method.

In the preceding example, we store the `SELECT` statement in the variable `query`. Note that we are using unquoted `%s`-markers where dates should have been. Connector/Python converts `hire_start` and `hire_end` from Python types to a data type that MySQL understands and adds the required quotes. In this case, it replaces the first `%s` with `'1999-01-01'`, and the second with `'1999-12-31'`.

We then execute the operation stored in the `query` variable using the `execute()` method. The data used to replace the `%s`-markers in the query is passed as a tuple: `(hire_start, hire_end)`.

After executing the query, the MySQL server is ready to send the data. The result set could be zero rows, one row, or 100 million rows. Depending on the expected volume, you can use different techniques to process this result set. In this example, we use the `cursor` object as an iterator. The first column in the row is stored in the variable `first_name`, the second in `last_name`, and the third in `hire_date`.

We print the result, formatting the output using Python's built-in `format()` function. Note that `hire_date` was converted automatically by Connector/Python to a Python `datetime.date` object. This means that we can easily format the date in a more human-readable form.

The output should be something like this:

```
..
Wilharm, LiMin was hired on 16 Dec 1999
Wielonsky, Lalit was hired on 16 Dec 1999
Kamble, Dannz was hired on 18 Dec 1999
DuBourdieu, Zhongwei was hired on 19 Dec 1999
Fujisawa, Rosita was hired on 20 Dec 1999
..
```

5.6 Connector/Python Tutorials

These tutorials illustrate how to develop Python applications and scripts that connect to a MySQL database server using MySQL Connector/Python.

5.6.1 Tutorial: Raise Employee's Salary Using a Buffered Cursor

The following example script gives a long-overdue 15% raise effective tomorrow to all employees who joined in the year 2000 and are still with the company.

To iterate through the selected employees, we use buffered cursors. (A buffered cursor fetches and buffers the rows of a result set after executing a query; see [Section 5.9.6.1, “cursor.MySQLCursorBuffered Class”](#).) This way, it is unnecessary to fetch the rows in a new variables. Instead, the cursor can be used as an iterator.

Note

This script is an example; there are other ways of doing this simple task.

```
from __future__ import print_function
from decimal import Decimal
from datetime import datetime, date, timedelta
import mysql.connector
# Connect with the MySQL Server
cnx = mysql.connector.connect(user='scott', database='employees')
```

```
# Get two buffered cursors
curA = cnx.cursor(buffered=True)
curB = cnx.cursor(buffered=True)
# Query to get employees who joined in a period defined by two dates
query = (
    "SELECT s.emp_no, salary, from_date, to_date FROM employees AS e "
    "LEFT JOIN salaries AS s USING (emp_no) "
    "WHERE to_date = DATE('9999-01-01') "
    "AND e.hire_date BETWEEN DATE(%s) AND DATE(%s)"
)
# UPDATE and INSERT statements for the old and new salary
update_old_salary = (
    "UPDATE salaries SET to_date = %s "
    "WHERE emp_no = %s AND from_date = %s"
)
insert_new_salary = (
    "INSERT INTO salaries (emp_no, from_date, to_date, salary) "
    "VALUES (%s, %s, %s, %s)"
)
# Select the employees getting a raise
curA.execute(query, (date(2000, 1, 1), date(2000, 12, 31)))
# Iterate through the result of curA
for (emp_no, salary, from_date, to_date) in curA:
    # Update the old and insert the new salary
    new_salary = int(round(salary * Decimal('1.15')))
    curB.execute(update_old_salary, (tomorrow, emp_no, from_date))
    curB.execute(insert_new_salary,
        (emp_no, tomorrow, date(9999, 1, 1), new_salary))
    # Commit the changes
    cnx.commit()
cnx.close()
```

5.7 Connector/Python Connection Establishment

Connector/Python provides a `connect()` call used to establish connections to the MySQL server. The following sections describe the permitted arguments for `connect()` and describe how to use option files that supply additional arguments.

5.7.1 Connector/Python Connection Arguments

A connection with the MySQL server can be established using either the `mysql.connector.connect()` function or the `mysql.connector.MySQLConnection()` class:

```
cnx = mysql.connector.connect(user='joe', database='test')
cnx = MySQLConnection(user='joe', database='test')
```

The following table describes the arguments that can be used to initiate a connection. An asterisk (*) following an argument indicates a synonymous argument name, available only for compatibility with other Python MySQL drivers. Oracle recommends not to use these alternative names.

Table 5.2 Connection Arguments for Connector/Python

Argument Name	Default	Description
<code>user</code> (<code>username</code> *)		The user name used to authenticate with the MySQL server.
<code>password</code> (<code>passwd</code> *)		The password to authenticate the user with the MySQL server.
<code>database</code> (<code>db</code> *)		The database name to use when connecting with the MySQL server.
<code>host</code>	127.0.0.1	The host name or IP address of the MySQL server.
<code>port</code>	3306	The TCP/IP port of the MySQL server. Must be an integer.
<code>unix_socket</code>		The location of the Unix socket file.
<code>auth_plugin</code>		Authentication plugin to use. Added in 1.2.1.
<code>use_unicode</code>	<code>True</code>	Whether to use Unicode.

Argument Name	Default	Description
<code>charset</code>	<code>utf8</code>	Which MySQL character set to use.
<code>collation</code>	<code>utf8_general_ci</code>	Which MySQL collation to use.
<code>autocommit</code>	<code>False</code>	Whether to <code>autocommit</code> transactions.
<code>time_zone</code>		Set the <code>time_zone</code> session variable at connection time.
<code>sql_mode</code>		Set the <code>sql_mode</code> session variable at connection time.
<code>get_warnings</code>	<code>False</code>	Whether to fetch warnings.
<code>raise_on_warnings</code>	<code>False</code>	Whether to raise an exception on warnings.
<code>connection_timeout</code> (<code>connect_timeout*</code>)		Timeout for the TCP and Unix socket connections.
<code>client_flags</code>		MySQL client flags.
<code>buffered</code>	<code>False</code>	Whether cursor objects fetch the results immediately after executing queries.
<code>raw</code>	<code>False</code>	Whether MySQL results are returned as is, rather than converted to Python types.
<code>consume_results</code>	<code>False</code>	Whether to automatically read result sets.
<code>ssl_ca</code>		File containing the SSL certificate authority.
<code>ssl_cert</code>		File containing the SSL certificate file.
<code>ssl_key</code>		File containing the SSL key.
<code>ssl_verify_cert</code>	<code>False</code>	When set to <code>True</code> , checks the server certificate against the certificate file specified by the <code>ssl_ca</code> option. Any mismatch causes a <code>ValueError</code> exception.
<code>force_ipv6</code>	<code>False</code>	When set to <code>True</code> , uses IPv6 when an address resolves to both IPv4 and IPv6. By default, IPv4 is used in such cases.
<code>dsn</code>		Not supported (raises <code>NotSupportedError</code> when used).
<code>pool_name</code>		Connection pool name. Added in 1.1.1.
<code>pool_size</code>	5	Connection pool size. Added in 1.1.1.
<code>pool_reset_session</code>	<code>True</code>	Whether to reset session variables when connection is returned to pool. Added in 1.1.5.
<code>compress</code>	<code>False</code>	Whether to use compressed client/server protocol. Added in 1.1.2.
<code>converter_class</code>		Converter class to use. Added in 1.1.2.
<code>fabric</code>		MySQL Fabric connection arguments. Added in 1.2.0.
<code>failover</code>		Server failover sequence. Added in 1.2.1.
<code>option_files</code>		Which option files to read. Added in 2.0.0.
<code>option_groups</code>	<code>['client', 'connector_python']</code>	Which groups to read from option files. Added in 2.0.0.
<code>allow_local_infile</code>	<code>True</code>	Whether to enable <code>LOAD DATA LOCAL INFILE</code> . Added in 2.0.0.
<code>use_pure</code>	<code>True</code>	Whether to use pure Python or C Extension. Added in 2.1.1.

MySQL Authentication Options

Authentication with MySQL uses `username` and `password`.

Note

MySQL Connector/Python does not support the old, less-secure password protocols of MySQL versions prior to 4.1.

When the `database` argument is given, the current database is set to the given value. To change the current database later, execute a `USE` SQL statement or set the `database` property of the `MySQLConnection` instance.

By default, Connector/Python tries to connect to a MySQL server running on the local host using TCP/IP. The `host` argument defaults to IP address 127.0.0.1 and `port` to 3306. Unix sockets are supported by setting `unix_socket`. Named pipes on the Windows platform are not supported.

Connector/Python 1.2.1 and up supports authentication plugins found in MySQL 5.6. This includes `mysql_clear_password` and `sha256_password`, both of which require an SSL connection. The `sha256_password` plugin does not work over a non-SSL connection because Connector/Python does not support RSA encryption.

The `connect()` method supports an `auth_plugin` argument that can be used to force use of a particular plugin. For example, if the server is configured to use `sha256_password` by default and you want to connect to an account that authenticates using `mysql_native_password`, either connect using SSL or specify `auth_plugin='mysql_native_password'`.

Character Encoding

By default, strings coming from MySQL are returned as Python Unicode literals. To change this behavior, set `use_unicode` to `False`. You can change the character setting for the client connection through the `charset` argument. To change the character set after connecting to MySQL, set the `charset` property of the `MySQLConnection` instance. This technique is preferred over using the `SET NAMES` SQL statement directly. Similar to the `charset` property, you can set the `collation` for the current MySQL session.

Transactions

The `autocommit` value defaults to `False`, so transactions are not automatically committed. Call the `commit()` method of the `MySQLConnection` instance within your application after doing a set of related insert, update, and delete operations. For data consistency and high throughput for write operations, it is best to leave the `autocommit` configuration option turned off when using `InnoDB` or other transactional tables.

Time Zones

The time zone can be set per connection using the `time_zone` argument. This is useful, for example, if the MySQL server is set to UTC and `TIMESTAMP` values should be returned by MySQL converted to the `PST` time zone.

SQL Modes

MySQL supports so-called SQL Modes, which change the behavior of the server globally or per connection. For example, to have warnings raised as errors, set `sql_mode` to `TRADITIONAL`. For more information, see [Server SQL Modes](#).

Troubleshooting and Error Handling

Warnings generated by queries are fetched automatically when `get_warnings` is set to `True`. You can also immediately raise an exception by setting `raise_on_warnings` to `True`. Consider using the MySQL `sql_mode` setting for turning warnings into errors.

To set a timeout value for connections, use `connection_timeout`.

Enabling and Disabling Features Using Client Flags

MySQL uses `client flags` to enable or disable features. Using the `client_flags` argument, you have control of what is set. To find out what flags are available, use the following:

```
from mysql.connector.constants import ClientFlag
print '\n'.join(ClientFlag.get_full_info())
```

If `client_flags` is not specified (that is, it is zero), defaults are used for MySQL v4.1 and later. If you specify an integer greater than 0, make sure all flags are set properly. A better way to set and unset flags individually is to use a list. For example, to set `FOUND_ROWS`, but disable the default `LONG_FLAG`:

```
flags = [ClientFlag.FOUND_ROWS, -ClientFlag.LONG_FLAG]
mysql.connector.connect(client_flags=flags)
```

Result Set Handling

By default, MySQL Connector/Python does not buffer or prefetch results. This means that after a query is executed, your program is responsible for fetching the data. This avoids excessive memory use when queries return large result sets. If you know that the result set is small enough to handle all at once, you can fetch the results immediately by setting `buffered` to `True`. It is also possible to set this per cursor (see [Section 5.9.2.6, “MySQLConnection.cursor\(\) Method”](#)).

Results generated by queries normally are not read until the client program fetches them. To automatically consume and discard result sets, set the `consume_results` option to `True`. The result is that all results are read, which for large result sets can be slow. (In this case, it might be preferable to close and reopen the connection.) This option is available as of Connector/Python 1.1.2.

Type Conversions

By default, MySQL types in result sets are converted automatically to Python types. For example, a `DATETIME` column value becomes a `datetime.datetime` object. To disable conversion, set the `raw` option to `True`. You might do this to get better performance or perform different types of conversion yourself.

Connecting through SSL

Using SSL connections is possible when your [Python installation supports SSL](#), that is, when it is compiled against the OpenSSL libraries. When you provide the `ssl_ca`, `ssl_key` and `ssl_cert` options, the connection switches to SSL, and the `client_flags` option includes the `ClientFlag.SSL` value automatically. You can use this in combination with the `compressed` option set to `True`.

As of Connector/Python 1.2.1, it is possible to establish an SSL connection using only the `ssl_ca` option. The `ssl_key` and `ssl_cert` arguments are optional. However, when either is given, both must be given or an `AttributeError` is raised.

```
# Note (Example is valid for Python v2 and v3)
from __future__ import print_function
import sys
#sys.path.insert(0, 'python{0}/'.format(sys.version_info[0]))
import mysql.connector
from mysql.connector.constants import ClientFlag
config = {
    'user': 'ssluser',
    'password': 'asecret',
    'host': '127.0.0.1',
    'client_flags': [ClientFlag.SSL],
    'ssl_ca': '/opt/mysql/ssl/ca.pem',
```

```
'ssl_cert': '/opt/mysql/ssl/client-cert.pem',
'ssl_key': '/opt/mysql/ssl/client-key.pem',
}
cnx = mysql.connector.connect(**config)
cur = cnx.cursor(buffered=True)
cur.execute("SHOW STATUS LIKE 'Ssl_cipher'")
print(cur.fetchone())
cur.close()
cnx.close()
```

Connection Pooling

With either the `pool_name` or `pool_size` argument present, Connector/Python creates the new pool. If the `pool_name` argument is not given, the `connect()` call automatically generates the name, composed from whichever of the `host`, `port`, `user`, and `database` connection arguments are given, in that order. If the `pool_size` argument is not given, the default size is 5 connections.

The `pool_reset_session` permits control over whether session variables are reset when the connection is returned to the pool. The default is to reset them.

Connection pooling is supported as of Connector/Python 1.1.1. See [Section 5.8.1, “Connector/Python Connection Pooling”](#).

Protocol Compression

The boolean `compress` argument indicates whether to use the compressed client/server protocol (default `False`). This provides an easier alternative to setting the `ClientFlag.COMPRESS` flag. This argument is available as of Connector/Python 1.1.2.

Converter Class

The `converter_class` argument takes a class and sets it when configuring the connection. An `AttributeError` is raised if the custom converter class is not a subclass of `conversion.MySQLConverterBase`. This argument is available as of Connector/Python 1.1.2. Before 1.1.2, setting a custom converter class is possible only after instantiating a new connection object.

The boolean `compress` argument indicates whether to use the compressed client/server protocol (default `False`). This provides an easier alternative to setting the `ClientFlag.COMPRESS` flag. This argument is available as of Connector/Python 1.1.2.

MySQL Fabric Support

To request a MySQL Fabric connection, provide a `fabric` argument that specifies to contact Fabric. For details, see [Requesting a Fabric Connection](#).

Server Failover

As of Connector/Python 1.2.1, the `connect()` method accepts a `failover` argument that provides information to use for server failover in the event of connection failures. The argument value is a tuple or list of dictionaries (tuple is preferred because it is nonmutable). Each dictionary contains connection arguments for a given server in the failover sequence. Permitted dictionary values are: `user`, `password`, `host`, `port`, `unix_socket`, `database`, `pool_name`, `pool_size`.

Option File Support

As of Connector/Python 2.0.0, option files are supported using two options for `connect()`:

- `option_files`: Which option files to read. The value can be a file path name (a string) or a sequence of path name strings. By default, Connector/Python reads no option files, so this argument must be given explicitly to cause option files to be read. Files are read in the order specified.

- `option_groups`: Which groups to read from option files, if option files are read. The value can be an option group name (a string) or a sequence of group name strings. If this argument is not given, the default value is `['client', 'connector_python']` to read the `[client]` and `[connector_python]` groups.

For more information, see [Section 5.7.2, “Connector/Python Option-File Support”](#).

LOAD DATA LOCAL INFILE

Prior to Connector/Python 2.0.0, to enable use of `LOAD DATA LOCAL INFILE`, clients had to explicitly set the `ClientFlag.LOCAL_FILES` flag. As of 2.0.0, this flag is enabled by default. To disable it, the `allow_local_infile` connection option can be set to `False` at connect time (the default is `True`).

Compatibility with Other Connection Interfaces

`passwd`, `db` and `connect_timeout` are valid for compatibility with other MySQL interfaces and are respectively the same as `password`, `database` and `connection_timeout`. The latter take precedence. Data source name syntax or `dsn` is not used; if specified, it raises a `NotSupportedError` exception.

Client/Server Protocol Implementation

Connector/Python can use a pure Python interface to MySQL, or a C Extension that uses the MySQL C client library. The `use_pure` connection argument determines which. The default is `True` (use the pure Python implementation). Setting `use_pure` to `False` causes the connection to use the C Extension if your Connector/Python installation includes it.

The `use_pure` argument is available as of Connector/Python 2.1.1. For more information, see [The Connector/Python C Extension](#).

5.7.2 Connector/Python Option-File Support

As of version 2.0.0, Connector/Python has the capability of reading options from option files. (For general information about option files in MySQL, see [Using Option Files](#).) Two arguments for the `connect()` call control use of option files in Connector/Python programs:

- `option_files`: Which option files to read. The value can be a file path name (a string) or a sequence of path name strings. By default, Connector/Python reads no option files, so this argument must be given explicitly to cause option files to be read. Files are read in the order specified.
- `option_groups`: Which groups to read from option files, if option files are read. The value can be an option group name (a string) or a sequence of group name strings. If this argument is not given, the default value is `['client', 'connector_python']`, to read the `[client]` and `[connector_python]` groups.

Connector/Python also supports the `!include` and `!includedir` inclusion directives within option files. These directives work the same way as for other MySQL programs (see [Using Option Files](#)).

This example specifies a single option file as a string:

```
cnx = mysql.connector.connect(option_files='/etc/mysql/connectors.cnf')
```

This example specifies multiple option files as a sequence of strings:

```
mysql_option_files = [
    '/etc/mysql/connectors.cnf',
    './development.cnf',
]
cnx = mysql.connector.connect(option_files=mysql_option_files)
```

Connector/Python reads no option files by default, for backward compatibility with versions older than 2.0.0. This differs from standard MySQL clients such as `mysql` or `mysqldump`, which do read option files by default. To find out which option files the standard clients read on your system, invoke one of them with its `--help` option and examine the output. For example:

```
shell> mysql --help
...
Default options are read from the following files in the given order:
/etc/my.cnf /etc/mysql/my.cnf /usr/local/mysql/etc/my.cnf ~/.my.cnf
...
```

If you specify the `option_files` connection argument to read option files, Connector/Python reads the `[client]` and `[connector_python]` option groups by default. To specify explicitly which groups to read, use the `option_groups` connection argument. The following example causes only the `[connector_python]` group to be read:

```
cnx = mysql.connector.connect(option_files='/etc/mysql/connectors.cnf',
                             option_groups='connector_python')
```

Other connection arguments specified in the `connect()` call take precedence over options read from option files. Suppose that `/etc/mysql/connectors.cnf` contains these lines:

```
[client]
database=cpyapp
```

The following `connect()` call includes no `database` connection argument. The resulting connection uses `cpyapp`, the database specified in the option file:

```
cnx = mysql.connector.connect(option_files='/etc/mysql/connectors.cnf')
```

By contrast, the following `connect()` call specifies a default database different from the one found in the option file. The resulting connection uses `cpyapp_dev` as the default database, not `cpyapp`:

```
cnx2 = mysql.connector.connect(option_files='/etc/mysql/connectors.cnf',
                              database='cpyapp_dev')
```

Connector/Python raises a `ValueError` if an option file cannot be read, or has already been read. This includes files read by inclusion directives.

For the `[connector_python]` group, only options supported by Connector/Python are accepted. Unrecognized options cause a `ValueError` to be raised.

For other option groups, Connector/Python ignores unrecognized options.

It is not an error for a named option group not to exist.

Connector/Python treats option values in option files as strings and evaluates them using `eval()`. This enables specification of option values more complex than simple scalars. For example, the `fabric` connection option requires a dictionary value, which can be specified like this:

```
[connector_python]
fabric={'host': 'localhost', 'username': 'webuser', 'password': 'webpass'}
```

Option values in option files must be specified on a single line, a constraint that does not apply to options specified directly in application code. For example, in an application, this is legal:

```
conn = mysql.connector.connect(
    fabric={'host': 'localhost',
           'username': 'webuser',
```

```
        'password': 'webpass'
    }
)
```

In an option file, attempting to use similar multiple-line formatting as shown here produces an error:

```
[connector_python]
fabric={'host': 'localhost',
        'username': 'webuser',
        'password': 'webpass'
}
```

5.8 Connector/Python Other Topics

This section describes additional Connection/Python features:

- Connection pooling: [Section 5.8.1, “Connector/Python Connection Pooling”](#)
- Django backend for MySQL: [Section 5.8.3, “Connector/Python Django Backend”](#)

5.8.1 Connector/Python Connection Pooling

MySQL Connector/Python 1.1.1 and up supports simple connection pooling that has these characteristics:

- The `mysql.connector.pooling` module implements pooling.
- A pool opens a number of connections and handles thread safety when providing connections to requesters.
- The size of a connection pool is configurable at pool creation time. It cannot be resized thereafter.
- A connection pool can be named at pool creation time. If no name is given, one is generated using the connection parameters.
- The connection pool name can be retrieved from the connection pool or connections obtained from it.
- It is possible to have multiple connection pools. This enables applications to support pools of connections to different MySQL servers, for example.
- For each connection request, the pool provides the next available connection. No round-robin or other scheduling algorithm is used. If a pool is exhausted, a `PoolError` is raised.
- It is possible to reconfigure the connection parameters used by a pool. These apply to connections obtained from the pool thereafter. Reconfiguring individual connections obtained from the pool by calling the connection `config()` method is not supported.

Applications that can benefit from connection-pooling capability include:

- Middleware that maintains multiple connections to multiple MySQL servers and requires connections to be readily available.
- Web sites that can have more “permanent” connections open to the MySQL server.

A connection pool can be created implicitly or explicitly.

To create a connection pool implicitly: Open a connection and specify one or more pool-related arguments (`pool_name`, `pool_size`). For example:

```
dbconfig = {
    "database": "test",
    "user":     "joe"
```

```

}
cnx = mysql.connector.connect(pool_name = "mypool",
                             pool_size = 3,
                             **dbconfig)

```

The pool name is restricted to alphanumeric characters and the special characters `.`, `_`, `*`, `$`, and `#`. The pool name must be no more than `pooling.CNX_POOL_MAXNAME_SIZE` characters long (default 64).

The pool size must be greater than 0 and less than `pooling.CNX_POOL_MAXSIZE` (default 32).

With either the `pool_name` or `pool_size` argument present, Connector/Python creates the new pool. If the `pool_name` argument is not given, the `connect()` call automatically generates the name, composed from whichever of the `host`, `port`, `user`, and `database` connection arguments are given, in that order. If the `pool_size` argument is not given, the default size is 5 connections.

Subsequent calls to `connect()` that name the same connection pool return connections from the existing pool. Any `pool_size` or connection parameter arguments are ignored, so the following `connect()` calls are equivalent to the original `connect()` call shown earlier:

```

cnx = mysql.connector.connect(pool_name = "mypool", pool_size = 3)
cnx = mysql.connector.connect(pool_name = "mypool", **dbconfig)
cnx = mysql.connector.connect(pool_name = "mypool")

```

Pooled connections obtained by calling `connect()` with a pool-related argument have a class of `PooledMySQLConnection` (see [Section 5.9.4, "pooling.PooledMySQLConnection Class"](#)). `PooledMySQLConnection` pooled connection objects are similar to `MySQLConnection` unpooled connection objects, with these differences:

- To release a pooled connection obtained from a connection pool, invoke its `close()` method, just as for any unpooled connection. However, for a pooled connection, `close()` does not actually close the connection but returns it to the pool and makes it available for subsequent connection requests.
- A pooled connection cannot be reconfigured using its `config()` method. Connection changes must be done through the pool object itself, as described shortly.
- A pooled connection has a `pool_name` property that returns the pool name.

To create a connection pool explicitly: Create a `MySQLConnectionPool` object (see [Section 5.9.3, "pooling.MySQLConnectionPool Class"](#)):

```

dbconfig = {
    "database": "test",
    "user": "joe"
}
cnxpool = mysql.connector.pooling.MySQLConnectionPool(pool_name = "mypool",
                                                       pool_size = 3,
                                                       **dbconfig)

```

To request a connection from the pool, use its `get_connection()` method:

```

cnx1 = cnxpool.get_connection()
cnx2 = cnxpool.get_connection()

```

When you create a connection pool explicitly, it is possible to use the pool object's `set_config()` method to reconfigure the pool connection parameters:

```

dbconfig = {
    "database": "performance_schema",
    "user": "admin",
    "password": "secret"
}

```



```
cnxpool.set_config(**dbconfig)
```

Connections requested from the pool after the configuration change use the new parameters. Connections obtained before the change remain unaffected, but when they are closed (returned to the pool) are reopened with the new parameters before being returned by the pool for subsequent connection requests.

5.8.2 Connector/Python Fabric Support

MySQL Fabric is a system for managing a farm of MySQL servers (and other components). Fabric provides an extensible and easy to use system for managing a MySQL deployment for sharding and high-availability.

Connector/Python 1.2.0 and up supports Fabric and provides these capabilities:

- Requesting a connection to a MySQL server managed by Fabric is as transparent as possible to users already familiar with Connector/Python.
- Connector/Python is able to get a MySQL server connection given a high-availability group and a mode specifying whether the connection is read-only or also permits updates (read-write).
- Connector/Python supports sharding and is able to find the correct MySQL server for a given table or tables and key based on scope (local or global) and mode (read-only or read-write). [RANGE](#) and [HASH](#) mechanisms are supported transparently to the user.
- Among secondary MySQL servers in the same group, load balancing of read-only operations occurs based on server weight.
- Faulty MySQL servers are reported to Fabric, and failover is supported when failure occurs for a server in a group.
- Connector/Python caches information coming from Fabric to speed up operations. Failures connecting to a MySQL server reset this cache. The time to live for cached information can be set by Fabric, or a default is used otherwise.
- Fabric support applies to versions of Python supported by Connector/Python itself (see [Section 5.3, “Connector/Python Versions”](#)). In particular, you can use Connector/Python with Python 3.1 and later to establish Fabric connections, even though Fabric does not support Python 3.

For more information, see [MySQL Fabric](#).

5.8.3 Connector/Python Django Backend

Connector/Python 1.1.1 and up includes a `mysql.connector.django` module that provides a Django backend for MySQL. This backend supports new features found in MySQL 5.6 such as fractional seconds support for temporal data types.

Django Configuration

Django uses a configuration file named `settings.py` that contains a variable called `DATABASES` (see <https://docs.djangoproject.com/en/1.5/ref/settings/#std:setting-DATABASES>). To configure Django to use Connector/Python as the MySQL backend, the example found in the Django manual can be used as a basis:

```
DATABASES = {
    'default': {
        'NAME': 'user_data',
        'ENGINE': 'mysql.connector.django',
        'USER': 'mysql_user',
        'PASSWORD': 'priv4te',
        'OPTIONS': {
```



```

        'autocommit': True,
    },
}

```

It is possible to add more connection arguments using [OPTIONS](#).

Support for MySQL Features

Django can launch the MySQL client application `mysql`. When the Connector/Python backend does this, it arranges for the `sql_mode` system variable to be set to `TRADITIONAL` at startup.

Some MySQL features are enabled depending on the server version. For example, support for fractional seconds precision is enabled when connecting to a server from MySQL 5.6.4 or higher. Django's `DateTimeField` is stored in a MySQL column defined as `DATETIME(6)`, and `TimeField` is stored as `TIME(6)`. For more information about fractional seconds support, see [Fractional Seconds in Time Values](#).

5.9 Connector/Python API Reference

This chapter contains the public API reference for Connector/Python. Examples should be considered working for Python 2.7, and Python 3.1 and greater. They might also work for older versions (such as Python 2.4) unless they use features introduced in newer Python versions. For example, exception handling using the `as` keyword was introduced in Python 2.6 and will not work in Python 2.4.

The following overview shows the `mysql.connector` package with its modules. Currently, only the most useful modules, classes, and methods for end users are documented.

```

mysql.connector
  errorcode
  errors
  connection
  constants
  conversion
  cursor
  dbapi
  locales
  eng
  client_error
  protocol
  utils

```

5.9.1 mysql.connector Module

The `mysql.connector` module provides top-level methods and properties.

5.9.1.1 mysql.connector.connect() Method

This method sets up a connection, establishing a session with the MySQL server. If no arguments are given, it uses the already configured or default values. For a complete list of possible arguments, see [Section 5.7.1, “Connector/Python Connection Arguments”](#).

A connection with the MySQL server can be established using either the `mysql.connector.connect()` method or the `mysql.connector.MySQLConnection()` class:

```

cnx = mysql.connector.connect(user='joe', database='test')
cnx = MySQLConnection(user='joe', database='test')

```

For descriptions of connection methods and properties, see [Section 5.9.2, “connection.MySQLConnection Class”](#).

5.9.1.2 mysql.connector.apilevel Property

This property is a string that indicates the supported DB API level.

```
>>> mysql.connector.apilevel
'2.0'
```

5.9.1.3 mysql.connector.paramstyle Property

This property is a string that indicates the Connector/Python default parameter style.

```
>>> mysql.connector.paramstyle
'pyformat'
```

5.9.1.4 mysql.connector.threadafety Property

This property is an integer that indicates the supported level of thread safety provided by Connector/Python.

```
>>> mysql.connector.threadafety
1
```

5.9.1.5 mysql.connector.__version__ Property

This property indicates the Connector/Python version as a string. It is available as of Connector/Python 1.1.0.

```
>>> mysql.connector.__version__
'1.1.0'
```

5.9.1.6 mysql.connector.__version_info__ Property

This property indicates the Connector/Python version as an array of version components. It is available as of Connector/Python 1.1.0.

```
>>> mysql.connector.__version_info__
(1, 1, 0, 'a', 0)
```

5.9.2 connection.MySQLConnection Class

The [MySQLConnection](#) class is used to open and manage a connection to a MySQL server. It also used to send commands and SQL statements and read the results.

5.9.2.1 connection.MySQLConnection() Constructor

Syntax:

```
cnx = MySQLConnection(**kwargs)
```

The [MySQLConnection](#) constructor initializes the attributes and when at least one argument is passed, it tries to connect to the MySQL server.

For a complete list of arguments, see [Section 5.7.1, “Connector/Python Connection Arguments”](#).

5.9.2.2 MySQLConnection.close() Method

Syntax:

```
cnx.close()
```

`close()` is a synonym for `disconnect()`. See [Section 5.9.2.20, “MySQLConnection.disconnect\(\) Method”](#).

For a connection obtained from a connection pool, `close()` does not actually close it but returns it to the pool and makes it available for subsequent connection requests. See [Section 5.8.1, “Connector/Python Connection Pooling”](#).

5.9.2.3 MySQLConnection.commit() Method

This method sends a `COMMIT` statement to the MySQL server, committing the current transaction. Since by default Connector/Python does not autocommit, it is important to call this method after every transaction that modifies data for tables that use transactional storage engines.

```
>>> cursor.execute("INSERT INTO employees (first_name) VALUES (%s)", ('Jane'))
>>> cnx.commit()
```

To roll back instead and discard modifications, see the [rollback\(\)](#) method.

5.9.2.4 MySQLConnection.config() Method

Syntax:

```
cnx.config(**kwargs)
```

Configures a `MySQLConnection` instance after it has been instantiated. For a complete list of possible arguments, see [Section 5.7.1, “Connector/Python Connection Arguments”](#).

Arguments:

- `kwargs`: Connection arguments.

You could use the `config()` method to change (for example) the user name, then call `reconnect()`.

Example:

```
cnx = mysql.connector.connect(user='joe', database='test')
# Connected as 'joe'
cnx.config(user='jane')
cnx.reconnect()
# Now connected as 'jane'
```

For a connection obtained from a connection pool, `config()` raises an exception. See [Section 5.8.1, “Connector/Python Connection Pooling”](#).

5.9.2.5 MySQLConnection.connect() Method

Syntax:

```
MySQLConnection.connect(**kwargs)
```

This method sets up a connection, establishing a session with the MySQL server. If no arguments are given, it uses the already configured or default values. For a complete list of possible arguments, see [Section 5.7.1, “Connector/Python Connection Arguments”](#).

Arguments:

- **kwargs**: Connection arguments.

Example:

```
cnx = MySQLConnection(user='joe', database='test')
```

For a connection obtained from a connection pool, the connection object class is [PooledMySQLConnection](#). A pooled connection differs from an unpooled connection as described in [Section 5.8.1, “Connector/Python Connection Pooling”](#).

5.9.2.6 MySQLConnection.cursor() Method

Syntax:

```
cursor = cnx.cursor([arg=value[, arg=value]...])
```

This method returns a [MySQLCursor\(\)](#) object, or a subclass of it depending on the passed arguments. The returned object is a [cursor.CursorBase](#) instance. For more information about cursor objects, see [Section 5.9.5, “cursor.MySQLCursor Class”](#), and [Section 5.9.6, “Subclasses cursor.MySQLCursor”](#).

Arguments may be passed to the [cursor\(\)](#) method to control what type of cursor to create:

- If **buffered** is **True**, the cursor fetches all rows from the server after an operation is executed. This is useful when queries return small result sets. **buffered** can be used alone, or in combination with the **dictionary** or **named_tuple** argument.

buffered can also be passed to [connect\(\)](#) to set the default buffering mode for all cursors created from the connection object. See [Section 5.7.1, “Connector/Python Connection Arguments”](#).

For information about the implications of buffering, see [Section 5.9.6.1, “cursor.MySQLCursorBuffered Class”](#).

- If **raw** is **True**, the cursor skips the conversion from MySQL data types to Python types when fetching rows. A raw cursor is usually used to get better performance or when you want to do the conversion yourself.

raw can also be passed to [connect\(\)](#) to set the default raw mode for all cursors created from the connection object. See [Section 5.7.1, “Connector/Python Connection Arguments”](#).
- If **dictionary** is **True**, the cursor returns rows as dictionaries. This argument is available as of Connector/Python 2.0.0.
- If **named_tuple** is **True**, the cursor returns rows as named tuples. This argument is available as of Connector/Python 2.0.0.
- If **prepared** is **True**, the cursor is used for executing prepared statements. This argument is available as of Connector/Python 1.1.2.
- The **cursor_class** argument can be used to pass a class to use for instantiating a new cursor. It must be a subclass of [cursor.CursorBase](#).

The returned object depends on the combination of the arguments. Examples:

- If not buffered and not raw: [MySQLCursor](#)
- If buffered and not raw: [MySQLCursorBuffered](#)
- If not buffered and raw: [MySQLCursorRaw](#)
- If buffered and raw: [MySQLCursorBufferedRaw](#)

5.9.2.7 MySQLConnection.cmd_change_user() Method

Changes the user using [username](#) and [password](#). It also causes the specified [database](#) to become the default (current) database. It is also possible to change the character set using the [charset](#) argument.

Syntax:

```
cnx.cmd_change_user(username='', password='', database='', charset=33)
```

Returns a dictionary containing the OK packet information.

5.9.2.8 MySQLConnection.cmd_debug() Method

Instructs the server to write debugging information to the error log. The connected user must have the [SUPER](#) privilege.

Returns a dictionary containing the OK packet information.

5.9.2.9 MySQLConnection.cmd_init_db() Method

Syntax:

```
cnx.cmd_init_db(db_name)
```

This method makes specified database the default (current) database. In subsequent queries, this database is the default for table references that include no explicit database qualifier.

Returns a dictionary containing the OK packet information.

5.9.2.10 MySQLConnection.cmd_ping() Method

Checks whether the connection to the server is working.

This method is not to be used directly. Use [ping\(\)](#) or [is_connected\(\)](#) instead.

Returns a dictionary containing the OK packet information.

5.9.2.11 MySQLConnection.cmd_process_info() Method

This method raises the `NotSupportedError` exception. Instead, use the [SHOW PROCESSLIST](#) statement or query the tables found in the database [INFORMATION_SCHEMA](#).

5.9.2.12 MySQLConnection.cmd_process_kill() Method

Syntax:

```
cnx.cmd_process_kill(mysql_pid)
```

Asks the server to kill the thread specified by [mysql_pid](#). Although still available, it is better to use the [KILL](#) SQL statement.

Returns a dictionary containing the OK packet information.

The following two lines have the same effect:

```
>>> cnx.cmd_process_kill(123)
>>> cnx.cmd_query('KILL 123')
```

5.9.2.13 MySQLConnection.cmd_query() Method

Syntax:

```
cnx.cmd_query(statement)
```

This method sends the given [statement](#) to the MySQL server and returns a result. To send multiple statements, use the [cmd_query_iter\(\)](#) method instead.

The returned dictionary contains information depending on what kind of query was executed. If the query is a [SELECT](#) statement, the result contains information about columns. Other statements return a dictionary containing OK or EOF packet information.

Errors received from the MySQL server are raised as exceptions. An [InterfaceError](#) is raised when multiple results are found.

Returns a dictionary.

5.9.2.14 MySQLConnection.cmd_query_iter() Method

Syntax:

```
cnx.cmd_query_iter(statement)
```

Similar to the [cmd_query\(\)](#) method, but returns a generator object to iterate through results. Use [cmd_query_iter\(\)](#) when sending multiple statements, and separate the statements with semicolons.

The following example shows how to iterate through the results after sending multiple statements:

```
statement = 'SELECT 1; INSERT INTO t1 VALUES (); SELECT 2'
for result in cnx.cmd_query_iter(statement):
    if 'columns' in result:
        columns = result['columns']
        rows = cnx.get_rows()
    else:
        # do something useful with INSERT result
```

Returns a generator object.

5.9.2.15 MySQLConnection.cmd_quit() Method

This method sends a [QUIT](#) command to the MySQL server, closing the current connection. Since there is no response from the MySQL server, the packet that was sent is returned.

5.9.2.16 MySQLConnection.cmd_refresh() Method

Syntax:

```
cnx.cmd_refresh(options)
```

This method flushes tables or caches, or resets replication server information. The connected user must have the [RELOAD](#) privilege.

The [options](#) argument should be a bitmask value constructed using constants from the [constants.RefreshOption](#) class.

For a list of options, see [Section 5.9.11, “constants.RefreshOption Class”](#).

Example:

```
>>> from mysql.connector import RefreshOption
>>> refresh = RefreshOption.LOG | RefreshOption.THREADS
>>> cnx.cmd_refresh(refresh)
```

5.9.2.17 MySQLConnection.cmd_reset_connection() Method

Syntax:

```
cnx.cmd_reset_connection()
```

Resets the connection by sending a [COM_RESET_CONNECTION](#) command to the server to clear the session state.

This method permits the session state to be cleared without reauthenticating. For MySQL servers older than 5.7.3 (when [COM_RESET_CONNECTION](#) was introduced), the [reset_session\(\)](#) method can be used instead. That method resets the session state by reauthenticating, which is more expensive.

This method was added in Connector/Python 1.2.1.

5.9.2.18 MySQLConnection.cmd_shutdown() Method

Asks the database server to shut down. The connected user must have the [SHUTDOWN](#) privilege.

Returns a dictionary containing the OK packet information.

5.9.2.19 MySQLConnection.cmd_statistics() Method

Returns a dictionary containing information about the MySQL server including uptime in seconds and the number of running threads, questions, reloads, and open tables.

5.9.2.20 MySQLConnection.disconnect() Method

This method tries to send a [QUIT](#) command and close the socket. It raises no exceptions.

[MySQLConnection.close\(\)](#) is a synonymous method name and more commonly used.

To shut down the connection without sending a [QUIT](#) command first, use [shutdown\(\)](#).

5.9.2.21 MySQLConnection.get_row() Method

This method retrieves the next row of a query result set, returning a tuple.

The tuple returned by [get_row\(\)](#) consists of:

- The row as a tuple containing byte objects, or [None](#) when no more rows are available.
- EOF packet information as a dictionary containing [status_flag](#) and [warning_count](#), or [None](#) when the row returned is not the last row.

The `get_row()` method is used by `MySQLCursor` to fetch rows.

5.9.2.22 MySQLConnection.get_rows() Method

Syntax:

```
cnx.get_rows(count=None)
```

This method retrieves all or remaining rows of a query result set, returning a tuple containing the rows as sequences and the EOF packet information. The count argument can be used to obtain a given number of rows. If count is not specified or is `None`, all rows are retrieved.

The tuple returned by `get_rows()` consists of:

- A list of tuples containing the row data as byte objects, or an empty list when no rows are available.
- EOF packet information as a dictionary containing `status_flag` and `warning_count`.

An `InterfaceError` is raised when all rows have been retrieved.

`MySQLCursor` uses the `get_rows()` method to fetch rows.

Returns a tuple.

5.9.2.23 MySQLConnection.get_server_info() Method

This method returns the MySQL server information verbatim as a string, for example `'5.6.11-log'`, or `None` when not connected.

5.9.2.24 MySQLConnection.get_server_version() Method

This method returns the MySQL server version as a tuple, or `None` when not connected.

5.9.2.25 MySQLConnection.is_connected() Method

Reports whether the connection to MySQL Server is available.

This method checks whether the connection to MySQL is available using the `ping()` method, but unlike `ping()`, `is_connected()` returns `True` when the connection is available, `False` otherwise.

5.9.2.26 MySQLConnection.isset_client_flag() Method

Syntax:

```
cnx.isset_client_flag(flag)
```

This method returns `True` if the client flag was set, `False` otherwise.

5.9.2.27 MySQLConnection.ping() Method

Syntax:

```
cnx.ping(attempts=1, delay=0)
```

Check whether the connection to the MySQL server is still available.

When `reconnect` is set to `True`, one or more attempts are made to try to reconnect to the MySQL server using the `reconnect()` method. Use the `delay` argument (seconds) if you want to wait between each retry.

When the connection is not available, an `InterfaceError` is raised. Use the `is_connected()` method to check the connection without raising an error.

Raises `InterfaceError` on errors.

5.9.2.28 MySQLConnection.reconnect() Method

Syntax:

```
cnx.reconnect(attempts=1, delay=0)
```

Attempt to reconnect to the MySQL server.

The argument `attempts` specifies the number of times a reconnect is tried. The `delay` argument is the number of seconds to wait between each retry.

You might set the number of attempts higher and use a longer delay when you expect the MySQL server to be down for maintenance, or when you expect the network to be temporarily unavailable.

5.9.2.29 MySQLConnection.reset_session() Method

Syntax:

```
cnx.reset_session(user_variables = None, session_variables = None)
```

Resets the connection by reauthenticating to clear the session state. `user_variables`, if given, is a dictionary of user variable names and values. `session_variables`, if given, is a dictionary of system variable names and values. The method sets each variable to the given value.

Example:

```
user_variables = {'var1': '1', 'var2': '10'}
session_variables = {'wait_timeout': 100000, 'sql_mode': 'TRADITIONAL'}
self.cnx.reset_session(user_variables, session_variables)
```

This method resets the session state by reauthenticating, which is expensive. For MySQL servers 5.7.3 or later, the `cmd_reset_connection()` method can be used instead. It is more lightweight because it permits the session state to be cleared without reauthenticating.

This method was added in Connector/Python 1.2.1.

5.9.2.30 MySQLConnection.rollback() Method

This method sends a `ROLLBACK` statement to the MySQL server, undoing all data changes from the current transaction. By default, Connector/Python does not autoccommit, so it is possible to cancel transactions when using transactional storage engines such as `InnoDB`.

```
>>> cursor.execute("INSERT INTO employees (first_name) VALUES (%s)", ('Jane'))
>>> cnx.rollback()
```

To `commit` modifications, see the `commit()` method.

5.9.2.31 MySQLConnection.set_charset_collation() Method

Syntax:

```
cnx.set_charset_collation(charset=None, collation=None)
```

This method sets the character set and collation to be used for the current connection. The `charset` argument can be either the name of a character set, or the numerical equivalent as defined in `constants.CharacterSet`.

When `collation` is `None`, the default collation for the character set is used.

In the following example, we set the character set to `latin1` and the collation to `latin1_swedish_ci` (the default collation for: `latin1`):

```
>>> cnx = mysql.connector.connect(user='scott')
>>> cnx.set_charset_collation('latin1')
```

Specify a given collation as follows:

```
>>> cnx = mysql.connector.connect(user='scott')
>>> cnx.set_charset_collation('latin1', 'latin1_general_ci')
```

5.9.2.32 MySQLConnection.set_client_flags() Method

Syntax:

```
cnx.set_client_flags(flags)
```

This method sets the client flags to use when connecting to the MySQL server, and returns the new value as an integer. The `flags` argument can be either an integer or a sequence of valid client flag values (see [Section 5.9.7, “constants.ClientFlag Class”](#)).

If `flags` is a sequence, each item in the sequence sets the flag when the value is positive or unsets it when negative. For example, to unset `LONG_FLAG` and set the `FOUND_ROWS` flags:

```
>>> from mysql.connector.constants import ClientFlag
>>> cnx.set_client_flags([ClientFlag.FOUND_ROWS, -ClientFlag.LONG_FLAG])
>>> cnx.reconnect()
```

Note

Client flags are only set or used when connecting to the MySQL server. It is therefore necessary to reconnect after making changes.

5.9.2.33 MySQLConnection.shutdown() Method

This method closes the socket. It raises no exceptions.

Unlike `disconnect()`, `shutdown()` closes the client connection without attempting to send a `QUIT` command to the server first. Thus, it will not block if the connection is disrupted for some reason such as network failure.

`shutdown()` was added in Connector/Python 2.0.1.

5.9.2.34 MySQLConnection.start_transaction() Method

This method starts a transaction. It accepts arguments indicating whether to use a consistent snapshot, which transaction isolation level to use, and the transaction access mode:

```
cnx.start_transaction(consistent_snapshot=bool,
                      isolation_level=level,
                      readonly=access_mode)
```

The default `consistent_snapshot` value is `False`. If the value is `True`, Connector/Python sends `WITH CONSISTENT SNAPSHOT` with the statement. MySQL ignores this for isolation levels for which that option does not apply.

The default `isolation_level` value is `None`, and permitted values are `'READ UNCOMMITTED'`, `'READ COMMITTED'`, `'REPEATABLE READ'`, and `'SERIALIZABLE'`. If the `isolation_level` value is `None`, no isolation level is sent, so the default level applies.

The `readonly` argument can be `True` to start the transaction in `READ ONLY` mode or `False` to start it in `READ WRITE` mode. If `readonly` is omitted, the server's default access mode is used. For details about transaction access mode, see the description for the `START TRANSACTION` statement at [START TRANSACTION, COMMIT, and ROLLBACK Syntax](#). If the server is older than MySQL 5.6.5, it does not support setting the access mode and Connector/Python raises a `ValueError`.

Invoking `start_transaction()` raises a `ProgrammingError` if invoked while a transaction is currently in progress. This differs from executing a `START TRANSACTION` SQL statement while a transaction is in progress; the statement implicitly commits the current transaction.

To determine whether a transaction is active for the connection, use the `in_transaction` property.

`start_transaction()` was added in MySQL Connector/Python 1.1.0. The `readonly` argument was added in Connector/Python 1.1.5.

5.9.2.35 MySQLConnection.autocommit Property

This property can be assigned a value of `True` or `False` to enable or disable the autocommit feature of MySQL. The property can be invoked to retrieve the current autocommit setting.

Note

Autocommit is disabled by default when connecting through Connector/Python. This can be enabled using the [autocommit connection parameter](#).

When the autocommit is turned off, you must `commit` transactions when using transactional storage engines such as `InnoDB` or `NDBCluster`.

```
>>> cnx.autocommit
False
>>> cnx.autocommit = True
>>> cnx.autocommit
True
```

5.9.2.36 MySQLConnection.can_consume_results Property

This property indicates the value of the `consume_results` connection parameter that controls whether result sets produced by queries are automatically read and discarded. See [Section 5.7.1, “Connector/Python Connection Arguments”](#).

This method was added in Connector/Python 2.1.1.

5.9.2.37 MySQLConnection.charset Property

This property returns a string indicating which character set is used for the connection, whether or not it is connected.

5.9.2.38 MySQLConnection.collation Property

This property returns a string indicating which collation is used for the connection, whether or not it is connected.

5.9.2.39 MySQLConnection.connection_id Property

This property returns the integer connection ID (thread ID or session ID) for the current connection or [None](#) when not connected.

5.9.2.40 MySQLConnection.database Property

This property sets the current (default) database by executing a [USE](#) statement. The property can also be used to retrieve the current database name.

```
>>> cnx.database = 'test'
>>> cnx.database = 'mysql'
>>> cnx.database
u'mysql'
```

Returns a string.

5.9.2.41 MySQLConnection.get_warnings Property

This property can be assigned a value of [True](#) or [False](#) to enable or disable whether warnings should be fetched automatically. The default is [False](#) (default). The property can be invoked to retrieve the current warnings setting.

Fetching warnings automatically can be useful when debugging queries. Cursors make warnings available through the method [MySQLCursor.fetchwarnings\(\)](#).

```
>>> cnx.get_warnings = True
>>> cursor.execute('SELECT "a"+1')
>>> cursor.fetchall()
[(1.0,)]
>>> cursor.fetchwarnings()
[(u'Warning', 1292, u'Truncated incorrect DOUBLE value: 'a'")]
```

Returns [True](#) or [False](#).

5.9.2.42 MySQLConnection.in_transaction Property

This property returns [True](#) or [False](#) to indicate whether a transaction is active for the connection. The value is [True](#) regardless of whether you start a transaction using the [start_transaction\(\)](#) API call or by directly executing an SQL statement such as [START TRANSACTION](#) or [BEGIN](#).

```
>>> cnx.start_transaction()
>>> cnx.in_transaction
True
>>> cnx.commit()
>>> cnx.in_transaction
False
```

[in_transaction](#) was added in MySQL Connector/Python 1.1.0.

5.9.2.43 MySQLConnection.raise_on_warnings Property

This property can be assigned a value of `True` or `False` to enable or disable whether warnings should raise exceptions. The default is `False` (default). The property can be invoked to retrieve the current exceptions setting.

Setting `raise_on_warnings` also sets `get_warnings` because warnings need to be fetched so they can be raised as exceptions.

Note

You might always want to set the SQL mode if you would like to have the MySQL server directly report warnings as errors (see [Section 5.9.2.46](#), “`MySQLConnection.sql_mode` Property”). It is also good to use transactional engines so transactions can be rolled back when catching the exception.

Result sets needs to be fetched completely before any exception can be raised. The following example shows the execution of a query that produces a warning:

```
>>> cnx.raise_on_warnings = True
>>> cursor.execute('SELECT "a"+1')
>>> cursor.fetchall()
..
mysql.connector.errors.DataError: 1292: Truncated incorrect DOUBLE value: 'a'
```

Returns `True` or `False`.

5.9.2.44 `MySQLConnection.server_host` Property

This read-only property returns the host name or IP address used for connecting to the MySQL server.

Returns a string.

5.9.2.45 `MySQLConnection.server_port` Property

This read-only property returns the TCP/IP port used for connecting to the MySQL server.

Returns an integer.

5.9.2.46 `MySQLConnection.sql_mode` Property

This property is used to retrieve and set the SQL Modes for the current connection. The value should be a list of different modes separated by comma (","), or a sequence of modes, preferably using the `constants.SQLMode` class.

To unset all modes, pass an empty string or an empty sequence.

```
>>> cnx.sql_mode = 'TRADITIONAL,NO_ENGINE_SUBSTITUTION'
>>> cnx.sql_mode.split(',')
[u'STRICT_TRANS_TABLES', u'STRICT_ALL_TABLES', u'NO_ZERO_IN_DATE',
u'NO_ZERO_DATE', u'ERROR_FOR_DIVISION_BY_ZERO', u'TRADITIONAL',
u'NO_AUTO_CREATE_USER', u'NO_ENGINE_SUBSTITUTION']
>>> from mysql.connector.constants import SQLMode
>>> cnx.sql_mode = [ SQLMode.NO_ZERO_DATE, SQLMode.REAL_AS_FLOAT]
>>> cnx.sql_mode
u'REAL_AS_FLOAT,NO_ZERO_DATE'
```

Returns a string.

5.9.2.47 `MySQLConnection.time_zone` Property

This property is used to set or retrieve the time zone session variable for the current connection.

```
>>> cnx.time_zone = '+00:00'
>>> cursor = cnx.cursor()
>>> cursor.execute('SELECT NOW()') ; cursor.fetchone()
(datetime.datetime(2012, 6, 15, 11, 24, 36),)
>>> cnx.time_zone = '-09:00'
>>> cursor.execute('SELECT NOW()') ; cursor.fetchone()
(datetime.datetime(2012, 6, 15, 2, 24, 44),)
>>> cnx.time_zone
u'-09:00'
```

Returns a string.

5.9.2.48 MySQLConnection.unix_socket Property

This read-only property returns the Unix socket file for connecting to the MySQL server.

Returns a string.

5.9.2.49 MySQLConnection.user Property

This read-only property returns the user name used for connecting to the MySQL server.

Returns a string.

5.9.3 pooling.MySQLConnectionPool Class

This class provides for the instantiation and management of connection pools.

5.9.3.1 pooling.MySQLConnectionPool Constructor

Syntax:

```
MySQLConnectionPool(pool_name=None,
                    pool_size=5,
                    pool_reset_session=True,
                    **kwargs)
```

This constructor instantiates an object that manages a connection pool.

Arguments:

- **pool_name**: The pool name. If this argument is not given, Connector/Python automatically generates the name, composed from whichever of the **host**, **port**, **user**, and **database** connection arguments are given in **kwargs**, in that order.

It is not an error for multiple pools to have the same name. An application that must distinguish pools by their **pool_name** property should create each pool with a distinct name.

- **pool_size**: The pool size. If this argument is not given, the default is 5.
- **pool_reset_session**: Whether to reset session variables when the connection is returned to the pool. This argument was added in Connector/Python 1.1.5. Before 1.1.5, session variables are not reset.
- **kwargs**: Optional additional connection arguments, as described in [Section 5.7.1, “Connector/Python Connection Arguments”](#).

Example:

```
dbconfig = {
    "database": "test",
    "user": "joe",
}
cnxpool = mysql.connector.pooling.MySQLConnectionPool(pool_name = "mypool",
                                                         pool_size = 3,
                                                         **dbconfig)
```

5.9.3.2 MySQLConnectionPool.add_connection() Method

Syntax:

```
cnxpool.add_connection(cnx = None)
```

This method adds a new or existing [MySQLConnection](#) to the pool, or raises a [PoolError](#) if the pool is full.

Arguments:

- **cnx**: The [MySQLConnection](#) object to be added to the pool. If this argument is missing, the pool creates a new connection and adds it.

Example:

```
cnxpool.add_connection() # add new connection to pool
cnxpool.add_connection(cnx) # add existing connection to pool
```

5.9.3.3 MySQLConnectionPool.get_connection() Method

Syntax:

```
cnxpool.get_connection()
```

This method returns a connection from the pool, or raises a [PoolError](#) if no connections are available.

Example:

```
cnx = cnxpool.get_connection()
```

5.9.3.4 MySQLConnectionPool.set_config() Method

Syntax:

```
cnxpool.set_config(**kwargs)
```

This method sets the configuration parameters for connections in the pool. Connections requested from the pool after the configuration change use the new parameters. Connections obtained before the change remain unaffected, but when they are closed (returned to the pool) are reopened with the new parameters before being returned by the pool for subsequent connection requests.

Arguments:

- **kwargs**: Connection arguments.

Example:

```
dbconfig = {
```

```
"database": "performance_schema",  
"user": "admin",  
"password": "secret",  
}  
cnxpool.set_config(**dbconfig)
```

5.9.3.5 MySQLConnectionPool.pool_name Property

Syntax:

```
cnxpool.pool_name
```

This property returns the connection pool name.

Example:

```
name = cnxpool.pool_name
```

5.9.4 pooling.PooledMySQLConnection Class

This class is used by `MySQLConnectionPool` to return a pooled connection instance. It is also the class used for connections obtained with calls to the `connect()` method that name a connection pool (see [Section 5.8.1, “Connector/Python Connection Pooling”](#)).

`PooledMySQLConnection` pooled connection objects are similar to `MySQLConnection` unpooled connection objects, with these differences:

- To release a pooled connection obtained from a connection pool, invoke its `close()` method, just as for any unpooled connection. However, for a pooled connection, `close()` does not actually close the connection but returns it to the pool and makes it available for subsequent connection requests.
- A pooled connection cannot be reconfigured using its `config()` method. Connection changes must be done through the pool object itself, as described shortly.
- A pooled connection has a `pool_name` property that returns the pool name.

5.9.4.1 pooling.PooledMySQLConnection Constructor

Syntax:

```
PooledMySQLConnection(cnxpool, cnx)
```

This constructor takes connection pool and connection arguments and returns a pooled connection. It is used by the `MySQLConnectionPool` class.

Arguments:

- `cnxpool`: A `MySQLConnectionPool` instance.
- `cnx`: A `MySQLConnection` instance.

Example:

```
pcnx = mysql.connector.pooling.PooledMySQLConnection(cnxpool, cnx)
```

5.9.4.2 PooledMySQLConnection.close() Method

Syntax:

```
cnx.close()
```

Returns a pooled connection to its connection pool.

For a pooled connection, `close()` does not actually close it but returns it to the pool and makes it available for subsequent connection requests.

If the pool configuration parameters are changed, a returned connection is closed and reopened with the new configuration before being returned from the pool again in response to a connection request.

5.9.4.3 PooledMySQLConnection.config() Method

For pooled connections, the `config()` method raises a `PoolError` exception. Configuration for pooled connections should be done using the pool object.

5.9.4.4 PooledMySQLConnection.pool_name Property

Syntax:

```
cnx.pool_name
```

This property returns the name of the connection pool to which the connection belongs.

Example:

```
cnx = cnxpool.get_connection()
name = cnx.pool_name
```

5.9.5 cursor.MySQLCursor Class

The `MySQLCursor` class instantiates objects that can execute operations such as SQL statements. Cursor objects interact with the MySQL server using a `MySQLConnection` object.

To create a cursor, use the `cursor()` method of a connection object:

```
import mysql.connector
cnx = mysql.connector.connect(database='world')
cursor = cnx.cursor()
```

Several related classes inherit from `MySQLCursor`. To create a cursor of one of these types, pass the appropriate arguments to `cursor()`:

- `MySQLCursorBuffered` creates a buffered cursor. See [Section 5.9.6.1, “cursor.MySQLCursorBuffered Class”](#).

```
cursor = cnx.cursor(buffered=True)
```

- `MySQLCursorRaw` creates a raw cursor. See [Section 5.9.6.2, “cursor.MySQLCursorRaw Class”](#).

```
cursor = cnx.cursor(raw=True)
```

- `MySQLCursorBufferedRaw` creates a buffered raw cursor. See [Section 5.9.6.3, “cursor.MySQLCursorBufferedRaw Class”](#).

```
cursor = cnx.cursor(raw=True, buffered=True)
```

- `MySQLCursorDict` creates a cursor that returns rows as dictionaries. See [Section 5.9.6.4, “cursor.MySQLCursorDict Class”](#).

```
cursor = cnx.cursor(dictionary=True)
```

- [MySQLCursorBufferedDict](#) creates a buffered cursor that returns rows as dictionaries. See [Section 5.9.6.5, “cursor.MySQLCursorBufferedDict Class”](#).

```
cursor = cnx.cursor(dictionary=True, buffered=True)
```

- [MySQLCursorNamedTuple](#) creates a cursor that returns rows as named tuples. See [Section 5.9.6.6, “cursor.MySQLCursorNamedTuple Class”](#).

```
cursor = cnx.cursor(named_tuple=True)
```

- [MySQLCursorBufferedNamedTuple](#) creates a buffered cursor that returns rows as named tuples. See [Section 5.9.6.7, “cursor.MySQLCursorBufferedNamedTuple Class”](#).

```
cursor = cnx.cursor(named_tuple=True, buffered=True)
```

- [MySQLCursorPrepared](#) creates a cursor for executing prepared statements. See [Section 5.9.6.8, “cursor.MySQLCursorPrepared Class”](#).

```
cursor = cnx.cursor(prepared=True)
```

5.9.5.1 cursor.MySQLCursor Constructor

In most cases, the [MySQLConnection](#) [cursor\(\)](#) method is used to instantiate a [MySQLCursor](#) object:

```
import mysql.connector
cnx = mysql.connector.connect(database='world')
cursor = cnx.cursor()
```

It is also possible to instantiate a cursor by passing a [MySQLConnection](#) object to [MySQLCursor](#):

```
import mysql.connector
from mysql.connector.cursor import MySQLCursor
cnx = mysql.connector.connect(database='world')
cursor = MySQLCursor(cnx)
```

The connection argument is optional. If omitted, the cursor is created but its [execute\(\)](#) method raises an exception.

5.9.5.2 MySQLCursor.callproc() Method

Syntax:

```
result_args = cursor.callproc(proc_name, args=())
```

This method calls the stored procedure named by the [proc_name](#) argument. The [args](#) sequence of parameters must contain one entry for each argument that the procedure expects. [callproc\(\)](#) returns a modified copy of the input sequence. Input parameters are left untouched. Output and input/output parameters may be replaced with new values.

Result sets produced by the stored procedure are automatically fetched and stored as [MySQLCursorBuffered](#) instances. For more information about using these result sets, see [stored_results\(\)](#).

Suppose that a stored procedure takes two parameters, multiplies the values, and returns the product:

```
CREATE PROCEDURE multiply(IN pFac1 INT, IN pFac2 INT, OUT pProd INT)
BEGIN
    SET pProd := pFac1 * pFac2;
END;
```

The following example shows how to execute the `multiply()` procedure:

```
>>> args = (5, 6, 0) # 0 is to hold value of the OUT parameter pProd
>>> cursor.callproc('multiply', args)
('5', '6', 30L)
```

Connector/Python 1.2.1 and up permits parameter types to be specified. To do this, specify a parameter as a two-item tuple consisting of the parameter value and type. Suppose that a procedure `sp1()` has this definition:

```
CREATE PROCEDURE sp1(IN pStr1 VARCHAR(20), IN pStr2 VARCHAR(20),
                    OUT pConcat VARCHAR(100))
BEGIN
    SET pConcat := CONCAT(pStr1, pStr2);
END;
```

To execute this procedure from Connector/Python, specifying a type for the `OUT` parameter, do this:

```
args = ('ham', 'eggs', (0, 'CHAR'))
result_args = cursor.callproc('sp1', args)
print(result_args[2])
```

5.9.5.3 MySQLCursor.close() Method

Syntax:

```
cursor.close()
```

Use `close()` when you are done using a cursor. This method closes the cursor, resets all results, and ensures that the cursor object has no reference to its original connection object.

5.9.5.4 MySQLCursor.execute() Method

Syntax:

```
cursor.execute(operation, params=None, multi=False)
iterator = cursor.execute(operation, params=None, multi=True)
```

This method executes the given database `operation` (query or command). The parameters found in the tuple or dictionary `params` are bound to the variables in the operation. Specify variables using `%s` or `%(name)s` parameter style (that is, using `format` or `pyformat` style). `execute()` returns an iterator if `multi` is `True`.

This example inserts information about a new employee, then selects the data for that person. The statements are executed as separate `execute()` operations:

```
insert_stmt = (
    "INSERT INTO employees (emp_no, first_name, last_name, hire_date) "
    "VALUES (%s, %s, %s, %s)"
)
data = (2, 'Jane', 'Doe', datetime.date(2012, 3, 23))
cursor.execute(insert_stmt, data)
select_stmt = "SELECT * FROM employees WHERE emp_no = %(emp_no)s"
cursor.execute(select_stmt, { 'emp_no': 2 })
```

The data values are converted as necessary from Python objects to something MySQL understands. In the preceding example, the `datetime.date()` instance is converted to `'2012-03-23'`.

If `multi` is set to `True`, `execute()` is able to execute multiple statements specified in the `operation` string. It returns an iterator that enables processing the result of each statement. However, using parameters does not work well in this case, and it is usually a good idea to execute each statement on its own.

The following example selects and inserts data in a single `execute()` operation and displays the result of each statement:

```
operation = 'SELECT 1; INSERT INTO t1 VALUES (); SELECT 2'
for result in cursor.execute(operation, multi=True):
    if result.with_rows:
        print("Rows produced by statement '{}':".format(
            result.statement))
        print(result.fetchall())
    else:
        print("Number of rows affected by statement '{}': {}".format(
            result.statement, result.rowcount))
```

If the connection is configured to fetch warnings, warnings generated by the operation are available through the `MySQLCursor.fetchwarnings()` method.

5.9.5.5 MySQLCursor.executemany() Method

Syntax:

```
cursor.executemany(operation, seq_of_params)
```

This method prepares a database `operation` (query or command) and executes it against all parameter sequences or mappings found in the sequence `seq_of_params`.

In most cases, the `executemany()` method iterates through the sequence of parameters, each time passing the current parameters to the `execute()` method.

An optimization is applied for inserts: The data values given by the parameter sequences are batched using multiple-row syntax. The following example inserts three records:

```
data = [
    ('Jane', date(2005, 2, 12)),
    ('Joe', date(2006, 5, 23)),
    ('John', date(2010, 10, 3)),
]
stmt = "INSERT INTO employees (first_name, hire_date) VALUES (%s, %s)"
cursor.executemany(stmt, data)
```

For the preceding example, the `INSERT` statement sent to MySQL is:

```
INSERT INTO employees (first_name, hire_date)
VALUES ('Jane', '2005-02-12'), ('Joe', '2006-05-23'), ('John', '2010-10-03')
```

With the `executemany()` method, it is not possible to specify multiple statements to execute in the `operation` argument. Doing so raises an `InternalError` exception. Consider using `execute()` with `multi=True` instead.

5.9.5.6 MySQLCursor.fetchall() Method

Syntax:

```
rows = cursor.fetchall()
```

The method fetches all (or all remaining) rows of a query result set and returns a list of tuples. If no more rows are available, it returns an empty list.

The following example shows how to retrieve the first two rows of a result set, and then retrieve any remaining rows:

```
>>> cursor.execute("SELECT * FROM employees ORDER BY emp_no")
>>> head_rows = cursor.fetchmany(size=2)
>>> remaining_rows = cursor.fetchall()
```

You must fetch all rows for the current query before executing new statements using the same connection.

5.9.5.7 MySQLCursor.fetchmany() Method

Syntax:

```
rows = cursor.fetchmany(size=1)
```

This method fetches the next set of rows of a query result and returns a list of tuples. If no more rows are available, it returns an empty list.

The number of rows returned can be specified using the `size` argument, which defaults to one. Fewer rows are returned if fewer rows are available than specified.

You must fetch all rows for the current query before executing new statements using the same connection.

5.9.5.8 MySQLCursor.fetchone() Method

Syntax:

```
row = cursor.fetchone()
```

This method retrieves the next row of a query result set and returns a single sequence, or `None` if no more rows are available. By default, the returned tuple consists of data returned by the MySQL server, converted to Python objects. If the cursor is a raw cursor, no such conversion occurs; see [Section 5.9.6.2, “cursor.MySQLCursorRaw Class”](#).

The `fetchone()` method is used by `fetchall()` and `fetchmany()`. It is also used when a cursor is used as an iterator.

The following example shows two equivalent ways to process a query result. The first uses `fetchone()` in a `while` loop, the second uses the cursor as an iterator:

```
# Using a while loop
cursor.execute("SELECT * FROM employees")
row = cursor.fetchone()
while row is not None:
    print(row)
    row = cursor.fetchone()
# Using the cursor as iterator
cursor.execute("SELECT * FROM employees")
for row in cursor:
    print(row)
```

You must fetch all rows for the current query before executing new statements using the same connection.

5.9.5.9 MySQLCursor.fetchwarnings() Method

Syntax:

```
tuples = cursor.fetchwarnings()
```

This method returns a list of tuples containing warnings generated by the previously executed operation. To set whether to fetch warnings, use the connection's [get_warnings](#) property.

The following example shows a [SELECT](#) statement that generates a warning:

```
>>> cnx.get_warnings = True
>>> cursor.execute("SELECT 'a'+1")
>>> cursor.fetchall()
[(1,0,)]
>>> cursor.fetchwarnings()
[(u'Warning', 1292, u'Truncated incorrect DOUBLE value: 'a'")]
```

When warnings are generated, it is possible to raise errors instead, using the connection's [raise_on_warnings](#) property.

5.9.5.10 MySQLCursor.stored_results() Method

Syntax:

```
iterator = cursor.stored_results()
```

This method returns a list iterator object that can be used to process result sets produced by a stored procedure executed using the [callproc\(\)](#) method. The result sets remain available until you use the cursor to execute another operation or call another stored procedure.

The following example executes a stored procedure that produces two result sets, then uses [stored_results\(\)](#) to retrieve them:

```
>>> cursor.callproc('myproc')
()
>>> for result in cursor.stored_results():
...     print result.fetchall()
...
[(1,)]
[(2,)]
```

5.9.5.11 MySQLCursor.column_names Property

Syntax:

```
sequence = cursor.column_names
```

This read-only property returns the column names of a result set as sequence of Unicode strings.

The following example shows how to create a dictionary from a tuple containing data with keys using [column_names](#):

```
cursor.execute("SELECT last_name, first_name, hire_date "
               "FROM employees WHERE emp_no = %s", (123,))
row = dict(zip(cursor.column_names, cursor.fetchone()))
print("{last_name}, {first_name}: {hire_date}".format(row))
```

Alternatively, as of Connector/Python 2.0.0, you can fetch rows as dictionaries directly; see [Section 5.9.6.4, “cursor.MySQLCursorDict Class”](#).

5.9.5.12 MySQLCursor.description Property

Syntax:

```
tuples = cursor.description
```

This read-only property returns a list of tuples describing the columns in a result set. Each tuple in the list contains values as follows:

```
(column_name,  
type,  
None,  
None,  
None,  
None,  
null_ok,  
column_flags)
```

The following example shows how to interpret `description` tuples:

```
import mysql.connector  
from mysql.connector import FieldType  
...  
cursor.execute("SELECT emp_no, last_name, hire_date "  
               "FROM employees WHERE emp_no = %s", (123,))  
for i in range(len(cursor.description)):  
    print("Column {}".format(i+1))  
    desc = cursor.description[i]  
    print("  column_name = {}".format(desc[0]))  
    print("  type = {} ({}).format(desc[1], FieldType.get_info(desc[1]))  
    print("  null_ok = {}".format(desc[6]))  
    print("  column_flags = {}".format(desc[7]))
```

The output looks like this:

```
Column 1:  
  column_name = emp_no  
  type = 3 (LONG)  
  null_ok = 0  
  column_flags = 20483  
Column 2:  
  column_name = last_name  
  type = 253 (VAR_STRING)  
  null_ok = 0  
  column_flags = 4097  
Column 3:  
  column_name = hire_date  
  type = 10 (DATE)  
  null_ok = 0  
  column_flags = 4225
```

The `column_flags` value is an instance of the `constants.FieldFlag` class. To see how to interpret it, do this:

```
>>> from mysql.connector import FieldFlag  
>>> FieldFlag.desc
```

5.9.5.13 MySQLCursor.lastrowid Property

Syntax:

```
id = cursor.lastrowid
```

This read-only property returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` or `UPDATE` statement or `None` when there is no such value available. For example, if you

perform an `INSERT` into a table that contains an `AUTO_INCREMENT` column, `lastrowid` returns the `AUTO_INCREMENT` value for the new row. For an example, see [Section 5.5.3, “Inserting Data Using Connector/Python”](#).

The `lastrowid` property is like the `mysql_insert_id()` C API function; see `mysql_insert_id()`.

5.9.5.14 MySQLCursor.rowcount Property

Syntax:

```
count = cursor.rowcount
```

This read-only property returns the number of rows returned for `SELECT` statements, or the number of rows affected by DML statements such as `INSERT` or `UPDATE`. For an example, see [Section 5.9.5.4, “MySQLCursor.execute\(\) Method”](#).

For nonbuffered cursors, the row count cannot be known before the rows have been fetched. In this case, the number of rows is -1 immediately after query execution and is incremented as rows are fetched.

The `rowcount` property is like the `mysql_affected_rows()` C API function; see `mysql_affected_rows()`.

5.9.5.15 MySQLCursor.statement Property

Syntax:

```
str = cursor.statement
```

This read-only property returns the last executed statement as a string. The `statement` property can be useful for debugging and displaying what was sent to the MySQL server.

The string can contain multiple statements if a multiple-statement string was executed. This occurs for `execute()` with `multi=True`. In this case, the `statement` property contains the entire statement string and the `execute()` call returns an iterator that can be used to process results from the individual statements. The `statement` property for this iterator shows statement strings for the individual statements.

5.9.5.16 MySQLCursor.with_rows Property

Syntax:

```
boolean = cursor.with_rows
```

This read-only property returns `True` or `False` to indicate whether the most recently executed operation produced rows.

The `with_rows` property is useful when it is necessary to determine whether a statement produces a result set and you need to fetch rows. The following example retrieves the rows returned by the `SELECT` statements, but reports only the affected-rows value for the `UPDATE` statement:

```
import mysql.connector
cnx = mysql.connector.connect(user='scott', database='test')
cursor = cnx.cursor()
operation = 'SELECT 1; UPDATE t1 SET c1 = 2; SELECT 2'
for result in cursor.execute(operation, multi=True):
    if result.with_rows:
        result.fetchall()
    else:
        print("Number of affected rows: {}".format(result.rowcount))
```


5.9.6 Subclasses `cursor.MySQLCursor`

The cursor classes described in the following sections inherit from the `MySQLCursor` class, which is described in [Section 5.9.5, “`cursor.MySQLCursor` Class”](#).

5.9.6.1 `cursor.MySQLCursorBuffered` Class

The `MySQLCursorBuffered` class inherits from `MySQLCursor`.

After executing a query, a `MySQLCursorBuffered` cursor fetches the entire result set from the server and buffers the rows.

For queries executed using a buffered cursor, row-fetching methods such as `fetchone()` return rows from the set of buffered rows. For nonbuffered cursors, rows are not fetched from the server until a row-fetching method is called. In this case, you must be sure to fetch all rows of the result set before executing any other statements on the same connection, or an `InternalError` (Unread result found) exception will be raised.

`MySQLCursorBuffered` can be useful in situations where multiple queries, with small result sets, need to be combined or computed with each other.

To create a buffered cursor, use the `buffered` argument when calling a connection's `cursor()` method. Alternatively, to make all cursors created from the connection buffered by default, use the [buffered connection argument](#).

Example:

```
import mysql.connector
cnx = mysql.connector.connect()
# Only this particular cursor will buffer results
cursor = cnx.cursor(buffered=True)
# All cursors created from cnx2 will be buffered by default
cnx2 = mysql.connector.connect(buffered=True)
```

For a practical use case, see [Section 5.6.1, “Tutorial: Raise Employee's Salary Using a Buffered Cursor”](#).

5.9.6.2 `cursor.MySQLCursorRaw` Class

The `MySQLCursorRaw` class inherits from `MySQLCursor`.

A `MySQLCursorRaw` cursor skips the conversion from MySQL data types to Python types when fetching rows. A raw cursor is usually used to get better performance or when you want to do the conversion yourself.

To create a raw cursor, use the `raw` argument when calling a connection's `cursor()` method. Alternatively, to make all cursors created from the connection raw by default, use the [raw connection argument](#).

Example:

```
import mysql.connector
cnx = mysql.connector.connect()
# Only this particular cursor will be raw
cursor = cnx.cursor(raw=True)
# All cursors created from cnx2 will be raw by default
cnx2 = mysql.connector.connect(raw=True)
```

5.9.6.3 `cursor.MySQLCursorBufferedRaw` Class

The `MySQLCursorBufferedRaw` class inherits from `MySQLCursor`.

A `MySQLCursorBufferedRaw` cursor is like a `MySQLCursorRaw` cursor, but is buffered: After executing a query, it fetches the entire result set from the server and buffers the rows. For information about the implications of buffering, see [Section 5.9.6.1, “cursor.MySQLCursorBuffered Class”](#).

To create a buffered raw cursor, use the `raw` and `buffered` arguments when calling a connection's `cursor()` method. Alternatively, to make all cursors created from the connection raw and buffered by default, use the `raw` and `buffered` connection arguments.

Example:

```
import mysql.connector
cnx = mysql.connector.connect()
# Only this particular cursor will be raw and buffered
cursor = cnx.cursor(raw=True, buffered=True)
# All cursors created from cnx2 will be raw and buffered by default
cnx2 = mysql.connector.connect(raw=True, buffered=True)
```

5.9.6.4 cursor.MySQLCursorDict Class

The `MySQLCursorDict` class inherits from `MySQLCursor`. This class is available as of Connector/Python 2.0.0.

A `MySQLCursorDict` cursor returns each row as a dictionary. The keys for each dictionary object are the column names of the MySQL result.

Example:

```
cnx = mysql.connector.connect(database='world')
cursor = cnx.cursor(dictionary=True)
cursor.execute("SELECT * FROM country WHERE Continent = 'Europe'")
print("Countries in Europe:")
for row in cursor:
    print("* {Name}".format(Name=row['Name'])
```

The preceding code produces output like this:

```
Countries in Europe:
* Albania
* Andorra
* Austria
* Belgium
* Bulgaria
...
```

It may be convenient to pass the dictionary to `format()` as follows:

```
cursor.execute("SELECT Name, Population FROM country WHERE Continent = 'Europe'")
print("Countries in Europe with population:")
for row in cursor:
    print("* {Name}: {Population}".format(**row))
```

5.9.6.5 cursor.MySQLCursorBufferedDict Class

The `MySQLCursorBufferedDict` class inherits from `MySQLCursor`. This class is available as of Connector/Python 2.0.0.

A `MySQLCursorBufferedDict` cursor is like a `MySQLCursorDict` cursor, but is buffered: After executing a query, it fetches the entire result set from the server and buffers the rows. For information about the implications of buffering, see [Section 5.9.6.1, “cursor.MySQLCursorBuffered Class”](#).

To get a buffered cursor that returns dictionaries, add the `buffered` argument when instantiating a new dictionary cursor:

```
cursor = cnx.cursor(dictionary=True, buffered=True)
```

5.9.6.6 `cursor.MySQLCursorNamedTuple` Class

The `MySQLCursorNamedTuple` class inherits from `MySQLCursor`. This class is available as of Connector/Python 2.0.0.

A `MySQLCursorNamedTuple` cursor returns each row as a named tuple. The attributes for each named-tuple object are the column names of the MySQL result.

Example:

```
cnx = mysql.connector.connect(database='world')
cursor = cnx.cursor(named_tuple=True)
cursor.execute("SELECT * FROM country WHERE Continent = 'Europe'")
print("Countries in Europe with population:")
for row in cursor:
    print("{} {Name}: {Population}".format(
        Name=row.Name,
        Population=row.Population
    ))
```

5.9.6.7 `cursor.MySQLCursorBufferedNamedTuple` Class

The `MySQLCursorBufferedNamedTuple` class inherits from `MySQLCursor`. This class is available as of Connector/Python 2.0.0.

A `MySQLCursorBufferedNamedTuple` cursor is like a `MySQLCursorNamedTuple` cursor, but is buffered: After executing a query, it fetches the entire result set from the server and buffers the rows. For information about the implications of buffering, see [Section 5.9.6.1, “`cursor.MySQLCursorBufferedClass`”](#).

To get a buffered cursor that returns named tuples, add the `buffered` argument when instantiating a new named-tuple cursor:

```
cursor = cnx.cursor(named_tuple=True, buffered=True)
```

5.9.6.8 `cursor.MySQLCursorPrepared` Class

The `MySQLCursorPrepared` class inherits from `MySQLCursor`. This class is available as of Connector/Python 1.1.0.

In MySQL, there are two ways to execute a prepared statement:

- Use the `PREPARE` and `EXECUTE` statements.
- Use the binary client/server protocol to send and receive data. To repeatedly execute the same statement with different data for different executions, this is more efficient than using `PREPARE` and `EXECUTE`. For information about the binary protocol, see [C API Prepared Statements](#).

In Connector/Python, there are two ways to create a cursor that enables execution of prepared statements using the binary protocol. In both cases, the `cursor()` method of the connection object returns a `MySQLCursorPrepared` object:

- The simpler syntax uses a `prepared=True` argument to the `cursor()` method. This syntax is available as of Connector/Python 1.1.2.

```
import mysql.connector
cnx = mysql.connector.connect(database='employees')
cursor = cnx.cursor(prepared=True)
```

- Alternatively, create an instance of the `MySQLCursorPrepared` class using the `cursor_class` argument to the `cursor()` method. This syntax is available as of Connector/Python 1.1.0.

```
import mysql.connector
from mysql.connector.cursor import MySQLCursorPrepared
cnx = mysql.connector.connect(database='employees')
cursor = cnx.cursor(cursor_class=MySQLCursorPrepared)
```

A cursor instantiated from the `MySQLCursorPrepared` class works like this:

- The first time you pass a statement to the cursor's `execute()` method, it prepares the statement. For subsequent invocations of `execute()`, the preparation phase is skipped if the statement is the same.
- The `execute()` method takes an optional second argument containing a list of data values to associate with parameter markers in the statement. If the list argument is present, there must be one value per parameter marker.

Example:

```
cursor = cnx.cursor(prepared=True)
stmt = "SELECT fullname FROM employees WHERE id = %s" # (1)
cursor.execute(stmt, (5,)) # (2)
# ... fetch data ...
cursor.execute(stmt, (10,)) # (3)
# ... fetch data ...
```

1. The `%s` within the statement is a parameter marker. Do not put quote marks around parameter markers.
2. For the first call to the `execute()` method, the cursor prepares the statement. If data is given in the same call, it also executes the statement and you should fetch the data.
3. For subsequent `execute()` calls that pass the same SQL statement, the cursor skips the preparation phase.

Prepared statements executed with `MySQLCursorPrepared` can use the `format` (`%s`) or `qmark` (`?`) parameterization style. This differs from nonprepared statements executed with `MySQLCursor`, which can use the `format` or `pyformat` parameterization style.

To use multiple prepared statements simultaneously, instantiate multiple cursors from the `MySQLCursorPrepared` class.

The MySQL client/server protocol has an option to send prepared statement parameters via the `COM_STMT_SEND_LONG_DATA` command. To use this from Connector/Python scripts, send the parameter in question using the `IOBase` interface. Example:

```
from io import IOBase
...
cur = cnx.cursor(prepared=True)
cur.execute("SELECT (%s)", (io.BytesIO(bytes("A", "latin1")), ))
```

5.9.7 constants.ClientFlag Class

This class provides constants defining MySQL client flags that can be used when the connection is established to configure the session. The `ClientFlag` class is available when importing `mysql.connector`.

```
>>> import mysql.connector
>>> mysql.connector.ClientFlag.FOUND_ROWS
```

2

See [Section 5.9.2.32, “MySQLConnection.set_client_flags\(\) Method”](#) and the [connection argument client_flag](#).

The [ClientFlag](#) class cannot be instantiated.

5.9.8 constants.FieldType Class

This class provides all supported MySQL field or data types. They can be useful when dealing with raw data or defining your own converters. The field type is stored with every cursor in the description for each column.

The following example shows how to print the name of the data type for each column in a result set.

```
from __future__ import print_function
import mysql.connector
from mysql.connector import FieldType
cnx = mysql.connector.connect(user='scott', database='test')
cursor = cnx.cursor()
cursor.execute(
    "SELECT DATE(NOW()) AS `c1`, TIME(NOW()) AS `c2`, "
    "NOW() AS `c3`, 'a string' AS `c4`, 42 AS `c5`")
rows = cursor.fetchall()
for desc in cursor.description:
    colname = desc[0]
    coltype = desc[1]
    print("Column {} has type {}".format(
        colname, FieldType.get_info(coltype)))
cursor.close()
cnx.close()
```

The [FieldType](#) class cannot be instantiated.

5.9.9 constants.SQLMode Class

This class provides all known MySQL [Server SQL Modes](#). It is mostly used when setting the SQL modes at connection time using the connection's [sql_mode](#) property. See [Section 5.9.2.46, “MySQLConnection.sql_mode Property”](#).

The [SQLMode](#) class cannot be instantiated.

5.9.10 constants.CharacterSet Class

This class provides all known MySQL characters sets and their default collations. For examples, see [Section 5.9.2.31, “MySQLConnection.set_charset_collation\(\) Method”](#).

The [CharacterSet](#) class cannot be instantiated.

5.9.11 constants.RefreshOption Class

This class performs various flush operations.

- [RefreshOption.GRANT](#)

Refresh the grant tables, like [FLUSH PRIVILEGES](#).

- [RefreshOption.LOG](#)

Flush the logs, like [FLUSH LOGS](#).

- [RefreshOption.TABLES](#)

Flush the table cache, like `FLUSH TABLES`.

- `RefreshOption.HOSTS`

Flush the host cache, like `FLUSH HOSTS`.

- `RefreshOption.STATUS`

Reset status variables, like `FLUSH STATUS`.

- `RefreshOption.THREADS`

Flush the thread cache.

- `RefreshOption.SLAVE`

On a slave replication server, reset the master server information and restart the slave, like `RESET SLAVE`.

- `RefreshOption.MASTER`

On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like `RESET MASTER`.

5.9.12 Errors and Exceptions

The `mysql.connector.errors` module defines exception classes for errors and warnings raised by MySQL Connector/Python. Most classes defined in this module are available when you import `mysql.connector`.

The exception classes defined in this module mostly follow the Python Database API Specification v2.0 (PEP 249). For some MySQL client or server errors it is not always clear which exception to raise. It is good to discuss whether an error should be reclassified by opening a bug report.

MySQL Server errors are mapped with Python exception based on their SQLSTATE value (see [Server Error Codes and Messages](#)). The following table shows the SQLSTATE classes and the exception Connector/Python raises. It is, however, possible to redefine which exception is raised for each server error. The default exception is `DatabaseError`.

Table 5.3 Mapping of Server Errors to Python Exceptions

SQLSTATE Class	Connector/Python Exception
02	<code>DataError</code>
02	<code>DataError</code>
07	<code>DatabaseError</code>
08	<code>OperationalError</code>
0A	<code>NotSupportedError</code>
21	<code>DataError</code>
22	<code>DataError</code>
23	<code>IntegrityError</code>
24	<code>ProgrammingError</code>
25	<code>ProgrammingError</code>
26	<code>ProgrammingError</code>
27	<code>ProgrammingError</code>
28	<code>ProgrammingError</code>

SQLSTATE Class	Connector/Python Exception
2A	ProgrammingError
2B	DatabaseError
2C	ProgrammingError
2D	DatabaseError
2E	DatabaseError
33	DatabaseError
34	ProgrammingError
35	ProgrammingError
37	ProgrammingError
3C	ProgrammingError
3D	ProgrammingError
3F	ProgrammingError
40	InternalError
42	ProgrammingError
44	InternalError
HZ	OperationalError
XA	IntegrityError
0K	OperationalError
HY	DatabaseError

5.9.12.1 errorcode Module

This module contains both MySQL server and client error codes defined as module attributes with the error number as value. Using error codes instead of error numbers could make reading the source code a bit easier.

```
>>> from mysql.connector import errorcode
>>> errorcode.ER_BAD_TABLE_ERROR
1051
```

See [Server Error Codes and Messages](#) and [Client Error Codes and Messages](#).

5.9.12.2 errors.Error Exception

This exception is the base class for all other exceptions in the `errors` module. It can be used to catch all errors in a single `except` statement.

The following example shows how we could catch syntax errors:

```
import mysql.connector
try:
    cnx = mysql.connector.connect(user='scott', database='employees')
    cursor = cnx.cursor()
    cursor.execute("SELECT * FORM employees") # Syntax error in query
    cnx.close()
except mysql.connector.Error as err:
    print("Something went wrong: {}".format(err))
```

Initializing the exception supports a few optional arguments, namely `msg`, `errno`, `values` and `sqlstate`. All of them are optional and default to `None`. `errors.Error` is internally used by

Connector/Python to raise MySQL client and server errors and should not be used by your application to raise exceptions.

The following examples show the result when using no arguments or a combination of the arguments:

```
>>> from mysql.connector.errors import Error
>>> str(Error())
'Unknown error'
>>> str(Error("Oops! There was an error."))
'Oops! There was an error.'
>>> str(Error(errno=2006))
'2006: MySQL server has gone away'
>>> str(Error(errno=2002, values=('/tmp/mysql.sock', 2)))
'2002: Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)'
>>> str(Error(errno=1146, sqlstate='42S02', msg="Table 'test.spam' doesn't exist"))
'1146 (42S02): Table 'test.spam' doesn't exist'
```

The example which uses error number 1146 is used when Connector/Python receives an error packet from the MySQL Server. The information is parsed and passed to the `Error` exception as shown.

Each exception subclassing from `Error` can be initialized using the previously mentioned arguments. Additionally, each instance has the attributes `errno`, `msg` and `sqlstate` which can be used in your code.

The following example shows how to handle errors when dropping a table which does not exist (when the `DROP TABLE` statement does not include a `IF EXISTS` clause):

```
import mysql.connector
from mysql.connector import errorcode
cnx = mysql.connector.connect(user='scott', database='test')
cursor = cnx.cursor()
try:
    cursor.execute("DROP TABLE spam")
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_BAD_TABLE_ERROR:
        print("Creating table spam")
    else:
        raise
```

Prior to Connector/Python 1.1.1, the original message passed to `errors.Error()` is not saved in such a way that it could be retrieved. Instead, the `Error.msg` attribute was formatted with the error number and SQLSTATE value. As of 1.1.1, only the original message is saved in the `Error.msg` attribute. The formatted value together with the error number and SQLSTATE value can be obtained by printing or getting the string representation of the error object. Example:

```
try:
    conn = mysql.connector.connect(database = "baddb")
except mysql.connector.Error as e:
    print "Error code:", e.errno          # error number
    print "SQLSTATE value:", e.sqlstate  # SQLSTATE value
    print "Error message:", e.msg        # error message
    print "Error:", e                    # errno, sqlstate, msg values
    s = str(e)
    print "Error:", s                    # errno, sqlstate, msg values
```

`errors.Error` is a subclass of the Python `StandardError`.

5.9.12.3 errors.DataError Exception

This exception is raised when there were problems with the data. Examples are a column set to `NULL` that cannot be `NULL`, out-of-range values for a column, division by zero, column count does not match value count, and so on.

`errors.DataError` is a subclass of `errors.DatabaseError`.

5.9.12.4 errors.DatabaseError Exception

This exception is the default for any MySQL error which does not fit the other exceptions.

`errors.DatabaseError` is a subclass of `errors.Error`.

5.9.12.5 errors.IntegrityError Exception

This exception is raised when the relational integrity of the data is affected. For example, a duplicate key was inserted or a foreign key constraint would fail.

The following example shows a duplicate key error raised as `IntegrityError`:

```
cursor.execute("CREATE TABLE t1 (id int, PRIMARY KEY (id))")
try:
    cursor.execute("INSERT INTO t1 (id) VALUES (1)")
    cursor.execute("INSERT INTO t1 (id) VALUES (1)")
except mysql.connector.IntegrityError as err:
    print("Error: {}".format(err))
```

`errors.IntegrityError` is a subclass of `errors.DatabaseError`.

5.9.12.6 errors.InterfaceError Exception

This exception is raised for errors originating from Connector/Python itself, not related to the MySQL server.

`errors.InterfaceError` is a subclass of `errors.Error`.

5.9.12.7 errors.InternalError Exception

This exception is raised when the MySQL server encounters an internal error, for example, when a deadlock occurred.

`errors.InternalError` is a subclass of `errors.DatabaseError`.

5.9.12.8 errors.NotSupportedError Exception

This exception is raised when some feature was used that is not supported by the version of MySQL that returned the error. It is also raised when using functions or statements that are not supported by stored routines.

`errors.NotSupportedError` is a subclass of `errors.DatabaseError`.

5.9.12.9 errors.OperationalError Exception

This exception is raised for errors which are related to MySQL's operations. For example: too many connections; a host name could not be resolved; bad handshake; server is shutting down, communication errors.

`errors.OperationalError` is a subclass of `errors.DatabaseError`.

5.9.12.10 errors.PoolError Exception

This exception is raised for connection pool errors. `errors.PoolError` is a subclass of `errors.Error`.

5.9.12.11 errors.ProgrammingError Exception

This exception is raised on programming errors, for example when you have a syntax error in your SQL or a table was not found.

The following example shows how to handle syntax errors:

```
try:
    cursor.execute("CREATE DESK t1 (id int, PRIMARY KEY (id))")
except mysql.connector.ProgrammingError as err:
    if err.errno == errorcode.ER_SYNTAX_ERROR:
        print("Check your syntax!")
    else:
        print("Error: {}".format(err))
```

`errors.ProgrammingError` is a subclass of `errors.DatabaseError`.

5.9.12.12 errors.Warning Exception

This exception is used for reporting important warnings, however, Connector/Python does not use it. It is included to be compliant with the Python Database Specification v2.0 (PEP-249).

Consider using either more strict [Server SQL Modes](#) or the `raise_on_warnings` connection argument to make Connector/Python raise errors when your queries produce warnings.

`errors.Warning` is a subclass of the Python `StandardError`.

5.9.12.13 errors.custom_error_exception() Function

Syntax:

```
errors.custom_error_exception(error=None, exception=None)
```

This method defines custom exceptions for MySQL server errors and returns current customizations.

If `error` is a MySQL Server error number, you must also pass the `exception` class. The `error` argument can be a dictionary, in which case the key is the server error number, and value the class of the exception to be raised.

To reset the customizations, supply an empty dictionary.

```
import mysql.connector
from mysql.connector import errorcode
# Server error 1028 should raise a DatabaseError
mysql.connector.custom_error_exception(1028, mysql.connector.DatabaseError)
# Or using a dictionary:
mysql.connector.custom_error_exception({
    1028: mysql.connector.DatabaseError,
    1029: mysql.connector.OperationalError,
})
# To reset, pass an empty dictionary:
mysql.connector.custom_error_exception({})
```

Chapter 6 MySQL and PHP

Table of Contents

6.1 Introduction to the MySQL PHP API	514
6.2 Overview of the MySQL PHP drivers	515
6.2.1 Introduction	515
6.2.2 Terminology overview	515
6.2.3 Choosing an API	516
6.2.4 Choosing a library	518
6.2.5 Concepts	519
6.3 MySQL Improved Extension	522
6.3.1 Overview	522
6.3.2 Quick start guide	525
6.3.3 Installing/Configuring	547
6.3.4 The mysqli Extension and Persistent Connections	550
6.3.5 Predefined Constants	551
6.3.6 Notes	554
6.3.7 The MySQLi Extension Function Summary	554
6.3.8 Examples	561
6.3.9 The mysqli class	562
6.3.10 The mysqli_stmt class	653
6.3.11 The mysqli_result class	693
6.3.12 The mysqli_driver class	721
6.3.13 The mysqli_warning class	725
6.3.14 The mysqli_sql_exception class	726
6.3.15 Aliases and deprecated Mysqli Functions	727
6.3.16 Changelog	735
6.4 MySQL Functions (PDO_MYSQL)	735
6.4.1 PDO_MYSQL DSN	738
6.5 Original MySQL API	739
6.5.1 Installing/Configuring	739
6.5.2 Changelog	742
6.5.3 Predefined Constants	743
6.5.4 Examples	744
6.5.5 MySQL Functions	744
6.6 MySQL Native Driver	810
6.6.1 Overview	811
6.6.2 Installation	812
6.6.3 Runtime Configuration	813
6.6.4 Incompatibilities	817
6.6.5 Persistent Connections	817
6.6.6 Statistics	818
6.6.7 Notes	831
6.6.8 Memory management	831
6.6.9 MySQL Native Driver Plugin API	833
6.7 Mysqlnd replication and load balancing plugin	845
6.7.1 Key Features	845
6.7.2 Limitations	846
6.7.3 On the name	847
6.7.4 Quickstart and Examples	847
6.7.5 Concepts	875
6.7.6 Installing/Configuring	899
6.7.7 Predefined Constants	954
6.7.8 Mysqlnd_ms Functions	956
6.7.9 Change History	978

6.8 Mysqlnd query result cache plugin	986
6.8.1 Key Features	986
6.8.2 Limitations	986
6.8.3 On the name	987
6.8.4 Quickstart and Examples	987
6.8.5 Installing/Configuring	1007
6.8.6 Predefined Constants	1010
6.8.7 mysqlnd_qc Functions	1012
6.8.8 Change History	1034
6.9 Mysqlnd user handler plugin	1036
6.9.1 Security considerations	1037
6.9.2 Documentation note	1037
6.9.3 On the name	1037
6.9.4 Quickstart and Examples	1037
6.9.5 Installing/Configuring	1042
6.9.6 Predefined Constants	1043
6.9.7 The MysqlndUhConnection class	1048
6.9.8 The MysqlndUhPreparedStatement class	1112
6.9.9 Mysqlnd_uh Functions	1115
6.9.10 Change History	1119
6.10 Mysqlnd connection multiplexing plugin	1119
6.10.1 Key Features	1119
6.10.2 Limitations	1120
6.10.3 About the name mysqlnd_mux	1120
6.10.4 Concepts	1120
6.10.5 Installing/Configuring	1121
6.10.6 Predefined Constants	1122
6.10.7 Change History	1122
6.11 Mysqlnd Memcache plugin	1123
6.11.1 Key Features	1124
6.11.2 Limitations	1124
6.11.3 On the name	1124
6.11.4 Quickstart and Examples	1124
6.11.5 Installing/Configuring	1126
6.11.6 Predefined Constants	1127
6.11.7 Mysqlnd_memcache Functions	1128
6.11.8 Change History	1132
6.12 Common Problems with MySQL and PHP	1132

6.1 Introduction to the MySQL PHP API

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic Web pages. It is available for most operating systems and Web servers, and can access most common databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with a Web server.

PHP provides three different MySQL API extensions:

- [Section 6.3, “MySQL Improved Extension”](#): Stands for “MySQL, Improved”; this extension is available as of PHP 5.0.0. It is intended for use with MySQL 4.1.1 and later. This extension fully supports the authentication protocol used in MySQL 5.0, as well as the Prepared Statements and Multiple Statements APIs. In addition, this extension provides an advanced, object-oriented programming interface.
- [Section 6.4, “MySQL Functions \(PDO_MYSQL\)”](#): Not its own API, but instead it's a MySQL driver for the PHP database abstraction layer PDO (PHP Data Objects). The PDO MySQL driver sits in the layer below PDO itself, and provides MySQL-specific functionality. This extension is available as of PHP 5.1.0.

- [Section 6.5, “Original MySQL API”](#): Available for PHP versions 4 and 5, this extension is intended for use with MySQL versions prior to MySQL 4.1. This extension does not support the improved authentication protocol used in MySQL 4.1, nor does it support prepared statements or multiple statements. To use this extension with MySQL 4.1, you will likely configure the MySQL server to set the `old_passwords` system variable to 1 (see [Client does not support authentication protocol](#)).

Warning

This extension was removed from PHP 5.5.0. All users must migrate to either `mysqli` or `PDO_MySQL`. For further information, see [Section 6.2.3, “Choosing an API”](#).

Note

This documentation, and other publications, sometimes uses the term `Connector/PHP`. This term refers to the full set of MySQL related functionality in PHP, which includes the three APIs that are described in the preceding discussion, along with the `mysqlnd` core library and all of its plugins.

The PHP distribution and documentation are available from the [PHP Web site](#).

Portions of this section are Copyright (c) 1997-2015 the PHP Documentation Group This material may be distributed only subject to the terms and conditions set forth in the Creative Commons Attribution 3.0 License or later. A copy of the Creative Commons Attribution 3.0 license is distributed with this manual. The latest version is presently available at <http://creativecommons.org/licenses/by/3.0/>.

6.2 Overview of the MySQL PHP drivers

[Copyright 1997-2014 the PHP Documentation Group.](#)

6.2.1 Introduction

Depending on the version of PHP, there are either two or three PHP APIs for accessing the MySQL database. PHP 5 users can choose between the deprecated `mysql` extension, `mysqli`, or `PDO_MySQL`. PHP 7 removes the `mysql` extension, leaving only the latter two options.

This guide explains the [terminology](#) used to describe each API, information about [choosing which API](#) to use, and also information to help choose which MySQL [library to use](#) with the API.

6.2.2 Terminology overview

[Copyright 1997-2014 the PHP Documentation Group.](#)

This section provides an introduction to the options available to you when developing a PHP application that needs to interact with a MySQL database.

What is an API?

An Application Programming Interface, or API, defines the classes, methods, functions and variables that your application will need to call in order to carry out its desired task. In the case of PHP applications that need to communicate with databases the necessary APIs are usually exposed via PHP extensions.

APIs can be procedural or object-oriented. With a procedural API you call functions to carry out tasks, with the object-oriented API you instantiate classes and then call methods on the resulting objects. Of the two the latter is usually the preferred interface, as it is more modern and leads to better organized code.

When writing PHP applications that need to connect to the MySQL server there are several API options available. This document discusses what is available and how to select the best solution for your application.

What is a Connector?

In the MySQL documentation, the term *connector* refers to a piece of software that allows your application to connect to the MySQL database server. MySQL provides connectors for a variety of languages, including PHP.

If your PHP application needs to communicate with a database server you will need to write PHP code to perform such activities as connecting to the database server, querying the database and other database-related functions. Software is required to provide the API that your PHP application will use, and also handle the communication between your application and the database server, possibly using other intermediate libraries where necessary. This software is known generically as a connector, as it allows your application to *connect* to a database server.

What is a Driver?

A driver is a piece of software designed to communicate with a specific type of database server. The driver may also call a library, such as the MySQL Client Library or the MySQL Native Driver. These libraries implement the low-level protocol used to communicate with the MySQL database server.

By way of an example, the [PHP Data Objects \(PDO\)](#) database abstraction layer may use one of several database-specific drivers. One of the drivers it has available is the PDO MYSQL driver, which allows it to interface with the MySQL server.

Sometimes people use the terms connector and driver interchangeably, this can be confusing. In the MySQL-related documentation the term “driver” is reserved for software that provides the database-specific part of a connector package.

What is an Extension?

In the PHP documentation you will come across another term - *extension*. The PHP code consists of a core, with optional extensions to the core functionality. PHP's MySQL-related extensions, such as the [mysqli](#) extension, and the [mysql](#) extension, are implemented using the PHP extension framework.

An extension typically exposes an API to the PHP programmer, to allow its facilities to be used programmatically. However, some extensions which use the PHP extension framework do not expose an API to the PHP programmer.

The PDO MySQL driver extension, for example, does not expose an API to the PHP programmer, but provides an interface to the PDO layer above it.

The terms API and extension should not be taken to mean the same thing, as an extension may not necessarily expose an API to the programmer.

6.2.3 Choosing an API

[Copyright 1997-2014 the PHP Documentation Group.](#)

PHP offers three different APIs to connect to MySQL. Below we show the APIs provided by the `mysql`, `mysqli`, and `PDO` extensions. Each code snippet creates a connection to a MySQL server running on “example.com” using the username “user” and the password “password”. And a query is run to greet the user.

Example 6.1 Comparing the three MySQL APIs

```
<?php
// mysqli
$mysqli = new mysqli("example.com", "user", "password", "database");
$result = $mysqli->query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = $result->fetch_assoc();
echo htmlentities($row['_message']);
// PDO
$pdo = new PDO('mysql:host=example.com;dbname=database', 'user', 'password');
```

```
$statement = $pdo->query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = $statement->fetch(PDO::FETCH_ASSOC);
echo htmlentities($row['_message']);
// mysql
$c = mysql_connect("example.com", "user", "password");
mysql_select_db("database");
$result = mysql_query("SELECT 'Hello, dear MySQL user!' AS _message FROM DUAL");
$row = mysql_fetch_assoc($result);
echo htmlentities($row['_message']);
?>
```

Recommended API

It is recommended to use either the [mysqli](#) or [PDO_MySQL](#) extensions. It is not recommended to use the old [mysql](#) extension for new development, as it was deprecated in PHP 5.5.0 and was removed in PHP 7. A detailed feature comparison matrix is provided below. The overall performance of all three extensions is considered to be about the same. Although the performance of the extension contributes only a fraction of the total run time of a PHP web request. Often, the impact is as low as 0.1%.

Feature comparison

	ext/mysqli	PDO_MySQL	ext/mysql
PHP version introduced	5.0	5.1	2.0
Included with PHP 5.x	Yes	Yes	Yes
Included with PHP 7.x	Yes	Yes	No
Development status	Active	Active	Maintenance only in 5.x; removed in 7.x
Lifecycle	Active	Active	Deprecated in 5.x; removed in 7.x
Recommended for new projects	Yes	Yes	No
OOP Interface	Yes	Yes	No
Procedural Interface	Yes	No	Yes
API supports non-blocking, asynchronous queries with mysqlnd	Yes	No	No
Persistent Connections	Yes	Yes	Yes
API supports Charsets	Yes	Yes	Yes
API supports server-side Prepared Statements	Yes	Yes	No
API supports client-side Prepared Statements	No	Yes	No
API supports Stored Procedures	Yes	Yes	No
API supports Multiple Statements	Yes	Most	No
API supports Transactions	Yes	Yes	No
Transactions can be controlled with SQL	Yes	Yes	Yes
Supports all MySQL 5.1+ functionality	Yes	Most	No

6.2.4 Choosing a library

Copyright 1997-2014 the PHP Documentation Group.

The `mysqli`, `PDO_MySQL` and `mysql` PHP extensions are lightweight wrappers on top of a C client library. The extensions can either use the [mysqlnd](#) library or the [libmysqlclient](#) library. Choosing a library is a compile time decision.

The `mysqlnd` library is part of the PHP distribution since 5.3.0. It offers features like lazy connections and query caching, features that are not available with `libmysqlclient`, so using the built-in `mysqlnd` library is highly recommended. See the [mysqlnd documentation](#) for additional details, and a listing of features and functionality that it offers.

Example 6.2 Configure commands for using `mysqlnd` or `libmysqlclient`

```
// Recommended, compiles with mysqlnd
$ ./configure --with-mysqli=mysqlnd --with-pdo-mysql=mysqlnd --with-mysql=mysqlnd
// Alternatively recommended, compiles with mysqlnd as of PHP 5.4
$ ./configure --with-mysqli --with-pdo-mysql --with-mysql
// Not recommended, compiles with libmysqlclient
$ ./configure --with-mysqli=/path/to/mysql_config --with-pdo-mysql=/path/to/mysql_config --with-mysql=/path/to/mysql_config
```

Library feature comparison

It is recommended to use the [mysqlnd](#) library instead of the MySQL Client Server library (`libmysqlclient`). Both libraries are supported and constantly being improved.

	MySQL native driver (mysqlnd)	MySQL client server library (libmysqlclient)
Part of the PHP distribution	Yes	No
PHP version introduced	5.3.0	N/A
License	PHP License 3.01	Dual-License
Development status	Active	Active
Lifecycle	No end announced	No end announced
PHP 5.4 and above; compile default (for all MySQL extensions)	Yes	No
PHP 5.3; compile default (for all MySQL extensions)	No	Yes
Compression protocol support	Yes (5.3.1+)	Yes
SSL support	Yes (5.3.3+)	Yes
Named pipe support	Yes (5.3.4+)	Yes
Non-blocking, asynchronous queries	Yes	No
Performance statistics	Yes	No
LOAD LOCAL INFILE respects the open_basedir directive	Yes	No
Uses PHP's native memory management system (e.g., follows PHP memory limits)	Yes	No
Return numeric column as double (COM_QUERY)	Yes	No

	MySQL native driver (mysqlnd)	MySQL client server library (libmysqlclient)
Return numeric column as string (COM_QUERY)	Yes	Yes
Plugin API	Yes	Limited
Read/Write splitting for MySQL Replication	Yes, with plugin	No
Load Balancing	Yes, with plugin	No
Fail over	Yes, with plugin	No
Lazy connections	Yes, with plugin	No
Query caching	Yes, with plugin	No
Transparent query manipulations (E.g., auto-EXPLAIN or monitoring)	Yes, with plugin	No

6.2.5 Concepts

Copyright 1997-2014 the PHP Documentation Group.

These concepts are specific to the MySQL drivers for PHP.

6.2.5.1 Buffered and Unbuffered queries

Copyright 1997-2014 the PHP Documentation Group.

Queries are using the buffered mode by default. This means that query results are immediately transferred from the MySQL Server to PHP and then are kept in the memory of the PHP process. This allows additional operations like counting the number of rows, and moving (seeking) the current result pointer. It also allows issuing further queries on the same connection while working on the result set. The downside of the buffered mode is that larger result sets might require quite a lot memory. The memory will be kept occupied till all references to the result set are unset or the result set was explicitly freed, which will automatically happen during request end the latest. The terminology "store result" is also used for buffered mode, as the whole result set is stored at once.

Note

When using `libmysqlclient` as library PHP's memory limit won't count the memory used for result sets unless the data is fetched into PHP variables. With `mysqlnd` the memory accounted for will include the full result set.

Unbuffered MySQL queries execute the query and then return a resource while the data is still waiting on the MySQL server for being fetched. This uses less memory on the PHP-side, but can increase the load on the server. Unless the full result set was fetched from the server no further queries can be sent over the same connection. Unbuffered queries can also be referred to as "use result".

Following these characteristics buffered queries should be used in cases where you expect only a limited result set or need to know the amount of returned rows before reading all rows. Unbuffered mode should be used when you expect larger results.

Because buffered queries are the default, the examples below will demonstrate how to execute unbuffered queries with each API.

Example 6.3 Unbuffered query example: `mysqli`

```
<?php
```

```
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
$result = $mysqli->query("SELECT Name FROM City", MYSQLI_USE_RESULT);
if ($result) {
    while ($row = $result->fetch_assoc()) {
        echo $row['Name'] . PHP_EOL;
    }
}
$result->close();
?>
```

Example 6.4 Unbuffered query example: pdo_mysql

```
<?php
$pdo = new PDO("mysql:host=localhost;dbname=world", 'my_user', 'my_pass');
$pdo->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, false);
$result = $pdo->query("SELECT Name FROM City");
if ($result) {
    while ($row = $result->fetch(PDO::FETCH_ASSOC)) {
        echo $row['Name'] . PHP_EOL;
    }
}
?>
```

Example 6.5 Unbuffered query example: mysql

```
<?php
$conn = mysql_connect("localhost", "my_user", "my_pass");
$db = mysql_select_db("world");
$result = mysql_unbuffered_query("SELECT Name FROM City");
if ($result) {
    while ($row = mysql_fetch_assoc($result)) {
        echo $row['Name'] . PHP_EOL;
    }
}
?>
```

6.2.5.2 Character sets

Copyright 1997-2014 the PHP Documentation Group.

Ideally a proper character set will be set at the server level, and doing this is described within the [Character Set Configuration](#) section of the MySQL Server manual. Alternatively, each MySQL API offers a method to set the character set at runtime.

The character set and character escaping

The character set should be understood and defined, as it has an affect on every action, and includes security implications. For example, the escaping mechanism (e.g., [mysqli_real_escape_string](#) for mysqli, [mysql_real_escape_string](#) for mysql, and [PDO::quote](#) for PDO_MySQL) will adhere to this setting. It is important to realize that these functions will not use the character set that is defined with a query, so for example the following will not have an effect on them:

Example 6.6 Problems with setting the character set with SQL

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
// Will NOT affect $mysqli->real_escape_string();
$mysqli->query("SET NAMES utf8");
// Will NOT affect $mysqli->real_escape_string();
$mysqli->query("SET CHARACTER SET utf8");
// But, this will affect $mysqli->real_escape_string();
$mysqli->set_charset('utf8');
// But, this will NOT affect it (utf-8 vs utf8) -- don't use dashes here
$mysqli->set_charset('utf-8');
?>
```

Below are examples that demonstrate how to properly alter the character set at runtime using each API.

Possible UTF-8 confusion

Because character set names in MySQL do not contain dashes, the string "utf8" is valid in MySQL to set the character set to UTF-8. The string "utf-8" is not valid, as using "utf-8" will fail to change the character set.

Example 6.7 Setting the character set example: mysqli

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
printf("Initial character set: %s\n", $mysqli->character_set_name());
if (!$mysqli->set_charset('utf8')) {
    printf("Error loading character set utf8: %s\n", $mysqli->error);
    exit;
}
echo "New character set information:\n";
print_r( $mysqli->get_charset() );
?>
```

Example 6.8 Setting the character set example: pdo_mysql

Note: This only works as of PHP 5.3.6.

```
<?php
$pdo = new PDO("mysql:host=localhost;dbname=world;charset=utf8", 'my_user', 'my_pass');
?>
```

Example 6.9 Setting the character set example: mysql

```
<?php
$conn = mysql_connect("localhost", "my_user", "my_pass");
$db = mysql_select_db("world");
echo 'Initial character set: ' . mysql_client_encoding($conn) . "\n";
if (!mysql_set_charset('utf8', $conn)) {
    echo "Error: Unable to set the character set.\n";
    exit;
}
echo 'Your current character set is: ' . mysql_client_encoding($conn);
?>
```

6.3 MySQL Improved Extension

Copyright 1997-2014 the PHP Documentation Group.

The `mysqli` extension allows you to access the functionality provided by MySQL 4.1 and above. More information about the MySQL Database server can be found at <http://www.mysql.com/>

An overview of software available for using MySQL from PHP can be found at [Section 6.3.1, "Overview"](#)

Documentation for MySQL can be found at <http://dev.mysql.com/doc/>.

Parts of this documentation included from MySQL manual with permissions of Oracle Corporation.

Examples use either the `world` or `sakila` database, which are freely available.

6.3.1 Overview

Copyright 1997-2014 the PHP Documentation Group.

This section provides an introduction to the options available to you when developing a PHP application that needs to interact with a MySQL database.

What is an API?

An Application Programming Interface, or API, defines the classes, methods, functions and variables that your application will need to call in order to carry out its desired task. In the case of PHP applications that need to communicate with databases the necessary APIs are usually exposed via PHP extensions.

APIs can be procedural or object-oriented. With a procedural API you call functions to carry out tasks, with the object-oriented API you instantiate classes and then call methods on the resulting objects. Of the two the latter is usually the preferred interface, as it is more modern and leads to better organized code.

When writing PHP applications that need to connect to the MySQL server there are several API options available. This document discusses what is available and how to select the best solution for your application.

What is a Connector?

In the MySQL documentation, the term *connector* refers to a piece of software that allows your application to connect to the MySQL database server. MySQL provides connectors for a variety of languages, including PHP.

If your PHP application needs to communicate with a database server you will need to write PHP code to perform such activities as connecting to the database server, querying the database and other database-related functions. Software is required to provide the API that your PHP application will use, and also handle the communication between your application and the database server, possibly using other intermediate libraries where necessary. This software is known generically as a connector, as it allows your application to *connect* to a database server.

What is a Driver?

A driver is a piece of software designed to communicate with a specific type of database server. The driver may also call a library, such as the MySQL Client Library or the MySQL Native Driver. These libraries implement the low-level protocol used to communicate with the MySQL database server.

By way of an example, the [PHP Data Objects \(PDO\)](#) database abstraction layer may use one of several database-specific drivers. One of the drivers it has available is the PDO MySQL driver, which allows it to interface with the MySQL server.

Sometimes people use the terms connector and driver interchangeably, this can be confusing. In the MySQL-related documentation the term “driver” is reserved for software that provides the database-specific part of a connector package.

What is an Extension?

In the PHP documentation you will come across another term - *extension*. The PHP code consists of a core, with optional extensions to the core functionality. PHP's MySQL-related extensions, such as the [mysql](#) extension, and the [mysqli](#) extension, are implemented using the PHP extension framework.

An extension typically exposes an API to the PHP programmer, to allow its facilities to be used programmatically. However, some extensions which use the PHP extension framework do not expose an API to the PHP programmer.

The PDO MySQL driver extension, for example, does not expose an API to the PHP programmer, but provides an interface to the PDO layer above it.

The terms API and extension should not be taken to mean the same thing, as an extension may not necessarily expose an API to the programmer.

What are the main PHP API offerings for using MySQL?

There are three main API options when considering connecting to a MySQL database server:

- PHP's MySQL Extension
- PHP's [mysqli](#) Extension
- PHP Data Objects (PDO)

Each has its own advantages and disadvantages. The following discussion aims to give a brief introduction to the key aspects of each API.

What is PHP's MySQL Extension?

This is the original extension designed to allow you to develop PHP applications that interact with a MySQL database. The [mysql](#) extension provides a procedural interface and is intended for use only with MySQL versions older than 4.1.3. This extension can be used with versions of MySQL 4.1.3 or newer, but not all of the latest MySQL server features will be available.

Note

If you are using MySQL versions 4.1.3 or later it is *strongly* recommended that you use the [mysqli](#) extension instead.

The [mysql](#) extension source code is located in the PHP extension directory [ext/mysql](#).

For further information on the [mysql](#) extension, see [Section 6.5, “Original MySQL API”](#).

What is PHP's [mysqli](#) Extension?

The [mysqli](#) extension, or as it is sometimes known, the MySQL *improved* extension, was developed to take advantage of new features found in MySQL systems versions 4.1.3 and newer. The [mysqli](#) extension is included with PHP versions 5 and later.

The [mysqli](#) extension has a number of benefits, the key enhancements over the [mysql](#) extension being:

- Object-oriented interface
- Support for Prepared Statements
- Support for Multiple Statements

- Support for Transactions
- Enhanced debugging capabilities
- Embedded server support

Note

If you are using MySQL versions 4.1.3 or later it is *strongly* recommended that you use this extension.

As well as the object-oriented interface the extension also provides a procedural interface.

The `mysqli` extension is built using the PHP extension framework, its source code is located in the directory `ext/mysqli`.

For further information on the `mysqli` extension, see [Section 6.3, “MySQL Improved Extension”](#).

What is PDO?

PHP Data Objects, or PDO, is a database abstraction layer specifically for PHP applications. PDO provides a consistent API for your PHP application regardless of the type of database server your application will connect to. In theory, if you are using the PDO API, you could switch the database server you used, from say Firebird to MySQL, and only need to make minor changes to your PHP code.

Other examples of database abstraction layers include JDBC for Java applications and DBI for Perl.

While PDO has its advantages, such as a clean, simple, portable API, its main disadvantage is that it doesn't allow you to use all of the advanced features that are available in the latest versions of MySQL server. For example, PDO does not allow you to use MySQL's support for Multiple Statements.

PDO is implemented using the PHP extension framework, its source code is located in the directory `ext/pdo`.

For further information on PDO, see the <http://www.php.net/book.pdo>.

What is the PDO MYSQL driver?

The PDO MYSQL driver is not an API as such, at least from the PHP programmer's perspective. In fact the PDO MYSQL driver sits in the layer below PDO itself and provides MySQL-specific functionality. The programmer still calls the PDO API, but PDO uses the PDO MYSQL driver to carry out communication with the MySQL server.

The PDO MYSQL driver is one of several available PDO drivers. Other PDO drivers available include those for the Firebird and PostgreSQL database servers.

The PDO MYSQL driver is implemented using the PHP extension framework. Its source code is located in the directory `ext/pdo_mysql`. It does not expose an API to the PHP programmer.

For further information on the PDO MYSQL driver, see [Section 6.4, “MySQL Functions \(PDO_MYSQL\)”](#).

What is PHP's MySQL Native Driver?

In order to communicate with the MySQL database server the `mysql` extension, `mysqli` and the PDO MYSQL driver each use a low-level library that implements the required protocol. In the past, the only available library was the MySQL Client Library, otherwise known as `libmysqlclient`.

However, the interface presented by `libmysqlclient` was not optimized for communication with PHP applications, as `libmysqlclient` was originally designed with C applications in mind. For this reason the MySQL Native Driver, `mysqlnd`, was developed as an alternative to `libmysqlclient` for PHP applications.

The [mysql](#) extension, the [mysqli](#) extension and the PDO MySQL driver can each be individually configured to use either [libmysqlclient](#) or [mysqlnd](#). As [mysqlnd](#) is designed specifically to be utilised in the PHP system it has numerous memory and speed enhancements over [libmysqlclient](#). You are strongly encouraged to take advantage of these improvements.

Note

The MySQL Native Driver can only be used with MySQL server versions 4.1.3 and later.

The MySQL Native Driver is implemented using the PHP extension framework. The source code is located in [ext/mysqlnd](#). It does not expose an API to the PHP programmer.

Comparison of Features

The following table compares the functionality of the three main methods of connecting to MySQL from PHP:

Table 6.1 Comparison of MySQL API options for PHP

	PHP's mysqli Extension	PDO (Using PDO MySQL Driver and MySQL Native Driver)	PHP's MySQL Extension
PHP version introduced	5.0	5.0	Prior to 3.0
Included with PHP 5.x	yes	yes	Yes
MySQL development status	Active development	Active development as of PHP 5.3	Maintenance only
Recommended by MySQL for new projects	Yes - preferred option	Yes	No
API supports Charsets	Yes	Yes	No
API supports server-side Prepared Statements	Yes	Yes	No
API supports client-side Prepared Statements	No	Yes	No
API supports Stored Procedures	Yes	Yes	No
API supports Multiple Statements	Yes	Most	No
Supports all MySQL 4.1+ functionality	Yes	Most	No

6.3.2 Quick start guide

[Copyright 1997-2014 the PHP Documentation Group.](#)

This quick start guide will help with choosing and gaining familiarity with the PHP MySQL API.

This quick start gives an overview on the [mysqli](#) extension. Code examples are provided for all major aspects of the API. Database concepts are explained to the degree needed for presenting concepts specific to MySQL.

Required: A familiarity with the PHP programming language, the SQL language, and basic knowledge of the MySQL server.

6.3.2.1 Dual procedural and object-oriented interface

[Copyright 1997-2014 the PHP Documentation Group.](#)

The mysqli extension features a dual interface. It supports the procedural and object-oriented programming paradigm.

Users migrating from the old mysql extension may prefer the procedural interface. The procedural interface is similar to that of the old mysql extension. In many cases, the function names differ only by prefix. Some mysqli functions take a connection handle as their first argument, whereas matching functions in the old mysql interface take it as an optional last argument.

Example 6.10 Easy migration from the old mysql extension

```
<?php
$mysqli = mysqli_connect("example.com", "user", "password", "database");
$res = mysqli_query($mysqli, "SELECT 'Please, do not use ' AS _msg FROM DUAL");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];
$mysql = mysql_connect("example.com", "user", "password");
mysql_select_db("test");
$res = mysql_query("SELECT 'the mysql extension for new developments.' AS _msg FROM DUAL", $mysqli);
$row = mysql_fetch_assoc($res);
echo $row['_msg'];
?>
```

The above example will output:

```
Please, do not use the mysql extension for new developments.
```

The object-oriented interface

In addition to the classical procedural interface, users can choose to use the object-oriented interface. The documentation is organized using the object-oriented interface. The object-oriented interface shows functions grouped by their purpose, making it easier to get started. The reference section gives examples for both syntax variants.

There are no significant performance differences between the two interfaces. Users can base their choice on personal preference.

Example 6.11 Object-oriented and procedural interface

```
<?php
$mysqli = mysqli_connect("example.com", "user", "password", "database");
if (mysqli_connect_errno($mysqli)) {
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
$res = mysqli_query($mysqli, "SELECT 'A world full of ' AS _msg FROM DUAL");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: " . $mysqli->connect_error;
}
$res = $mysqli->query("SELECT 'choices to please everybody.' AS _msg FROM DUAL");
$row = $res->fetch_assoc();
echo $row['_msg'];
?>
```

The above example will output:

A world full of choices to please everybody.

The object oriented interface is used for the quickstart because the reference section is organized that way.

Mixing styles

It is possible to switch between styles at any time. Mixing both styles is not recommended for code clarity and coding style reasons.

Example 6.12 Bad coding style

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: " . $mysqli->connect_error;
}
$res = mysqli_query($mysqli, "SELECT 'Possible but bad style.' AS _msg FROM DUAL");
if (!$res) {
    echo "Failed to run query: (" . $mysqli->errno . ") " . $mysqli->error;
}
if ($row = $res->fetch_assoc()) {
    echo $row['_msg'];
}
?>
```

The above example will output:

Possible but bad style.

See also

[mysqli::__construct](#)
[mysqli::query](#)
[mysqli_result::fetch_assoc](#)
[\\$mysqli::connect_errno](#)
[\\$mysqli::connect_error](#)
[\\$mysqli::errno](#)
[\\$mysqli::error](#)
[The MySQLi Extension Function Summary](#)

6.3.2.2 Connections

[Copyright 1997-2014 the PHP Documentation Group.](#)

The MySQL server supports the use of different transport layers for connections. Connections use TCP/IP, Unix domain sockets or Windows named pipes.

The hostname [localhost](#) has a special meaning. It is bound to the use of Unix domain sockets. It is not possible to open a TCP/IP connection using the hostname [localhost](#) you must use [127.0.0.1](#) instead.

Example 6.13 Special meaning of localhost

```
<?php
$mysqli = new mysqli("localhost", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
echo $mysqli->host_info . "\n";
$mysqli = new mysqli("127.0.0.1", "user", "password", "database", 3306);
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
echo $mysqli->host_info . "\n";
?>
```

The above example will output:

```
Localhost via UNIX socket
127.0.0.1 via TCP/IP
```

Connection parameter defaults

Depending on the connection function used, assorted parameters can be omitted. If a parameter is not provided, then the extension attempts to use the default values that are set in the PHP configuration file.

Example 6.14 Setting defaults

```
mysqli.default_host=192.168.2.27
mysqli.default_user=root
mysqli.default_pw=""
mysqli.default_port=3306
mysqli.default_socket=/tmp/mysql.sock
```

The resulting parameter values are then passed to the client library that is used by the extension. If the client library detects empty or unset parameters, then it may default to the library built-in values.

Built-in connection library defaults

If the host value is unset or empty, then the client library will default to a Unix socket connection on [localhost](#). If socket is unset or empty, and a Unix socket connection is requested, then a connection to the default socket on [/tmp/mysql.sock](#) is attempted.

On Windows systems, the host name [.](#) is interpreted by the client library as an attempt to open a Windows named pipe based connection. In this case the socket parameter is interpreted as the pipe name. If not given or empty, then the socket (pipe name) defaults to [\\.\pipe\MySQL](#).

If neither a Unix domain socket based not a Windows named pipe based connection is to be established and the port parameter value is unset, the library will default to port [3306](#).

The [mysqlnd](#) library and the MySQL Client Library (libmysqlclient) implement the same logic for determining defaults.

Connection options

Connection options are available to, for example, set init commands which are executed upon connect, or for requesting use of a certain charset. Connection options must be set before a network connection is established.

For setting a connection option, the connect operation has to be performed in three steps: creating a connection handle with `mysqli_init`, setting the requested options using `mysqli_options`, and establishing the network connection with `mysqli_real_connect`.

Connection pooling

The `mysqli` extension supports persistent database connections, which are a special kind of pooled connections. By default, every database connection opened by a script is either explicitly closed by the user during runtime or released automatically at the end of the script. A persistent connection is not. Instead it is put into a pool for later reuse, if a connection to the same server using the same username, password, socket, port and default database is opened. Reuse saves connection overhead.

Every PHP process is using its own `mysqli` connection pool. Depending on the web server deployment model, a PHP process may serve one or multiple requests. Therefore, a pooled connection may be used by one or more scripts subsequently.

Persistent connection

If a unused persistent connection for a given combination of host, username, password, socket, port and default database can not be found in the connection pool, then `mysqli` opens a new connection. The use of persistent connections can be enabled and disabled using the PHP directive `mysqli.allow_persistent`. The total number of connections opened by a script can be limited with `mysqli.max_links`. The maximum number of persistent connections per PHP process can be restricted with `mysqli.max_persistent`. Please note, that the web server may spawn many PHP processes.

A common complain about persistent connections is that their state is not reset before reuse. For example, open and unfinished transactions are not automatically rolled back. But also, authorization changes which happened in the time between putting the connection into the pool and reusing it are not reflected. This may be seen as an unwanted side-effect. On the contrary, the name `persistent` may be understood as a promise that the state is persisted.

The `mysqli` extension supports both interpretations of a persistent connection: state persisted, and state reset before reuse. The default is reset. Before a persistent connection is reused, the `mysqli` extension implicitly calls `mysqli_change_user` to reset the state. The persistent connection appears to the user as if it was just opened. No artifacts from previous usages are visible.

The `mysqli_change_user` function is an expensive operation. For best performance, users may want to recompile the extension with the compile flag `MYSQLI_NO_CHANGE_USER_ON_PCONNECT` being set.

It is left to the user to choose between safe behavior and best performance. Both are valid optimization goals. For ease of use, the safe behavior has been made the default at the expense of maximum performance.

See also

```
mysqli::__construct  
mysqli::init  
mysqli::options  
mysqli::real_connect  
mysqli::change_user  
$mysqli::host_info  
MySQLi Configuration Options  
Persistent Database Connections
```

6.3.2.3 Executing statements

Copyright 1997-2014 the PHP Documentation Group.

Statements can be executed with the `mysqli_query`, `mysqli_real_query` and `mysqli_multi_query` functions. The `mysqli_query` function is the most common, and combines

the executing statement with a buffered fetch of its result set, if any, in one call. Calling `mysqli_query` is identical to calling `mysqli_real_query` followed by `mysqli_store_result`.

Example 6.15 Connecting to MySQL

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Buffered result sets

After statement execution results can be retrieved at once to be buffered by the client or by read row by row. Client-side result set buffering allows the server to free resources associated with the statement results as early as possible. Generally speaking, clients are slow consuming result sets. Therefore, it is recommended to use buffered result sets. `mysqli_query` combines statement execution and result set buffering.

PHP applications can navigate freely through buffered results. Navigation is fast because the result sets are held in client memory. Please, keep in mind that it is often easier to scale by client than it is to scale the server.

Example 6.16 Navigation through buffered results

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
$res = $mysqli->query("SELECT id FROM test ORDER BY id ASC");
echo "Reverse order...\n";
for ($row_no = $res->num_rows - 1; $row_no >= 0; $row_no--) {
    $res->data_seek($row_no);
    $row = $res->fetch_assoc();
    echo " id = " . $row['id'] . "\n";
}
echo "Result set order...\n";
$res->data_seek(0);
while ($row = $res->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
?>
```

The above example will output:

```
Reverse order...
 id = 3
```

```

id = 2
id = 1
Result set order...
id = 1
id = 2
id = 3

```

Unbuffered result sets

If client memory is a short resource and freeing server resources as early as possible to keep server load low is not needed, unbuffered results can be used. Scrolling through unbuffered results is not possible before all rows have been read.

Example 6.17 Navigation through unbuffered results

```

<?php
$mysqli->real_query("SELECT id FROM test ORDER BY id ASC");
$res = $mysqli->use_result();
echo "Result set order...\n";
while ($row = $res->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
?>

```

Result set values data types

The `mysqli_query`, `mysqli_real_query` and `mysqli_multi_query` functions are used to execute non-prepared statements. At the level of the MySQL Client Server Protocol, the command `COM_QUERY` and the text protocol are used for statement execution. With the text protocol, the MySQL server converts all data of a result sets into strings before sending. This conversion is done regardless of the SQL result set column data type. The mysql client libraries receive all column values as strings. No further client-side casting is done to convert columns back to their native types. Instead, all values are provided as PHP strings.

Example 6.18 Text protocol returns strings by default

```

<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
$res = $mysqli->query("SELECT id, label FROM test WHERE id = 1");
$row = $res->fetch_assoc();
printf("id = %s (%s)\n", $row['id'], gettype($row['id']));
printf("label = %s (%s)\n", $row['label'], gettype($row['label']));
?>

```

The above example will output:

```

id = 1 (string)
label = a (string)

```

It is possible to convert integer and float columns back to PHP numbers by setting the `MYSQLI_OPT_INT_AND_FLOAT_NATIVE` connection option, if using the `mysqlnd` library. If set, the `mysqlnd` library will check the result set meta data column types and convert numeric SQL columns to PHP numbers, if the PHP data type value range allows for it. This way, for example, SQL INT columns are returned as integers.

Example 6.19 Native data types with `mysqlnd` and connection option

```
<?php
$mysqli = mysqli_init();
$mysqli->options(MYSQLI_OPT_INT_AND_FLOAT_NATIVE, 1);
$mysqli->real_connect("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
$res = $mysqli->query("SELECT id, label FROM test WHERE id = 1");
$row = $res->fetch_assoc();
printf("id = %s (%s)\n", $row['id'], gettype($row['id']));
printf("label = %s (%s)\n", $row['label'], gettype($row['label']));
?>
```

The above example will output:

```
id = 1 (integer)
label = a (string)
```

See also

```
mysqli::__construct
mysqli::init
mysqli::options
mysqli::real_connect
mysqli::query
mysqli::multi_query
mysqli::use_result
mysqli::store_result
mysqli_result::free
```

6.3.2.4 Prepared Statements

Copyright 1997-2014 the PHP Documentation Group.

The MySQL database supports prepared statements. A prepared statement or a parameterized statement is used to execute the same statement repeatedly with high efficiency.

Basic workflow

The prepared statement execution consists of two stages: prepare and execute. At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use.

The MySQL server supports using anonymous, positional placeholder with `?`.

Example 6.20 First stage: prepare

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
/* Non-prepared statement */
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
/* Prepared statement, stage 1: prepare */
if (!$stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)")) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Prepare is followed by execute. During execute the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values to execute it using the previously created internal resources.

Example 6.21 Second stage: bind and execute

```
<?php
/* Prepared statement, stage 2: bind and execute */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
    echo "Binding parameters failed: (" . $stmt->errno . ") " . $stmt->error;
}
if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
?>
```

Repeated execution

A prepared statement can be executed repeatedly. Upon every execution the current value of the bound variable is evaluated and sent to the server. The statement is not parsed again. The statement template is not transferred to the server again.

Example 6.22 INSERT prepared once, executed multiple times

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
/* Non-prepared statement */
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
/* Prepared statement, stage 1: prepare */
if (!$stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)")) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
/* Prepared statement, stage 2: bind and execute */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
```

```

        echo "Binding parameters failed: (" . $stmt->errno . ") " . $stmt->error;
    }
    if (!$stmt->execute()) {
        echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
    }
    /* Prepared statement: repeated execution, only data transferred from client to server */
    for ($id = 2; $id < 5; $id++) {
        if (!$stmt->execute()) {
            echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
        }
    }
    /* explicit close recommended */
    $stmt->close();
    /* Non-prepared statement */
    $res = $mysqli->query("SELECT id FROM test");
    var_dump($res->fetch_all());
?>

```

The above example will output:

```

array(4) {
  [0]=>
  array(1) {
    [0]=>
    string(1) "1"
  }
  [1]=>
  array(1) {
    [0]=>
    string(1) "2"
  }
  [2]=>
  array(1) {
    [0]=>
    string(1) "3"
  }
  [3]=>
  array(1) {
    [0]=>
    string(1) "4"
  }
}

```

Every prepared statement occupies server resources. Statements should be closed explicitly immediately after use. If not done explicitly, the statement will be closed when the statement handle is freed by PHP.

Using a prepared statement is not always the most efficient way of executing a statement. A prepared statement executed only once causes more client-server round-trips than a non-prepared statement. This is why the [SELECT](#) is not run as a prepared statement above.

Also, consider the use of the MySQL multi-INSERT SQL syntax for INSERTs. For the example, multi-INSERT requires less round-trips between the server and client than the prepared statement shown above.

Example 6.23 Less round trips using multi-INSERT SQL

```

<?php
if (!$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3), (4)")) {
    echo "Multi-INSERT failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>

```


Result set values data types

The MySQL Client Server Protocol defines a different data transfer protocol for prepared statements and non-prepared statements. Prepared statements are using the so called binary protocol. The MySQL server sends result set data "as is" in binary format. Results are not serialized into strings before sending. The client libraries do not receive strings only. Instead, they will receive binary data and try to convert the values into appropriate PHP data types. For example, results from an SQL `INT` column will be provided as PHP integer variables.

Example 6.24 Native datatypes

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
$stmt = $mysqli->prepare("SELECT id, label FROM test WHERE id = 1");
$stmt->execute();
$res = $stmt->get_result();
$row = $res->fetch_assoc();
printf("id = %s (%s)\n", $row['id'], gettype($row['id']));
printf("label = %s (%s)\n", $row['label'], gettype($row['label']));
?>
```

The above example will output:

```
id = 1 (integer)
label = a (string)
```

This behavior differs from non-prepared statements. By default, non-prepared statements return all results as strings. This default can be changed using a connection option. If the connection option is used, there are no differences.

Fetching results using bound variables

Results from prepared statements can either be retrieved by binding output variables, or by requesting a `mysqli_result` object.

Output variables must be bound after statement execution. One variable must be bound for every column of the statements result set.

Example 6.25 Output variable binding

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
```

```

!$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
!$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')") {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!($stmt = $mysqli->prepare("SELECT id, label FROM test"))) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$stmt->execute()) {
    echo "Execute failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
$out_id = NULL;
$out_label = NULL;
if (!$stmt->bind_result($out_id, $out_label)) {
    echo "Binding output parameters failed: (" . $stmt->errno . ") " . $stmt->error;
}
while ($stmt->fetch()) {
    printf("id = %s (%s), label = %s (%s)\n", $out_id, gettype($out_id), $out_label, gettype($out_label));
}
?>

```

The above example will output:

```
id = 1 (integer), label = a (string)
```

Prepared statements return unbuffered result sets by default. The results of the statement are not implicitly fetched and transferred from the server to the client for client-side buffering. The result set takes server resources until all results have been fetched by the client. Thus it is recommended to consume results timely. If a client fails to fetch all results or the client closes the statement before having fetched all data, the data has to be fetched implicitly by [mysqli](#).

It is also possible to buffer the results of a prepared statement using [mysqli_stmt_store_result](#).

Fetching results using mysqli_result interface

Instead of using bound results, results can also be retrieved through the [mysqli_result](#) interface. [mysqli_stmt_get_result](#) returns a buffered result set.

Example 6.26 Using mysqli_result to fetch results

```

<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
!$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
!$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!($stmt = $mysqli->prepare("SELECT id, label FROM test ORDER BY id ASC"))) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
if (!($res = $stmt->get_result())) {
    echo "Getting result set failed: (" . $stmt->errno . ") " . $stmt->error;
}
var_dump($res->fetch_all());
?>

```

The above example will output:

```
array(1) {
  [0]=>
    array(2) {
      [0]=>
        int(1)
      [1]=>
        string(1) "a"
    }
}
```

Using the [mysqli_result interface](#) offers the additional benefit of flexible client-side result set navigation.

Example 6.27 Buffered result set for flexible read out

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT, label CHAR(1))") ||
    !$mysqli->query("INSERT INTO test(id, label) VALUES (1, 'a'), (2, 'b'), (3, 'c')")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!($stmt = $mysqli->prepare("SELECT id, label FROM test"))) {
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
if (!($res = $stmt->get_result())) {
    echo "Getting result set failed: (" . $stmt->errno . ") " . $stmt->error;
}
for ($row_no = ($res->num_rows - 1); $row_no >= 0; $row_no--) {
    $res->data_seek($row_no);
    var_dump($res->fetch_assoc());
}
$res->close();
?>
```

The above example will output:

```
array(2) {
  ["id"]=>
    int(3)
  ["label"]=>
    string(1) "c"
}
array(2) {
  ["id"]=>
    int(2)
  ["label"]=>
    string(1) "b"
}
array(2) {
  ["id"]=>
```

```

int(1)
["label"]=>
string(1) "a"
}

```

Escaping and SQL injection

Bound variables are sent to the server separately from the query and thus cannot interfere with it. The server uses these values directly at the point of execution, after the statement template is parsed. Bound parameters do not need to be escaped as they are never substituted into the query string directly. A hint must be provided to the server for the type of bound variable, to create an appropriate conversion. See the [mysqli_stmt_bind_param](#) function for more information.

Such a separation sometimes considered as the only security feature to prevent SQL injection, but the same degree of security can be achieved with non-prepared statements, if all the values are formatted correctly. It should be noted that correct formatting is not the same as escaping and involves more logic than simple escaping. Thus, prepared statements are simply a more convenient and less error-prone approach to this element of database security.

Client-side prepared statement emulation

The API does not include emulation for client-side prepared statement emulation.

Quick prepared - non-prepared statement comparison

The table below compares server-side prepared and non-prepared statements.

Table 6.2 Comparison of prepared and non-prepared statements

	Prepared Statement	Non-prepared statement
Client-server round trips, SELECT, single execution	2	1
Statement string transferred from client to server	1	1
Client-server round trips, SELECT, repeated (n) execution	1 + n	n
Statement string transferred from client to server	1 template, n times bound parameter, if any	n times together with parameter, if any
Input parameter binding API	Yes, automatic input escaping	No, manual input escaping
Output variable binding API	Yes	No
Supports use of mysqli_result API	Yes, use mysqli_stmt_get_result	Yes
Buffered result sets	Yes, use mysqli_stmt_get_result or binding with mysqli_stmt_store_result	Yes, default of mysqli_query
Unbuffered result sets	Yes, use output binding API	Yes, use mysqli_real_query with mysqli_use_result
MySQL Client Server protocol data transfer flavor	Binary protocol	Text protocol
Result set values SQL data types	Preserved when fetching	Converted to string or preserved when fetching
Supports all SQL statements	Recent MySQL versions support most but not all	Yes

See also

```
mysqli::__construct  
mysqli::query  
mysqli::prepare  
mysqli_stmt::prepare  
mysqli_stmt::execute  
mysqli_stmt::bind_param  
mysqli_stmt::bind_result
```

6.3.2.5 Stored Procedures

[Copyright 1997-2014 the PHP Documentation Group.](#)

The MySQL database supports stored procedures. A stored procedure is a subroutine stored in the database catalog. Applications can call and execute the stored procedure. The [CALL](#) SQL statement is used to execute a stored procedure.

Parameter

Stored procedures can have [IN](#), [INOUT](#) and [OUT](#) parameters, depending on the MySQL version. The [mysqli](#) interface has no special notion for the different kinds of parameters.

IN parameter

Input parameters are provided with the [CALL](#) statement. Please, make sure values are escaped correctly.

Example 6.28 Calling a stored procedure

```
<?php  
$mysqli = new mysqli("example.com", "user", "password", "database");  
if ($mysqli->connect_errno) {  
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;  
}  
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {  
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;  
}  
if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||  
    !$mysqli->query("CREATE PROCEDURE p(IN id_val INT) BEGIN INSERT INTO test(id) VALUES(id_val); END;")) {  
    echo "Stored procedure creation failed: (" . $mysqli->errno . ") " . $mysqli->error;  
}  
if (!$mysqli->query("CALL p(1)")) {  
    echo "CALL failed: (" . $mysqli->errno . ") " . $mysqli->error;  
}  
if (!$res = $mysqli->query("SELECT id FROM test")) {  
    echo "SELECT failed: (" . $mysqli->errno . ") " . $mysqli->error;  
}  
var_dump($res->fetch_assoc());  
?>
```

The above example will output:

```
array(1) {  
    ["id"]=>  
        string(1) "1"  
}
```

INOUT/OUT parameter

The values of [INOUT/OUT](#) parameters are accessed using session variables.

Example 6.29 Using session variables

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||
    !$mysqli->query('CREATE PROCEDURE p(OUT msg VARCHAR(50)) BEGIN SELECT "Hi!" INTO msg; END;')) {
    echo "Stored procedure creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$mysqli->query("SET @msg = ''") || !$mysqli->query("CALL p(@msg)")) {
    echo "CALL failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$res = $mysqli->query("SELECT @msg as _p_out")) {
    echo "Fetch failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
$row = $res->fetch_assoc();
echo $row['_p_out'];
?>
```

The above example will output:

```
Hi!
```

Application and framework developers may be able to provide a more convenient API using a mix of session variables and databased catalog inspection. However, please note the possible performance impact of a custom solution based on catalog inspection.

Handling result sets

Stored procedures can return result sets. Result sets returned from a stored procedure cannot be fetched correctly using [mysqli_query](#). The [mysqli_query](#) function combines statement execution and fetching the first result set into a buffered result set, if any. However, there are additional stored procedure result sets hidden from the user which cause [mysqli_query](#) to fail returning the user expected result sets.

Result sets returned from a stored procedure are fetched using [mysqli_real_query](#) or [mysqli_multi_query](#). Both functions allow fetching any number of result sets returned by a statement, such as [CALL](#). Failing to fetch all result sets returned by a stored procedure causes an error.

Example 6.30 Fetching results from stored procedures

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
}
```

```
if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||
    !$mysqli->query('CREATE PROCEDURE p() READS SQL DATA BEGIN SELECT id FROM test; SELECT id + 1 FROM
    echo "Stored procedure creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$mysqli->multi_query("CALL p()")) {
    echo "CALL failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
do {
    if ($res = $mysqli->store_result()) {
        printf("---\n");
        var_dump($res->fetch_all());
        $res->free();
    } else {
        if ($mysqli->errno) {
            echo "Store failed: (" . $mysqli->errno . ") " . $mysqli->error;
        }
    }
} while ($mysqli->more_results() && $mysqli->next_result());
?>
```

The above example will output:

```
---
array(3) {
    [0]=>
    array(1) {
        [0]=>
        string(1) "1"
    }
    [1]=>
    array(1) {
        [0]=>
        string(1) "2"
    }
    [2]=>
    array(1) {
        [0]=>
        string(1) "3"
    }
}
---
array(3) {
    [0]=>
    array(1) {
        [0]=>
        string(1) "2"
    }
    [1]=>
    array(1) {
        [0]=>
        string(1) "3"
    }
    [2]=>
    array(1) {
        [0]=>
        string(1) "4"
    }
}
```

Use of prepared statements

No special handling is required when using the prepared statement interface for fetching results from the same stored procedure as above. The prepared statement and non-prepared statement interfaces are similar. Please note, that not every MYSQL server version may support preparing the [CALL](#) SQL statement.

Example 6.31 Stored Procedures and Prepared Statements

```

<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$mysqli->query("DROP PROCEDURE IF EXISTS p") ||
    !$mysqli->query('CREATE PROCEDURE p() READS SQL DATA BEGIN SELECT id FROM test; SELECT id + 1 FROM test;')) {
    echo "Stored procedure creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!($stmt = $mysqli->prepare("CALL p()"))){
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
do {
    if ($res = $stmt->get_result()) {
        printf("---\n");
        var_dump(mysqli_fetch_all($res));
        mysqli_free_result($res);
    } else {
        if ($stmt->errno) {
            echo "Store failed: (" . $stmt->errno . ") " . $stmt->error;
        }
    }
} while ($stmt->more_results() && $stmt->next_result());
?>

```

Of course, use of the bind API for fetching is supported as well.

Example 6.32 Stored Procedures and Prepared Statements using bind API

```

<?php
if (!($stmt = $mysqli->prepare("CALL p()"))){
    echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
if (!$stmt->execute()) {
    echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;
}
do {
    $id_out = NULL;
    if (!$stmt->bind_result($id_out)) {
        echo "Bind failed: (" . $stmt->errno . ") " . $stmt->error;
    }

    while ($stmt->fetch()) {
        echo "id = $id_out\n";
    }
} while ($stmt->more_results() && $stmt->next_result());
?>

```

See also

[mysqli::query](#)
[mysqli::multi_query](#)
[mysqli_result::next-result](#)

[mysqli_result::more-results](#)

6.3.2.6 Multiple Statements

Copyright 1997-2014 the PHP Documentation Group.

MySQL optionally allows having multiple statements in one statement string. Sending multiple statements at once reduces client-server round trips but requires special handling.

Multiple statements or multi queries must be executed with [mysqli_multi_query](#). The individual statements of the statement string are separated by semicolon. Then, all result sets returned by the executed statements must be fetched.

The MySQL server allows having statements that do return result sets and statements that do not return result sets in one multiple statement.

Example 6.33 Multiple Statements

```

<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
$sql = "SELECT COUNT(*) AS _num FROM test; ";
$sql.= "INSERT INTO test(id) VALUES (1); ";
$sql.= "SELECT COUNT(*) AS _num FROM test; ";
if (!$mysqli->multi_query($sql)) {
    echo "Multi query failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
do {
    if ($res = $mysqli->store_result()) {
        var_dump($res->fetch_all(MYSQLI_ASSOC));
        $res->free();
    }
} while ($mysqli->more_results() && $mysqli->next_result());
?>

```

The above example will output:

```

array(1) {
    [0]=>
    array(1) {
        ["_num"]=>
        string(1) "0"
    }
}
array(1) {
    [0]=>
    array(1) {
        ["_num"]=>
        string(1) "1"
    }
}

```

Security considerations

The API functions [mysqli_query](#) and [mysqli_real_query](#) do not set a connection flag necessary for activating multi queries in the server. An extra API call is used for multiple statements to reduce

the likeliness of accidental SQL injection attacks. An attacker may try to add statements such as `; DROP DATABASE mysql` or `; SELECT SLEEP(999)`. If the attacker succeeds in adding SQL to the statement string but `mysqli_multi_query` is not used, the server will not execute the second, injected and malicious SQL statement.

Example 6.34 SQL Injection

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
$res    = $mysqli->query("SELECT 1; DROP TABLE mysql.user");
if (!$res) {
    echo "Error executing query: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

The above example will output:

```
Error executing query: (1064) You have an error in your SQL syntax;
check the manual that corresponds to your MySQL server version for the right syntax
to use near 'DROP TABLE mysql.user' at line 1
```

Prepared statements

Use of the multiple statement with prepared statements is not supported.

See also

```
mysqli::query
mysqli::multi_query
mysqli_result::next_result
mysqli_result::more_results
```

6.3.2.7 API support for transactions

Copyright 1997-2014 the PHP Documentation Group.

The MySQL server supports transactions depending on the storage engine used. Since MySQL 5.5, the default storage engine is InnoDB. InnoDB has full ACID transaction support.

Transactions can either be controlled using SQL or API calls. It is recommended to use API calls for enabling and disabling the auto commit mode and for committing and rolling back transactions.

Example 6.35 Setting auto commit mode with SQL and through the API

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
/* Recommended: using API to control transactional settings */
$mysqli->autocommit(false);
/* Won't be monitored and recognized by the replication and the load balancing plugin */
if (!$mysqli->query('SET AUTOCOMMIT = 0')) {
    echo "Query failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Optional feature packages, such as the replication and load balancing plugin, can easily monitor API calls. The replication plugin offers transaction aware load balancing, if transactions are controlled with API calls. Transaction aware load balancing is not available if SQL statements are used for setting auto commit mode, committing or rolling back a transaction.

Example 6.36 Commit and rollback

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
$mysqli->autocommit(false);
$mysqli->query("INSERT INTO test(id) VALUES (1)");
$mysqli->rollback();
$mysqli->query("INSERT INTO test(id) VALUES (2)");
$mysqli->commit();
?>
```

Please note, that the MySQL server cannot roll back all statements. Some statements cause an implicit commit.

See also

[mysqli::autocommit](#)
[mysqli_result::commit](#)
[mysqli_result::rollback](#)

6.3.2.8 Metadata

[Copyright 1997-2014 the PHP Documentation Group.](#)

A MySQL result set contains metadata. The metadata describes the columns found in the result set. All metadata sent by MySQL is accessible through the [mysqli](#) interface. The extension performs no or negligible changes to the information it receives. Differences between MySQL server versions are not aligned.

Meta data is access through the [mysqli_result](#) interface.

Example 6.37 Accessing result set meta data

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
$res = $mysqli->query("SELECT 1 AS _one, 'Hello' AS _two FROM DUAL");
var_dump($res->fetch_fields());
?>
```

The above example will output:

```
array(2) {
  [0]=>
  object(stdClass)#3 (13) {
    ["name"]=>
    string(4) "_one"
    ["orgname"]=>
    string(0) ""
    ["table"]=>
    string(0) ""
  }
}
```

```
["orgtable"]=>
string(0) ""
["def"]=>
string(0) ""
["db"]=>
string(0) ""
["catalog"]=>
string(3) "def"
["max_length"]=>
int(1)
["length"]=>
int(1)
["charsetnr"]=>
int(63)
["flags"]=>
int(32897)
["type"]=>
int(8)
["decimals"]=>
int(0)
}
[1]=>
object(stdClass)#4 (13) {
    ["name"]=>
    string(4) "_two"
    ["orgname"]=>
    string(0) ""
    ["table"]=>
    string(0) ""
    ["orgtable"]=>
    string(0) ""
    ["def"]=>
    string(0) ""
    ["db"]=>
    string(0) ""
    ["catalog"]=>
    string(3) "def"
    ["max_length"]=>
    int(5)
    ["length"]=>
    int(5)
    ["charsetnr"]=>
    int(8)
    ["flags"]=>
    int(1)
    ["type"]=>
    int(253)
    ["decimals"]=>
    int(31)
}
}
```

Prepared statements

Meta data of result sets created using prepared statements are accessed the same way. A suitable `mysqli_result` handle is returned by `mysqli_stmt_result_metadata`.

Example 6.38 Prepared statements metadata

```
<?php
$stmt = $mysqli->prepare("SELECT 1 AS _one, 'Hello' AS _two FROM DUAL");
$stmt->execute();
$res = $stmt->result_metadata();
var_dump($res->fetch_fields());
?>
```

See also

```
mysqli::query
mysqli_result::fetch_fields
```

6.3.3 Installing/Configuring

Copyright 1997-2014 the PHP Documentation Group.

6.3.3.1 Requirements

Copyright 1997-2014 the PHP Documentation Group.

In order to have these functions available, you must compile PHP with support for the mysqli extension.

Note

The mysqli extension is designed to work with MySQL version 4.1.13 or newer, or 5.0.7 or newer. For previous versions, please see the [MySQL](#) extension documentation.

6.3.3.2 Installation

Copyright 1997-2014 the PHP Documentation Group.

The `mysqli` extension was introduced with PHP version 5.0.0. The MySQL Native Driver was included in PHP version 5.3.0.

Installation on Linux

Copyright 1997-2014 the PHP Documentation Group.

The common Unix distributions include binary versions of PHP that can be installed. Although these binary versions are typically built with support for the MySQL extensions, the extension libraries themselves may need to be installed using an additional package. Check the package manager that comes with your chosen distribution for availability.

For example, on Ubuntu the `php5-mysql` package installs the `ext/mysql`, `ext/mysqli`, and `pdo_mysql` PHP extensions. On CentOS, the `php-mysql` package also installs these three PHP extensions.

Alternatively, you can compile this extension yourself. Building PHP from source allows you to specify the MySQL extensions you want to use, as well as your choice of client library for each extension.

The MySQL Native Driver is the recommended client library option, as it results in improved performance and gives access to features not available when using the MySQL Client Library. Refer to [What is PHP's MySQL Native Driver?](#) for a brief overview of the advantages of MySQL Native Driver.

The `/path/to/mysqli_config` represents the location of the `mysqli_config` program that comes with MySQL Server.

Table 6.3 mysqli compile time support matrix

PHP Version	Default	Configure Options: <code>mysqli</code>	Configure Options: <code>libmysqlclient</code>	Changelog
5.4.x and above	<code>mysqli</code>	<code>--with-mysqli</code>	<code>--with-mysqli=/path/to/mysqli_config</code>	<code>mysqli</code> is the default
5.3.x	<code>libmysqlclient</code>	<code>--with-mysqli=mysqli</code>	<code>--with-mysqli=/</code>	<code>mysqli</code> is supported

PHP Version	Default	Configure Options: <code>mysqlnd</code>	Configure Options: <code>libmysqlclient</code>	Changelog
			<code>path/to/mysql_config</code>	
5.0.x, 5.1.x, 5.2.x	<code>libmysqlclient</code>	Not Available	<code>--with-mysql=/path/to/mysql_config</code>	<code>mysqlnd</code> is not supported

Note that it is possible to freely mix MySQL extensions and client libraries. For example, it is possible to enable the MySQL extension to use the MySQL Client Library (`libmysqlclient`), while configuring the `mysqli` extension to use the MySQL Native Driver. However, all permutations of extension and client library are possible.

Installation on Windows Systems

Copyright 1997-2014 the PHP Documentation Group.

On Windows, PHP is most commonly installed using the binary installer.

PHP 5.3.0 and newer

Copyright 1997-2014 the PHP Documentation Group.

On Windows, for PHP versions 5.3 and newer, the `mysqli` extension is enabled and uses the MySQL Native Driver by default. This means you don't need to worry about configuring access to `libmysql.dll`.

PHP 5.0, 5.1, 5.2

Copyright 1997-2014 the PHP Documentation Group.

On these old unsupported PHP versions (PHP 5.2 reached EOL on '6 Jan 2011'), additional configuration procedures are required to enable `mysqli` and specify the client library you want it to use.

The `mysqli` extension is not enabled by default, so the `php_mysqli.dll` DLL must be enabled inside of `php.ini`. In order to do this you need to find the `php.ini` file (typically located in `c:\php`), and make sure you remove the comment (semi-colon) from the start of the line `extension=php_mysqli.dll`, in the section marked `[PHP_MYSQLI]`.

Also, if you want to use the MySQL Client Library with `mysqli`, you need to make sure PHP can access the client library file. The MySQL Client Library is included as a file named `libmysql.dll` in the Windows PHP distribution. This file needs to be available in the Windows system's `PATH` environment variable, so that it can be successfully loaded. See the FAQ titled "[How do I add my PHP directory to the PATH on Windows](#)" for information on how to do this. Copying `libmysql.dll` to the Windows system directory (typically `c:\Windows\system`) also works, as the system directory is by default in the system's `PATH`. However, this practice is strongly discouraged.

As with enabling any PHP extension (such as `php_mysqli.dll`), the PHP directive `extension_dir` should be set to the directory where the PHP extensions are located. See also the [Manual Windows Installation Instructions](#). An example `extension_dir` value for PHP 5 is `c:\php\ext`.

Note

If when starting the web server an error similar to the following occurs: "`Unable to load dynamic library './php_mysqli.dll'`", this is because `php_mysqli.dll` and/or `libmysql.dll` cannot be found by the system.

6.3.3.3 Runtime Configuration

Copyright 1997-2014 the PHP Documentation Group.

The behaviour of these functions is affected by settings in [php.ini](#).

Table 6.4 MySQLi Configuration Options

Name	Default	Changeable	Changelog
mysqli.allow_local_infile	"1"	PHP_INI_SYSTEM	Available since PHP 5.2.4.
mysqli.allow_persistent	"1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqli.max_persistent	"-1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqli.max_links	"-1"	PHP_INI_SYSTEM	Available since PHP 5.0.0.
mysqli.default_port	"3306"	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.default_socket	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.default_host	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.default_user	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.default_pw	NULL	PHP_INI_ALL	Available since PHP 5.0.0.
mysqli.reconnect	"0"	PHP_INI_SYSTEM	Available since PHP 4.3.5.
mysqli.rollback_on_cached_stored_proc	TRUE	PHP_INI_SYSTEM	Available since PHP 5.6.0.
mysqli.cache_size	"2000"	PHP_INI_SYSTEM	Available since PHP 5.3.0.

For further details and definitions of the preceding `PHP_INI_*` constants, see the chapter on [configuration changes](#).

Here's a short explanation of the configuration directives.

mysqli.allow_local_infile integer	Allow accessing, from PHP's perspective, local files with <code>LOAD DATA</code> statements
mysqli.allow_persistent integer	Enable the ability to create persistent connections using mysqli_connect .
mysqli.max_persistent integer	Maximum of persistent connections that can be made. Set to 0 for unlimited.
mysqli.max_links integer	The maximum number of MySQL connections per process.
mysqli.default_port integer	The default TCP port number to use when connecting to the database server if no other port is specified. If no default is specified, the port will be obtained from the <code>MYSQL_TCP_PORT</code> environment variable, the <code>mysql-tcp</code> entry in <code>/etc/services</code> or the compile-time <code>MYSQL_PORT</code> constant, in that order. Win32 will only use the <code>MYSQL_PORT</code> constant.

<code>mysqli.default_socket</code> string	The default socket name to use when connecting to a local database server if no other socket name is specified.
<code>mysqli.default_host</code> string	The default server host to use when connecting to the database server if no other host is specified. Doesn't apply in safe mode .
<code>mysqli.default_user</code> string	The default user name to use when connecting to the database server if no other name is specified. Doesn't apply in safe mode .
<code>mysqli.default_pw</code> string	The default password to use when connecting to the database server if no other password is specified. Doesn't apply in safe mode .
<code>mysqli.reconnect</code> integer	Automatically reconnect if the connection was lost.
<code>mysqli.rollback_on_cached_ipool</code> bool	Used for rolling back connections put back into the persistent connection pool.
<code>mysqli.cache_size</code> integer	Available only with mysqlnd .

Users cannot set `MYSQL_OPT_READ_TIMEOUT` through an API call or runtime configuration setting. Note that if it were possible there would be differences between how `libmysqlclient` and streams would interpret the value of `MYSQL_OPT_READ_TIMEOUT`.

6.3.3.4 Resource Types

Copyright 1997-2014 the PHP Documentation Group.

This extension has no resource types defined.

6.3.4 The mysqli Extension and Persistent Connections

Copyright 1997-2014 the PHP Documentation Group.

Persistent connection support was introduced in PHP 5.3 for the `mysqli` extension. Support was already present in PDO MySQL and ext/mysql. The idea behind persistent connections is that a connection between a client process and a database can be reused by a client process, rather than being created and destroyed multiple times. This reduces the overhead of creating fresh connections every time one is required, as unused connections are cached and ready to be reused.

Unlike the mysql extension, mysqli does not provide a separate function for opening persistent connections. To open a persistent connection you must prepend `p:` to the hostname when connecting.

The problem with persistent connections is that they can be left in unpredictable states by clients. For example, a table lock might be activated before a client terminates unexpectedly. A new client process reusing this persistent connection will get the connection “as is”. Any cleanup would need to be done by the new client process before it could make good use of the persistent connection, increasing the burden on the programmer.

The persistent connection of the `mysqli` extension however provides built-in cleanup handling code. The cleanup carried out by `mysqli` includes:

- Rollback active transactions
- Close and drop temporary tables
- Unlock tables
- Reset session variables
- Close prepared statements (always happens with PHP)
- Close handler

- Release locks acquired with [GET_LOCK](#)

This ensures that persistent connections are in a clean state on return from the connection pool, before the client process uses them.

The [mysqli](#) extension does this cleanup by automatically calling the C-API function [mysqli_change_user\(\)](#).

The automatic cleanup feature has advantages and disadvantages though. The advantage is that the programmer no longer needs to worry about adding cleanup code, as it is called automatically. However, the disadvantage is that the code could *potentially* be a little slower, as the code to perform the cleanup needs to run each time a connection is returned from the connection pool.

It is possible to switch off the automatic cleanup code, by compiling PHP with [MYSQLI_NO_CHANGE_USER_ON_PCONNECT](#) defined.

Note

The [mysqli](#) extension supports persistent connections when using either MySQL Native Driver or MySQL Client Library.

6.3.5 Predefined Constants

Copyright 1997-2014 the PHP Documentation Group.

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

MYSQLI_READ_DEFAULT_GROUP	Read options from the named group from my.cnf or the file specified with MYSQLI_READ_DEFAULT_FILE
MYSQLI_READ_DEFAULT_FILE	Read options from the named option file instead of from my.cnf
MYSQLI_OPT_CONNECT_TIMEOUT	Connect timeout in seconds
MYSQLI_OPT_LOCAL_INFILE	Enables command LOAD LOCAL INFILE
MYSQLI_INIT_COMMAND	Command to execute when connecting to MySQL server. Will automatically be re-executed when reconnecting.
MYSQLI_CLIENT_SSL	Use SSL (encrypted protocol). This option should not be set by application programs; it is set internally in the MySQL client library
MYSQLI_CLIENT_COMPRESS	Use compression protocol
MYSQLI_CLIENT_INTERACTIVE	Allow interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable will be set to the value of the session interactive_timeout variable.
MYSQLI_CLIENT_IGNORE_SPACE	Allow spaces after function names. Makes all functions names reserved words.
MYSQLI_CLIENT_NO_SCHEMA	Don't allow the db_name.tbl_name.col_name syntax.
MYSQLI_CLIENT_MULTI_QUERIES	Allows multiple semicolon-delimited queries in a single mysqli_query call.
MYSQLI_STORE_RESULT	For using buffered resultsets
MYSQLI_USE_RESULT	For using unbuffered resultsets

<code>MYSQLI_ASSOC</code>	Columns are returned into the array having the fieldname as the array index.
<code>MYSQLI_NUM</code>	Columns are returned into the array having an enumerated index.
<code>MYSQLI_BOTH</code>	Columns are returned into the array having both a numerical index and the fieldname as the associative index.
<code>MYSQLI_NOT_NULL_FLAG</code>	Indicates that a field is defined as <code>NOT NULL</code>
<code>MYSQLI_PRI_KEY_FLAG</code>	Field is part of a primary index
<code>MYSQLI_UNIQUE_KEY_FLAG</code>	Field is part of a unique index.
<code>MYSQLI_MULTIPLE_KEY_FLAG</code>	Field is part of an index.
<code>MYSQLI_BLOB_FLAG</code>	Field is defined as <code>BLOB</code>
<code>MYSQLI_UNSIGNED_FLAG</code>	Field is defined as <code>UNSIGNED</code>
<code>MYSQLI_ZEROFILL_FLAG</code>	Field is defined as <code>ZEROFILL</code>
<code>MYSQLI_AUTO_INCREMENT_FLAG</code>	Field is defined as <code>AUTO_INCREMENT</code>
<code>MYSQLI_TIMESTAMP_FLAG</code>	Field is defined as <code>TIMESTAMP</code>
<code>MYSQLI_SET_FLAG</code>	Field is defined as <code>SET</code>
<code>MYSQLI_NUM_FLAG</code>	Field is defined as <code>NUMERIC</code>
<code>MYSQLI_PART_KEY_FLAG</code>	Field is part of an multi-index
<code>MYSQLI_GROUP_FLAG</code>	Field is part of <code>GROUP BY</code>
<code>MYSQLI_TYPE_DECIMAL</code>	Field is defined as <code>DECIMAL</code>
<code>MYSQLI_TYPE_NEWDECIMAL</code>	Precision math <code>DECIMAL</code> or <code>NUMERIC</code> field (MySQL 5.0.3 and up)
<code>MYSQLI_TYPE_BIT</code>	Field is defined as <code>BIT</code> (MySQL 5.0.3 and up)
<code>MYSQLI_TYPE_TINY</code>	Field is defined as <code>TINYINT</code>
<code>MYSQLI_TYPE_SHORT</code>	Field is defined as <code>SMALLINT</code>
<code>MYSQLI_TYPE_LONG</code>	Field is defined as <code>INT</code>
<code>MYSQLI_TYPE_FLOAT</code>	Field is defined as <code>FLOAT</code>
<code>MYSQLI_TYPE_DOUBLE</code>	Field is defined as <code>DOUBLE</code>
<code>MYSQLI_TYPE_NULL</code>	Field is defined as <code>DEFAULT NULL</code>
<code>MYSQLI_TYPE_TIMESTAMP</code>	Field is defined as <code>TIMESTAMP</code>
<code>MYSQLI_TYPE_LONGLONG</code>	Field is defined as <code>BIGINT</code>
<code>MYSQLI_TYPE_INT24</code>	Field is defined as <code>MEDIUMINT</code>
<code>MYSQLI_TYPE_DATE</code>	Field is defined as <code>DATE</code>
<code>MYSQLI_TYPE_TIME</code>	Field is defined as <code>TIME</code>
<code>MYSQLI_TYPE_DATETIME</code>	Field is defined as <code>DATETIME</code>
<code>MYSQLI_TYPE_YEAR</code>	Field is defined as <code>YEAR</code>

<code>MYSQLI_TYPE_NEWDATE</code>	Field is defined as DATE
<code>MYSQLI_TYPE_INTERVAL</code>	Field is defined as INTERVAL
<code>MYSQLI_TYPE_ENUM</code>	Field is defined as ENUM
<code>MYSQLI_TYPE_SET</code>	Field is defined as SET
<code>MYSQLI_TYPE_TINY_BLOB</code>	Field is defined as TINYBLOB
<code>MYSQLI_TYPE_MEDIUM_BLOB</code>	Field is defined as MEDIUMBLOB
<code>MYSQLI_TYPE_LONG_BLOB</code>	Field is defined as LONGBLOB
<code>MYSQLI_TYPE_BLOB</code>	Field is defined as BLOB
<code>MYSQLI_TYPE_VAR_STRING</code>	Field is defined as VARCHAR
<code>MYSQLI_TYPE_STRING</code>	Field is defined as CHAR or BINARY
<code>MYSQLI_TYPE_CHAR</code>	Field is defined as TINYINT . For CHAR , see MYSQLI_TYPE_STRING
<code>MYSQLI_TYPE_GEOMETRY</code>	Field is defined as GEOMETRY
<code>MYSQLI_NEED_DATA</code>	More data available for bind variable
<code>MYSQLI_NO_DATA</code>	No more data available for bind variable
<code>MYSQLI_DATA_TRUNCATED</code>	Data truncation occurred. Available since PHP 5.1.0 and MySQL 5.0.5.
<code>MYSQLI_ENUM_FLAG</code>	Field is defined as ENUM . Available since PHP 5.3.0.
<code>MYSQLI_BINARY_FLAG</code>	Field is defined as BINARY . Available since PHP 5.3.0.
<code>MYSQLI_CURSOR_TYPE_FOR_UPDATE</code>	
<code>MYSQLI_CURSOR_TYPE_NO_CURSOR</code>	
<code>MYSQLI_CURSOR_TYPE_READ_ONLY</code>	
<code>MYSQLI_CURSOR_TYPE_SCROLLABLE</code>	
<code>MYSQLI_STMT_ATTR_CURSOR_TYPE</code>	
<code>MYSQLI_STMT_ATTR_PREFETCH_ROWS</code>	
<code>MYSQLI_STMT_ATTR_UPDATE_MAX_LENGTH</code>	
<code>MYSQLI_SET_CHARSET_NAME</code>	
<code>MYSQLI_REPORT_INDEX</code>	Report if no index or bad index was used in a query.
<code>MYSQLI_REPORT_ERROR</code>	Report errors from <code>mysqli</code> function calls.
<code>MYSQLI_REPORT_STRICT</code>	Throw a mysqli_sql_exception for errors instead of warnings.
<code>MYSQLI_REPORT_ALL</code>	Set all options on (report all).
<code>MYSQLI_REPORT_OFF</code>	Turns reporting off.
<code>MYSQLI_DEBUG_TRACE_ENABLED</code>	Is set to 1 if mysqli_debug functionality is enabled.
<code>MYSQLI_SERVER_QUERY_NO_GOOD_INDEX_USED</code>	

MYSQLI_SERVER_QUERY_NO_INDEX_USED

MYSQLI_REFRESH_GRANT	Refreshes the grant tables.
MYSQLI_REFRESH_LOG	Flushes the logs, like executing the FLUSH LOGS SQL statement.
MYSQLI_REFRESH_TABLES	Flushes the table cache, like executing the FLUSH TABLES SQL statement.
MYSQLI_REFRESH_HOSTS	Flushes the host cache, like executing the FLUSH HOSTS SQL statement.
MYSQLI_REFRESH_STATUS	Reset the status variables, like executing the FLUSH STATUS SQL statement.
MYSQLI_REFRESH_THREADS	Flushes the thread cache.
MYSQLI_REFRESH_SLAVE	On a slave replication server: resets the master server information, and restarts the slave. Like executing the RESET SLAVE SQL statement.
MYSQLI_REFRESH_MASTER	On a master replication server: removes the binary log files listed in the binary log index, and truncates the index file. Like executing the RESET MASTER SQL statement.
MYSQLI_TRANS_COR_AND_CHAIN	Appends "AND CHAIN" to <code>mysqli_commit</code> or <code>mysqli_rollback</code> .
MYSQLI_TRANS_COR_AND_NO_CHAIN	Appends "AND NO CHAIN" to <code>mysqli_commit</code> or <code>mysqli_rollback</code> .
MYSQLI_TRANS_COR_RELEASE	Appends "RELEASE" to <code>mysqli_commit</code> or <code>mysqli_rollback</code> .
MYSQLI_TRANS_COR_NO_RELEASE	Appends "NO RELEASE" to <code>mysqli_commit</code> or <code>mysqli_rollback</code> .
MYSQLI_TRANS_START_READ_ONLY	Start the transaction as "START TRANSACTION READ ONLY" with <code>mysqli_begin_transaction</code> .
MYSQLI_TRANS_START_READ_WRITE	Start the transaction as "START TRANSACTION READ WRITE" with <code>mysqli_begin_transaction</code> .
MYSQLI_TRANS_START_CONSISTENT_SNAPSHOT	Start the transaction as "START TRANSACTION WITH CONSISTENT SNAPSHOT" with <code>mysqli_begin_transaction</code> .

6.3.6 Notes

Copyright 1997-2014 the PHP Documentation Group.

Some implementation notes:

1. Support was added for **MYSQL_TYPE_GEOMETRY** to the MySQLi extension in PHP 5.3.
2. Note there are different internal implementations within `libmysqlclient` and `mysqlnd` for handling columns of type **MYSQL_TYPE_GEOMETRY**. Generally speaking, `mysqlnd` will allocate significantly less memory. For example, if there is a **POINT** column in a result set, `libmysqlclient` may pre-allocate up to 4GB of RAM although less than 50 bytes are needed for holding a **POINT** column in memory. Memory allocation is much lower, less than 50 bytes, if using `mysqlnd`.

6.3.7 The MySQLi Extension Function Summary

Copyright 1997-2014 the PHP Documentation Group.

Table 6.5 Summary of **mysqli** methods

mysqli Class			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Properties</i>			
<code>\$mysqli::affected_rows</code>	<code>mysqli_affected_rows</code>	N/A	Gets the number of affected rows in a previous MySQL operation
<code>\$mysqli::client_info</code>	<code>mysqli_get_client_info</code>	N/A	Returns the MySQL client version as a string
<code>\$mysqli::client_version</code>	<code>mysqli_get_client_version</code>	N/A	Returns MySQL client version info as an integer
<code>\$mysqli::connect_errno</code>	<code>mysqli_connect_errno</code>	N/A	Returns the error code from last connect call
<code>\$mysqli::connect_error</code>	<code>mysqli_connect_error</code>	N/A	Returns a string description of the last connect error
<code>\$mysqli::errno</code>	<code>mysqli_errno</code>	N/A	Returns the error code for the most recent function call
<code>\$mysqli::error</code>	<code>mysqli_error</code>	N/A	Returns a string description of the last error
<code>\$mysqli::field_count</code>	<code>mysqli_field_count</code>	N/A	Returns the number of columns for the most recent query
<code>\$mysqli::host_info</code>	<code>mysqli_get_host_info</code>	N/A	Returns a string representing the type of connection used
<code>\$mysqli::protocol_version</code>	<code>mysqli_get_proto_info</code>	N/A	Returns the version of the MySQL protocol used
<code>\$mysqli::server_info</code>	<code>mysqli_get_server_info</code>	N/A	Returns the version of the MySQL server
<code>\$mysqli::server_version</code>	<code>mysqli_get_server_version</code>	N/A	Returns the version of the MySQL server as an integer
<code>\$mysqli::info</code>	<code>mysqli_info</code>	N/A	Retrieves information about the most recently executed query
<code>\$mysqli::insert_id</code>	<code>mysqli_insert_id</code>	N/A	Returns the auto generated id used in the last query
<code>\$mysqli::sqlstate</code>	<code>mysqli_sqlstate</code>	N/A	Returns the SQLSTATE error from previous MySQL operation
<code>\$mysqli::warning_count</code>	<code>mysqli_warning_count</code>	N/A	Returns the number of warnings from the last query for the given link

mysqli Class			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Methods</i>			
<code>mysqli::autocommit</code>	<code>mysqli_autocommit</code>	N/A	Turns on or off auto-committing database modifications
<code>mysqli::change_user</code>	<code>mysqli_change_user</code>	N/A	Changes the user of the specified database connection
<code>mysqli::character_set_name</code> <code>mysqli::client_encoding</code>	<code>mysqli_character_set_name</code>	<code>mysqli_client_encoding</code>	Returns the default character set for the database connection
<code>mysqli::close</code>	<code>mysqli_close</code>	N/A	Closes a previously opened database connection
<code>mysqli::commit</code>	<code>mysqli_commit</code>	N/A	Commits the current transaction
<code>mysqli::__construct</code>	<code>mysqli_connect</code>	N/A	Open a new connection to the MySQL server [Note: static (i.e. class) method]
<code>mysqli::debug</code>	<code>mysqli_debug</code>	N/A	Performs debugging operations
<code>mysqli::dump_debug</code>	<code>mysqli_dump_debug_info</code>	N/A	Dump debugging information into the log
<code>mysqli::get_charset</code>	<code>mysqli_get_charset</code>	N/A	Returns a character set object
<code>mysqli::get_connection_stats</code>	<code>mysqli_get_connection_stats</code>	N/A	Returns client connection statistics. Available only with mysqlind .
<code>mysqli::get_client_info</code>	<code>mysqli_get_client_info</code>	N/A	Returns the MySQL client version as a string
<code>mysqli::get_client_stats</code>	<code>mysqli_get_client_stats</code>	N/A	Returns client per-process statistics. Available only with mysqlind .
<code>mysqli::get_cache_stats</code>	<code>mysqli_get_cache_stats</code>	N/A	Returns client Zval cache statistics. Available only with mysqlind .
<code>mysqli::get_server_info</code>	<code>mysqli_get_server_info</code>	N/A	Returns a string representing the version of the MySQL server that the MySQLi extension is connected to
<code>mysqli::get_warnings</code>	<code>mysqli_get_warnings</code>	N/A	NOT DOCUMENTED
<code>mysqli::init</code>	<code>mysqli_init</code>	N/A	Initializes MySQLi and returns a resource for use with

mysqli Class			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
			mysqli_real_connect. [Not called on an object, as it returns a \$mysqli object.]
mysqli::kill	mysqli_kill	N/A	Asks the server to kill a MySQL thread
mysqli::more_results	mysqli_more_results	N/A	Check if there are any more query results from a multi query
mysqli::multi_query	mysqli_multi_query	N/A	Performs a query on the database
mysqli::next_result	mysqli_next_result	N/A	Prepare next result from multi_query
mysqli::options	mysqli_options	mysqli_set_opt	Set options
mysqli::ping	mysqli_ping	N/A	Pings a server connection, or tries to reconnect if the connection has gone down
mysqli::prepare	mysqli_prepare	N/A	Prepare an SQL statement for execution
mysqli::query	mysqli_query	N/A	Performs a query on the database
mysqli::real_connect	mysqli_real_connect	N/A	Opens a connection to a mysql server
mysqli::real_escape_string mysqli::escape_string	mysqli_real_escape_string mysqli_escape_string	mysqli_real_escape_string mysqli_escape_string	Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection
mysqli::real_query	mysqli_real_query	N/A	Execute an SQL query
mysqli::refresh	mysqli_refresh	N/A	Flushes tables or caches, or resets the replication server information
mysqli::rollback	mysqli_rollback	N/A	Rolls back current transaction
mysqli::select_db	mysqli_select_db	N/A	Selects the default database for database queries
mysqli::set_charset	mysqli_set_charset	N/A	Sets the default client character set
mysqli::set_local_infile_default	mysqli_set_local_infile_default	N/A	Unsets user defined handler for load local infile command

mysqli Class			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>mysqli::set_local_infile_handler</code>	<code>mysqli_set_local_infile_handler</code>	N/A	Set callback function for LOAD DATA LOCAL INFILE command
<code>mysqli::ssl_set</code>	<code>mysqli_ssl_set</code>	N/A	Used for establishing secure connections using SSL
<code>mysqli::stat</code>	<code>mysqli_stat</code>	N/A	Gets the current system status
<code>mysqli::stmt_init</code>	<code>mysqli_stmt_init</code>	N/A	Initializes a statement and returns an object for use with <code>mysqli_stmt_prepare</code>
<code>mysqli::store_result</code>	<code>mysqli_store_result</code>	N/A	Transfers a result set from the last query
<code>mysqli::thread_id</code>	<code>mysqli_thread_id</code>	N/A	Returns the thread ID for the current connection
<code>mysqli::thread_safe</code>	<code>mysqli_thread_safe</code>	N/A	Returns whether thread safety is given or not
<code>mysqli::use_result</code>	<code>mysqli_use_result</code>	N/A	Initiate a result set retrieval

Table 6.6 Summary of `mysqli_stmt` methods

MySQL_STMT			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Properties</i>			
<code>\$mysqli_stmt::affected_rows</code>	<code>mysqli_stmt_affected_rows</code>	N/A	Returns the total number of rows changed, deleted, or inserted by the last executed statement
<code>\$mysqli_stmt::errno</code>	<code>mysqli_stmt_errno</code>	N/A	Returns the error code for the most recent statement call
<code>\$mysqli_stmt::error</code>	<code>mysqli_stmt_error</code>	N/A	Returns a string description for last statement error
<code>\$mysqli_stmt::field_count</code>	<code>mysqli_stmt_field_count</code>	N/A	Returns the number of field in the given statement - not documented
<code>\$mysqli_stmt::insert_id</code>	<code>mysqli_stmt_insert_id</code>	N/A	Get the ID generated from the previous INSERT operation
<code>\$mysqli_stmt::num_rows</code>	<code>mysqli_stmt_num_rows</code>	N/A	Return the number of rows in statements result set

MySQL_STMT			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>\$mysqli_stmt::param_count</code>	<code>mysqli_stmt_param_count</code>	<code>mysqli_param_count</code>	Returns the number of parameter for the given statement
<code>\$mysqli_stmt::sqlstate</code>	<code>mysqli_stmt_sqlstate</code>	N/A	Returns SQLSTATE error from previous statement operation
<i>Methods</i>			
<code>mysqli_stmt::attr_get</code>	<code>mysqli_stmt_attr_get</code>	N/A	Used to get the current value of a statement attribute
<code>mysqli_stmt::attr_set</code>	<code>mysqli_stmt_attr_set</code>	N/A	Used to modify the behavior of a prepared statement
<code>mysqli_stmt::bind_param</code>	<code>mysqli_stmt_bind_param</code>	<code>mysqli_bind_param</code>	Binds variables to a prepared statement as parameters
<code>mysqli_stmt::bind_result</code>	<code>mysqli_stmt_bind_result</code>	<code>mysqli_bind_result</code>	Binds variables to a prepared statement for result storage
<code>mysqli_stmt::close</code>	<code>mysqli_stmt_close</code>	N/A	Closes a prepared statement
<code>mysqli_stmt::data_seek</code>	<code>mysqli_stmt_data_seek</code>	N/A	Seeks to an arbitrary row in statement result set
<code>mysqli_stmt::execute</code>	<code>mysqli_stmt_execute</code>	<code>mysqli_execute</code>	Executes a prepared Query
<code>mysqli_stmt::fetch</code>	<code>mysqli_stmt_fetch</code>	<code>mysqli_fetch</code>	Fetch results from a prepared statement into the bound variables
<code>mysqli_stmt::free_result</code>	<code>mysqli_stmt_free_result</code>	N/A	Frees stored result memory for the given statement handle
<code>mysqli_stmt::get_result</code>	<code>mysqli_stmt_get_result</code>	N/A	Gets a result set from a prepared statement. Available only with mysqlind .
<code>mysqli_stmt::get_warnings</code>	<code>mysqli_stmt_get_warnings</code>	N/A	NOT DOCUMENTED
<code>\$mysqli_stmt::more_results</code>	<code>mysqli_stmt_more_results</code>	N/A	Checks if there are more query results from a multiple query
<code>\$mysqli_stmt::next_result</code>	<code>mysqli_stmt_next_result</code>	N/A	Reads the next result from a multiple query
<code>mysqli_stmt::num_rows</code>	<code>mysqli_stmt_num_rows</code>	N/A	See also property \$mysqli_stmt::num_rows
<code>mysqli_stmt::prepare</code>	<code>mysqli_stmt_prepare</code>	N/A	Prepare an SQL statement for execution
<code>mysqli_stmt::reset</code>	<code>mysqli_stmt_reset</code>	N/A	Resets a prepared statement

MySQL_STMT			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>mysqli_stmt::result_metadata</code>	<code>mysqli_stmt_result_metadata</code>	<code>mysqli_stmt_get_metadata</code>	Returns result set metadata from a prepared statement
<code>mysqli_stmt::send_long_data</code>	<code>mysqli_stmt_send_long_data</code>	<code>mysqli_stmt_send_long_data</code>	Sends data in blocks
<code>mysqli_stmt::store_result</code>	<code>mysqli_stmt_store_result</code>	N/A	Transfers a result set from a prepared statement

Table 6.7 Summary of `mysqli_result` methods

mysqli_result			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Properties</i>			
<code>\$mysqli_result::current_field</code>	<code>mysqli_field_tell</code>	N/A	Get current field offset of a result pointer
<code>\$mysqli_result::field_count</code>	<code>mysqli_num_fields</code>	N/A	Get the number of fields in a result
<code>\$mysqli_result::lengths</code>	<code>mysqli_fetch_lengths</code>	N/A	Returns the lengths of the columns of the current row in the result set
<code>\$mysqli_result::num_rows</code>	<code>mysqli_num_rows</code>	N/A	Gets the number of rows in a result
<i>Methods</i>			
<code>mysqli_result::data_seek</code>	<code>mysqli_data_seek</code>	N/A	Adjusts the result pointer to an arbitrary row in the result
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_all</code>	N/A	Fetches all result rows and returns the result set as an associative array, a numeric array, or both. Available only with <code>mysqlind</code> .
<code>mysqli_result::fetch_array</code>	<code>mysqli_fetch_array</code>	N/A	Fetch a result row as an associative, a numeric array, or both
<code>mysqli_result::fetch_assoc</code>	<code>mysqli_fetch_assoc</code>	N/A	Fetch a result row as an associative array
<code>mysqli_result::fetch_field</code>	<code>mysqli_fetch_field</code>	N/A	Fetch meta-data for a single field
<code>mysqli_result::fetch_field_direct</code>	<code>mysqli_fetch_field_direct</code>	N/A	Fetch meta-data for a single field
<code>mysqli_result::fetch_fields</code>	<code>mysqli_fetch_fields</code>	N/A	Returns the next field in the result set
<code>mysqli_result::fetch_object</code>	<code>mysqli_fetch_object</code>	N/A	Returns an array of objects representing the fields in a result set
<code>mysqli_result::fetch_row</code>	<code>mysqli_fetch_row</code>	N/A	Returns the current row of a result set as an object

mysqli_result			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<code>mysqli_result::fetch</code>	<code>mysqli_fetch_row</code>	N/A	Get a result row as an enumerated array
<code>mysqli_result::field_seek</code>	<code>mysqli_field_seek</code>	N/A	Set result pointer to a specified field offset
<code>mysqli_result::free</code> <code>mysqli_result::close</code> , <code>mysqli_result::free_result</code>	<code>mysqli_free_result</code>	N/A	Frees the memory associated with a result

Table 6.8 Summary of `mysqli_driver` methods

MySQL_Driver			
OOP Interface	Procedural Interface	Alias (Do not use)	Description
<i>Properties</i>			
N/A			
<i>Methods</i>			
<code>mysqli_driver::embedded_server_end</code>	<code>mysql_embedded_server_end</code>	N/A	NOT DOCUMENTED
<code>mysqli_driver::embedded_server_start</code>	<code>mysql_embedded_server_start</code>	N/A	NOT DOCUMENTED

Note

Alias functions are provided for backward compatibility purposes only. Do not use them in new projects.

6.3.8 Examples

Copyright 1997-2014 the PHP Documentation Group.

6.3.8.1 MySQLi extension basic examples

Copyright 1997-2014 the PHP Documentation Group.

This example shows how to connect, execute a query, use basic error handling, print resulting rows, and disconnect from a MySQL database.

This example uses the freely available Sakila database that can be downloaded from dev.mysql.com, as described here. To get this example to work, (a) install sakila and (b) modify the connection variables (host, your_user, your_pass).

Example 6.39 MySQLi extension overview example

```
<?php
// Let's pass in a $_GET variable to our example, in this case
// it's aid for actor_id in our Sakila database. Let's make it
// default to 1, and cast it to an integer as to avoid SQL injection
// and/or related security problems. Handling all of this goes beyond
// the scope of this simple example. Example:
// http://example.org/script.php?aid=42
if (isset($_GET['aid']) && is_numeric($_GET['aid'])) {
    $aid = (int) $_GET['aid'];
} else {
    $aid = 1;
}
// Connecting to and selecting a MySQL database named sakila
// Hostname: 127.0.0.1, username: your_user, password: your_pass, db: sakila
$mysqli = new mysqli('127.0.0.1', 'your_user', 'your_pass', 'sakila');
// Oh no! A connect_errno exists so the connection attempt failed!
```

```
if ($mysqli->connect_errno) {
    // The connection failed. What do you want to do?
    // You could contact yourself (email?), log the error, show a nice page, etc.
    // You do not want to reveal sensitive information
    // Let's try this:
    echo "Sorry, this website is experiencing problems.";
    // Something you should not do on a public site, but this example will show you
    // anyways, is print out MySQL error related information -- you might log this
    echo "Error: Failed to make a MySQL connection, here is why: \n";
    echo "Errno: " . $mysqli->connect_errno . "\n";
    echo "Error: " . $mysqli->connect_error . "\n";

    // You might want to show them something nice, but we will simply exit
    exit;
}
// Perform an SQL query
$sql = "SELECT actor_id, first_name, last_name FROM actor WHERE actor_id = $aid";
if (!$result = $mysqli->query($sql)) {
    // Oh no! The query failed.
    echo "Sorry, the website is experiencing problems.";
    // Again, do not do this on a public site, but we'll show you how
    // to get the error information
    echo "Error: Our query failed to execute and here is why: \n";
    echo "Query: " . $sql . "\n";
    echo "Errno: " . $mysqli->errno . "\n";
    echo "Error: " . $mysqli->error . "\n";
    exit;
}
// Phew, we made it. We know our MySQL connection and query
// succeeded, but do we have a result?
if ($result->num_rows === 0) {
    // Oh, no rows! Sometimes that's expected and okay, sometimes
    // it is not. You decide. In this case, maybe actor_id was too
    // large?
    echo "We could not find a match for ID $aid, sorry about that. Please try again.";
    exit;
}
// Now, we know only one result will exist in this example so let's
// fetch it into an associated array where the array's keys are the
// table's column names
$actor = $result->fetch_assoc();
echo "Sometimes I see " . $actor['first_name'] . " " . $actor['last_name'] . " on TV.";
// Now, let's fetch five random actors and output their names to a list.
// We'll add less error handling here as you can do that on your own now
$sql = "SELECT actor_id, first_name, last_name FROM actor ORDER BY rand() LIMIT 5";
if (!$result = $mysqli->query($sql)) {
    echo "Sorry, the website is experiencing problems.";
    exit;
}
// Print our 5 random actors in a list, and link to each actor
echo "<ul>\n";
while ($actor = $result->fetch_assoc()) {
    echo "<li><a href='" . $_SERVER['SCRIPT_FILENAME'] . "?aid=" . $actor['actor_id'] . "'>\n";
    echo $actor['first_name'] . ' ' . $actor['last_name'];
    echo "</a></li>\n";
}
echo "</ul>\n";
// The script will automatically free the result and close the MySQL
// connection when it exits, but let's just do it anyways
$result->free();
$mysqli->close();
?>
```

6.3.9 The mysqli class

Copyright 1997-2014 the PHP Documentation Group.

Represents a connection between PHP and a MySQL database.

```
mysqli {
    mysqli
        Properties

    int
        mysqli->affected_rows ;

    string
        mysqli->client_info ;

    int
        mysqli->client_version ;

    int
        mysqli->connect_errno ;

    string
        mysqli->connect_error ;

    int
        mysqli->errno ;

    array
        mysqli->error_list ;

    string
        mysqli->error ;

    int
        mysqli->field_count ;

    int
        mysqli->client_version ;

    string
        mysqli->host_info ;

    string
        mysqli->protocol_version ;

    string
        mysqli->server_info ;

    int
        mysqli->server_version ;

    string
        mysqli->info ;

    mixed
        mysqli->insert_id ;

    string
        mysqli->sqlstate ;

    int
        mysqli->thread_id ;

    int
        mysqli->warning_count ;

    Methods

    mysqli::__construct(
        string host
            = =ini_get("mysqli.default_host"),
        string username
            = =ini_get("mysqli.default_user"),
        string passwd
            = =ini_get("mysqli.default_pw"),
```

```
    string dbname
        = "",
    int port
        = ini_get("mysqli.default_port"),
    string socket
        = ini_get("mysqli.default_socket"));

bool mysqli::autocommit(
    bool mode);

bool mysqli::change_user(
    string user,
    string password,
    string database);

string mysqli::character_set_name();

bool mysqli::close();

bool mysqli::commit(
    int flags,
    string name);

bool mysqli::debug(
    string message);

bool mysqli::dump_debug_info();

object mysqli::get_charset();

string mysqli::get_client_info();

bool mysqli::get_connection_stats();

mysqli_warning mysqli::get_warnings();

mysqli mysqli::init();

bool mysqli::kill(
    int processid);

bool mysqli::more_results();

bool mysqli::multi_query(
    string query);

bool mysqli::next_result();

bool mysqli::options(
    int option,
    mixed value);

bool mysqli::ping();

public static int mysqli::poll(
    array read,
    array error,
    array reject,
    int sec,
    int usec);

mysqli_stmt mysqli::prepare(
    string query);

mixed mysqli::query(
    string query,
    int resultmode
        = MYSQLI_STORE_RESULT);

bool mysqli::real_connect(
```

```
    string host,
    string username,
    string passwd,
    string dbname,
    int port,
    string socket,
    int flags);

string mysqli::escape_string(
    string escapestr);

bool mysqli::real_query(
    string query);

public mysqli_result mysqli::reap_async_query();

public bool mysqli::refresh(
    int options);

bool mysqli::rollback(
    int flags,
    string name);

int mysqli::rpl_query_type(
    string query);

bool mysqli::select_db(
    string dbname);

bool mysqli::send_query(
    string query);

bool mysqli::set_charset(
    string charset);

bool mysqli::set_local_infile_handler(
    mysqli link,
    callable read_func);

bool mysqli::ssl_set(
    string key,
    string cert,
    string ca,
    string capath,
    string cipher);

string mysqli::stat();

mysqli_stmt mysqli::stmt_init();

mysqli_result mysqli::store_result(
    int option);

mysqli_result mysqli::use_result();
}
```

6.3.9.1 `mysqli::$affected_rows`, `mysqli_affected_rows`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::$affected_rows`

`mysqli_affected_rows`

Gets the number of affected rows in a previous MySQL operation

Description

Object oriented style

```
int  
mysqli->affected_rows ;
```

Procedural style

```
int mysqli_affected_rows(  
    mysqli link);
```

Returns the number of rows affected by the last [INSERT](#), [UPDATE](#), [REPLACE](#) or [DELETE](#) query.

For [SELECT](#) statements [mysqli_affected_rows](#) works like [mysqli_num_rows](#).

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an [UPDATE](#) statement, no rows matched the [WHERE](#) clause in the query or that no query has yet been executed. -1 indicates that the query returned an error.

Note

If the number of affected rows is greater than the maximum integer value([PHP_INT_MAX](#)), the number of affected rows will be returned as a string.

Examples

Example 6.40 [\\$mysqli->affected_rows](#) example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* Insert rows */  
$mysqli->query("CREATE TABLE Language SELECT * from CountryLanguage");  
printf("Affected rows (INSERT): %d\n", $mysqli->affected_rows);  
$mysqli->query("ALTER TABLE Language ADD Status int default 0");  
/* update rows */  
$mysqli->query("UPDATE Language SET Status=1 WHERE Percentage > 50");  
printf("Affected rows (UPDATE): %d\n", $mysqli->affected_rows);  
/* delete rows */  
$mysqli->query("DELETE FROM Language WHERE Percentage < 50");  
printf("Affected rows (DELETE): %d\n", $mysqli->affected_rows);  
/* select all rows */  
$result = $mysqli->query("SELECT CountryCode FROM Language");  
printf("Affected rows (SELECT): %d\n", $mysqli->affected_rows);  
$result->close();  
/* Delete table Language */  
$mysqli->query("DROP TABLE Language");  
/* close connection */  
$mysqli->close();  
?>
```


Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
if (!$link) {
    printf("Can't connect to localhost. Error: %s\n", mysqli_connect_error());
    exit();
}
/* Insert rows */
mysqli_query($link, "CREATE TABLE Language SELECT * from CountryLanguage");
printf("Affected rows (INSERT): %d\n", mysqli_affected_rows($link));
mysqli_query($link, "ALTER TABLE Language ADD Status int default 0");
/* update rows */
mysqli_query($link, "UPDATE Language SET Status=1 WHERE Percentage > 50");
printf("Affected rows (UPDATE): %d\n", mysqli_affected_rows($link));
/* delete rows */
mysqli_query($link, "DELETE FROM Language WHERE Percentage < 50");
printf("Affected rows (DELETE): %d\n", mysqli_affected_rows($link));
/* select all rows */
$result = mysqli_query($link, "SELECT CountryCode FROM Language");
printf("Affected rows (SELECT): %d\n", mysqli_affected_rows($link));
mysqli_free_result($result);
/* Delete table Language */
mysqli_query($link, "DROP TABLE Language");
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Affected rows (INSERT): 984
Affected rows (UPDATE): 168
Affected rows (DELETE): 815
Affected rows (SELECT): 169
```

See Also

[mysqli_num_rows](#)
[mysqli_info](#)

6.3.9.2 `mysqli::autocommit`, `mysqli_autocommit`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::autocommit`

`mysqli_autocommit`

Turns on or off auto-committing database modifications

Description

Object oriented style

```
bool mysqli::autocommit(
    bool mode);
```

Procedural style

```
bool mysqli_autocommit(
```

```
mysqli link,  
bool mode);
```

Turns on or off auto-commit mode on queries for the database connection.

To determine the current state of autocommit use the SQL command `SELECT @@autocommit`.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by <code>mysqli_connect</code> or <code>mysqli_init</code>
<i>mode</i>	Whether to turn on auto-commit or not.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Notes

Note

This function doesn't work with non transactional table types (like MyISAM or ISAM).

Examples

Example 6.41 `mysqli::autocommit` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* turn autocommit on */  
$mysqli->autocommit(TRUE);  
if ($result = $mysqli->query("SELECT @@autocommit")) {  
    $row = $result->fetch_row();  
    printf("Autocommit is %s\n", $row[0]);  
    $result->free();  
}  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password", "world");  
if (!$link) {  
    printf("Can't connect to localhost. Error: %s\n", mysqli_connect_error());  
    exit();  
}  
/* turn autocommit on */  
mysqli_autocommit($link, TRUE);  
if ($result = mysqli_query($link, "SELECT @@autocommit")) {  
    $row = mysqli_fetch_row($result);
```

```
        printf("Autocommit is %s\n", $row[0]);
        mysqli_free_result($result);
    }
    /* close connection */
    mysqli_close($link);
?>
```

The above examples will output:

```
Autocommit is 1
```

See Also

[mysqli_begin_transaction](#)
[mysqli_commit](#)
[mysqli_rollback](#)

6.3.9.3 `mysqli::begin_transaction`, `mysqli_begin_transaction`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::begin_transaction`

[mysqli_begin_transaction](#)

Starts a transaction

Description

Object oriented style (method):

```
public bool mysqli::begin_transaction(
    int flags,
    string name);
```

Procedural style:

```
bool mysqli_begin_transaction(
    mysqli link,
    int flags,
    string name);
```

Begins a transaction. Requires MySQL 5.6 and above, and the InnoDB engine (it is enabled by default). For additional details about how MySQL transactions work, see <http://dev.mysql.com/doc/mysql/en/commit.html>.

Parameters

link

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

flags

Valid flags are:

- `MYSQLI_TRANS_START_READ_ONLY`: Start the transaction as "START TRANSACTION READ ONLY".
- `MYSQLI_TRANS_START_READ_WRITE`: Start the transaction as "START TRANSACTION READ WRITE".

- [MYSQLI_TRANS_START_WITH_CONSISTENT_SNAPSHOT](#): Start the transaction as "START TRANSACTION WITH CONSISTENT SNAPSHOT".

name Savepoint name for the transaction.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 6.42 [\\$mysqli->begin_transaction](#) example

Object oriented style

```
<?php
$mysqli = new mysqli("127.0.0.1", "my_user", "my_password", "sakila");
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
$mysqli->begin_transaction(MYSQLI_TRANS_START_READ_ONLY);
$mysqli->query("SELECT first_name, last_name FROM actor");
$mysqli->commit();
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("127.0.0.1", "my_user", "my_password", "sakila");
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_begin_transaction($link, MYSQLI_TRANS_START_READ_ONLY);
mysqli_query($link, "SELECT first_name, last_name FROM actor LIMIT 1");
mysqli_commit($link);
mysqli_close($link);
?>
```

See Also

[mysqli_autocommit](#)
[mysqli_commit](#)
[mysqli_rollback](#)

6.3.9.4 [mysqli::change_user](#), [mysqli_change_user](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::change_user](#)

[mysqli_change_user](#)

Changes the user of the specified database connection

Description

Object oriented style

```
bool mysqli::change_user(  
    string user,  
    string password,  
    string database);
```

Procedural style

```
bool mysqli_change_user(  
    mysqli link,  
    string user,  
    string password,  
    string database);
```

Changes the user of the specified database connection and sets the current database.

In order to successfully change users a valid *username* and *password* parameters must be provided and that user must have sufficient permissions to access the desired database. If for any reason authorization fails, the current user authentication will remain.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
<i>user</i>	The MySQL user name.
<i>password</i>	The MySQL password.
<i>database</i>	The database to change to. If desired, the NULL value may be passed resulting in only changing the user and not selecting a database. To select a database in this case use the mysqli_select_db function.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Notes

Note

Using this command will always cause the current database connection to behave as if was a completely new database connection, regardless of if the operation was completed successfully. This reset includes performing a rollback on any active transactions, closing all temporary tables, and unlocking all locked tables.

Examples

Example 6.43 [mysqli::change_user](#) example

Object oriented style

```
<?php  
/* connect database test */  
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");  
/* check connection */  
if (mysqli_connect_errno()) {
```

```
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Set Variable a */
$mysqli->query("SET @a:=1");
/* reset all and select a new database */
$mysqli->change_user("my_user", "my_password", "world");
if ($result = $mysqli->query("SELECT DATABASE())) {
    $row = $result->fetch_row();
    printf("Default database: %s\n", $row[0]);
    $result->close();
}
if ($result = $mysqli->query("SELECT @a")) {
    $row = $result->fetch_row();
    if ($row[0] === NULL) {
        printf("Value of variable a is NULL\n");
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
/* connect database test */
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Set Variable a */
mysqli_query($link, "SET @a:=1");
/* reset all and select a new database */
mysqli_change_user($link, "my_user", "my_password", "world");
if ($result = mysqli_query($link, "SELECT DATABASE())) {
    $row = mysqli_fetch_row($result);
    printf("Default database: %s\n", $row[0]);
    mysqli_free_result($result);
}
if ($result = mysqli_query($link, "SELECT @a")) {
    $row = mysqli_fetch_row($result);
    if ($row[0] === NULL) {
        printf("Value of variable a is NULL\n");
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Default database: world
Value of variable a is NULL
```

See Also

[mysqli_connect](#)

`mysqli_select_db`

6.3.9.5 `mysqli::character_set_name`, `mysqli_character_set_name`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::character_set_name`

`mysqli_character_set_name`

Returns the default character set for the database connection

Description

Object oriented style

```
string mysqli::character_set_name();
```

Procedural style

```
string mysqli_character_set_name(  
    mysqli link);
```

Returns the current character set for the database connection.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

The default character set for the current connection

Examples

Example 6.44 `mysqli::character_set_name` example

Object oriented style

```
<?php  
/* Open a connection */  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* Print current character set */  
$charset = $mysqli->character_set_name();  
printf ("Current character set is %s\n", $charset);  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
/* Open a connection */  
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Print current character set */
$charset = mysqli_character_set_name($link);
printf("Current character set is %s\n", $charset);
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Current character set is latin1_swedish_ci
```

See Also

[mysqli_set_charset](#)
[mysqli_client_encoding](#)
[mysqli_real_escape_string](#)

6.3.9.6 [mysqli::\\$client_info](#), [mysqli_get_client_info](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$client_info](#)
[mysqli_get_client_info](#)

Get MySQL client info

Description

Object oriented style

```
string
mysqli->client_info ;
```

Procedural style

```
string mysqli_get_client_info(
    mysqli link);
```

Returns a string that represents the MySQL client library version.

Return Values

A string that represents the MySQL client library version

Examples

Example 6.45 [mysqli_get_client_info](#)

```
<?php
/* We don't need a connection to determine
   the version of mysql client library */
printf("Client library version: %s\n", mysqli_get_client_info());
```



```
?>
```

See Also

```
mysqli_get_client_version  
mysqli_get_server_info  
mysqli_get_server_version
```

6.3.9.7 mysqli::\$client_version, mysqli_get_client_version

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::$client_version`

```
mysqli_get_client_version
```

Returns the MySQL client version as a string

Description

Object oriented style

```
int  
mysqli->client_version ;
```

Procedural style

```
int mysqli_get_client_version(  
    mysqli link);
```

Returns client version number as an integer.

Return Values

A number that represents the MySQL client library version in format: `main_version*10000 + minor_version *100 + sub_version`. For example, 4.1.0 is returned as 40100.

This is useful to quickly determine the version of the client library to know if some capability exists.

Examples**Example 6.46 mysqli_get_client_version**

```
<?php  
/* We don't need a connection to determine  
   the version of mysql client library */  
printf("Client library version: %d\n", mysqli_get_client_version());  
?>
```

See Also

```
mysqli_get_client_info  
mysqli_get_server_info  
mysqli_get_server_version
```

6.3.9.8 mysqli::close, mysqli_close

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::close`

`mysqli_close`

Closes a previously opened database connection

Description

Object oriented style

```
bool mysqli::close();
```

Procedural style

```
bool mysqli_close(  
    mysqli link);
```

Closes a previously opened database connection.

Open non-persistent MySQL connections and result sets are automatically destroyed when a PHP script finishes its execution. So, while explicitly closing open connections and freeing result sets is optional, doing so is recommended. This will immediately return resources to PHP and MySQL, which can improve performance. For related information, see [freeing resources](#)

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

See `mysqli_connect`.

Notes

Note

`mysqli_close` will not close persistent connections. For additional details, see the manual page on [persistent connections](#).

See Also

`mysqli::__construct`
`mysqli_init`
`mysqli_real_connect`
`mysqli_free_result`

6.3.9.9 `mysqli::commit`, `mysqli_commit`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::commit`

`mysqli_commit`

Commits the current transaction

Description

Object oriented style

```
bool mysqli::commit(
    int flags,
    string name);
```

Procedural style

```
bool mysqli_commit(
    mysqli link,
    int flags,
    string name);
```

Commits the current transaction for the database connection.

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

flags A bitmask of `MYSQLI_TRANS_COR_*` constants.

name If provided then `COMMIT/*name*/` is executed.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Changelog

Version	Description
5.5.0	Added flags and name parameters.

Examples

Example 6.47 `mysqli::commit` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE Language LIKE CountryLanguage");
/* set autocommit to off */
$mysqli->autocommit(FALSE);
/* Insert some values */
$mysqli->query("INSERT INTO Language VALUES ('DEU', 'Bavarian', 'F', 11.2)");
$mysqli->query("INSERT INTO Language VALUES ('DEU', 'Swabian', 'F', 9.4)");
/* commit transaction */
if (!$mysqli->commit()) {
    print("Transaction commit failed\n");
    exit();
}
/* drop table */
$mysqli->query("DROP TABLE Language");
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* set autocommit to off */
mysqli_autocommit($link, FALSE);
mysqli_query($link, "CREATE TABLE Language LIKE CountryLanguage");
/* Insert some values */
mysqli_query($link, "INSERT INTO Language VALUES ('DEU', 'Bavarian', 'F', 11.2)");
mysqli_query($link, "INSERT INTO Language VALUES ('DEU', 'Swabian', 'F', 9.4)");
/* commit transaction */
if (!mysqli_commit($link)) {
    print("Transaction commit failed\n");
    exit();
}
/* close connection */
mysqli_close($link);
?>
```

See Also

[mysqli_autocommit](#)
[mysqli_begin_transaction](#)
[mysqli_rollback](#)
[mysqli_savepoint](#)

6.3.9.10 [mysqli::\\$connect_errno](#), [mysqli_connect_errno](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$connect_errno](#)
[mysqli_connect_errno](#)

Returns the error code from last connect call

Description

Object oriented style

```
int
mysqli->connect_errno ;
```

Procedural style

```
int mysqli_connect_errno();
```

Returns the last error code number from the last call to [mysqli_connect](#).

Note

Client error message numbers are listed in the MySQL [errmsg.h](#) header file, server error message numbers are listed in [mysqld_error.h](#). In the MySQL source distribution you can find a complete list of error messages and error numbers in the file [Docs/mysqld_error.txt](#).

Return Values

An error code value for the last call to [mysqli_connect](#), if it failed. zero means no error occurred.

Examples

Example 6.48 [\\$mysqli->connect_errno](#) example

Object oriented style

```
<?php
$mysqli = @new mysqli('localhost', 'fake_user', 'my_password', 'my_db');
if ($mysqli->connect_errno) {
    die('Connect Error: ' . $mysqli->connect_errno);
}
?>
```

Procedural style

```
<?php
$link = @mysqli_connect('localhost', 'fake_user', 'my_password', 'my_db');
if (!$link) {
    die('Connect Error: ' . mysqli_connect_errno());
}
?>
```

The above examples will output:

```
Connect Error: 1045
```

See Also

[mysqli_connect](#)
[mysqli_connect_error](#)
[mysqli_errno](#)
[mysqli_error](#)
[mysqli_sqlstate](#)

6.3.9.11 [mysqli::\\$connect_error](#), [mysqli_connect_error](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$connect_error](#)

[mysqli_connect_error](#)

Returns a string description of the last connect error

Description

Object oriented style

```
string
```

```
mysqli->connect_error ;
```

Procedural style

```
string mysqli_connect_error();
```

Returns the last error message string from the last call to [mysqli_connect](#).

Return Values

A string that describes the error. [NULL](#) is returned if no error occurred.

Examples

Example 6.49 [\\$mysqli->connect_error](#) example

Object oriented style

```
<?php
$mysqli = @new mysqli('localhost', 'fake_user', 'my_password', 'my_db');
// Works as of PHP 5.2.9 and 5.3.0.
if ($mysqli->connect_error) {
    die('Connect Error: ' . $mysqli->connect_error);
}
?>
```

Procedural style

```
<?php
$link = @mysqli_connect('localhost', 'fake_user', 'my_password', 'my_db');
if (!$link) {
    die('Connect Error: ' . mysqli_connect_error());
}
?>
```

The above examples will output:

```
Connect Error: Access denied for user 'fake_user'@'localhost' (using password: YES)
```

Notes

Warning

The `mysqli->connect_error` property only works properly as of PHP versions 5.2.9 and 5.3.0. Use the [mysqli_connect_error](#) function if compatibility with earlier PHP versions is required.

See Also

[mysqli_connect](#)
[mysqli_connect_errno](#)
[mysqli_errno](#)
[mysqli_error](#)
[mysqli_sqlstate](#)

6.3.9.12 `mysqli::__construct`, `mysqli_connect`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::__construct`

`mysqli_connect`

Open a new connection to the MySQL server

Description

Object oriented style

```
mysqli::__construct(  
    string host  
        = =ini_get("mysqli.default_host"),  
    string username  
        = =ini_get("mysqli.default_user"),  
    string passwd  
        = =ini_get("mysqli.default_pw"),  
    string dbname  
        = "",  
    int port  
        = =ini_get("mysqli.default_port"),  
    string socket  
        = =ini_get("mysqli.default_socket"));
```

Procedural style

```
mysqli mysqli_connect(  
    string host  
        = =ini_get("mysqli.default_host"),  
    string username  
        = =ini_get("mysqli.default_user"),  
    string passwd  
        = =ini_get("mysqli.default_pw"),  
    string dbname  
        = "",  
    int port  
        = =ini_get("mysqli.default_port"),  
    string socket  
        = =ini_get("mysqli.default_socket"));
```

Opens a connection to the MySQL Server running on.

Parameters

host

Can be either a host name or an IP address. Passing the `NULL` value or the string "localhost" to this parameter, the local host is assumed. When possible, pipes will be used instead of the TCP/IP protocol.

Prepending host by `p:` opens a persistent connection. `mysqli_change_user` is automatically called on connections opened from the connection pool.

username

The MySQL user name.

passwd

If not provided or `NULL`, the MySQL server will attempt to authenticate the user against those user records which have no password only. This allows one username to be used with different permissions (depending on if a password as provided or not).

dbname

If provided will specify the default database to be used when performing queries.

port

Specifies the port number to attempt to connect to the MySQL server.

socket

Specifies the socket or named pipe that should be used.

Note

Specifying the *socket* parameter will not explicitly determine the type of connection to be used when connecting to the MySQL server. How the connection is made to the MySQL database is determined by the *host* parameter.

Return Values

Returns an object which represents the connection to a MySQL Server.

Changelog

Version	Description
5.3.0	Added the ability of persistent connections.

Examples**Example 6.50 `mysqli::__construct` example**

Object oriented style

```
<?php
$mysqli = new mysqli('localhost', 'my_user', 'my_password', 'my_db');
/*
 * This is the "official" OO way to do it,
 * BUT $connect_error was broken until PHP 5.2.9 and 5.3.0.
 */
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
        . $mysqli->connect_error);
}
/*
 * Use this instead of $connect_error if you need to ensure
 * compatibility with PHP versions prior to 5.2.9 and 5.3.0.
 */
if (mysqli_connect_error()) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}
echo 'Success... ' . $mysqli->host_info . "\n";
$mysqli->close();
?>
```

Object oriented style when extending mysqli class

```
<?php
class foo_mysqli extends mysqli {
    public function __construct($host, $user, $pass, $db) {
        parent::__construct($host, $user, $pass, $db);
        if (mysqli_connect_error()) {
            die('Connect Error (' . mysqli_connect_errno() . ') '
                . mysqli_connect_error());
        }
    }
}
```



```
        . mysqli_connect_error());
    }
}
$db = new mysqli('localhost', 'my_user', 'my_password', 'my_db');
echo 'Success... ' . $db->host_info . "\n";
$db->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'my_db');
if (!$link) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}
echo 'Success... ' . mysqli_get_host_info($link) . "\n";
mysqli_close($link);
?>
```

The above examples will output:

```
Success... MySQL host info: localhost via TCP/IP
```

Notes

Note

MySQLnd always assumes the server default charset. This charset is sent during connection hand-shake/authentication, which MySQLnd will use.

Libmysqlclient uses the default charset set in the [my.cnf](#) or by an explicit call to [mysqli_options](#) prior to calling [mysqli_real_connect](#), but after [mysqli_init](#).

Note

OO syntax only: If a connection fails an object is still returned. To check if the connection failed then use either the [mysqli_connect_error](#) function or the [mysqli->connect_error](#) property as in the preceding examples.

Note

If it is necessary to set options, such as the connection timeout, [mysqli_real_connect](#) must be used instead.

Note

Calling the constructor with no parameters is the same as calling [mysqli_init](#).

Note

Error "Can't create TCP/IP socket (10106)" usually means that the [variables_order](#) configure directive doesn't contain character [E](#). On Windows, if

the environment is not copied the `SYSTEMROOT` environment variable won't be available and PHP will have problems loading Winsock.

See Also

`mysqli_real_connect`
`mysqli_options`
`mysqli_connect_errno`
`mysqli_connect_error`
`mysqli_close`

6.3.9.13 `mysqli::debug`, `mysqli_debug`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::debug`

`mysqli_debug`

Performs debugging operations

Description

Object oriented style

```
bool mysqli::debug(  
    string message);
```

Procedural style

```
bool mysqli_debug(  
    string message);
```

Performs debugging operations using the Fred Fish debugging library.

Parameters

message A string representing the debugging operation to perform

Return Values

Returns `TRUE`.

Notes

Note

To use the `mysqli_debug` function you must compile the MySQL client library to support debugging.

Examples

Example 6.51 Generating a Trace File

```
<?php  
/* Create a trace file in '/tmp/client.trace' on the local (client) machine: */  
mysqli_debug("d:t:o,/tmp/client.trace");  
?>
```

See Also

```
mysqli_dump_debug_info  
mysqli_report
```

6.3.9.14 **mysqli::dump_debug_info, mysqli_dump_debug_info**

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::dump_debug_info`

`mysqli_dump_debug_info`

Dump debugging information into the log

Description

Object oriented style

```
bool mysqli::dump_debug_info();
```

Procedural style

```
bool mysqli_dump_debug_info(  
    mysqli link);
```

This function is designed to be executed by an user with the SUPER privilege and is used to dump debugging information into the log for the MySQL Server relating to the connection.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns `TRUE` on success or `FALSE` on failure.

See Also

`mysqli_debug`

6.3.9.15 **mysqli::\$errno, mysqli_errno**

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::$errno`

`mysqli_errno`

Returns the error code for the most recent function call

Description

Object oriented style

```
int  
mysqli->errno ;
```

Procedural style

```
int mysqli_errno(  
    mysqli link);
```

Returns the last error code for the most recent MySQLi function call that can succeed or fail.

Client error message numbers are listed in the MySQL [errmsg.h](#) header file, server error message numbers are listed in [mysqld_error.h](#). In the MySQL source distribution you can find a complete list of error messages and error numbers in the file [Docs/mysqld_error.txt](#).

Parameters

[link](#)

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

An error code value for the last call, if it failed. zero means no error occurred.

Examples

Example 6.52 [\\$mysqli->errno](#) example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
if (!$mysqli->query("SET a=1")) {
    printf("Errorcode: %d\n", $mysqli->errno);
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if (!mysqli_query($link, "SET a=1")) {
    printf("Errorcode: %d\n", mysqli_errno($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Errorcode: 1193
```

See Also

[mysqli_connect_errno](#)

```
mysqli_connect_error  
mysqli_error  
mysqli_sqlstate
```

6.3.9.16 `mysqli::$error_list, mysqli_error_list`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::$error_list`

`mysqli_error_list`

Returns a list of errors from the last command executed

Description

Object oriented style

```
array  
    mysqli->error_list ;
```

Procedural style

```
array mysqli_error_list(  
    mysqli link);
```

Returns a array of errors for the most recent MySQLi function call that can succeed or fail.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

A list of errors, each as an associative array containing the errno, error, and sqlstate.

Examples

Example 6.53 `$mysqli->error_list` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "nobody", "");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
if (!$mysqli->query("SET a=1")) {  
    print_r($mysqli->error_list);  
}  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if (!mysqli_query($link, "SET a=1")) {
    print_r(mysqli_error_list($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Array
(
    [0] => Array
        (
            [errno] => 1193
            [sqlstate] => HY000
            [error] => Unknown system variable 'a'
        )
)
```

See Also

[mysqli_connect_errno](#)
[mysqli_connect_error](#)
[mysqli_error](#)
[mysqli_sqlstate](#)

6.3.9.17 [mysqli::\\$error](#), [mysqli_error](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$error](#)

[mysqli_error](#)

Returns a string description of the last error

Description

Object oriented style

```
string
mysqli->error ;
```

Procedural style

```
string mysqli_error(
    mysqli link);
```

Returns the last error message for the most recent MySQLi function call that can succeed or fail.

Parameters

[link](#)

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

A string that describes the error. An empty string if no error occurred.

Examples

Example 6.54 `$mysqli->error` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
if (!$mysqli->query("SET a=1")) {
    printf("Errormessage: %s\n", $mysqli->error);
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if (!mysqli_query($link, "SET a=1")) {
    printf("Errormessage: %s\n", mysqli_error($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Errormessage: Unknown system variable 'a'
```

See Also

[mysqli_connect_errno](#)
[mysqli_connect_error](#)
[mysqli_errno](#)
[mysqli_sqlstate](#)

6.3.9.18 `mysqli::$field_count`, `mysqli_field_count`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$field_count](#)

mysqli_field_count

Returns the number of columns for the most recent query

Description

Object oriented style

```
int  
mysqli->field_count ;
```

Procedural style

```
int mysqli_field_count(  
    mysqli link);
```

Returns the number of columns for the most recent query on the connection represented by the [link](#) parameter. This function can be useful when using the [mysqli_store_result](#) function to determine if the query should have produced a non-empty result set or not without knowing the nature of the query.

Parameters

[link](#)

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

An integer representing the number of fields in a result set.

Examples

Example 6.55 `$mysqli->field_count` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");  
$mysqli->query( "DROP TABLE IF EXISTS friends");  
$mysqli->query( "CREATE TABLE friends (id int, name varchar(20))");  
$mysqli->query( "INSERT INTO friends VALUES (1,'Hartmut'), (2, 'Ulf')");  
$mysqli->real_query("SELECT * FROM friends");  
if ($mysqli->field_count) {  
    /* this was a select/show or describe query */  
    $result = $mysqli->store_result();  
    /* process resultset */  
    $row = $result->fetch_row();  
    /* free resultset */  
    $result->close();  
}  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password", "test");  
mysqli_query($link, "DROP TABLE IF EXISTS friends");
```



```
mysqli_query($link, "CREATE TABLE friends (id int, name varchar(20))");
mysqli_query($link, "INSERT INTO friends VALUES (1, 'Hartmut'), (2, 'Ulf')");
mysqli_real_query($link, "SELECT * FROM friends");
if (mysqli_field_count($link)) {
    /* this was a select/show or describe query */
    $result = mysqli_store_result($link);
    /* process resultset */
    $row = mysqli_fetch_row($result);
    /* free resultset */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

6.3.9.19 `mysqli::get_charset`, `mysqli_get_charset`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::get_charset`

`mysqli_get_charset`

Returns a character set object

Description

Object oriented style

```
object mysqli::get_charset();
```

Procedural style

```
object mysqli_get_charset(
    mysqli link);
```

Returns a character set object providing several properties of the current active character set.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

The function returns a character set object with the following properties:

<i>charset</i>	Character set name
<i>collation</i>	Collation name
<i>dir</i>	Directory the charset description was fetched from (?) or "" for built-in character sets
<i>min_length</i>	Minimum character length in bytes
<i>max_length</i>	Maximum character length in bytes
<i>number</i>	Internal character set number
<i>state</i>	Character set status (?)

Examples

Example 6.56 `mysqli::get_charset` example

Object oriented style

```
<?php
$db = mysqli_init();
$db->real_connect("localhost","root","","test");
var_dump($db->get_charset());
?>
```

Procedural style

```
<?php
$db = mysqli_init();
mysqli_real_connect($db, "localhost","root","","test");
var_dump(mysqli_get_charset($db));
?>
```

The above examples will output:

```
object(stdClass)#2 (7) {
  ["charset"]=>
  string(6) "latin1"
  ["collation"]=>
  string(17) "latin1_swedish_ci"
  ["dir"]=>
  string(0) ""
  ["min_length"]=>
  int(1)
  ["max_length"]=>
  int(1)
  ["number"]=>
  int(8)
  ["state"]=>
  int(801)
}
```

See Also

[mysqli_character_set_name](#)
[mysqli_set_charset](#)

6.3.9.20 `mysqli::get_client_info`, `mysqli_get_client_info`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::get_client_info](#)

[mysqli_get_client_info](#)

Get MySQL client info

Description

Object oriented style

```
string mysqli::get_client_info();
```

Procedural style

```
string mysqli_get_client_info(  
    mysqli link);
```

Returns a string that represents the MySQL client library version.

Return Values

A string that represents the MySQL client library version

Examples

Example 6.57 `mysqli_get_client_info`

```
<?php  
/* We don't need a connection to determine  
   the version of mysql client library */  
printf("Client library version: %s\n", mysqli_get_client_info());  
?>
```

See Also

[mysqli_get_client_version](#)
[mysqli_get_server_info](#)
[mysqli_get_server_version](#)

6.3.9.21 `mysqli_get_client_stats`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_get_client_stats](#)

Returns client per-process statistics

Description

```
array mysqli_get_client_stats();
```

Returns client per-process statistics. Available only with [mysqlnd](#).

Parameters

Return Values

Returns an array with client stats if success, [FALSE](#) otherwise.

Examples

Example 6.58 A `mysqli_get_client_stats` example

```
<?php  
$link = mysqli_connect();  
print_r(mysqli_get_client_stats());  
?>
```

The above example will output something similar to:

```
Array
(
    [bytes_sent] => 43
    [bytes_received] => 80
    [packets_sent] => 1
    [packets_received] => 2
    [protocol_overhead_in] => 8
    [protocol_overhead_out] => 4
    [bytes_received_ok_packet] => 11
    [bytes_received_eof_packet] => 0
    [bytes_received_rset_header_packet] => 0
    [bytes_received_rset_field_meta_packet] => 0
    [bytes_received_rset_row_packet] => 0
    [bytes_received_prepare_response_packet] => 0
    [bytes_received_change_user_packet] => 0
    [packets_sent_command] => 0
    [packets_received_ok] => 1
    [packets_received_eof] => 0
    [packets_received_rset_header] => 0
    [packets_received_rset_field_meta] => 0
    [packets_received_rset_row] => 0
    [packets_received_prepare_response] => 0
    [packets_received_change_user] => 0
    [result_set_queries] => 0
    [non_result_set_queries] => 0
    [no_index_used] => 0
    [bad_index_used] => 0
    [slow_queries] => 0
    [buffered_sets] => 0
    [unbuffered_sets] => 0
    [ps_buffered_sets] => 0
    [ps_unbuffered_sets] => 0
    [flushed_normal_sets] => 0
    [flushed_ps_sets] => 0
    [ps_prepared_never_executed] => 0
    [ps_prepared_once_executed] => 0
    [rows_fetched_from_server_normal] => 0
    [rows_fetched_from_server_ps] => 0
    [rows_buffered_from_client_normal] => 0
    [rows_buffered_from_client_ps] => 0
    [rows_fetched_from_client_normal_buffered] => 0
    [rows_fetched_from_client_normal_unbuffered] => 0
    [rows_fetched_from_client_ps_buffered] => 0
    [rows_fetched_from_client_ps_unbuffered] => 0
    [rows_fetched_from_client_ps_cursor] => 0
    [rows_skipped_normal] => 0
    [rows_skipped_ps] => 0
    [copy_on_write_saved] => 0
    [copy_on_write_performed] => 0
    [command_buffer_too_small] => 0
    [connect_success] => 1
    [connect_failure] => 0
    [connection_reused] => 0
    [reconnect] => 0
    [pconnect_success] => 0
    [active_connections] => 1
    [active_persistent_connections] => 0
    [explicit_close] => 0
    [implicit_close] => 0
    [disconnect_close] => 0
    [in_middle_of_command_close] => 0
    [explicit_free_result] => 0
    [implicit_free_result] => 0
    [explicit_stmt_close] => 0
    [implicit_stmt_close] => 0
    [mem_emalloc_count] => 0
)
```

```
[mem_eralloc_ammount] => 0
[mem_eralloc_count] => 0
[mem_eralloc_ammount] => 0
[mem_erealloc_count] => 0
[mem_erealloc_ammount] => 0
[mem_efree_count] => 0
[mem_malloc_count] => 0
[mem_malloc_ammount] => 0
[mem_calloc_count] => 0
[mem_calloc_ammount] => 0
[mem_realloc_count] => 0
[mem_realloc_ammount] => 0
[mem_free_count] => 0
[proto_text_fetched_null] => 0
[proto_text_fetched_bit] => 0
[proto_text_fetched_tinyint] => 0
[proto_text_fetched_short] => 0
[proto_text_fetched_int24] => 0
[proto_text_fetched_int] => 0
[proto_text_fetched_bigint] => 0
[proto_text_fetched_decimal] => 0
[proto_text_fetched_float] => 0
[proto_text_fetched_double] => 0
[proto_text_fetched_date] => 0
[proto_text_fetched_year] => 0
[proto_text_fetched_time] => 0
[proto_text_fetched_datetime] => 0
[proto_text_fetched_timestamp] => 0
[proto_text_fetched_string] => 0
[proto_text_fetched_blob] => 0
[proto_text_fetched_enum] => 0
[proto_text_fetched_set] => 0
[proto_text_fetched_geometry] => 0
[proto_text_fetched_other] => 0
[proto_binary_fetched_null] => 0
[proto_binary_fetched_bit] => 0
[proto_binary_fetched_tinyint] => 0
[proto_binary_fetched_short] => 0
[proto_binary_fetched_int24] => 0
[proto_binary_fetched_int] => 0
[proto_binary_fetched_bigint] => 0
[proto_binary_fetched_decimal] => 0
[proto_binary_fetched_float] => 0
[proto_binary_fetched_double] => 0
[proto_binary_fetched_date] => 0
[proto_binary_fetched_year] => 0
[proto_binary_fetched_time] => 0
[proto_binary_fetched_datetime] => 0
[proto_binary_fetched_timestamp] => 0
[proto_binary_fetched_string] => 0
[proto_binary_fetched_blob] => 0
[proto_binary_fetched_enum] => 0
[proto_binary_fetched_set] => 0
[proto_binary_fetched_geometry] => 0
[proto_binary_fetched_other] => 0
)
```

See Also

[Stats description](#)

6.3.9.22 `mysqli_get_client_version`, `mysqli::$client_version`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_get_client_version`

`mysqli::$client_version`

Returns the MySQL client version as an integer

Description

Object oriented style

```
int  
mysqli->client_version ;
```

Procedural style

```
int mysqli_get_client_version(  
    mysqli link);
```

Returns client version number as an integer.

Return Values

A number that represents the MySQL client library version in format: `main_version*10000 + minor_version *100 + sub_version`. For example, 4.1.0 is returned as 40100.

This is useful to quickly determine the version of the client library to know if some capability exists.

Examples

Example 6.59 mysqli_get_client_version

```
<?php  
/* We don't need a connection to determine  
   the version of mysql client library */  
printf("Client library version: %d\n", mysqli_get_client_version());  
?>
```

See Also

`mysqli_get_client_info`
`mysqli_get_server_info`
`mysqli_get_server_version`

6.3.9.23 `mysqli::get_connection_stats`, `mysqli_get_connection_stats`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::get_connection_stats`

`mysqli_get_connection_stats`

Returns statistics about the client connection

Description

Object oriented style

```
bool mysqli::get_connection_stats();
```

Procedural style

```
array mysqli_get_connection_stats(  
    mysqli link);
```

Returns statistics about the client connection. Available only with `mysqlnd`.

Parameters

link

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns an array with connection stats if success, [FALSE](#) otherwise.

Examples

Example 6.60 A [mysqli_get_connection_stats](#) example

```
<?php
$link = mysqli_connect();
print_r(mysqli_get_connection_stats($link));
?>
```

The above example will output something similar to:

```
Array
(
    [bytes_sent] => 43
    [bytes_received] => 80
    [packets_sent] => 1
    [packets_received] => 2
    [protocol_overhead_in] => 8
    [protocol_overhead_out] => 4
    [bytes_received_ok_packet] => 11
    [bytes_received_eof_packet] => 0
    [bytes_received_rset_header_packet] => 0
    [bytes_received_rset_field_meta_packet] => 0
    [bytes_received_rset_row_packet] => 0
    [bytes_received_prepare_response_packet] => 0
    [bytes_received_change_user_packet] => 0
    [packets_sent_command] => 0
    [packets_received_ok] => 1
    [packets_received_eof] => 0
    [packets_received_rset_header] => 0
    [packets_received_rset_field_meta] => 0
    [packets_received_rset_row] => 0
    [packets_received_prepare_response] => 0
    [packets_received_change_user] => 0
    [result_set_queries] => 0
    [non_result_set_queries] => 0
    [no_index_used] => 0
    [bad_index_used] => 0
    [slow_queries] => 0
    [buffered_sets] => 0
    [unbuffered_sets] => 0
    [ps_buffered_sets] => 0
    [ps_unbuffered_sets] => 0
    [flushed_normal_sets] => 0
    [flushed_ps_sets] => 0
    [ps_prepared_never_executed] => 0
    [ps_prepared_once_executed] => 0
    [rows_fetched_from_server_normal] => 0
    [rows_fetched_from_server_ps] => 0
    [rows_buffered_from_client_normal] => 0
    [rows_buffered_from_client_ps] => 0
    [rows_fetched_from_client_normal_buffered] => 0
    [rows_fetched_from_client_normal_unbuffered] => 0
    [rows_fetched_from_client_ps_buffered] => 0
    [rows_fetched_from_client_ps_unbuffered] => 0
    [rows_fetched_from_client_ps_cursor] => 0
    [rows_skipped_normal] => 0
    [rows_skipped_ps] => 0
    [copy_on_write_saved] => 0
)
```

```
[copy_on_write_performed] => 0
[command_buffer_too_small] => 0
[connect_success] => 1
[connect_failure] => 0
[connection_reused] => 0
[reconnect] => 0
[pconnect_success] => 0
[active_connections] => 1
[active_persistent_connections] => 0
[explicit_close] => 0
[implicit_close] => 0
[disconnect_close] => 0
[in_middle_of_command_close] => 0
[explicit_free_result] => 0
[implicit_free_result] => 0
[explicit_stmt_close] => 0
[implicit_stmt_close] => 0
[mem_emalloc_count] => 0
[mem_emalloc_ammount] => 0
[mem_ecalloc_count] => 0
[mem_ecalloc_ammount] => 0
[mem_erealloc_count] => 0
[mem_erealloc_ammount] => 0
[mem_efree_count] => 0
[mem_malloc_count] => 0
[mem_malloc_ammount] => 0
[mem_calloc_count] => 0
[mem_calloc_ammount] => 0
[mem_realloc_count] => 0
[mem_realloc_ammount] => 0
[mem_free_count] => 0
[proto_text_fetched_null] => 0
[proto_text_fetched_bit] => 0
[proto_text_fetched_tinyint] => 0
[proto_text_fetched_short] => 0
[proto_text_fetched_int24] => 0
[proto_text_fetched_int] => 0
[proto_text_fetched_bigint] => 0
[proto_text_fetched_decimal] => 0
[proto_text_fetched_float] => 0
[proto_text_fetched_double] => 0
[proto_text_fetched_date] => 0
[proto_text_fetched_year] => 0
[proto_text_fetched_time] => 0
[proto_text_fetched_datetime] => 0
[proto_text_fetched_timestamp] => 0
[proto_text_fetched_string] => 0
[proto_text_fetched_blob] => 0
[proto_text_fetched_enum] => 0
[proto_text_fetched_set] => 0
[proto_text_fetched_geometry] => 0
[proto_text_fetched_other] => 0
[proto_binary_fetched_null] => 0
[proto_binary_fetched_bit] => 0
[proto_binary_fetched_tinyint] => 0
[proto_binary_fetched_short] => 0
[proto_binary_fetched_int24] => 0
[proto_binary_fetched_int] => 0
[proto_binary_fetched_bigint] => 0
[proto_binary_fetched_decimal] => 0
[proto_binary_fetched_float] => 0
[proto_binary_fetched_double] => 0
[proto_binary_fetched_date] => 0
[proto_binary_fetched_year] => 0
[proto_binary_fetched_time] => 0
[proto_binary_fetched_datetime] => 0
[proto_binary_fetched_timestamp] => 0
[proto_binary_fetched_string] => 0
[proto_binary_fetched_blob] => 0
[proto_binary_fetched_enum] => 0
[proto_binary_fetched_set] => 0
[proto_binary_fetched_geometry] => 0
```



```
[proto_binary_fetched_other] => 0
)
```

See Also

[Stats description](#)

6.3.9.24 `mysqli::$host_info`, `mysqli_get_host_info`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::$host_info`
`mysqli_get_host_info`

Returns a string representing the type of connection used

Description

Object oriented style

```
string
mysqli->host_info ;
```

Procedural style

```
string mysqli_get_host_info(
    mysqli link);
```

Returns a string describing the connection represented by the [link](#) parameter (including the server host name).

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

A character string representing the server hostname and the connection type.

Examples**Example 6.61 `$mysqli->host_info` example**

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print host information */
printf("Host info: %s\n", $mysqli->host_info);
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print host information */
printf("Host info: %s\n", mysqli_get_host_info($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Host info: Localhost via UNIX socket
```

See Also

[mysqli_get_proto_info](#)

6.3.9.25 [mysqli::\\$protocol_version](#), [mysqli_get_proto_info](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$protocol_version](#)

[mysqli_get_proto_info](#)

Returns the version of the MySQL protocol used

Description

Object oriented style

```
string
mysqli->protocol_version ;
```

Procedural style

```
int mysqli_get_proto_info(
    mysqli link);
```

Returns an integer representing the MySQL protocol version used by the connection represented by the [link](#) parameter.

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns an integer representing the protocol version.

Examples

Example 6.62 `$mysqli->protocol_version` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print protocol version */
printf("Protocol version: %d\n", $mysqli->protocol_version);
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print protocol version */
printf("Protocol version: %d\n", mysqli_get_proto_info($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Protocol version: 10
```

See Also

[mysqli_get_host_info](#)

6.3.9.26 `mysqli::$server_info`, `mysqli_get_server_info`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$server_info](#)

[mysqli_get_server_info](#)

Returns the version of the MySQL server

Description

Object oriented style

```
string  
mysqli->server_info ;
```

Procedural style

```
string mysqli_get_server_info(  
    mysqli link);
```

Returns a string representing the version of the MySQL server that the MySQLi extension is connected to.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

A character string representing the server version.

Examples

Example 6.63 `$mysqli->server_info` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* print server version */  
printf("Server version: %s\n", $mysqli->server_info);  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* print server version */  
printf("Server version: %s\n", mysqli_get_server_info($link));  
/* close connection */  
mysqli_close($link);  
?>
```

The above examples will output:

```
Server version: 4.1.2-alpha-debug
```

See Also

[mysqli_get_client_info](#)
[mysqli_get_client_version](#)
[mysqli_get_server_version](#)

6.3.9.27 [mysqli::\\$server_version](#), [mysqli_get_server_version](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$server_version](#)
[mysqli_get_server_version](#)

Returns the version of the MySQL server as an integer

Description

Object oriented style

```
int  
    mysqli->server_version ;
```

Procedural style

```
int mysqli_get_server_version(  
    mysqli link);
```

The [mysqli_get_server_version](#) function returns the version of the server connected to (represented by the [link](#) parameter) as an integer.

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

An integer representing the server version.

The form of this version number is `main_version * 10000 + minor_version * 100 + sub_version` (i.e. version 4.1.0 is 40100).

Examples**Example 6.64 [\\$mysqli->server_version](#) example**

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* print server version */  
printf("Server version: %d\n", $mysqli->server_version);  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* print server version */
printf("Server version: %d\n", mysqli_get_server_version($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Server version: 40102
```

See Also

[mysqli_get_client_info](#)
[mysqli_get_client_version](#)
[mysqli_get_server_info](#)

6.3.9.28 [mysqli::get_warnings](#), [mysqli_get_warnings](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::get_warnings](#)

[mysqli_get_warnings](#)

Get result of SHOW WARNINGS

Description

Object oriented style

```
mysqli_warning mysqli::get_warnings();
```

Procedural style

```
mysqli_warning mysqli_get_warnings(
    mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

6.3.9.29 [mysqli::\\$info](#), [mysqli_info](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$info](#)

mysqli_info

Retrieves information about the most recently executed query

Description

Object oriented style

```
string  
mysqli->info ;
```

Procedural style

```
string mysqli_info(  
    mysqli link);
```

The `mysqli_info` function returns a string providing information about the last query executed. The nature of this string is provided below:

Table 6.9 Possible `mysqli_info` return values

Query type	Example result string
INSERT INTO...SELECT...	Records: 100 Duplicates: 0 Warnings: 0
INSERT INTO...VALUES (...),(...),(...)	Records: 3 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...	Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE ...	Records: 3 Duplicates: 0 Warnings: 0
UPDATE ...	Rows matched: 40 Changed: 40 Warnings: 0

Note

Queries which do not fall into one of the preceding formats are not supported. In these situations, `mysqli_info` will return an empty string.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

A character string representing additional information about the most recently executed query.

Examples

Example 6.65 `$mysqli->info` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TEMPORARY TABLE t1 LIKE City");
/* INSERT INTO .. SELECT */
$mysqli->query("INSERT INTO t1 SELECT * FROM City ORDER BY ID LIMIT 150");
printf("%s\n", $mysqli->info);
```

```
/* close connection */
mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TEMPORARY TABLE t1 LIKE City");
/* INSERT INTO .. SELECT */
mysqli_query($link, "INSERT INTO t1 SELECT * FROM City ORDER BY ID LIMIT 150");
printf("%s\n", mysqli_info($link));
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Records: 150  Duplicates: 0  Warnings: 0
```

See Also

[mysqli_affected_rows](#)
[mysqli_warning_count](#)
[mysqli_num_rows](#)

6.3.9.30 [mysqli::init](#), [mysqli_init](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::init](#)

[mysqli_init](#)

Initializes MySQLi and returns a resource for use with [mysqli_real_connect\(\)](#)

Description

Object oriented style

```
mysqli mysqli::init();
```

Procedural style

```
mysqli mysqli_init();
```

Allocates or initializes a MYSQL object suitable for [mysqli_options](#) and [mysqli_real_connect](#).

Note

Any subsequent calls to any mysqli function (except [mysqli_options](#)) will fail until [mysqli_real_connect](#) was called.

Return Values

Returns an object.

Examples

See [mysqli_real_connect](#).

See Also

[mysqli_options](#)
[mysqli_close](#)
[mysqli_real_connect](#)
[mysqli_connect](#)

6.3.9.31 [mysqli::\\$insert_id](#), [mysqli_insert_id](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$insert_id](#)
[mysqli_insert_id](#)

Returns the auto generated id used in the last query

Description

Object oriented style

```
mixed  
mysqli->insert_id ;
```

Procedural style

```
mixed mysqli_insert_id(  
    mysqli link);
```

The [mysqli_insert_id](#) function returns the ID generated by a query on a table with a column having the AUTO_INCREMENT attribute. If the last query wasn't an INSERT or UPDATE statement or if the modified table does not have a column with the AUTO_INCREMENT attribute, this function will return zero.

Note

Performing an INSERT or UPDATE statement using the LAST_INSERT_ID() function will also modify the value returned by the [mysqli_insert_id](#) function.

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

The value of the [AUTO_INCREMENT](#) field that was updated by the previous query. Returns zero if there was no previous query on the connection or if the query did not update an [AUTO_INCREMENT](#) value.

Note

If the number is greater than maximal int value, [mysqli_insert_id](#) will return a string.

Examples

Example 6.66 `$mysqli->insert_id` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCity LIKE City");
$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
$mysqli->query($query);
printf("New Record has id %d.\n", $mysqli->insert_id);
/* drop table */
$mysqli->query("DROP TABLE myCity");
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCity LIKE City");
$query = "INSERT INTO myCity VALUES (NULL, 'Stuttgart', 'DEU', 'Stuttgart', 617000)";
mysqli_query($link, $query);
printf("New Record has id %d.\n", mysqli_insert_id($link));
/* drop table */
mysqli_query($link, "DROP TABLE myCity");
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
New Record has id 1.
```

6.3.9.32 `mysqli::kill`, `mysqli_kill`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::kill`

`mysqli_kill`

Asks the server to kill a MySQL thread

Description

Object oriented style

```
bool mysqli::kill(
    int processid);
```

Procedural style

```
bool mysqli_kill(
    mysqli link,
    int processid);
```

This function is used to ask the server to kill a MySQL thread specified by the *processid* parameter. This value must be retrieved by calling the *mysqli_thread_id* function.

To stop a running query you should use the SQL command **KILL QUERY processid**.

Parameters

link Procedural style only: A link identifier returned by *mysqli_connect* or *mysqli_init*

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example 6.67 *mysqli::kill* example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* determine our thread id */
$thread_id = $mysqli->thread_id;
/* Kill connection */
$mysqli->kill($thread_id);
/* This should produce an error */
if (!$mysqli->query("CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", $mysqli->error);
    exit;
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* determine our thread id */
```

```
$thread_id = mysqli_thread_id($link);
/* Kill connection */
mysqli_kill($link, $thread_id);
/* This should produce an error */
if (!mysqli_query($link, "CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", mysqli_error($link));
    exit;
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: MySQL server has gone away
```

See Also

[mysqli_thread_id](#)

6.3.9.33 [mysqli::more_results](#), [mysqli_more_results](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::more_results](#)

[mysqli_more_results](#)

Check if there are any more query results from a multi query

Description

Object oriented style

```
bool mysqli::more_results();
```

Procedural style

```
bool mysqli_more_results(
    mysqli link);
```

Indicates if one or more result sets are available from a previous call to [mysqli_multi_query](#).

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns [TRUE](#) if one or more result sets are available from a previous call to [mysqli_multi_query](#), otherwise [FALSE](#).

Examples

See [mysqli_multi_query](#).

See Also

```
mysqli_multi_query  
mysqli_next_result  
mysqli_store_result  
mysqli_use_result
```

6.3.9.34 `mysqli::multi_query`, `mysqli_multi_query`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::multi_query`

`mysqli_multi_query`

Performs a query on the database

Description

Object oriented style

```
bool mysqli::multi_query(  
    string query);
```

Procedural style

```
bool mysqli_multi_query(  
    mysqli link,  
    string query);
```

Executes one or multiple queries which are concatenated by a semicolon.

To retrieve the resultset from the first query you can use `mysqli_use_result` or `mysqli_store_result`. All subsequent query results can be processed using `mysqli_more_results` and `mysqli_next_result`.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by <code>mysqli_connect</code> or <code>mysqli_init</code>
<i>query</i>	The query, as a string. Data inside the query should be properly escaped .

Return Values

Returns `FALSE` if the first statement failed. To retrieve subsequent errors from other statements you have to call `mysqli_next_result` first.

Examples

Example 6.68 `mysqli::multi_query` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}
```

```
$query = "SELECT CURRENT_USER();";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
/* execute multi query */
if ($mysqli->multi_query($query)) {
    do {
        /* store first result set */
        if ($result = $mysqli->store_result()) {
            while ($row = $result->fetch_row()) {
                printf("%s\n", $row[0]);
            }
            $result->free();
        }
        /* print divider */
        if ($mysqli->more_results()) {
            printf("-----\n");
        }
    } while ($mysqli->next_result());
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT CURRENT_USER();";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
/* execute multi query */
if (mysqli_multi_query($link, $query)) {
    do {
        /* store first result set */
        if ($result = mysqli_store_result($link)) {
            while ($row = mysqli_fetch_row($result)) {
                printf("%s\n", $row[0]);
            }
            mysqli_free_result($result);
        }
        /* print divider */
        if (mysqli_more_results($link)) {
            printf("-----\n");
        }
    } while (mysqli_next_result($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output something similar to:

```
my_user@localhost
-----
Amersfoort
Maastricht
Dordrecht
Leiden
Haarlemmermeer
```

See Also

[mysqli_query](#)
[mysqli_use_result](#)
[mysqli_store_result](#)
[mysqli_next_result](#)
[mysqli_more_results](#)

6.3.9.35 [mysqli::next_result](#), [mysqli_next_result](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::next_result](#)

[mysqli_next_result](#)

Prepare next result from [multi_query](#)

Description

Object oriented style

```
bool mysqli::next_result();
```

Procedural style

```
bool mysqli_next_result(  
    mysqli link);
```

Prepares next result set from a previous call to [mysqli_multi_query](#) which can be retrieved by [mysqli_store_result](#) or [mysqli_use_result](#).

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

See [mysqli_multi_query](#).

See Also

[mysqli_multi_query](#)
[mysqli_more_results](#)
[mysqli_store_result](#)
[mysqli_use_result](#)

6.3.9.36 [mysqli::options](#), [mysqli_options](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::options](#)

[mysqli_options](#)

Set options

Description

Object oriented style

```
bool mysqli::options(  
    int option,  
    mixed value);
```

Procedural style

```
bool mysqli_options(  
    mysqli link,  
    int option,  
    mixed value);
```

Used to set extra connect options and affect behavior for a connection.

This function may be called multiple times to set several options.

`mysqli_options` should be called after `mysqli_init` and before `mysqli_real_connect`.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

option

The option that you want to set. It can be one of the following values:

Table 6.10 Valid options

Name	Description
<code>MYSQLI_OPT_CONNECT_TIMEOUT</code>	Connection timeout in seconds (supported on Windows with TCP/IP since PHP 5.3.1)
<code>MYSQLI_OPT_LOCAL_INFILE</code>	enable/disable use of <code>LOAD LOCAL INFILE</code>
<code>MYSQLI_INIT_COMMAND</code>	command to execute after when connecting to MySQL server
<code>MYSQLI_READ_DEFAULT_FILE</code>	Read options from named option file instead of <code>my.cnf</code>
<code>MYSQLI_READ_DEFAULT_GROUP</code>	Read options from the named group from <code>my.cnf</code> or the file specified with <code>MYSQLI_READ_DEFAULT_FILE</code> .
<code>MYSQLI_SERVER_PUBLIC_KEY</code>	RSA public key file used with the SHA-256 based authentication.
<code>MYSQLI_OPT_NET_CMD_BUFFER_SIZE</code>	The size of the internal command/network buffer. Only valid for <code>mysqlnd</code> .
<code>MYSQLI_OPT_NET_READ_BUFFER_SIZE</code>	Maximum read chunk size in bytes when reading the body of a MySQL command packet. Only valid for <code>mysqlnd</code> .
<code>MYSQLI_OPT_INT_AND_FLOAT_CONVERT</code>	Convert integer and float columns back to PHP numbers. Only valid for <code>mysqlnd</code> .
<code>MYSQLI_OPT_SSL_VERIFY_SERVER_CERT</code>	

value The value for the option.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Changelog

Version	Description
5.5.0	The MYSQLI_SERVER_PUBLIC_KEY and MYSQLI_SERVER_PUBLIC_KEY options were added.
5.3.0	The MYSQLI_OPT_INT_AND_FLOAT_NATIVE , MYSQLI_OPT_NET_CMD_BUFFER_SIZE , MYSQLI_OPT_NET_READ_BUFFER_SIZE , and MYSQLI_OPT_SSL_VERIFY_SERVER_CERT options were added.

Examples

See [mysqli_real_connect](#).

Notes

Note

MySQLnd always assumes the server default charset. This charset is sent during connection hand-shake/authentication, which mysqlnd will use.

Libmysqlclient uses the default charset set in the [my.cnf](#) or by an explicit call to [mysqli_options](#) prior to calling [mysqli_real_connect](#), but after [mysqli_init](#).

See Also

[mysqli_init](#)
[mysqli_real_connect](#)

6.3.9.37 [mysqli::ping](#), [mysqli_ping](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::ping](#)

[mysqli_ping](#)

Pings a server connection, or tries to reconnect if the connection has gone down

Description

Object oriented style

```
bool mysqli::ping();
```

Procedural style

```
bool mysqli_ping(  
    mysqli link);
```

Checks whether the connection to the server is working. If it has gone down, and global option [mysqli.reconnect](#) is enabled an automatic reconnection is attempted.

This function can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 6.69 `mysqli::ping` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
/* check if server is alive */
if ($mysqli->ping()) {
    printf("Our connection is ok!\n");
} else {
    printf("Error: %s\n", $mysqli->error);
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* check if server is alive */
if (mysqli_ping($link)) {
    printf("Our connection is ok!\n");
} else {
    printf("Error: %s\n", mysqli_error($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Our connection is ok!
```

6.3.9.38 `mysqli::poll`, `mysqli_poll`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::poll`

`mysqli_poll`

Poll connections

Description

Object oriented style

```
public static int mysqli::poll(
    array read,
    array error,
    array reject,
    int sec,
    int usec);
```

Procedural style

```
int mysqli_poll(
    array read,
    array error,
    array reject,
    int sec,
    int usec);
```

Poll connections. Available only with [mysqlnd](#). The method can be used as [static](#).

Parameters

<i>read</i>	List of connections to check for outstanding results that can be read.
<i>error</i>	List of connections on which an error occurred, for example, query failure or lost connection.
<i>reject</i>	List of connections rejected because no asynchronous query has been run on for which the function could poll results.
<i>sec</i>	Number of seconds to wait, must be non-negative.
<i>usec</i>	Number of microseconds to wait, must be non-negative.

Return Values

Returns number of ready connections upon success, [FALSE](#) otherwise.

Examples

Example 6.70 A `mysqli_poll` example

```
<?php
$link1 = mysqli_connect();
$link1->query("SELECT 'test'", MYSQLI_ASYNC);
$arr_links = array($link1);
$processed = 0;
do {
```

```
$links = $errors = $reject = array();
foreach ($all_links as $link) {
    $links[] = $errors[] = $reject[] = $link;
}
if (!mysqli_poll($links, $errors, $reject, 1)) {
    continue;
}
foreach ($links as $link) {
    if ($result = $link->reap_async_query()) {
        print_r($result->fetch_row());
        if (is_object($result))
            mysqli_free_result($result);
    } else die(sprintf("MySQLi Error: %s", mysqli_error($link)));
    $processed++;
}
} while ($processed < count($all_links));
?>
```

The above example will output:

```
Array
(
    [0] => test
)
```

See Also

[mysqli_query](#)
[mysqli_reap_async_query](#)

6.3.9.39 [mysqli::prepare](#), [mysqli_prepare](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::prepare](#)

[mysqli_prepare](#)

Prepare an SQL statement for execution

Description

Object oriented style

```
mysqli_stmt mysqli::prepare(
    string query);
```

Procedural style

```
mysqli_stmt mysqli_prepare(
    mysqli link,
    string query);
```

Prepares the SQL query, and returns a statement handle to be used for further operations on the statement. The query must consist of a single SQL statement.

The parameter markers must be bound to application variables using [mysqli_stmt_bind_param](#) and/or [mysqli_stmt_bind_result](#) before executing the statement or fetching rows.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

query

The query, as a string.

Note

You should not add a terminating semicolon or `\g` to the statement.

This parameter can include one or more parameter markers in the SQL statement by embedding question mark (?) characters at the appropriate positions.

Note

The markers are legal only in certain places in SQL statements. For example, they are allowed in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value.

However, they are not allowed for identifiers (such as table or column names), in the select list that names the columns to be returned by a `SELECT` statement, or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. It's not allowed to compare marker with `NULL` by `? IS NULL` too. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

Return Values

`mysqli_prepare` returns a statement object or `FALSE` if an error occurred.

Examples**Example 6.71 `mysqli::prepare` example**

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$city = "Amersfoort";
/* create a prepared statement */
if ($stmt = $mysqli->prepare("SELECT District FROM City WHERE Name=?")) {
    /* bind parameters for markers */
    $stmt->bind_param("s", $city);
    /* execute query */
    $stmt->execute();
}
```

```
/* bind result variables */
$stmt->bind_result($district);
/* fetch value */
$stmt->fetch();
printf("%s is in district %s\n", $city, $district);
/* close statement */
$stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$city = "Amersfoort";
/* create a prepared statement */
if ($stmt = mysqli_prepare($link, "SELECT District FROM City WHERE Name=?")) {
    /* bind parameters for markers */
    mysqli_stmt_bind_param($stmt, "s", $city);
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $district);
    /* fetch value */
    mysqli_stmt_fetch($stmt);
    printf("%s is in district %s\n", $city, $district);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Amersfoort is in district Utrecht
```

See Also

[mysqli_stmt_execute](#)
[mysqli_stmt_fetch](#)
[mysqli_stmt_bind_param](#)
[mysqli_stmt_bind_result](#)
[mysqli_stmt_close](#)

6.3.9.40 [mysqli::query](#), [mysqli_query](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::query](#)
[mysqli_query](#)

Performs a query on the database

Description

Object oriented style

```
mixed mysqli::query(  
    string query,  
    int resultmode  
        = MYSQLI_STORE_RESULT);
```

Procedural style

```
mixed mysqli_query(  
    mysqli link,  
    string query,  
    int resultmode  
        = MYSQLI_STORE_RESULT);
```

Performs a *query* against the database.

For non-DML queries (not INSERT, UPDATE or DELETE), this function is similar to calling *mysqli_real_query* followed by either *mysqli_use_result* or *mysqli_store_result*.

Note

In the case where you pass a statement to *mysqli_query* that is longer than *max_allowed_packet* of the server, the returned error codes are different depending on whether you are using MySQL Native Driver (*mysqlnd*) or MySQL Client Library (*libmysqlclient*). The behavior is as follows:

- *mysqlnd* on Linux returns an error code of 1153. The error message means “got a packet bigger than *max_allowed_packet* bytes”.
- *mysqlnd* on Windows returns an error code 2006. This error message means “server has gone away”.
- *libmysqlclient* on all platforms returns an error code 2006. This error message means “server has gone away”.

Parameters

link

Procedural style only: A link identifier returned by *mysqli_connect* or *mysqli_init*

query

The query string.

Data inside the query should be *properly escaped*.

resultmode

Either the constant *MYSQLI_USE_RESULT* or *MYSQLI_STORE_RESULT* depending on the desired behavior. By default, *MYSQLI_STORE_RESULT* is used.

If you use *MYSQLI_USE_RESULT* all subsequent calls will return error *Commands out of sync* unless you call *mysqli_free_result*

With *MYSQLI_ASYNC* (available with *mysqlnd*), it is possible to perform query asynchronously. *mysqli_poll* is then used to get results from such queries.

Return Values

Returns **FALSE** on failure. For successful **SELECT**, **SHOW**, **DESCRIBE** or **EXPLAIN** queries **mysqli_query** will return a **mysqli_result** object. For other successful queries **mysqli_query** will return **TRUE**.

Changelog

Version	Description
5.3.0	Added the ability of async queries.

Examples

Example 6.72 **mysqli::query** example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
/* Create table doesn't return a resultset */
if ($mysqli->query("CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
    printf("Table myCity successfully created.\n");
}
/* Select queries return a resultset */
if ($result = $mysqli->query("SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", $result->num_rows);
    /* free result set */
    $result->close();
}
/* If we have to retrieve large amount of data we use MYSQLI_USE_RESULT */
if ($result = $mysqli->query("SELECT * FROM City", MYSQLI_USE_RESULT)) {
    /* Note, that we can't execute any functions which interact with the
       server until result set was closed. All calls will return an
       'out of sync' error */
    if (!$mysqli->query("SET @a:='this will not work'")) {
        printf("Error: %s\n", $mysqli->error);
    }
    $result->close();
}
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Create table doesn't return a resultset */
if (mysqli_query($link, "CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
    printf("Table myCity successfully created.\n");
}
/* Select queries return a resultset */
if ($result = mysqli_query($link, "SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", mysqli_num_rows($result));
    /* free result set */
    mysqli_free_result($result);
}
```



```
}
/* If we have to retrieve large amount of data we use MYSQLI_USE_RESULT */
if ($result = mysqli_query($link, "SELECT * FROM City", MYSQLI_USE_RESULT)) {
    /* Note, that we can't execute any functions which interact with the
       server until result set was closed. All calls will return an
       'out of sync' error */
    if (!mysqli_query($link, "SET @a:='this will not work'")) {
        printf("Error: %s\n", mysqli_error($link));
    }
    mysqli_free_result($result);
}
mysqli_close($link);
?>
```

The above examples will output:

```
Table myCity successfully created.
Select returned 10 rows.
Error: Commands out of sync; You can't run this command now
```

See Also

[mysqli_real_query](#)
[mysqli_multi_query](#)
[mysqli_free_result](#)

6.3.9.41 [mysqli::real_connect](#), [mysqli_real_connect](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::real_connect](#)

[mysqli_real_connect](#)

Opens a connection to a mysql server

Description

Object oriented style

```
bool mysqli::real_connect(
    string host,
    string username,
    string passwd,
    string dbname,
    int port,
    string socket,
    int flags);
```

Procedural style

```
bool mysqli_real_connect(
    mysqli link,
    string host,
    string username,
    string passwd,
    string dbname,
    int port,
    string socket,
    int flags);
```

Establish a connection to a MySQL database engine.

This function differs from `mysqli_connect`:

- `mysqli_real_connect` needs a valid object which has to be created by function `mysqli_init`.
- With the `mysqli_options` function you can set various options for connection.
- There is a `flags` parameter.

Parameters

<code>link</code>	Procedural style only: A link identifier returned by <code>mysqli_connect</code> or <code>mysqli_init</code>
<code>host</code>	Can be either a host name or an IP address. Passing the <code>NULL</code> value or the string "localhost" to this parameter, the local host is assumed. When possible, pipes will be used instead of the TCP/IP protocol.
<code>username</code>	The MySQL user name.
<code>passwd</code>	If provided or <code>NULL</code> , the MySQL server will attempt to authenticate the user against those user records which have no password only. This allows one username to be used with different permissions (depending on if a password as provided or not).
<code>dbname</code>	If provided will specify the default database to be used when performing queries.
<code>port</code>	Specifies the port number to attempt to connect to the MySQL server.
<code>socket</code>	Specifies the socket or named pipe that should be used.

Note

Specifying the `socket` parameter will not explicitly determine the type of connection to be used when connecting to the MySQL server. How the connection is made to the MySQL database is determined by the `host` parameter.

`flags` With the parameter `flags` you can set different connection options:

Table 6.11 Supported flags

Name	Description
<code>MYSQLI_CLIENT_COMPRESS</code>	Use compression protocol
<code>MYSQLI_CLIENT_FOUND_ROWS</code>	return number of matched rows, not the number of affected rows
<code>MYSQLI_CLIENT_IGNORE_SPACE</code>	Allow spaces after function names. Makes all function names reserved words.
<code>MYSQLI_CLIENT_INTERACTIVE</code>	Allow <code>interactive_timeout</code> seconds (instead of <code>wait_timeout</code> seconds) of inactivity before closing the connection
<code>MYSQLI_CLIENT_SSL</code>	Use SSL (encryption)

Name	Description
<code>MYSQLI_CLIENT_SSL_DONT_VERIFY_SERVER_CERT</code>	Like <code>MYSQLI_CLIENT_SSL</code> , but disables validation of the provided SSL certificate. This is only for installations using MySQL Native Driver and MySQL 5.6 or later.

Note

For security reasons the `MULTI_STATEMENT` flag is not supported in PHP. If you want to execute multiple queries use the `mysqli_multi_query` function.

Changelog

Version	Description
5.6.16	Added the <code>MYSQLI_CLIENT_SSL_DONT_VERIFY_SERVER_CERT</code> flag for MySQL Native Driver

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples**Example 6.73 `mysqli::real_connect` example**

Object oriented style

```
<?php
$mysqli = mysqli_init();
if (!$mysqli) {
    die('mysqli_init failed');
}
if (!$mysqli->options(MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
    die('Setting MYSQLI_INIT_COMMAND failed');
}
if (!$mysqli->options(MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
    die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
}
if (!$mysqli->real_connect('localhost', 'my_user', 'my_password', 'my_db')) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}
echo 'Success... ' . $mysqli->host_info . "\n";
$mysqli->close();
?>
```

Object oriented style when extending mysqli class

```
<?php
class foo_mysqli extends mysqli {
    public function __construct($host, $user, $pass, $db) {
        parent::init();
        if (!parent::options(MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
```

```
        die('Setting MYSQLI_INIT_COMMAND failed');
    }
    if (!parent::options(MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
        die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
    }
    if (!parent::real_connect($host, $user, $pass, $db)) {
        die('Connect Error (' . mysqli_connect_errno() . ') '
            . mysqli_connect_error());
    }
}
$db = new foo_mysqli('localhost', 'my_user', 'my_password', 'my_db');
echo 'Success... ' . $db->host_info . "\n";
$db->close();
?>
```

Procedural style

```
<?php
$link = mysqli_init();
if (!$link) {
    die('mysqli_init failed');
}
if (!mysqli_options($link, MYSQLI_INIT_COMMAND, 'SET AUTOCOMMIT = 0')) {
    die('Setting MYSQLI_INIT_COMMAND failed');
}
if (!mysqli_options($link, MYSQLI_OPT_CONNECT_TIMEOUT, 5)) {
    die('Setting MYSQLI_OPT_CONNECT_TIMEOUT failed');
}
if (!mysqli_real_connect($link, 'localhost', 'my_user', 'my_password', 'my_db')) {
    die('Connect Error (' . mysqli_connect_errno() . ') '
        . mysqli_connect_error());
}
echo 'Success... ' . mysqli_get_host_info($link) . "\n";
mysqli_close($link);
?>
```

The above examples will output:

```
Success... MySQL host info: localhost via TCP/IP
```

Notes

Note

MySQLnd always assumes the server default charset. This charset is sent during connection hand-shake/authentication, which mysqlnd will use.

Libmysqlclient uses the default charset set in the [my.cnf](#) or by an explicit call to [mysqli_options](#) prior to calling [mysqli_real_connect](#), but after [mysqli_init](#).

See Also

[mysqli_connect](#)
[mysqli_init](#)
[mysqli_options](#)
[mysqli_ssl_set](#)

`mysqli_close`

6.3.9.42 `mysqli::real_escape_string`, `mysqli_real_escape_string`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::real_escape_string`

`mysqli_real_escape_string`

Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection

Description

Object oriented style

```
string mysqli::escape_string(  
    string escapestr);
```

```
string mysqli::real_escape_string(  
    string escapestr);
```

Procedural style

```
string mysqli_real_escape_string(  
    mysqli link,  
    string escapestr);
```

This function is used to create a legal SQL string that you can use in an SQL statement. The given string is encoded to an escaped SQL string, taking into account the current character set of the connection.

Security: the default character set

The character set must be set either at the server level, or with the API function `mysqli_set_charset` for it to affect `mysqli_real_escape_string`. See the concepts section on [character sets](#) for more information.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

escapestr

The string to be escaped.

Characters encoded are NUL (ASCII 0), `\n`, `\r`, `\`, `'`, `"`, and Control-Z.

Return Values

Returns an escaped string.

Examples

Example 6.74 `mysqli::real_escape_string` example

Object oriented style

```
<?php
```

```

$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TEMPORARY TABLE myCity LIKE City");
$city = "'s Hertogenbosch";
/* this query will fail, cause we didn't escape $city */
if (!$mysqli->query("INSERT into myCity (Name) VALUES ('$city')")) {
    printf("Error: %s\n", $mysqli->sqlstate);
}
$city = $mysqli->real_escape_string($city);
/* this query with escaped $city will work */
if ($mysqli->query("INSERT into myCity (Name) VALUES ('$city')")) {
    printf("%d Row inserted.\n", $mysqli->affected_rows);
}
$mysqli->close();
?>

```

Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TEMPORARY TABLE myCity LIKE City");
$city = "'s Hertogenbosch";
/* this query will fail, cause we didn't escape $city */
if (!mysqli_query($link, "INSERT into myCity (Name) VALUES ('$city')")) {
    printf("Error: %s\n", mysqli_sqlstate($link));
}
$city = mysqli_real_escape_string($link, $city);
/* this query with escaped $city will work */
if (mysqli_query($link, "INSERT into myCity (Name) VALUES ('$city')")) {
    printf("%d Row inserted.\n", mysqli_affected_rows($link));
}
mysqli_close($link);
?>

```

The above examples will output:

```

Error: 42000
1 Row inserted.

```

Notes

Note

For those accustomed to using [mysql_real_escape_string](#), note that the arguments of [mysqli_real_escape_string](#) differ from what [mysql_real_escape_string](#) expects. The *link* identifier comes first in [mysqli_real_escape_string](#), whereas the string to be escaped comes first in [mysql_real_escape_string](#).

See Also

[mysqli_set_charset](#)
[mysqli_character_set_name](#)

6.3.9.43 [mysqli::real_query](#), [mysqli_real_query](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::real_query](#)

[mysqli_real_query](#)

Execute an SQL query

Description

Object oriented style

```
bool mysqli::real_query(  
    string query);
```

Procedural style

```
bool mysqli_real_query(  
    mysqli link,  
    string query);
```

Executes a single query against the database whose result can then be retrieved or stored using the [mysqli_store_result](#) or [mysqli_use_result](#) functions.

In order to determine if a given query should return a result set or not, see [mysqli_field_count](#).

Parameters

<i>link</i>	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
<i>query</i>	The query, as a string. Data inside the query should be properly escaped .

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

See Also

[mysqli_query](#)
[mysqli_store_result](#)
[mysqli_use_result](#)

6.3.9.44 [mysqli::reap_async_query](#), [mysqli_reap_async_query](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::reap_async_query](#)

[mysqli_reap_async_query](#)

Get result from async query

Description

Object oriented style

```
public mysqli_result mysqli::reap_async_query();
```

Procedural style

```
mysqli_result mysqli_reap_async_query(  
    mysqli link);
```

Get result from async query. Available only with [mysqlind](#).

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

Returns [mysqli_result](#) in success, [FALSE](#) otherwise.

See Also

[mysqli_poll](#)

6.3.9.45 [mysqli::refresh](#), [mysqli_refresh](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::refresh](#)

[mysqli_refresh](#)

Refreshes

Description

Object oriented style

```
public bool mysqli::refresh(  
    int options);
```

Procedural style

```
int mysqli_refresh(  
    resource link,  
    int options);
```

Flushes tables or caches, or resets the replication server information.

Parameters

[link](#) Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

[options](#) The options to refresh, using the `MYSQLI_REFRESH_*` constants as documented within the [MySQLi constants](#) documentation.

See also the official [MySQL Refresh](#) documentation.

Return Values

[TRUE](#) if the refresh was a success, otherwise [FALSE](#)

See Also[mysqli_poll](#)**6.3.9.46 [mysqli::release_savepoint](#), [mysqli_release_savepoint](#)**

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::release_savepoint](#)

[mysqli_release_savepoint](#)

Removes the named savepoint from the set of savepoints of the current transaction

Description

Object oriented style (method):

```
public bool mysqli::release_savepoint(
    string name);
```

Procedural style:

```
bool mysqli_release_savepoint(
    mysqli link,
    string name);
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

link

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

name

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

See Also[mysqli_rollback](#)**6.3.9.47 [mysqli::rollback](#), [mysqli_rollback](#)**

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::rollback](#)

[mysqli_rollback](#)

Rolls back current transaction

Description

Object oriented style

```
bool mysqli::rollback(
```

```
int flags,  
string name);
```

Procedural style

```
bool mysqli_rollback(  
    mysqli link,  
    int flags,  
    string name);
```

Rollbacks the current transaction for the database.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by <code>mysqli_connect</code> or <code>mysqli_init</code>
<i>flags</i>	A bitmask of <code>MYSQLI_TRANS_COR_*</code> constants.
<i>name</i>	If provided then <code>ROLLBACK/*name*/</code> is executed.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Changelog

Version	Description
5.5.0	Added <i>flags</i> and <i>name</i> parameters.

Examples

Example 6.75 `mysqli::rollback` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* disable autocommit */  
$mysqli->autocommit(FALSE);  
$mysqli->query("CREATE TABLE myCity LIKE City");  
$mysqli->query("ALTER TABLE myCity Type=InnoDB");  
$mysqli->query("INSERT INTO myCity SELECT * FROM City LIMIT 50");  
/* commit insert */  
$mysqli->commit();  
/* delete all rows */  
$mysqli->query("DELETE FROM myCity");  
if ($result = $mysqli->query("SELECT COUNT(*) FROM myCity")) {  
    $row = $result->fetch_row();  
    printf("%d rows in table myCity.\n", $row[0]);  
    /* Free result */  
    $result->close();  
}  
/* Rollback */  
$mysqli->rollback();  
if ($result = $mysqli->query("SELECT COUNT(*) FROM myCity")) {  
    $row = $result->fetch_row();  
    printf("%d rows in table myCity (after rollback).\n", $row[0]);  
    /* Free result */
```

```
        $result->close();
    }
    /* Drop table myCity */
    $mysqli->query("DROP TABLE myCity");
    $mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* disable autocommit */
mysqli_autocommit($link, FALSE);
mysqli_query($link, "CREATE TABLE myCity LIKE City");
mysqli_query($link, "ALTER TABLE myCity Type=InnoDB");
mysqli_query($link, "INSERT INTO myCity SELECT * FROM City LIMIT 50");
/* commit insert */
mysqli_commit($link);
/* delete all rows */
mysqli_query($link, "DELETE FROM myCity");
if ($result = mysqli_query($link, "SELECT COUNT(*) FROM myCity")) {
    $row = mysqli_fetch_row($result);
    printf("%d rows in table myCity.\n", $row[0]);
    /* Free result */
    mysqli_free_result($result);
}
/* Rollback */
mysqli_rollback($link);
if ($result = mysqli_query($link, "SELECT COUNT(*) FROM myCity")) {
    $row = mysqli_fetch_row($result);
    printf("%d rows in table myCity (after rollback).\n", $row[0]);
    /* Free result */
    mysqli_free_result($result);
}
/* Drop table myCity */
mysqli_query($link, "DROP TABLE myCity");
mysqli_close($link);
?>
```

The above examples will output:

```
0 rows in table myCity.
50 rows in table myCity (after rollback).
```

See Also

[mysqli_begin_transaction](#)
[mysqli_commit](#)
[mysqli_autocommit](#)
[mysqli_release_savepoint](#)

6.3.9.48 [mysqli::rpl_query_type](#), [mysqli_rpl_query_type](#)

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::rpl_query_type`

`mysqli_rpl_query_type`

Returns RPL query type

Description

Object oriented style

```
int mysqli::rpl_query_type(
    string query);
```

Procedural style

```
int mysqli_rpl_query_type(
    mysqli link,
    string query);
```

Returns `MYSQLI_RPL_MASTER`, `MYSQLI_RPL_SLAVE` or `MYSQLI_RPL_ADMIN` depending on a query type. `INSERT`, `UPDATE` and similar are *master* queries, `SELECT` is *slave*, and `FLUSH`, `REPAIR` and similar are *admin*.

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.9.49 `mysqli::savepoint`, `mysqli_savepoint`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::savepoint`

`mysqli_savepoint`

Set a named transaction savepoint

Description

Object oriented style (method):

```
public bool mysqli::savepoint(
    string name);
```

Procedural style:

```
bool mysqli_savepoint(
    mysqli link,
    string name);
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

name

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

[mysqli_commit](#)

6.3.9.50 **mysqli::select_db, mysqli_select_db**

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::select_db](#)

[mysqli_select_db](#)

Selects the default database for database queries

Description

Object oriented style

```
bool mysqli::select_db(  
    string dbname);
```

Procedural style

```
bool mysqli_select_db(  
    mysqli link,  
    string dbname);
```

Selects the default database to be used when performing queries against the database connection.

Note

This function should only be used to change the default database for the connection. You can select the default database with 4th parameter in [mysqli_connect](#).

Parameters

link

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

dbname

The database name.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example 6.76 **mysqli::select_db** example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");  
/* check connection */
```

```
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* return name of current default database */
if ($result = $mysqli->query("SELECT DATABASE())) {
    $row = $result->fetch_row();
    printf("Default database is %s.\n", $row[0]);
    $result->close();
}
/* change db to world db */
$mysqli->select_db("world");
/* return name of current default database */
if ($result = $mysqli->query("SELECT DATABASE())) {
    $row = $result->fetch_row();
    printf("Default database is %s.\n", $row[0]);
    $result->close();
}
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* return name of current default database */
if ($result = mysqli_query($link, "SELECT DATABASE())) {
    $row = mysqli_fetch_row($result);
    printf("Default database is %s.\n", $row[0]);
    mysqli_free_result($result);
}
/* change db to world db */
mysqli_select_db($link, "world");
/* return name of current default database */
if ($result = mysqli_query($link, "SELECT DATABASE())) {
    $row = mysqli_fetch_row($result);
    printf("Default database is %s.\n", $row[0]);
    mysqli_free_result($result);
}
mysqli_close($link);
?>
```

The above examples will output:

```
Default database is test.
Default database is world.
```

See Also

[mysqli_connect](#)
[mysqli_real_connect](#)

6.3.9.51 `mysqli::send_query`, `mysqli_send_query`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::send_query](#)

[mysqli_send_query](#)

Send the query and return

Description

Object oriented style

```
bool mysqli::send_query(  
    string query);
```

Procedural style

```
bool mysqli_send_query(  
    mysqli link,  
    string query);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.9.52 [mysqli::set_charset](#), [mysqli_set_charset](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::set_charset](#)

[mysqli_set_charset](#)

Sets the default client character set

Description

Object oriented style

```
bool mysqli::set_charset(  
    string charset);
```

Procedural style

```
bool mysqli_set_charset(  
    mysqli link,  
    string charset);
```

Sets the default character set to be used when sending data from and to the database server.

Parameters

link

Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

charset

The charset to be set as default.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Notes

Note

To use this function on a Windows platform you need MySQL client library version 4.1.11 or above (for MySQL 5.0 you need 5.0.6 or above).

Note

This is the preferred way to change the charset. Using `mysqli_query` to set it (such as `SET NAMES utf8`) is not recommended. See the [MySQL character set concepts](#) section for more information.

Examples

Example 6.77 `mysqli::set_charset` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
printf("Initial character set: %s\n", $mysqli->character_set_name());
/* change character set to utf8 */
if (!$mysqli->set_charset("utf8")) {
    printf("Error loading character set utf8: %s\n", $mysqli->error);
    exit();
} else {
    printf("Current character set: %s\n", $mysqli->character_set_name());
}
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'test');
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
printf("Initial character set: %s\n", mysqli_character_set_name($link));
/* change character set to utf8 */
if (!mysqli_set_charset($link, "utf8")) {
    printf("Error loading character set utf8: %s\n", mysqli_error($link));
    exit();
} else {
    printf("Current character set: %s\n", mysqli_character_set_name($link));
}
mysqli_close($link);
?>
```

The above examples will output something similar to:

```
Initial character set: latin1
Current character set: utf8
```

See Also

[mysqli_character_set_name](#)
[mysqli_real_escape_string](#)
[MySQL character set concepts](#)
[List of character sets that MySQL supports](#)

**6.3.9.53 `mysqli::set_local_infile_default`,
`mysqli_set_local_infile_default`**

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::set_local_infile_default`

`mysqli_set_local_infile_default`

Unsets user defined handler for load local infile command

Description

```
void mysqli_set_local_infile_default(
    mysqli link);
```

Deactivates a `LOAD DATA INFILE LOCAL` handler previously set with `mysqli_set_local_infile_handler`.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

No value is returned.

Examples

See `mysqli_set_local_infile_handler` examples

See Also

`mysqli_set_local_infile_handler`

**6.3.9.54 `mysqli::set_local_infile_handler`,
`mysqli_set_local_infile_handler`**

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::set_local_infile_handler`

`mysqli_set_local_infile_handler`

Set callback function for `LOAD DATA LOCAL INFILE` command

Description

Object oriented style

```
bool mysqli::set_local_infile_handler(  
    mysqli link,  
    callable read_func);
```

Procedural style

```
bool mysqli_set_local_infile_handler(  
    mysqli link,  
    callable read_func);
```

Set callback function for LOAD DATA LOCAL INFILE command

The callback task is to read input from the file specified in the [LOAD DATA LOCAL INFILE](#) and to reformat it into the format understood by [LOAD DATA INFILE](#).

The returned data needs to match the format specified in the [LOAD DATA](#)

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init	
read_func	A callback function or object method taking the following parameters:	
	stream	A PHP stream associated with the SQL commands INFILE
	&buffer	A string buffer to store the rewritten input into
	buflen	The maximum number of characters to be stored in the buffer
	&errmsg	If an error occurs you can store an error message in here

The callback function should return the number of characters stored in the [buffer](#) or a negative value if an error occurred.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 6.78 [mysqli::set_local_infile_handler](#) example

Object oriented style

```
<?php  
$db = mysqli_init();  
$db->real_connect("localhost","root","","test");  
function callme($stream, &$buffer, $buflen, &$errmsg)  
{  
    $buffer = fgets($stream);  
    echo $buffer;
```

```
// convert to upper case and replace "," delimiter with [TAB]
$buffer = strtoupper(str_replace(",", "\t", $buffer));
return strlen($buffer);
}
echo "Input:\n";
$db->set_local_infile_handler("callme");
$db->query("LOAD DATA LOCAL INFILE 'input.txt' INTO TABLE t1");
$db->set_local_infile_default();
$res = $db->query("SELECT * FROM t1");
echo "\nResult:\n";
while ($row = $res->fetch_assoc()) {
    echo join(",", $row)."\n";
}
?>
```

Procedural style

```
<?php
$db = mysqli_init();
mysqli_real_connect($db, "localhost", "root", "", "test");
function callme($stream, &$amp;buffer, $buflen, &$amp;errmsg)
{
    $buffer = fgets($stream);
    echo $buffer;
    // convert to upper case and replace "," delimiter with [TAB]
    $buffer = strtoupper(str_replace(",", "\t", $buffer));
    return strlen($buffer);
}
echo "Input:\n";
mysqli_set_local_infile_handler($db, "callme");
mysqli_query($db, "LOAD DATA LOCAL INFILE 'input.txt' INTO TABLE t1");
mysqli_set_local_infile_default($db);
$res = mysqli_query($db, "SELECT * FROM t1");
echo "\nResult:\n";
while ($row = mysqli_fetch_assoc($res)) {
    echo join(",", $row)."\n";
}
?>
```

The above examples will output:

```
Input:
23,foo
42,bar
Output:
23,F00
42,BAR
```

See Also

[mysqli_set_local_infile_default](#)

6.3.9.55 **mysqli::\$sqlstate, mysqli_sqlstate**

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$sqlstate](#)

[mysqli_sqlstate](#)

Returns the SQLSTATE error from previous MySQL operation

Description

Object oriented style

```
string  
mysqli->sqlstate ;
```

Procedural style

```
string mysqli_sqlstate(  
    mysqli link);
```

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. '00000' means no error. The values are specified by ANSI SQL and ODBC. For a list of possible values, see <http://dev.mysql.com/doc/mysql/en/error-handling.html>.

Note

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value **HY000** (general error) is used for unmapped errors.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. '00000' means no error.

Examples

Example 6.79 `$mysqli->sqlstate` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* Table City already exists, so we should get an error */  
if (!$mysqli->query("CREATE TABLE City (ID INT, Name VARCHAR(30))")) {  
    printf("Error - SQLSTATE %s.\n", $mysqli->sqlstate);  
}  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* Table City already exists, so we should get an error */
if (!mysqli_query($link, "CREATE TABLE City (ID INT, Name VARCHAR(30))")) {
    printf("Error - SQLSTATE %s.\n", mysqli_sqlstate($link));
}
mysqli_close($link);
?>
```

The above examples will output:

```
Error - SQLSTATE 42S01.
```

See Also

[mysqli_errno](#)

[mysqli_error](#)

6.3.9.56 [mysqli::ssl_set](#), [mysqli_ssl_set](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::ssl_set](#)

[mysqli_ssl_set](#)

Used for establishing secure connections using SSL

Description

Object oriented style

```
bool mysqli::ssl_set(
    string key,
    string cert,
    string ca,
    string capath,
    string cipher);
```

Procedural style

```
bool mysqli_ssl_set(
    mysqli link,
    string key,
    string cert,
    string ca,
    string capath,
    string cipher);
```

Used for establishing secure connections using SSL. It must be called before [mysqli_real_connect](#). This function does nothing unless OpenSSL support is enabled.

Note that MySQL Native Driver does not support SSL before PHP 5.3.3, so calling this function when using MySQL Native Driver will result in an error. MySQL Native Driver is enabled by default on Microsoft Windows from PHP version 5.3 onwards.

Parameters

<i>link</i>	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
<i>key</i>	The path name to the key file.
<i>cert</i>	The path name to the certificate file.
<i>ca</i>	The path name to the certificate authority file.
<i>capath</i>	The pathname to a directory that contains trusted SSL CA certificates in PEM format.
<i>cipher</i>	A list of allowable ciphers to use for SSL encryption.

Any unused SSL parameters may be given as [NULL](#)

Return Values

This function always returns [TRUE](#) value. If SSL setup is incorrect [mysqli_real_connect](#) will return an error when you attempt to connect.

See Also

[mysqli_options](#)
[mysqli_real_connect](#)

6.3.9.57 [mysqli::stat](#), [mysqli_stat](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::stat](#)

[mysqli_stat](#)

Gets the current system status

Description

Object oriented style

```
string mysqli::stat();
```

Procedural style

```
string mysqli_stat(  
    mysqli link);
```

[mysqli_stat](#) returns a string containing information similar to that provided by the 'mysqladmin status' command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

Return Values

A string describing the server status. [FALSE](#) if an error occurred.

Examples

Example 6.80 `mysqli::stat` example

Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
printf ("System status: %s\n", $mysqli->stat());
$mysqli->close();
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
printf("System status: %s\n", mysqli_stat($link));
mysqli_close($link);
?>
```

The above examples will output:

```
System status: Uptime: 272  Threads: 1  Questions: 5340  Slow queries: 0
Opens: 13  Flush tables: 1  Open tables: 0  Queries per second avg: 19.632
Memory in use: 8496K  Max memory used: 8560K
```

See Also

[mysqli_get_server_info](#)

6.3.9.58 `mysqli::stmt_init`, `mysqli_stmt_init`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::stmt_init](#)

[mysqli_stmt_init](#)

Initializes a statement and returns an object for use with `mysqli_stmt_prepare`

Description

Object oriented style

```
mysqli_stmt $mysqli::stmt_init();
```

Procedural style

```
mysqli_stmt mysqli_stmt_init(
    mysqli link);
```

Allocates and initializes a statement object suitable for `mysqli_stmt_prepare`.

Note

Any subsequent calls to any `mysqli_stmt` function will fail until `mysqli_stmt_prepare` was called.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns an object.

See Also

`mysqli_stmt_prepare`

6.3.9.59 `mysqli::store_result`, `mysqli_store_result`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::store_result`

`mysqli_store_result`

Transfers a result set from the last query

Description

Object oriented style

```
mysqli_result mysqli::store_result(
    int option);
```

Procedural style

```
mysqli_result mysqli_store_result(
    mysqli link,
    int option);
```

Transfers the result set from the last query on the database connection represented by the *link* parameter to be used with the `mysqli_data_seek` function.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

option The option that you want to set. It can be one of the following values:

Table 6.12 Valid options

Name	Description
<code>MYSQLI_STORE_RESULT_COPY_DATA</code>	Copy results from the internal mysqlnd buffer into the PHP

Name	Description
	variables fetched. By default, mysqlnd will use a reference logic to avoid copying and duplicating results held in memory. For certain result sets, for example, result sets with many small rows, the copy approach can reduce the overall memory usage because PHP variables holding results may be released earlier (available with mysqlnd only, since PHP 5.6.0)

Return Values

Returns a buffered result object or **FALSE** if an error occurred.

Note

`mysqli_store_result` returns **FALSE** in case the query didn't return a result set (if the query was, for example an INSERT statement). This function also returns **FALSE** if the reading of the result set failed. You can check if you have got an error by checking if `mysqli_error` doesn't return an empty string, if `mysqli_errno` returns a non zero value, or if `mysqli_field_count` returns a non zero value. Also possible reason for this function returning **FALSE** after successful call to `mysqli_query` can be too large result set (memory for it cannot be allocated). If `mysqli_field_count` returns a non-zero value, the statement should have produced a non-empty result set.

Notes

Note

Although it is always good practice to free the memory used by the result of a query using the `mysqli_free_result` function, when transferring large result sets using the `mysqli_store_result` this becomes particularly important.

Examples

See `mysqli_multi_query`.

See Also

`mysqli_real_query`
`mysqli_use_result`

6.3.9.60 `mysqli::$thread_id`, `mysqli_thread_id`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::$thread_id`

`mysqli_thread_id`

Returns the thread ID for the current connection

Description

Object oriented style

```
int  
mysqli->thread_id ;
```

Procedural style

```
int mysqli_thread_id(  
    mysqli link);
```

The `mysqli_thread_id` function returns the thread ID for the current connection which can then be killed using the `mysqli_kill` function. If the connection is lost and you reconnect with `mysqli_ping`, the thread ID will be other. Therefore you should get the thread ID only when you need it.

Note

The thread ID is assigned on a connection-by-connection basis. Hence, if the connection is broken and then re-established a new thread ID will be assigned.

To kill a running query you can use the SQL command `KILL QUERY processid`.

Parameters

link Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Returns the Thread ID for the current connection.

Examples

Example 6.81 `$mysqli->thread_id` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* determine our thread id */  
$thread_id = $mysqli->thread_id;  
/* Kill connection */  
$mysqli->kill($thread_id);  
/* This should produce an error */  
if (!$mysqli->query("CREATE TABLE myCity LIKE City")) {  
    printf("Error: %s\n", $mysqli->error);  
    exit;  
}  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* determine our thread id */
$thread_id = mysqli_thread_id($link);
/* Kill connection */
mysqli_kill($link, $thread_id);
/* This should produce an error */
if (!mysqli_query($link, "CREATE TABLE myCity LIKE City")) {
    printf("Error: %s\n", mysqli_error($link));
    exit;
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: MySQL server has gone away
```

See Also

[mysqli_kill](#)

6.3.9.61 [mysqli::thread_safe](#), [mysqli_thread_safe](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::thread_safe](#)

[mysqli_thread_safe](#)

Returns whether thread safety is given or not

Description

Procedural style

```
bool mysqli_thread_safe();
```

Tells whether the client library is compiled as thread-safe.

Return Values

TRUE if the client library is thread-safe, otherwise **FALSE**.

6.3.9.62 [mysqli::use_result](#), [mysqli_use_result](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::use_result](#)

[mysqli_use_result](#)

Initiate a result set retrieval

Description

Object oriented style

```
mysqli_result mysqli::use_result();
```

Procedural style

```
mysqli_result mysqli_use_result(  
    mysqli link);
```

Used to initiate the retrieval of a result set from the last query executed using the `mysqli_real_query` function on the database connection.

Either this or the `mysqli_store_result` function must be called before the results of a query can be retrieved, and one or the other must be called to prevent the next query on that database connection from failing.

Note

The `mysqli_use_result` function does not transfer the entire result set from the database and hence cannot be used functions such as `mysqli_data_seek` to move to a particular row within the set. To use this functionality, the result set must be stored using `mysqli_store_result`. One should not use `mysqli_use_result` if a lot of processing on the client side is performed, since this will tie up the server and prevent other threads from updating any tables from which the data is being fetched.

Return Values

Returns an unbuffered result object or `FALSE` if an error occurred.

Examples

Example 6.82 `mysqli::use_result` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
$query = "SELECT CURRENT_USER()";  
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";  
/* execute multi query */  
if ($mysqli->multi_query($query)) {  
    do {  
        /* store first result set */  
        if ($result = $mysqli->use_result()) {  
            while ($row = $result->fetch_row()) {  
                printf("%s\n", $row[0]);  
            }  
            $result->close();  
        }  
        /* print divider */  
        if ($mysqli->more_results()) {  
            printf("-----\n");  
        }  
    } while ($mysqli->next_result());  
}  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT CURRENT_USER();";
$query .= "SELECT Name FROM City ORDER BY ID LIMIT 20, 5";
/* execute multi query */
if (mysqli_multi_query($link, $query)) {
    do {
        /* store first result set */
        if ($result = mysqli_use_result($link)) {
            while ($row = mysqli_fetch_row($result)) {
                printf("%s\n", $row[0]);
            }
            mysqli_free_result($result);
        }
        /* print divider */
        if (mysqli_more_results($link)) {
            printf("-----\n");
        }
    } while (mysqli_next_result($link));
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
my_user@localhost
-----
Amersfoort
Maastricht
Dordrecht
Leiden
Haarlemmermeer
```

See Also

[mysqli_real_query](#)
[mysqli_store_result](#)

6.3.9.63 [mysqli::\\$warning_count](#), [mysqli_warning_count](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::\\$warning_count](#)

[mysqli_warning_count](#)

Returns the number of warnings from the last query for the given link

Description

Object oriented style

```
int  
mysqli->warning_count ;
```

Procedural style

```
int mysqli_warning_count(  
    mysqli link);
```

Returns the number of warnings from the last query in the connection.

Note

For retrieving warning messages you can use the SQL command `SHOW WARNINGS [limit row_count]`.

Parameters

link

Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`

Return Values

Number of warnings or zero if there are no warnings.

Examples

Example 6.83 `$mysqli->warning_count` example

Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
$mysqli->query("CREATE TABLE myCity LIKE City");  
/* a remarkable city in Wales */  
$query = "INSERT INTO myCity (CountryCode, Name) VALUES('GBR',  
    'Llanfairpwllgwyngyllgogerychwyrndrobwl'llantysiliogogogoch')";  
$mysqli->query($query);  
if ($mysqli->warning_count) {  
    if ($result = $mysqli->query("SHOW WARNINGS")) {  
        $row = $result->fetch_row();  
        printf("%s (%d): %s\n", $row[0], $row[1], $row[2]);  
        $result->close();  
    }  
}  
/* close connection */  
$mysqli->close();  
?>
```

Procedural style

```
<?php  
$link = mysqli_connect("localhost", "my_user", "my_password", "world");  
/* check connection */
```

```
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCity LIKE City");
/* a remarkable long city name in Wales */
$query = "INSERT INTO myCity (CountryCode, Name) VALUES('GBR',
    'Llanfairpwllgwyngyllgogerychwyrndrobwlllllantysiliogogogoch')";
mysqli_query($link, $query);
if (mysqli_warning_count($link)) {
    if ($result = mysqli_query($link, "SHOW WARNINGS")) {
        $row = mysqli_fetch_row($result);
        printf("%s (%d): %s\n", $row[0], $row[1], $row[2]);
        mysqli_free_result($result);
    }
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Warning (1264): Data truncated for column 'Name' at row 1
```

See Also

[mysqli_errno](#)
[mysqli_error](#)
[mysqli_sqlstate](#)

6.3.10 The mysqli_stmt class

Copyright 1997-2014 the PHP Documentation Group.

Represents a prepared statement.

```
mysqli_stmt {
    mysqli_stmt
        Properties

    int
        mysqli_stmt->affected_rows ;

    int
        mysqli_stmt->errno ;

    array
        mysqli_stmt->error_list ;

    string
        mysqli_stmt->error ;

    int
        mysqli_stmt->field_count ;

    int
        mysqli_stmt->insert_id ;

    int
        mysqli_stmt->num_rows ;
```

```
int
    mysqli_stmt->param_count ;

string
    mysqli_stmt->sqlstate ;

Methods

mysqli_stmt::__construct(
    mysqli link,
    string query);

int mysqli_stmt::attr_get(
    int attr);

bool mysqli_stmt::attr_set(
    int attr,
    int mode);

bool mysqli_stmt::bind_param(
    string types,
    mixed var1,
    mixed ...);

bool mysqli_stmt::bind_result(
    mixed var1,
    mixed ...);

bool mysqli_stmt::close();

void mysqli_stmt::data_seek(
    int offset);

bool mysqli_stmt::execute();

bool mysqli_stmt::fetch();

void mysqli_stmt::free_result();

mysqli_result mysqli_stmt::get_result();

object mysqli_stmt::get_warnings(
    mysqli_stmt stmt);

mixed mysqli_stmt::prepare(
    string query);

bool mysqli_stmt::reset();

mysqli_result mysqli_stmt::result_metadata();

bool mysqli_stmt::send_long_data(
    int param_nr,
    string data);

bool mysqli_stmt::store_result();
}
```

6.3.10.1 `mysqli_stmt::$affected_rows`, `mysqli_stmt_affected_rows`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::$affected_rows`

`mysqli_stmt_affected_rows`

Returns the total number of rows changed, deleted, or inserted by the last executed statement

Description

Object oriented style

```
int  
mysqli_stmt->affected_rows ;
```

Procedural style

```
int mysqli_stmt_affected_rows(  
    mysqli_stmt stmt);
```

Returns the number of rows affected by [INSERT](#), [UPDATE](#), or [DELETE](#) query.

This function only works with queries which update a table. In order to get the number of rows from a [SELECT](#) query, use [mysqli_stmt_num_rows](#) instead.

Parameters

[stmt](#) Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an [UPDATE](#)/[DELETE](#) statement, no rows matched the [WHERE](#) clause in the query or that no query has yet been executed. -1 indicates that the query has returned an error. NULL indicates an invalid argument was supplied to the function.

Note

If the number of affected rows is greater than maximal PHP int value, the number of affected rows will be returned as a string value.

Examples

Example 6.84 Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* create temp table */  
$mysqli->query("CREATE TEMPORARY TABLE myCountry LIKE Country");  
$query = "INSERT INTO myCountry SELECT * FROM Country WHERE Code LIKE ?";  
/* prepare statement */  
if ($stmt = $mysqli->prepare($query)) {  
    /* Bind variable for placeholder */  
    $code = 'A%';  
    $stmt->bind_param("s", $code);  
    /* execute statement */  
    $stmt->execute();  
    printf("rows inserted: %d\n", $stmt->affected_rows);  
    /* close statement */  
    $stmt->close();  
}  
/* close connection */  
$mysqli->close();  
?>
```

Example 6.85 Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* create temp table */
mysqli_query($link, "CREATE TEMPORARY TABLE myCountry LIKE Country");
$query = "INSERT INTO myCountry SELECT * FROM Country WHERE Code LIKE ?";
/* prepare statement */
if ($stmt = mysqli_prepare($link, $query)) {
    /* Bind variable for placeholder */
    $code = 'A%';
    mysqli_stmt_bind_param($stmt, "s", $code);
    /* execute statement */
    mysqli_stmt_execute($stmt);
    printf("rows inserted: %d\n", mysqli_stmt_affected_rows($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```
rows inserted: 17
```

See Also

[mysqli_stmt_num_rows](#)
[mysqli_prepare](#)

6.3.10.2 [mysqli_stmt::attr_get](#), [mysqli_stmt_attr_get](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::attr_get](#)
[mysqli_stmt_attr_get](#)

Used to get the current value of a statement attribute

Description

Object oriented style

```
int mysqli_stmt::attr_get(
    int attr);
```

Procedural style

```
int mysqli_stmt_attr_get(
    mysqli_stmt stmt,
    int attr);
```

Gets the current value of a statement attribute.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

attr The attribute that you want to get.

Return Values

Returns `FALSE` if the attribute is not found, otherwise returns the value of the attribute.

6.3.10.3 `mysqli_stmt::attr_set`, `mysqli_stmt_attr_set`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::attr_set`

`mysqli_stmt_attr_set`

Used to modify the behavior of a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::attr_set(  
    int attr,  
    int mode);
```

Procedural style

```
bool mysqli_stmt_attr_set(  
    mysqli_stmt stmt,  
    int attr,  
    int mode);
```

Used to modify the behavior of a prepared statement. This function may be called multiple times to set several attributes.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

attr The attribute that you want to set. It can have one of the following values:

Table 6.13 Attribute values

Character	Description
<code>MYSQLI_STMT_ATTR_UPDATE_MAX_LENGTH</code>	Setting <code>UPDATE_MAX_LENGTH</code> causes <code>mysqli_stmt_store_result</code> to update the metadata <code>MYSQL_FIELD->max_length</code> value.
<code>MYSQLI_STMT_ATTR_CURSOR_TYPE</code>	Specifies cursor to open for statement when <code>mysqli_stmt_execute</code> is invoked. <i>mode</i> can be <code>MYSQLI_CURSOR_TYPE_NO_CURSOR</code>

Character	Description
	(the default) or <code>MYSQLI_CURSOR_TYPE_READ_ONLY</code> .
<code>MYSQLI_STMT_ATTR_PREFETCH_ROWS</code>	Number of rows to fetch from server at a time when using a cursor. <i>mode</i> can be in the range from 1 to the maximum value of unsigned long. The default is 1.

If you use the `MYSQLI_STMT_ATTR_CURSOR_TYPE` option with `MYSQLI_CURSOR_TYPE_READ_ONLY`, a cursor is opened for the statement when you invoke `mysqli_stmt_execute`. If there is already an open cursor from a previous `mysqli_stmt_execute` call, it closes the cursor before opening a new one. `mysqli_stmt_reset` also closes any open cursor before preparing the statement for re-execution. `mysqli_stmt_free_result` closes any open cursor.

If you open a cursor for a prepared statement, `mysqli_stmt_store_result` is unnecessary.

mode

The value to assign to the attribute.

See Also

[Connector/MySQL `mysqli\_stmt\_attr\_set\(\)`](#)

6.3.10.4 `mysqli_stmt::bind_param`, `mysqli_stmt_bind_param`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::bind_param`

`mysqli_stmt_bind_param`

Binds variables to a prepared statement as parameters

Description

Object oriented style

```
bool mysqli_stmt::bind_param(
    string types,
    mixed var1,
    mixed ...);
```

Procedural style

```
bool mysqli_stmt_bind_param(
    mysqli_stmt stmt,
    string types,
    mixed var1,
    mixed ...);
```

Bind variables for the parameter markers in the SQL statement that was passed to `mysqli_prepare`.

Note

If data size of a variable exceeds max. allowed packet size (`max_allowed_packet`), you have to specify `b` in *types* and use `mysqli_stmt_send_long_data` to send the data in packets.

Note

Care must be taken when using `mysqli_stmt_bind_param` in conjunction with `call_user_func_array`. Note that `mysqli_stmt_bind_param` requires parameters to be passed by reference, whereas `call_user_func_array` can accept as a parameter a list of variables that can represent references or values.

Parameters*stmt*

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

types

A string that contains one or more characters which specify the types for the corresponding bind variables:

Table 6.14 Type specification chars

Character	Description
i	corresponding variable has type integer
d	corresponding variable has type double
s	corresponding variable has type string
b	corresponding variable is a blob and will be sent in packets

var1

The number of variables and length of string *types* must match the parameters in the statement.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples**Example 6.86 Object oriented style**

```
<?php
$mysqli = new mysqli('localhost', 'my_user', 'my_password', 'world');
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$stmt = $mysqli->prepare("INSERT INTO CountryLanguage VALUES (?, ?, ?, ?)");
$stmt->bind_param('sssd', $code, $language, $official, $percent);
$code = 'DEU';
$language = 'Bavarian';
$official = "F";
$percent = 11.2;
/* execute prepared statement */
$stmt->execute();
printf("%d Row inserted.\n", $stmt->affected_rows);
/* close statement and connection */
$stmt->close();
/* Clean up table CountryLanguage */
$mysqli->query("DELETE FROM CountryLanguage WHERE Language='Bavarian'");
printf("%d Row deleted.\n", $mysqli->affected_rows);
/* close connection */
```

```
$mysqli->close();  
?>
```

Example 6.87 Procedural style

```
<?php  
$link = mysqli_connect('localhost', 'my_user', 'my_password', 'world');  
/* check connection */  
if (!$link) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
$stmt = mysqli_prepare($link, "INSERT INTO CountryLanguage VALUES (?, ?, ?, ?)");  
mysqli_stmt_bind_param($stmt, 'sssd', $code, $language, $official, $percent);  
$code = 'DEU';  
$language = 'Bavarian';  
$official = "F";  
$percent = 11.2;  
/* execute prepared statement */  
mysqli_stmt_execute($stmt);  
printf("%d Row inserted.\n", mysqli_stmt_affected_rows($stmt));  
/* close statement and connection */  
mysqli_stmt_close($stmt);  
/* Clean up table CountryLanguage */  
mysqli_query($link, "DELETE FROM CountryLanguage WHERE Language='Bavarian'");  
printf("%d Row deleted.\n", mysqli_affected_rows($link));  
/* close connection */  
mysqli_close($link);  
?>
```

The above examples will output:

```
1 Row inserted.  
1 Row deleted.
```

See Also

[mysqli_stmt_bind_result](#)
[mysqli_stmt_execute](#)
[mysqli_stmt_fetch](#)
[mysqli_prepare](#)
[mysqli_stmt_send_long_data](#)
[mysqli_stmt_errno](#)
[mysqli_stmt_error](#)

6.3.10.5 [mysqli_stmt::bind_result](#), [mysqli_stmt_bind_result](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::bind_result](#)

[mysqli_stmt_bind_result](#)

Binds variables to a prepared statement for result storage

Description

Object oriented style

```
bool mysqli_stmt::bind_result(  
    mixed var1,  
    mixed ...);
```

Procedural style

```
bool mysqli_stmt_bind_result(  
    mysqli_stmt stmt,  
    mixed var1,  
    mixed ...);
```

Binds columns in the result set to variables.

When `mysqli_stmt_fetch` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified variables `var1`, ...

Note

Note that all columns must be bound after `mysqli_stmt_execute` and prior to calling `mysqli_stmt_fetch`. Depending on column types bound variables can silently change to the corresponding PHP type.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysqli_stmt_fetch` is called.

Parameters

`stmt` Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

`var1` The variable to be bound.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 6.88 Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
/* prepare statement */  
if ($stmt = $mysqli->prepare("SELECT Code, Name FROM Country ORDER BY Name LIMIT 5")) {  
    $stmt->execute();  
    /* bind variables to prepared statement */  
    $stmt->bind_result($col1, $col2);  
    /* fetch values */  
    while ($stmt->fetch()) {  
        printf("%s %s\n", $col1, $col2);  
    }  
    /* close statement */  
    $stmt->close();  
}  
/* close connection */  
$mysqli->close();  
?>
```

Example 6.89 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* prepare statement */
if ($stmt = mysqli_prepare($link, "SELECT Code, Name FROM Country ORDER BY Name LIMIT 5")) {
    mysqli_stmt_execute($stmt);
    /* bind variables to prepared statement */
    mysqli_stmt_bind_result($stmt, $col1, $col2);
    /* fetch values */
    while (mysqli_stmt_fetch($stmt)) {
        printf("%s %s\n", $col1, $col2);
    }
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
AFG Afghanistan
ALB Albania
DZA Algeria
ASM American Samoa
AND Andorra
```

See Also

[mysqli_stmt_get_result](#)
[mysqli_stmt_bind_param](#)
[mysqli_stmt_execute](#)
[mysqli_stmt_fetch](#)
[mysqli_prepare](#)
[mysqli_stmt_prepare](#)
[mysqli_stmt_init](#)
[mysqli_stmt_errno](#)
[mysqli_stmt_error](#)

6.3.10.6 [mysqli_stmt::close](#), [mysqli_stmt_close](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::close](#)

[mysqli_stmt_close](#)

Closes a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::close();
```

Procedural style

```
bool mysqli_stmt_close(  
    mysqli_stmt stmt);
```

Closes a prepared statement. [mysqli_stmt_close](#) also deallocates the statement handle. If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Parameters

stmt Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

See Also

[mysqli_prepare](#)

6.3.10.7 [mysqli_stmt::__construct](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::__construct](#)

Constructs a new [mysqli_stmt](#) object

Description

```
mysqli_stmt::__construct(  
    mysqli link,  
    string query);
```

This method constructs a new [mysqli_stmt](#) object.

Note

In general, you should use either [mysqli_prepare](#) or [mysqli_stmt_init](#) to create a [mysqli_stmt](#) object, rather than directly instantiating the object with `new mysqli_stmt`. This method (and the ability to directly instantiate [mysqli_stmt](#) objects) may be deprecated and removed in the future.

Parameters

link Procedural style only: A link identifier returned by [mysqli_connect](#) or [mysqli_init](#)

query The query, as a string. If this parameter is omitted, then the constructor behaves identically to [mysqli_stmt_init](#), if provided, then it behaves as per [mysqli_prepare](#).

See Also

[mysqli_prepare](#)

[mysqli_stmt_init](#)

6.3.10.8 [mysqli_stmt::data_seek](#), [mysqli_stmt_data_seek](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::data_seek](#)

[mysqli_stmt_data_seek](#)

Seeks to an arbitrary row in statement result set

Description

Object oriented style

```
void mysqli_stmt::data_seek(  
    int offset);
```

Procedural style

```
void mysqli_stmt_data_seek(  
    mysqli_stmt stmt,  
    int offset);
```

Seeks to an arbitrary result pointer in the statement result set.

[mysqli_stmt_store_result](#) must be called prior to [mysqli_stmt_data_seek](#).

Parameters

stmt

Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

offset

Must be between zero and the total number of rows minus one (0.. [mysqli_stmt_num_rows](#) - 1).

Return Values

No value is returned.

Examples

Example 6.90 Object oriented style

```
<?php  
/* Open a connection */  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";  
if ($stmt = $mysqli->prepare($query)) {  
    /* execute query */  
    $stmt->execute();  
    /* bind result variables */  
    $stmt->bind_result($name, $code);  
    /* store result */  
    $stmt->store_result();  
    /* seek to row no. 400 */  
    $stmt->data_seek(399);
```

```
        /* fetch values */
        $stmt->fetch();
        printf ("City: %s Countrycode: %s\n", $name, $code);
        /* close statement */
        $stmt->close();
    }
    /* close connection */
    $mysqli->close();
?>
```

Example 6.91 Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $name, $code);
    /* store result */
    mysqli_stmt_store_result($stmt);
    /* seek to row no. 400 */
    mysqli_stmt_data_seek($stmt, 399);
    /* fetch values */
    mysqli_stmt_fetch($stmt);
    printf ("City: %s Countrycode: %s\n", $name, $code);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
City: Benin City Countrycode: NGA
```

See Also

[mysqli_prepare](#)

6.3.10.9 [mysqli_stmt::\\$errno](#), [mysqli_stmt_errno](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::\\$errno](#)

[mysqli_stmt_errno](#)

Returns the error code for the most recent statement call

Description

Object oriented style

```
int
mysqli_stmt->errno ;
```

Procedural style

```
int mysqli_stmt_errno(
    mysqli_stmt stmt);
```

Returns the error code for the most recently invoked statement function that can succeed or fail.

Client error message numbers are listed in the MySQL [errmsg.h](#) header file, server error message numbers are listed in [mysqld_error.h](#). In the MySQL source distribution you can find a complete list of error messages and error numbers in the file [Docs/mysqld_error.txt](#).

Parameters

stmt Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

An error code value. Zero means no error occurred.

Examples

Example 6.92 Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {
    /* drop table */
    $mysqli->query("DROP TABLE myCountry");
    /* execute query */
    $stmt->execute();
    printf("Error: %d.\n", $stmt->errno);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.93 Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
```

```
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");
    /* execute query */
    mysqli_stmt_execute($stmt);
    printf("Error: %d.\n", mysqli_stmt_errno($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Error: 1146.
```

See Also

[mysqli_stmt_error](#)
[mysqli_stmt_sqlstate](#)

6.3.10.10 [mysqli_stmt::\\$error_list](#), [mysqli_stmt_error_list](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::\\$error_list](#)

[mysqli_stmt_error_list](#)

Returns a list of errors from the last statement executed

Description

Object oriented style

```
array
mysqli_stmt->error_list ;
```

Procedural style

```
array mysqli_stmt_error_list(
    mysqli_stmt stmt);
```

Returns an array of errors for the most recently invoked statement function that can succeed or fail.

Parameters

[stmt](#) Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

A list of errors, each as an associative array containing the errno, error, and sqlstate.

Examples

Example 6.94 Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {
    /* drop table */
    $mysqli->query("DROP TABLE myCountry");
    /* execute query */
    $stmt->execute();

    echo "Error:\n";
    print_r($stmt->error_list);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.95 Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");
    /* execute query */
    mysqli_stmt_execute($stmt);

    echo "Error:\n";
    print_r(mysqli_stmt_error_list($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Array
(
    [0] => Array
        (
            [errno] => 1146
            [sqlstate] => 42S02
            [error] => Table 'world.myCountry' doesn't exist
        )
)
```

See Also

[mysqli_stmt_error](#)
[mysqli_stmt_errno](#)
[mysqli_stmt_sqlstate](#)

6.3.10.11 [mysqli_stmt::\\$error](#), [mysqli_stmt_error](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::\\$error](#)

[mysqli_stmt_error](#)

Returns a string description for last statement error

Description

Object oriented style

```
string
mysqli_stmt->error ;
```

Procedural style

```
string mysqli_stmt_error(
    mysqli_stmt stmt);
```

Returns a string containing the error message for the most recently invoked statement function that can succeed or fail.

Parameters

[stmt](#) Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

A string that describes the error. An empty string if no error occurred.

Examples**Example 6.96 Object oriented style**

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
```

```

}
$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {
    /* drop table */
    $mysqli->query("DROP TABLE myCountry");
    /* execute query */
    $stmt->execute();
    printf("Error: %s.\n", $stmt->error);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>

```

Example 6.97 Procedural style

```

<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");
    /* execute query */
    mysqli_stmt_execute($stmt);
    printf("Error: %s.\n", mysqli_stmt_error($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```
Error: Table 'world.myCountry' doesn't exist.
```

See Also

[mysqli_stmt_errno](#)
[mysqli_stmt_sqlstate](#)

6.3.10.12 [mysqli_stmt::execute](#), [mysqli_stmt_execute](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::execute](#)
[mysqli_stmt_execute](#)

Executes a prepared Query

Description

Object oriented style

```
bool mysqli_stmt::execute();
```

Procedural style

```
bool mysqli_stmt_execute(  
    mysqli_stmt stmt);
```

Executes a query that has been previously prepared using the [mysqli_prepare](#) function. When executed any parameter markers which exist will automatically be replaced with the appropriate data.

If the statement is [UPDATE](#), [DELETE](#), or [INSERT](#), the total number of affected rows can be determined by using the [mysqli_stmt_affected_rows](#) function. Likewise, if the query yields a result set the [mysqli_stmt_fetch](#) function is used.

Note

When using [mysqli_stmt_execute](#), the [mysqli_stmt_fetch](#) function must be used to fetch the data prior to performing any additional queries.

Parameters

stmt

Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 6.98 Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
$mysqli->query("CREATE TABLE myCity LIKE City");  
/* Prepare an insert statement */  
$query = "INSERT INTO myCity (Name, CountryCode, District) VALUES (?, ?, ?)";  
$stmt = $mysqli->prepare($query);  
$stmt->bind_param("sss", $val1, $val2, $val3);  
$val1 = 'Stuttgart';  
$val2 = 'DEU';  
$val3 = 'Baden-Wuerttemberg';  
/* Execute the statement */  
$stmt->execute();  
$val1 = 'Bordeaux';  
$val2 = 'FRA';  
$val3 = 'Aquitaine';  
/* Execute the statement */  
$stmt->execute();  
/* close statement */  
$stmt->close();  
/* retrieve all rows from myCity */
```

```

$query = "SELECT Name, CountryCode, District FROM myCity";
if ($result = $mysqli->query($query)) {
    while ($row = $result->fetch_row()) {
        printf("%s (%s,%s)\n", $row[0], $row[1], $row[2]);
    }
    /* free result set */
    $result->close();
}
/* remove table */
$mysqli->query("DROP TABLE myCity");
/* close connection */
$mysqli->close();
?>

```

Example 6.99 Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCity LIKE City");
/* Prepare an insert statement */
$query = "INSERT INTO myCity (Name, CountryCode, District) VALUES (?, ?, ?)";
$stmt = mysqli_prepare($link, $query);
mysqli_stmt_bind_param($stmt, "sss", $val1, $val2, $val3);
$val1 = 'Stuttgart';
$val2 = 'DEU';
$val3 = 'Baden-Wuerttemberg';
/* Execute the statement */
mysqli_stmt_execute($stmt);
$val1 = 'Bordeaux';
$val2 = 'FRA';
$val3 = 'Aquitaine';
/* Execute the statement */
mysqli_stmt_execute($stmt);
/* close statement */
mysqli_stmt_close($stmt);
/* retrieve all rows from myCity */
$query = "SELECT Name, CountryCode, District FROM myCity";
if ($result = mysqli_query($link, $query)) {
    while ($row = mysqli_fetch_row($result)) {
        printf("%s (%s,%s)\n", $row[0], $row[1], $row[2]);
    }
    /* free result set */
    mysqli_free_result($result);
}
/* remove table */
mysqli_query($link, "DROP TABLE myCity");
/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```

Stuttgart (DEU,Baden-Wuerttemberg)
Bordeaux (FRA,Aquitaine)

```

See Also

```
mysqli_prepare  
mysqli_stmt_bind_param  
mysqli_stmt_get_result
```

6.3.10.13 `mysqli_stmt::fetch`, `mysqli_stmt_fetch`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::fetch`

`mysqli_stmt_fetch`

Fetch results from a prepared statement into the bound variables

Description

Object oriented style

```
bool mysqli_stmt::fetch();
```

Procedural style

```
bool mysqli_stmt_fetch(  
    mysqli_stmt stmt);
```

Fetch the result from a prepared statement into the variables bound by `mysqli_stmt_bind_result`.

Note

Note that all columns must be bound by the application before calling `mysqli_stmt_fetch`.

Note

Data are transferred unbuffered without calling `mysqli_stmt_store_result` which can decrease performance (but reduces memory cost).

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Table 6.15 Return Values

Value	Description
<code>TRUE</code>	Success. Data has been fetched
<code>FALSE</code>	Error occurred
<code>NULL</code>	No more rows/data exists or data truncation occurred

Examples

Example 6.100 Object oriented style

```
<?php
```

```
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 150,5";
if ($stmt = $mysqli->prepare($query)) {
    /* execute statement */
    $stmt->execute();
    /* bind result variables */
    $stmt->bind_result($name, $code);
    /* fetch values */
    while ($stmt->fetch()) {
        printf ("%s (%s)\n", $name, $code);
    }
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.101 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 150,5";
if ($stmt = mysqli_prepare($link, $query)) {
    /* execute statement */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $name, $code);
    /* fetch values */
    while (mysqli_stmt_fetch($stmt)) {
        printf ("%s (%s)\n", $name, $code);
    }
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Rockford (USA)
Tallahassee (USA)
Salinas (USA)
Santa Clarita (USA)
Springfield (USA)
```

See Also

[mysqli_prepare](#)

```
mysqli_stmt_errno  
mysqli_stmt_error  
mysqli_stmt_bind_result
```

6.3.10.14 `mysqli_stmt::$field_count`, `mysqli_stmt_field_count`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::$field_count`

`mysqli_stmt_field_count`

Returns the number of field in the given statement

Description

Object oriented style

```
int  
mysqli_stmt->field_count ;
```

Procedural style

```
int mysqli_stmt_field_count(  
mysqli_stmt stmt);
```

Warning

This function is currently not documented; only its argument list is available.

6.3.10.15 `mysqli_stmt::$free_result`, `mysqli_stmt_free_result`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::$free_result`

`mysqli_stmt_free_result`

Frees stored result memory for the given statement handle

Description

Object oriented style

```
void mysqli_stmt::$free_result();
```

Procedural style

```
void mysqli_stmt_free_result(  
mysqli_stmt stmt);
```

Frees the result memory associated with the statement, which was allocated by `mysqli_stmt_store_result`.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

No value is returned.

See Also[mysqli_stmt_store_result](#)**6.3.10.16 mysqli_stmt::get_result, mysqli_stmt_get_result**

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::get_result](#)

[mysqli_stmt_get_result](#)

Gets a result set from a prepared statement

Description

Object oriented style

```
mysqli_result mysqli_stmt::get_result();
```

Procedural style

```
mysqli_result mysqli_stmt_get_result(  
    mysqli_stmt stmt);
```

Call to return a result set from a prepared statement query.

Parameters

<i>stmt</i>	Procedural style only: A statement identifier returned by mysqli_stmt_init .

Return Values

Returns a resultset for successful SELECT queries, or [FALSE](#) for other DML queries or on failure. The [mysqli_errno](#) function can be used to distinguish between the two types of failure.

MySQL Native Driver Only

Available only with [mysqlnd](#).

Examples**Example 6.102 Object oriented style**

```
<?php  
$mysqli = new mysqli("127.0.0.1", "user", "password", "world");  
if($mysqli->connect_error)  
{  
    die("$mysqli->connect_errno: $mysqli->connect_error");  
}  
$query = "SELECT Name, Population, Continent FROM Country WHERE Continent=? ORDER BY Name LIMIT 1";  
$stmt = $mysqli->stmt_init();  
if(!$stmt->prepare($query))  
{  
    print "Failed to prepare statement\n";  
}  
else  
{  
    $stmt->bind_param("s", $continent);  
    $continent_array = array('Europe', 'Africa', 'Asia', 'North America');  
    foreach($continent_array as $continent)
```

```
{
    $stmt->execute();
    $result = $stmt->get_result();
    while ($row = $result->fetch_array(MYSQLI_NUM))
    {
        foreach ($row as $r)
        {
            print "$r ";
        }
        print "\n";
    }
}
$stmt->close();
$mysqli->close();
?>
```

Example 6.103 Procedural style

```
<?php
$link = mysqli_connect("127.0.0.1", "user", "password", "world");
if (!$link)
{
    $error = mysqli_connect_error();
    $errno = mysqli_connect_errno();
    print "errno: $error\n";
    exit();
}
$query = "SELECT Name, Population, Continent FROM Country WHERE Continent=? ORDER BY Name LIMIT 1";
$stmt = mysqli_stmt_init($link);
if(!mysqli_stmt_prepare($stmt, $query))
{
    print "Failed to prepare statement\n";
}
else
{
    mysqli_stmt_bind_param($stmt, "s", $continent);
    $continent_array = array('Europe', 'Africa', 'Asia', 'North America');
    foreach($continent_array as $continent)
    {
        mysqli_stmt_execute($stmt);
        $result = mysqli_stmt_get_result($stmt);
        while ($row = mysqli_fetch_array($result, MYSQLI_NUM))
        {
            foreach ($row as $r)
            {
                print "$r ";
            }
            print "\n";
        }
    }
}
mysqli_stmt_close($stmt);
mysqli_close($link);
?>
```

The above examples will output:

```
Albania 3401200 Europe
Algeria 31471000 Africa
Afghanistan 22720000 Asia
Anguilla 8000 North America
```

See Also

`mysqli_prepare`
`mysqli_stmt_result_metadata`
`mysqli_stmt_fetch`
`mysqli_fetch_array`
`mysqli_stmt_store_result`
`mysqli_errno`

6.3.10.17 `mysqli_stmt::get_warnings`, `mysqli_stmt_get_warnings`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::get_warnings`

`mysqli_stmt_get_warnings`

Get result of SHOW WARNINGS

Description

Object oriented style

```
object mysqli_stmt::get_warnings(  
    mysqli_stmt stmt);
```

Procedural style

```
object mysqli_stmt_get_warnings(  
    mysqli_stmt stmt);
```

Warning

This function is currently not documented; only its argument list is available.

6.3.10.18 `mysqli_stmt::$insert_id`, `mysqli_stmt_insert_id`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::$insert_id`

`mysqli_stmt_insert_id`

Get the ID generated from the previous INSERT operation

Description

Object oriented style

```
int  
mysqli_stmt->insert_id ;
```

Procedural style

```
mixed mysqli_stmt_insert_id(  
    mysqli_stmt stmt);
```

Warning

This function is currently not documented; only its argument list is available.

6.3.10.19 `mysqli_stmt::more_results`, `mysqli_stmt_more_results`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::more_results`

`mysqli_stmt_more_results`

Check if there are more query results from a multiple query

Description

Object oriented style (method):

```
public bool mysqli_stmt::more_results();
```

Procedural style:

```
bool mysqli_stmt_more_results(  
    mysqli_stmt stmt);
```

Checks if there are more query results from a multiple query.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns `TRUE` if more results exist, otherwise `FALSE`.

MySQL Native Driver Only

Available only with `mysqlnd`.

See Also

`mysqli_stmt::next_result`
`mysqli::multi_query`

6.3.10.20 `mysqli_stmt::next_result`, `mysqli_stmt_next_result`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::next_result`

`mysqli_stmt_next_result`

Reads the next result from a multiple query

Description

Object oriented style (method):

```
public bool mysqli_stmt::next_result();
```

Procedural style:

```
bool mysqli_stmt_next_result(  
    mysqli_stmt stmt);
```

Reads the next result from a multiple query.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Errors/Exceptions

Emits an `E_STRICT` level error if a result set does not exist, and suggests using `mysqli_stmt::more_results` in these cases, before calling `mysqli_stmt::next_result`.

MySQL Native Driver Only

Available only with `mysqlnd`.

See Also

`mysqli_stmt::more_results`
`mysqli::multi_query`

6.3.10.21 `mysqli_stmt::$num_rows`, `mysqli_stmt_num_rows`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::$num_rows`
`mysqli_stmt_num_rows`

Return the number of rows in statements result set

Description

Object oriented style

```
int  
mysqli_stmt->num_rows ;
```

Procedural style

```
int mysqli_stmt_num_rows(  
    mysqli_stmt stmt);
```

Returns the number of rows in the result set. The use of `mysqli_stmt_num_rows` depends on whether or not you used `mysqli_stmt_store_result` to buffer the entire result set in the statement handle.

If you use `mysqli_stmt_store_result`, `mysqli_stmt_num_rows` may be called immediately.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

An integer representing the number of rows in result set.

Examples

Example 6.104 Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = $mysqli->prepare($query)) {
    /* execute query */
    $stmt->execute();
    /* store result */
    $stmt->store_result();
    printf("Number of rows: %d.\n", $stmt->num_rows);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.105 Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = mysqli_prepare($link, $query)) {
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* store result */
    mysqli_stmt_store_result($stmt);
    printf("Number of rows: %d.\n", mysqli_stmt_num_rows($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Number of rows: 20.
```

See Also

[mysqli_stmt_affected_rows](#)

`mysqli_prepare`
`mysqli_stmt_store_result`

6.3.10.22 `mysqli_stmt::$param_count`, `mysqli_stmt_param_count`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::$param_count`

`mysqli_stmt_param_count`

Returns the number of parameter for the given statement

Description

Object oriented style

```
int  
mysqli_stmt->param_count ;
```

Procedural style

```
int mysqli_stmt_param_count(  
    mysqli_stmt stmt);
```

Returns the number of parameter markers present in the prepared statement.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns an integer representing the number of parameters.

Examples

Example 6.106 Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");  
/* check connection */  
if (mysqli_connect_errno()) {  
    printf("Connect failed: %s\n", mysqli_connect_error());  
    exit();  
}  
if ($stmt = $mysqli->prepare("SELECT Name FROM Country WHERE Name=? OR Code=?")) {  
    $marker = $stmt->param_count;  
    printf("Statement has %d markers.\n", $marker);  
    /* close statement */  
    $stmt->close();  
}  
/* close connection */  
$mysqli->close();  
?>
```

Example 6.107 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($stmt = mysqli_prepare($link, "SELECT Name FROM Country WHERE Name=? OR Code=?")) {
    $marker = mysqli_stmt_param_count($stmt);
    printf("Statement has %d markers.\n", $marker);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Statement has 2 markers.
```

See Also

[mysqli_prepare](#)

6.3.10.23 [mysqli_stmt::prepare](#), [mysqli_stmt_prepare](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::prepare](#)

[mysqli_stmt_prepare](#)

Prepare an SQL statement for execution

Description

Object oriented style

```
mixed mysqli_stmt::prepare(
    string query);
```

Procedural style

```
bool mysqli_stmt_prepare(
    mysqli_stmt stmt,
    string query);
```

Prepares the SQL query pointed to by the null-terminated string query.

The parameter markers must be bound to application variables using [mysqli_stmt_bind_param](#) and/or [mysqli_stmt_bind_result](#) before executing the statement or fetching rows.

Note

In the case where you pass a statement to [mysqli_stmt_prepare](#) that is longer than [max_allowed_packet](#) of the server, the returned error codes are different depending on whether you are using MySQL Native Driver ([mysqlnd](#)) or MySQL Client Library ([libmysqlclient](#)). The behavior is as follows:

- `mysqlnd` on Linux returns an error code of 1153. The error message means “got a packet bigger than `max_allowed_packet` bytes”.
- `mysqlnd` on Windows returns an error code 2006. This error message means “server has gone away”.
- `libmysqlclient` on all platforms returns an error code 2006. This error message means “server has gone away”.

Parameters

stmt

Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

query

The query, as a string. It must consist of a single SQL statement.

You can include one or more parameter markers in the SQL statement by embedding question mark (?) characters at the appropriate positions.

Note

You should not add a terminating semicolon or `\g` to the statement.

Note

The markers are legal only in certain places in SQL statements. For example, they are allowed in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value.

However, they are not allowed for identifiers (such as table or column names), in the select list that names the columns to be returned by a `SELECT` statement), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 6.108 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
```

```

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$city = "Amersfoort";
/* create a prepared statement */
$stmt = $mysqli->stmt_init();
if ($stmt->prepare("SELECT District FROM City WHERE Name=?")) {
    /* bind parameters for markers */
    $stmt->bind_param("s", $city);
    /* execute query */
    $stmt->execute();
    /* bind result variables */
    $stmt->bind_result($district);
    /* fetch value */
    $stmt->fetch();
    printf("%s is in district %s\n", $city, $district);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>

```

Example 6.109 Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$city = "Amersfoort";
/* create a prepared statement */
$stmt = mysqli_stmt_init($link);
if (mysqli_stmt_prepare($stmt, 'SELECT District FROM City WHERE Name=?')) {
    /* bind parameters for markers */
    mysqli_stmt_bind_param($stmt, "s", $city);
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $district);
    /* fetch value */
    mysqli_stmt_fetch($stmt);
    printf("%s is in district %s\n", $city, $district);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```
Amersfoort is in district Utrecht
```

See Also

[mysqli_stmt_init](#)

```
mysqli_stmt_execute  
mysqli_stmt_fetch  
mysqli_stmt_bind_param  
mysqli_stmt_bind_result  
mysqli_stmt_get_result  
mysqli_stmt_close
```

6.3.10.24 `mysqli_stmt::reset`, `mysqli_stmt_reset`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::reset`

`mysqli_stmt_reset`

Resets a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::reset();
```

Procedural style

```
bool mysqli_stmt_reset(  
    mysqli_stmt stmt);
```

Resets a prepared statement on client and server to state after prepare.

It resets the statement on the server, data sent using `mysqli_stmt_send_long_data`, unbuffered result sets and current errors. It does not clear bindings or stored result sets. Stored result sets will be cleared when executing the prepared statement (or closing it).

To prepare a statement with another query use function `mysqli_stmt_prepare`.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

See Also

`mysqli_prepare`

6.3.10.25 `mysqli_stmt::result_metadata`, `mysqli_stmt_result_metadata`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_stmt::result_metadata`

`mysqli_stmt_result_metadata`

Returns result set metadata from a prepared statement

Description

Object oriented style

```
mysqli_result mysqli_stmt::result_metadata();
```

Procedural style

```
mysqli_result mysqli_stmt_result_metadata(  
    mysqli_stmt stmt);
```

If a statement passed to [mysqli_prepare](#) is one that produces a result set, [mysqli_stmt_result_metadata](#) returns the result object that can be used to process the meta information such as total number of fields and individual field information.

Note

This result set pointer can be passed as an argument to any of the field-based functions that process result set metadata, such as:

- [mysqli_num_fields](#)
- [mysqli_fetch_field](#)
- [mysqli_fetch_field_direct](#)
- [mysqli_fetch_fields](#)
- [mysqli_field_count](#)
- [mysqli_field_seek](#)
- [mysqli_field_tell](#)
- [mysqli_free_result](#)

The result set structure should be freed when you are done with it, which you can do by passing it to [mysqli_free_result](#)

Note

The result set returned by [mysqli_stmt_result_metadata](#) contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with [mysqli_stmt_fetch](#).

Parameters

stmt

Procedural style only: A statement identifier returned by [mysqli_stmt_init](#).

Return Values

Returns a result object or [FALSE](#) if an error occurred.

Examples

Example 6.110 Object oriented style

```
<?php  
$mysqli = new mysqli("localhost", "my_user", "my_password", "test");  
$mysqli->query("DROP TABLE IF EXISTS friends");  
$mysqli->query("CREATE TABLE friends (id int, name varchar(20))");  
$mysqli->query("INSERT INTO friends VALUES (1, 'Hartmut'), (2, 'Ulf')");
```

```
$stmt = $mysqli->prepare("SELECT id, name FROM friends");
$stmt->execute();
/* get resultset for metadata */
$result = $stmt->result_metadata();
/* retrieve field information from metadata result set */
$field = $result->fetch_field();
printf("Fieldname: %s\n", $field->name);
/* close resultset */
$result->close();
/* close connection */
$mysqli->close();
?>
```

Example 6.111 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "test");
mysqli_query($link, "DROP TABLE IF EXISTS friends");
mysqli_query($link, "CREATE TABLE friends (id int, name varchar(20))");
mysqli_query($link, "INSERT INTO friends VALUES (1, 'Hartmut'), (2, 'Ulf')");
$stmt = mysqli_prepare($link, "SELECT id, name FROM friends");
mysqli_stmt_execute($stmt);
/* get resultset for metadata */
$result = mysqli_stmt_result_metadata($stmt);
/* retrieve field information from metadata result set */
$field = mysqli_fetch_field($result);
printf("Fieldname: %s\n", $field->name);
/* close resultset */
mysqli_free_result($result);
/* close connection */
mysqli_close($link);
?>
```

See Also

[mysqli_prepare](#)
[mysqli_free_result](#)

6.3.10.26 [mysqli_stmt::send_long_data](#), [mysqli_stmt_send_long_data](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::send_long_data](#)

[mysqli_stmt_send_long_data](#)

Send data in blocks

Description

Object oriented style

```
bool mysqli_stmt::send_long_data(
    int param_nr,
    string data);
```

Procedural style

```
bool mysqli_stmt_send_long_data(
    mysqli_stmt stmt,
    int param_nr,
```

```
string data);
```

Allows to send parameter data to the server in pieces (or chunks), e.g. if the size of a blob exceeds the size of [max_allowed_packet](#). This function can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the TEXT or BLOB datatypes.

Parameters

<i>stmt</i>	Procedural style only: A statement identifier returned by mysqli_stmt_init .
<i>param_nr</i>	Indicates which parameter to associate the data with. Parameters are numbered beginning with 0.
<i>data</i>	A string containing data to be sent.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 6.112 Object oriented style

```
<?php
$stmt = $mysqli->prepare("INSERT INTO messages (message) VALUES (?)");
$null = NULL;
$stmt->bind_param("b", $null);
$fp = fopen("messages.txt", "r");
while (!feof($fp)) {
    $stmt->send_long_data(0, fread($fp, 8192));
}
fclose($fp);
$stmt->execute();
?>
```

See Also

[mysqli_prepare](#)
[mysqli_stmt_bind_param](#)

6.3.10.27 [mysqli_stmt::\\$sqlstate](#), [mysqli_stmt::\\$sqlstate](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::\\$sqlstate](#)

[mysqli_stmt::\\$sqlstate](#)

Returns SQLSTATE error from previous statement operation

Description

Object oriented style

```
string
mysqli_stmt->sqlstate ;
```

Procedural style

```
string mysqli_stmt_sqlstate(
    mysqli_stmt stmt);
```

Returns a string containing the SQLSTATE error code for the most recently invoked prepared statement function that can succeed or fail. The error code consists of five characters. '00000' means no error. The values are specified by ANSI SQL and ODBC. For a list of possible values, see <http://dev.mysql.com/doc/mysql/en/error-handling.html>.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns a string containing the SQLSTATE error code for the last error. The error code consists of five characters. '00000' means no error.

Notes

Note

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value `HY000` (general error) is used for unmapped errors.

Examples

Example 6.113 Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$mysqli->query("CREATE TABLE myCountry LIKE Country");
$mysqli->query("INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = $mysqli->prepare($query)) {
    /* drop table */
    $mysqli->query("DROP TABLE myCountry");
    /* execute query */
    $stmt->execute();
    printf("Error: %s.\n", $stmt->sqlstate);
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.114 Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
```

```

    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
mysqli_query($link, "CREATE TABLE myCountry LIKE Country");
mysqli_query($link, "INSERT INTO myCountry SELECT * FROM Country");
$query = "SELECT Name, Code FROM myCountry ORDER BY Name";
if ($stmt = mysqli_prepare($link, $query)) {
    /* drop table */
    mysqli_query($link, "DROP TABLE myCountry");
    /* execute query */
    mysqli_stmt_execute($stmt);
    printf("Error: %s.\n", mysqli_stmt_sqlstate($stmt));
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```
Error: 42S02.
```

See Also

[mysqli_stmt_errno](#)
[mysqli_stmt_error](#)

6.3.10.28 [mysqli_stmt::store_result](#), [mysqli_stmt_store_result](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_stmt::store_result](#)

[mysqli_stmt_store_result](#)

Transfers a result set from a prepared statement

Description

Object oriented style

```
bool mysqli_stmt::store_result();
```

Procedural style

```
bool mysqli_stmt_store_result(
    mysqli_stmt stmt);
```

You must call [mysqli_stmt_store_result](#) for every query that successfully produces a result set ([SELECT](#), [SHOW](#), [DESCRIBE](#), [EXPLAIN](#)), if and only if you want to buffer the complete result set by the client, so that the subsequent [mysqli_stmt_fetch](#) call returns buffered data.

Note

It is unnecessary to call [mysqli_stmt_store_result](#) for other queries, but if you do, it will not harm or cause any notable performance loss in all cases. You can detect whether the query produced a result set by checking if [mysqli_stmt_result_metadata](#) returns NULL.

Parameters

stmt Procedural style only: A statement identifier returned by `mysqli_stmt_init`.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example 6.115 Object oriented style

```
<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = $mysqli->prepare($query)) {
    /* execute query */
    $stmt->execute();
    /* store result */
    $stmt->store_result();
    printf("Number of rows: %d.\n", $stmt->num_rows);
    /* free result */
    $stmt->free_result();
    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.116 Procedural style

```
<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name LIMIT 20";
if ($stmt = mysqli_prepare($link, $query)) {
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* store result */
    mysqli_stmt_store_result($stmt);
    printf("Number of rows: %d.\n", mysqli_stmt_num_rows($stmt));
    /* free result */
    mysqli_stmt_free_result($stmt);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Number of rows: 20.
```

See Also

[mysqli_prepare](#)
[mysqli_stmt_result_metadata](#)
[mysqli_stmt_fetch](#)

6.3.11 The mysqli_result class

Copyright 1997-2014 the PHP Documentation Group.

Represents the result set obtained from a query against the database.

Changelog

Table 6.16 Changelog

Version	Description
5.4.0	Iterator support was added, as mysqli_result now implements Traversable .

```
mysqli_result {
    mysqli_result
        Traversable
        Properties

    int
        mysqli_result->current_field ;

    int
        mysqli_result->field_count ;

    array
        mysqli_result->lengths ;

    int
        mysqli_result->num_rows ;

Methods

    bool mysqli_result::data_seek(
        int offset);

    mixed mysqli_result::fetch_all(
        int resulttype
            = =MYSQLI_NUM);

    mixed mysqli_result::fetch_array(
        int resulttype
            = =MYSQLI_BOTH);

    array mysqli_result::fetch_assoc();

    object mysqli_result::fetch_field_direct(
```

```
int fieldnr);

object mysqli_result::fetch_field();

array mysqli_result::fetch_fields();

object mysqli_result::fetch_object(
    string class_name
        = "stdClass",
    array params);

mixed mysqli_result::fetch_row();

bool mysqli_result::field_seek(
    int fieldnr);

void mysqli_result::free();
}
```

6.3.11.1 `mysqli_result::$current_field`, `mysqli_field_tell`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_result::$current_field`

`mysqli_field_tell`

Get current field offset of a result pointer

Description

Object oriented style

```
int
mysqli_result->current_field ;
```

Procedural style

```
int mysqli_field_tell(
    mysqli_result result);
```

Returns the position of the field cursor used for the last `mysqli_fetch_field` call. This value can be used as an argument to `mysqli_field_seek`.

Parameters

<i>result</i>	Procedural style only: A result set identifier returned by <code>mysqli_query</code> , <code>mysqli_store_result</code> or <code>mysqli_use_result</code> .

Return Values

Returns current offset of field cursor.

Examples

Example 6.117 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
```



```

if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for all columns */
    while ($finfo = $result->fetch_field()) {
        /* get fieldpointer offset */
        $currentfield = $result->current_field;
        printf("Column %d:\n", $currentfield);
        printf("Name:      %s\n", $finfo->name);
        printf("Table:     %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:      %d\n", $finfo->flags);
        printf("Type:       %d\n\n", $finfo->type);
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>

```

Example 6.118 Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for all fields */
    while ($finfo = mysqli_fetch_field($result)) {
        /* get fieldpointer offset */
        $currentfield = mysqli_field_tell($result);
        printf("Column %d:\n", $currentfield);
        printf("Name:      %s\n", $finfo->name);
        printf("Table:     %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:      %d\n", $finfo->flags);
        printf("Type:       %d\n\n", $finfo->type);
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```

Column 1:
Name:      Name
Table:     Country
max. Len:  11
Flags:      1
Type:       254
Column 2:
Name:      SurfaceArea
Table:     Country
max. Len:  10

```

```
Flags:    32769
Type:     4
```

See Also

[mysqli_fetch_field](#)
[mysqli_field_seek](#)

6.3.11.2 [mysqli_result::data_seek](#), [mysqli_data_seek](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_result::data_seek](#)

[mysqli_data_seek](#)

Adjusts the result pointer to an arbitrary row in the result

Description

Object oriented style

```
bool mysqli_result::data_seek(
    int offset);
```

Procedural style

```
bool mysqli_data_seek(
    mysqli_result result,
    int offset);
```

The [mysqli_data_seek](#) function seeks to an arbitrary result pointer specified by the *offset* in the result set.

Parameters

result

Procedural style only: A result set identifier returned by [mysqli_query](#), [mysqli_store_result](#) or [mysqli_use_result](#).

offset

The field offset. Must be between zero and the total number of rows minus one (0..[mysqli_num_rows](#) - 1).

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Notes

Note

This function can only be used with buffered results attained from the use of the [mysqli_store_result](#) or [mysqli_query](#) functions.

Examples

Example 6.119 Object oriented style

```

<?php
/* Open a connection */
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($result = $mysqli->query($query)) {
    /* seek to row no. 400 */
    $result->data_seek(399);
    /* fetch row */
    $row = $result->fetch_row();
    printf("City: %s Countrycode: %s\n", $row[0], $row[1]);
    /* free result set*/
    $result->close();
}
/* close connection */
$mysqli->close();
?>

```

Example 6.120 Procedural style

```

<?php
/* Open a connection */
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (!$link) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER BY Name";
if ($result = mysqli_query($link, $query)) {
    /* seek to row no. 400 */
    mysqli_data_seek($result, 399);
    /* fetch row */
    $row = mysqli_fetch_row($result);
    printf("City: %s Countrycode: %s\n", $row[0], $row[1]);
    /* free result set*/
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```
City: Benin City Countrycode: NGA
```

See Also

[mysqli_store_result](#)
[mysqli_fetch_row](#)
[mysqli_fetch_array](#)
[mysqli_fetch_assoc](#)
[mysqli_fetch_object](#)
[mysqli_query](#)
[mysqli_num_rows](#)

6.3.11.3 `mysqli_result::fetch_all`, `mysqli_fetch_all`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_result::fetch_all`

`mysqli_fetch_all`

Fetches all result rows as an associative array, a numeric array, or both

Description

Object oriented style

```
mixed mysqli_result::fetch_all(  
    int resulttype  
    = MYSQLI_NUM);
```

Procedural style

```
mixed mysqli_fetch_all(  
    mysqli_result result,  
    int resulttype  
    = MYSQLI_NUM);
```

`mysqli_fetch_all` fetches all result rows and returns the result set as an associative array, a numeric array, or both.

Parameters

result

Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

resulttype

This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants `MYSQLI_ASSOC`, `MYSQLI_NUM`, or `MYSQLI_BOTH`.

Return Values

Returns an array of associative or numeric arrays holding result rows.

MySQL Native Driver Only

Available only with `mysqlnd`.

As `mysqli_fetch_all` returns all the rows as an array in a single step, it may consume more memory than some similar functions such as `mysqli_fetch_array`, which only returns one row at a time from the result set. Further, if you need to iterate over the result set, you will need a looping construct that will further impact performance. For these reasons `mysqli_fetch_all` should only be used in those situations where the fetched result set will be sent to another layer for processing.

See Also

`mysqli_fetch_array`
`mysqli_query`

6.3.11.4 `mysqli_result::fetch_array`, `mysqli_fetch_array`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_result::fetch_array`

`mysqli_fetch_array`

Fetch a result row as an associative, a numeric array, or both

Description

Object oriented style

```
mixed mysqli_result::fetch_array(  
    int resulttype  
    = MYSQLI_BOTH);
```

Procedural style

```
mixed mysqli_fetch_array(  
    mysqli_result result,  
    int resulttype  
    = MYSQLI_BOTH);
```

Returns an array that corresponds to the fetched row or `NULL` if there are no more rows for the resultset represented by the *result* parameter.

`mysqli_fetch_array` is an extended version of the `mysqli_fetch_row` function. In addition to storing the data in the numeric indices of the result array, the `mysqli_fetch_array` function can also store the data in associative indices, using the field names of the result set as keys.

Note

Field names returned by this function are *case-sensitive*.

Note

This function sets `NULL` fields to the PHP `NULL` value.

If two or more columns of the result have the same field names, the last column will take precedence and overwrite the earlier data. In order to access multiple columns with the same name, the numerically indexed version of the row must be used.

Parameters

result

Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

resulttype

This optional parameter is a constant indicating what type of array should be produced from the current row data. The possible values for this parameter are the constants `MYSQLI_ASSOC`, `MYSQLI_NUM`, or `MYSQLI_BOTH`.

By using the `MYSQLI_ASSOC` constant this function will behave identically to the `mysqli_fetch_assoc`, while `MYSQLI_NUM` will behave identically to the `mysqli_fetch_row` function. The final option `MYSQLI_BOTH` will create a single array with the attributes of both.

Return Values

Returns an array of strings that corresponds to the fetched row or `NULL` if there are no more rows in resultset.

Examples

Example 6.121 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID LIMIT 3";
$result = $mysqli->query($query);
/* numeric array */
$row = $result->fetch_array(MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);
/* associative array */
$row = $result->fetch_array(MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
/* associative and numeric array */
$row = $result->fetch_array(MYSQLI_BOTH);
printf ("%s (%s)\n", $row[0], $row["CountryCode"]);
/* free result set */
$result->free();
/* close connection */
$mysqli->close();
?>
```

Example 6.122 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID LIMIT 3";
$result = mysqli_query($link, $query);
/* numeric array */
$row = mysqli_fetch_array($result, MYSQLI_NUM);
printf ("%s (%s)\n", $row[0], $row[1]);
/* associative array */
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
/* associative and numeric array */
$row = mysqli_fetch_array($result, MYSQLI_BOTH);
printf ("%s (%s)\n", $row[0], $row["CountryCode"]);
/* free result set */
mysqli_free_result($result);
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Kabul (AFG)
Qandahar (AFG)
Herat (AFG)
```

See Also

[mysqli_fetch_assoc](#)
[mysqli_fetch_row](#)
[mysqli_fetch_object](#)
[mysqli_query](#)
[mysqli_data_seek](#)

6.3.11.5 [mysqli_result::fetch_assoc](#), [mysqli_fetch_assoc](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_result::fetch_assoc](#)

[mysqli_fetch_assoc](#)

Fetch a result row as an associative array

Description

Object oriented style

```
array mysqli_result::fetch_assoc();
```

Procedural style

```
array mysqli_fetch_assoc(  
    mysqli_result result);
```

Returns an associative array that corresponds to the fetched row or [NULL](#) if there are no more rows.

Note

Field names returned by this function are *case-sensitive*.

Note

This function sets NULL fields to the PHP [NULL](#) value.

Parameters

result

Procedural style only: A result set identifier returned by [mysqli_query](#), [mysqli_store_result](#) or [mysqli_use_result](#).

Return Values

Returns an associative array of strings representing the fetched row in the result set, where each key in the array represents the name of one of the result set's columns or [NULL](#) if there are no more rows in resultset.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you either need to access the result with numeric indices by using [mysqli_fetch_row](#) or add alias names.

Examples

Example 6.123 Object oriented style

```

<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = $mysqli->query($query)) {
    /* fetch associative array */
    while ($row = $result->fetch_assoc()) {
        printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
    }
    /* free result set */
    $result->free();
}
/* close connection */
$mysqli->close();
?>

```

Example 6.124 Procedural style

```

<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = mysqli_query($link, $query)) {
    /* fetch associative array */
    while ($row = mysqli_fetch_assoc($result)) {
        printf ("%s (%s)\n", $row["Name"], $row["CountryCode"]);
    }
    /* free result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>

```

The above examples will output:

```

Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)

```

Example 6.125 A `mysqli_result` example comparing `iterator` usage

```

<?php
$c = mysqli_connect('127.0.0.1', 'user', 'pass');
// Using iterators (support was added with PHP 5.4)
foreach ( $c->query('SELECT user,host FROM mysql.user') as $row ) {
    printf("%s'@'%s'\n", $row['user'], $row['host']);
}

```



```
}
echo "\n=====\\n";
// Not using iterators
$result = $c->query('SELECT user,host FROM mysql.user');
while ($row = $result->fetch_assoc()) {
    printf("%s'@'%s'\\n", $row['user'], $row['host']);
}
?>
```

The above example will output something similar to:

```
'root'@'192.168.1.1'
'root'@'127.0.0.1'
'dude'@'localhost'
'lebowski'@'localhost'
=====
'root'@'192.168.1.1'
'root'@'127.0.0.1'
'dude'@'localhost'
'lebowski'@'localhost'
```

See Also

[mysqli_fetch_array](#)
[mysqli_fetch_row](#)
[mysqli_fetch_object](#)
[mysqli_query](#)
[mysqli_data_seek](#)

6.3.11.6 [mysqli_result::fetch_field_direct](#), [mysqli_fetch_field_direct](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_result::fetch_field_direct](#)

[mysqli_fetch_field_direct](#)

Fetch meta-data for a single field

Description

Object oriented style

```
object mysqli_result::fetch_field_direct(
    int fieldnr);
```

Procedural style

```
object mysqli_fetch_field_direct(
    mysqli_result result,
    int fieldnr);
```

Returns an object which contains field definition information from the specified result set.

Parameters

result Procedural style only: A result set identifier returned by [mysqli_query](#), [mysqli_store_result](#) or [mysqli_use_result](#).

fieldnr

The field number. This value must be in the range from 0 to *number of fields - 1*.

Return Values

Returns an object which contains field definition information or **FALSE** if no field information for specified *fieldnr* is available.

Table 6.17 Object attributes

Attribute	Description
name	The name of the column
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
def	The default value for this field, represented as a string
max_length	The maximum width of the field for the result set.
length	The width of the field, as specified in the table definition.
charsetnr	The character set number for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for numeric fields)

Examples

Example 6.126 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Name LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for column 'SurfaceArea' */
    $finfo = $result->fetch_field_direct(1);
    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len: %d\n", $finfo->max_length);
    printf("Flags:    %d\n", $finfo->flags);
    printf("Type:     %d\n", $finfo->type);
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.127 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Name LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for column 'SurfaceArea' */
    $finfo = mysqli_fetch_field_direct($result, 1);
    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len: %d\n", $finfo->max_length);
    printf("Flags:      %d\n", $finfo->flags);
    printf("Type:       %d\n", $finfo->type);
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:      SurfaceArea
Table:     Country
max. Len:  10
Flags:     32769
Type:      4
```

See Also

[mysqli_num_fields](#)
[mysqli_fetch_field](#)
[mysqli_fetch_fields](#)

6.3.11.7 [mysqli_result::fetch_field](#), [mysqli_fetch_field](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_result::fetch_field](#)

[mysqli_fetch_field](#)

Returns the next field in the result set

Description

Object oriented style

```
object mysqli_result::fetch_field();
```

Procedural style

```
object mysqli_fetch_field(
    mysqli_result result);
```

Returns the definition of one column of a result set as an object. Call this function repeatedly to retrieve information about all columns in the result set.

Parameters

result

Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns an object which contains field definition information or `FALSE` if no field information is available.

Table 6.18 Object properties

Property	Description
name	The name of the column
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
def	Reserved for default value, currently always ""
db	Database (since PHP 5.3.6)
catalog	The catalog name, always "def" (since PHP 5.3.6)
max_length	The maximum width of the field for the result set.
length	The width of the field, as specified in the table definition.
charsetnr	The character set number for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for integer fields)

Examples

Example 6.128 Object oriented style

```

<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for all columns */
    while ($finfo = $result->fetch_field()) {
        printf("Name:      %s\n", $finfo->name);
        printf("Table:     %s\n", $finfo->table);
        printf("Max. Len: %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>

```

Example 6.129 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for all fields */
    while ($finfo = mysqli_fetch_field($result)) {
        printf("Name:      %s\n", $finfo->name);
        printf("Table:     %s\n", $finfo->table);
        printf("max. Len: %d\n", $finfo->max_length);
        printf("Flags:    %d\n", $finfo->flags);
        printf("Type:     %d\n\n", $finfo->type);
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:      Name
Table:     Country
max. Len:  11
Flags:     1
Type:      254
Name:      SurfaceArea
Table:     Country
max. Len:  10
Flags:     32769
Type:      4
```

See Also

[mysqli_num_fields](#)
[mysqli_fetch_field_direct](#)
[mysqli_fetch_fields](#)
[mysqli_field_seek](#)

6.3.11.8 [mysqli_result::fetch_fields](#), [mysqli_fetch_fields](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_result::fetch_fields](#)

[mysqli_fetch_fields](#)

Returns an array of objects representing the fields in a result set

Description

Object oriented style

```
array mysqli_result::fetch_fields();
```

Procedural style

```
array mysqli_fetch_fields(
    mysqli_result result);
```

This function serves an identical purpose to the [mysqli_fetch_field](#) function with the single difference that, instead of returning one object at a time for each field, the columns are returned as an array of objects.

Parameters

result

Procedural style only: A result set identifier returned by [mysqli_query](#), [mysqli_store_result](#) or [mysqli_use_result](#).

Return Values

Returns an array of objects which contains field definition information or [FALSE](#) if no field information is available.

Table 6.19 Object properties

Property	Description
name	The name of the column
orgname	Original column name if an alias was specified
table	The name of the table this field belongs to (if not calculated)
orgtable	Original table name if an alias was specified
max_length	The maximum width of the field for the result set.
length	The width of the field, in bytes, as specified in the table definition. Note that this number (bytes) might differ from your table definition value (characters), depending on the character set you use. For example, the character set utf8 has 3 bytes per character, so varchar(10) will return a length of 30 for utf8 (10*3), but return 10 for latin1 (10*1).
charsetnr	The character set number (id) for the field.
flags	An integer representing the bit-flags for the field.
type	The data type used for this field
decimals	The number of decimals used (for integer fields)

Examples

Example 6.130 Object oriented style

```
<?php
$mysqli = new mysqli("127.0.0.1", "root", "foofoo", "sakila");
/* check connection */
if ($mysqli->connect_errno) {
    printf("Connect failed: %s\n", $mysqli->connect_error);
```

```

        exit();
    }
    foreach (array('latin1', 'utf8') as $charset) {
        // Set character set, to show its impact on some values (e.g., length in bytes)
        $mysqli->set_charset($charset);
        $query = "SELECT actor_id, last_name from actor ORDER BY actor_id";
        echo "=====\n";
        echo "Character Set: $charset\n";
        echo "=====\n";

        if ($result = $mysqli->query($query)) {
            /* Get field information for all columns */
            $finfo = $result->fetch_fields();
            foreach ($finfo as $val) {
                printf("Name:      %s\n",    $val->name);
                printf("Table:      %s\n",    $val->table);
                printf("Max. Len:   %d\n",    $val->max_length);
                printf("Length:     %d\n",    $val->length);
                printf("charsetnr: %d\n",    $val->charsetnr);
                printf("Flags:      %d\n",    $val->flags);
                printf("Type:       %d\n\n", $val->type);
            }
            $result->free();
        }
    }
    $mysqli->close();
?>

```

Example 6.131 Procedural style

```

<?php
$link = mysqli_connect("127.0.0.1", "my_user", "my_password", "sakila");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
foreach (array('latin1', 'utf8') as $charset) {
    // Set character set, to show its impact on some values (e.g., length in bytes)
    mysqli_set_charset($link, $charset);
    $query = "SELECT actor_id, last_name from actor ORDER BY actor_id";
    echo "=====\n";
    echo "Character Set: $charset\n";
    echo "=====\n";
    if ($result = mysqli_query($link, $query)) {
        /* Get field information for all columns */
        $finfo = mysqli_fetch_fields($result);
        foreach ($finfo as $val) {
            printf("Name:      %s\n",    $val->name);
            printf("Table:      %s\n",    $val->table);
            printf("Max. Len:   %d\n",    $val->max_length);
            printf("Length:     %d\n",    $val->length);
            printf("charsetnr: %d\n",    $val->charsetnr);
            printf("Flags:      %d\n",    $val->flags);
            printf("Type:       %d\n\n", $val->type);
        }
        mysqli_free_result($result);
    }
}
mysqli_close($link);
?>

```

The above examples will output:

```
=====
Character Set: latin1
=====
Name:      actor_id
Table:     actor
Max. Len:  3
Length:    5
charsetnr: 63
Flags:     49699
Type:      2
Name:      last_name
Table:     actor
Max. Len:  12
Length:    45
charsetnr: 8
Flags:     20489
Type:      253
=====
Character Set: utf8
=====
Name:      actor_id
Table:     actor
Max. Len:  3
Length:    5
charsetnr: 63
Flags:     49699
Type:      2
Name:      last_name
Table:     actor
Max. Len:  12
Length:    135
charsetnr: 33
Flags:     20489
```

See Also

[mysqli_num_fields](#)
[mysqli_fetch_field_direct](#)
[mysqli_fetch_field](#)

6.3.11.9 `mysqli_result::fetch_object`, `mysqli_fetch_object`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_result::fetch_object`

[mysqli_fetch_object](#)

Returns the current row of a result set as an object

Description

Object oriented style

```
object mysqli_result::fetch_object(
    string class_name
        = "stdClass",
    array params);
```

Procedural style

```
object mysqli_fetch_object(
    mysqli_result result,
    string class_name
        = "stdClass",
```



```
array params);
```

The `mysqli_fetch_object` will return the current row result set as an object where the attributes of the object represent the names of the fields found within the result set.

Note that `mysqli_fetch_object` sets the properties of the object before calling the object constructor.

Parameters

<code>result</code>	Procedural style only: A result set identifier returned by <code>mysqli_query</code> , <code>mysqli_store_result</code> or <code>mysqli_use_result</code> .
<code>class_name</code>	The name of the class to instantiate, set the properties of and return. If not specified, a <code>stdClass</code> object is returned.
<code>params</code>	An optional array of parameters to pass to the constructor for <code>class_name</code> objects.

Return Values

Returns an object with string properties that corresponds to the fetched row or `NULL` if there are no more rows in resultset.

Note

Field names returned by this function are *case-sensitive*.

Note

This function sets NULL fields to the PHP `NULL` value.

Examples

Example 6.132 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = $mysqli->query($query)) {
    /* fetch object array */
    while ($obj = $result->fetch_object()) {
        printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
    }
    /* free result set */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.133 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = mysqli_query($link, $query)) {
    /* fetch associative array */
    while ($obj = mysqli_fetch_object($result)) {
        printf ("%s (%s)\n", $obj->Name, $obj->CountryCode);
    }
    /* free result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)
```

See Also

[mysqli_fetch_array](#)
[mysqli_fetch_assoc](#)
[mysqli_fetch_row](#)
[mysqli_query](#)
[mysqli_data_seek](#)

6.3.11.10 [mysqli_result::fetch_row](#), [mysqli_fetch_row](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_result::fetch_row](#)

[mysqli_fetch_row](#)

Get a result row as an enumerated array

Description

Object oriented style

```
mixed mysqli_result::fetch_row();
```

Procedural style

```
mixed mysqli_fetch_row(
    mysqli_result result);
```

Fetches one row of data from the result set and returns it as an enumerated array, where each column is stored in an array offset starting from 0 (zero). Each subsequent call to this function will return the next row within the result set, or [NULL](#) if there are no more rows.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

`mysqli_fetch_row` returns an array of strings that corresponds to the fetched row or `NULL` if there are no more rows in result set.

Note

This function sets NULL fields to the PHP `NULL` value.

Examples

Example 6.134 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = $mysqli->query($query)) {
    /* fetch object array */
    while ($row = $result->fetch_row()) {
        printf ("%s (%s)\n", $row[0], $row[1]);
    }
    /* free result set */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.135 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, CountryCode FROM City ORDER by ID DESC LIMIT 50,5";
if ($result = mysqli_query($link, $query)) {
    /* fetch associative array */
    while ($row = mysqli_fetch_row($result)) {
        printf ("%s (%s)\n", $row[0], $row[1]);
    }
    /* free result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)
```

See Also

[mysqli_fetch_array](#)
[mysqli_fetch_assoc](#)
[mysqli_fetch_object](#)
[mysqli_query](#)
[mysqli_data_seek](#)

6.3.11.11 `mysqli_result::$field_count`, `mysqli_num_fields`

[Copyright 1997-2014 the PHP Documentation Group.](#)

- `mysqli_result::$field_count`

`mysqli_num_fields`

Get the number of fields in a result

Description

Object oriented style

```
int
mysqli_result->field_count ;
```

Procedural style

```
int mysqli_num_fields(
    mysqli_result result);
```

Returns the number of fields from specified result set.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

The number of fields from a result set.

Examples

Example 6.136 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($result = $mysqli->query("SELECT * FROM City ORDER BY ID LIMIT 1")) {
    /* determine number of fields in result set */
    $field_cnt = $result->field_count;
    printf("Result set has %d fields.\n", $field_cnt);
    /* close result set */
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.137 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($result = mysqli_query($link, "SELECT * FROM City ORDER BY ID LIMIT 1")) {
    /* determine number of fields in result set */
    $field_cnt = mysqli_num_fields($result);
    printf("Result set has %d fields.\n", $field_cnt);
    /* close result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Result set has 5 fields.
```

See Also

[mysqli_fetch_field](#)

6.3.11.12 [mysqli_result::field_seek](#), [mysqli_field_seek](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_result::field_seek](#)

[mysqli_field_seek](#)

Set result pointer to a specified field offset

Description

Object oriented style

```
bool mysqli_result::field_seek(
    int fieldnr);
```

Procedural style

```
bool mysqli_field_seek(
    mysqli_result result,
    int fieldnr);
```

Sets the field cursor to the given offset. The next call to [mysqli_fetch_field](#) will retrieve the field definition of the column associated with that offset.

Note

To seek to the beginning of a row, pass an offset value of zero.

Parameters

result

Procedural style only: A result set identifier returned by [mysqli_query](#), [mysqli_store_result](#) or [mysqli_use_result](#).

fieldnr

The field number. This value must be in the range from 0 to [number of fields](#) - 1.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 6.138 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = $mysqli->query($query)) {
    /* Get field information for 2nd column */
    $result->field_seek(1);
    $finfo = $result->fetch_field();
    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len: %d\n", $finfo->max_length);
    printf("Flags:      %d\n", $finfo->flags);
    printf("Type:       %d\n\n", $finfo->type);
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.139 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT Name, SurfaceArea from Country ORDER BY Code LIMIT 5";
if ($result = mysqli_query($link, $query)) {
    /* Get field information for 2nd column */
    mysqli_field_seek($result, 1);
    $finfo = mysqli_fetch_field($result);
    printf("Name:      %s\n", $finfo->name);
    printf("Table:     %s\n", $finfo->table);
    printf("max. Len: %d\n", $finfo->max_length);
    printf("Flags:      %d\n", $finfo->flags);
    printf("Type:       %d\n\n", $finfo->type);
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Name:      SurfaceArea
Table:     Country
max. Len:  10
Flags:     32769
Type:      4
```

See Also

[mysqli_fetch_field](#)

6.3.11.13 [mysqli_result::free](#), [mysqli_free_result](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_result::free](#)

[mysqli_free_result](#)

Frees the memory associated with a result

Description

Object oriented style

```
void mysqli_result::free();
```

```
void mysqli_result::close();
```

```
void mysqli_result::free_result();
```

Procedural style

```
void mysqli_free_result(
    mysqli_result result);
```

Frees the memory associated with the result.

Note

You should always free your result with `mysqli_free_result`, when your result object is not needed anymore.

Parameters*result*

Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

No value is returned.

See Also

`mysqli_query`
`mysqli_stmt_store_result`
`mysqli_store_result`
`mysqli_use_result`

6.3.11.14 `mysqli_result::$lengths`, `mysqli_fetch_lengths`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_result::$lengths`

mysqli_fetch_lengths

Returns the lengths of the columns of the current row in the result set

Description

Object oriented style

```
array  
mysqli_result->lengths ;
```

Procedural style

```
array mysqli_fetch_lengths(  
    mysqli_result result);
```

The `mysqli_fetch_lengths` function returns an array containing the lengths of every column of the current row within the result set.

Parameters*result*

Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

An array of integers representing the size of each column (not including any terminating null characters). `FALSE` if an error occurred.

`mysqli_fetch_lengths` is valid only for the current row of the result set. It returns `FALSE` if you call it before calling `mysqli_fetch_row/array/object` or after retrieving all rows in the result.

Examples

Example 6.140 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT * from Country ORDER BY Code LIMIT 1";
if ($result = $mysqli->query($query)) {
    $row = $result->fetch_row();
    /* display column lengths */
    foreach ($result->lengths as $i => $val) {
        printf("Field %2d has Length %2d\n", $i+1, $val);
    }
    $result->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.141 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
$query = "SELECT * from Country ORDER BY Code LIMIT 1";
if ($result = mysqli_query($link, $query)) {
    $row = mysqli_fetch_row($result);
    /* display column lengths */
    foreach (mysqli_fetch_lengths($result) as $i => $val) {
        printf("Field %2d has Length %2d\n", $i+1, $val);
    }
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Field  1 has Length  3
Field  2 has Length  5
Field  3 has Length 13
Field  4 has Length  9
Field  5 has Length  6
Field  6 has Length  1
Field  7 has Length  6
Field  8 has Length  4
Field  9 has Length  6
Field 10 has Length  6
Field 11 has Length  5
Field 12 has Length 44
```

```
Field 13 has Length 7
Field 14 has Length 3
Field 15 has Length 2
```

6.3.11.15 `mysqli_result::$num_rows`, `mysqli_num_rows`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_result::$num_rows`
`mysqli_num_rows`

Gets the number of rows in a result

Description

Object oriented style

```
int
mysqli_result->num_rows ;
```

Procedural style

```
int mysqli_num_rows(
    mysqli_result result);
```

Returns the number of rows in the result set.

The behaviour of `mysqli_num_rows` depends on whether buffered or unbuffered result sets are being used. For unbuffered result sets, `mysqli_num_rows` will not return the correct number of rows until all the rows in the result have been retrieved.

Parameters

result Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns number of rows in the result set.

Note

If the number of rows is greater than `PHP_INT_MAX`, the number will be returned as a string.

Examples

Example 6.142 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($result = $mysqli->query("SELECT Code, Name FROM Country ORDER BY Name")) {
```

```
/* determine number of rows result set */
$row_cnt = $result->num_rows;
printf("Result set has %d rows.\n", $row_cnt);
/* close result set */
$result->close();
}
/* close connection */
$mysqli->close();
?>
```

Example 6.143 Procedural style

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
if ($result = mysqli_query($link, "SELECT Code, Name FROM Country ORDER BY Name")) {
    /* determine number of rows result set */
    $row_cnt = mysqli_num_rows($result);
    printf("Result set has %d rows.\n", $row_cnt);
    /* close result set */
    mysqli_free_result($result);
}
/* close connection */
mysqli_close($link);
?>
```

The above examples will output:

```
Result set has 239 rows.
```

See Also

[mysqli_affected_rows](#)
[mysqli_store_result](#)
[mysqli_use_result](#)
[mysqli_query](#)

6.3.12 The mysqli_driver class

Copyright 1997-2014 the PHP Documentation Group.

MySQLi Driver.

```
mysqli_driver {
    mysqli_driver
        Properties

    public readonly string
        client_info ;

    public readonly string
```

```
    client_version ;

    public readonly string
        driver_version ;

    public readonly string
        embedded ;

    public bool
        reconnect ;

    public int
        report_mode ;

Methods

    void mysqli_driver::embedded_server_end();

    bool mysqli_driver::embedded_server_start(
        bool start,
        array arguments,
        array groups);
}
```

<code>client_info</code>	The Client API header version
<code>client_version</code>	The Client version
<code>driver_version</code>	The MySQLi Driver version
<code>embedded</code>	Whether MySQLi Embedded support is enabled
<code>reconnect</code>	Allow or prevent reconnect (see the <code>mysqli.reconnect</code> INI directive)
<code>report_mode</code>	Set to <code>MYSQLI_REPORT_OFF</code> , <code>MYSQLI_REPORT_ALL</code> or any combination of <code>MYSQLI_REPORT_STRICT</code> (throw Exceptions for errors), <code>MYSQLI_REPORT_ERROR</code> (report errors) and <code>MYSQLI_REPORT_INDEX</code> (errors regarding indexes). See also <code>mysqli_report</code> .

6.3.12.1 `mysqli_driver::embedded_server_end`, `mysqli_embedded_server_end`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_driver::embedded_server_end`

`mysqli_embedded_server_end`

Stop embedded server

Description

Object oriented style

```
void mysqli_driver::embedded_server_end();
```

Procedural style

```
void mysqli_embedded_server_end();
```

Warning

This function is currently not documented; only its argument list is available.

6.3.12.2 `mysqli_driver::embedded_server_start`, `mysqli_embedded_server_start`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_driver::embedded_server_start`

`mysqli_embedded_server_start`

Initialize and start embedded server

Description

Object oriented style

```
bool mysqli_driver::embedded_server_start(  
    bool start,  
    array arguments,  
    array groups);
```

Procedural style

```
bool mysqli_embedded_server_start(  
    bool start,  
    array arguments,  
    array groups);
```

Warning

This function is currently not documented; only its argument list is available.

6.3.12.3 `mysqli_driver::$report_mode`, `mysqli_report`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_driver::$report_mode`

`mysqli_report`

Enables or disables internal report functions

Description

Object oriented style

```
int  
mysqli_driver->report_mode ;
```

Procedural style

```
bool mysqli_report(  
    int flags);
```

A function helpful in improving queries during code development and testing. Depending on the flags, it reports errors from mysqli function calls or queries that don't use an index (or use a bad index).

Parameters

flags

Table 6.20 Supported flags

Name	Description
<code>MYSQLI_REPORT_OFF</code>	Turns reporting off

Name	Description
<code>MYSQLI_REPORT_ERROR</code>	Report errors from mysqli function calls
<code>MYSQLI_REPORT_STRICT</code>	Throw <code>mysqli_sql_exception</code> for errors instead of warnings
<code>MYSQLI_REPORT_INDEX</code>	Report if no index or bad index was used in a query
<code>MYSQLI_REPORT_ALL</code>	Set all options (report all)

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Changelog

Version	Description
5.3.4	Changing the reporting mode is now be per-request, rather than per-process.
5.2.15	Changing the reporting mode is now be per-request, rather than per-process.

Examples

Example 6.144 Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* activate reporting */
$driver = new mysqli_driver();
$driver->report_mode = MYSQLI_REPORT_ALL;
try {
    /* this query should report an error */
    $result = $mysqli->query("SELECT Name FROM Nonexistingtable WHERE population > 50000");
    /* this query should report a bad index */
    $result = $mysqli->query("SELECT Name FROM City WHERE population > 50000");
    $result->close();
    $mysqli->close();
} catch (mysqli_sql_exception $e) {
    echo $e->__toString();
}
?>
```

Example 6.145 Procedural style

```
<?php
/* activate reporting */
mysqli_report(MYSQLI_REPORT_ALL);
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */
if (mysqli_connect_errno()) {
```

```
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}
/* this query should report an error */
$result = mysqli_query("SELECT Name FROM Nonexistingtable WHERE population > 50000");
/* this query should report a bad index */
$result = mysqli_query("SELECT Name FROM City WHERE population > 50000");
mysqli_free_result($result);
mysqli_close($link);
?>
```

See Also

[mysqli_debug](#)
[mysqli_dump_debug_info](#)
[mysqli_sql_exception](#)
[set_exception_handler](#)
[error_reporting](#)

6.3.13 The mysqli_warning class

Copyright 1997-2014 the PHP Documentation Group.

Represents a MySQL warning.

```
mysqli_warning {
    mysqli_warning
        Properties

    public
        message ;

    public
        sqlstate ;

    public
        errno ;

    Methods

    public mysqli_warning::__construct();

    public void mysqli_warning::next();
}
```

message	Message string
sqlstate	SQL state
errno	Error number

6.3.13.1 [mysqli_warning::__construct](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_warning::__construct](#)

The `__construct` purpose

Description

```
public mysqli_warning::__construct();
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

6.3.13.2 `mysqli_warning::next`

[Copyright 1997-2014 the PHP Documentation Group.](#)

- `mysqli_warning::next`

The next purpose

Description

```
public void mysqli_warning::next();
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

6.3.14 The `mysqli_sql_exception` class

[Copyright 1997-2014 the PHP Documentation Group.](#)

The mysqli exception handling class.

```
mysqli_sql_exception {  
    mysqli_sql_exception extends RuntimeException  
        Properties  
  
    protected string  
        sqlstate ;  
  
    Inherited properties  
  
    protected string  
        message ;  
  
    protected int  
        code ;  
  
    protected string  
        file ;  
  
    protected int
```



```
line ;  
}
```

`sqlstate`

The sql state with the error.

6.3.15 Aliases and deprecated Mysqli Functions

Copyright 1997-2014 the PHP Documentation Group.

6.3.15.1 `mysqli_bind_param`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_bind_param`

Alias for `mysqli_stmt_bind_param`

Description

This function is an alias of `mysqli_stmt_bind_param`.

Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_bind_param`

6.3.15.2 `mysqli_bind_result`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_bind_result`

Alias for `mysqli_stmt_bind_result`

Description

This function is an alias of `mysqli_stmt_bind_result`.

Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_bind_result`

6.3.15.3 `mysqli_client_encoding`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_client_encoding`

Alias of `mysqli_character_set_name`

Description

This function is an alias of [mysqli_character_set_name](#).

Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

[mysqli_real_escape_string](#)

6.3.15.4 [mysqli_connect](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_connect](#)

Alias of [mysqli::__construct](#)

Description

This function is an alias of: [mysqli::__construct](#)

Although the [mysqli::__construct](#) documentation also includes procedural examples that use the [mysqli_connect](#) function, here is a short example:

Examples**Example 6.146 [mysqli_connect](#) example**

```
<?php
$link = mysqli_connect("127.0.0.1", "my_user", "my_password", "my_db");
if (!$link) {
    echo "Error: Unable to connect to MySQL." . PHP_EOL;
    echo "Debugging errno: " . mysqli_connect_errno() . PHP_EOL;
    echo "Debugging error: " . mysqli_connect_error() . PHP_EOL;
    exit;
}
echo "Success: A proper connection to MySQL was made! The my_db database is great." . PHP_EOL;
echo "Host information: " . mysqli_get_host_info($link) . PHP_EOL;
mysqli_close($link);
?>
```

The above examples will output:

```
Success: A proper connection to MySQL was made! The my_db database is great.
Host information: localhost via TCP/IP
```

6.3.15.5 [mysqli::disable_reads_from_master](#), [mysqli_disable_reads_from_master](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli::disable_reads_from_master](#)

[mysqli_disable_reads_from_master](#)

Disable reads from master

Description

Object oriented style

```
void mysqli::disable_reads_from_master();
```

Procedural style

```
bool mysqli_disable_reads_from_master(  
    mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.15.6 `mysqli_disable_rpl_parse`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_disable_rpl_parse`

Disable RPL parse

Description

```
bool mysqli_disable_rpl_parse(  
    mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.15.7 `mysqli_enable_reads_from_master`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_enable_reads_from_master`

Enable reads from master

Description

```
bool mysqli_enable_reads_from_master(  
    mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.15.8 `mysqli_enable_rpl_parse`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_enable_rpl_parse`

Enable RPL parse

Description

```
bool mysqli_enable_rpl_parse(
    mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.15.9 `mysqli_escape_string`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_escape_string`

Alias of `mysqli_real_escape_string`

Description

This function is an alias of: `mysqli_real_escape_string`.

6.3.15.10 `mysqli_execute`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_execute`

Alias for `mysqli_stmt_execute`

Description

This function is an alias of `mysqli_stmt_execute`.

Notes

Note

`mysqli_execute` is deprecated and will be removed.

See Also

`mysqli_stmt_execute`

6.3.15.11 `mysqli_fetch`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_fetch`

Alias for [mysqli_stmt_fetch](#)

Description

This function is an alias of [mysqli_stmt_fetch](#).

Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

[mysqli_stmt_fetch](#)

6.3.15.12 [mysqli_get_cache_stats](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_get_cache_stats](#)

Returns client Zval cache statistics

Warning

This function has been *REMOVED* as of PHP 5.4.0.

Description

```
array mysqli_get_cache_stats();
```

Returns an empty array. Available only with [mysqlnd](#).

Parameters

Return Values

Returns an empty array on success, [FALSE](#) otherwise.

Changelog

Version	Description
5.4.0	The mysqli_get_cache_stats was removed.
5.3.0	The mysqli_get_cache_stats was added as stub.

6.3.15.13 [mysqli_get_links_stats](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_get_links_stats](#)

Return information about open and cached links

Description

```
array mysqli_get_links_stats();
```

[mysqli_get_links_stats](#) returns information about open and cached MySQL links.

Parameters

This function has no parameters.

Return Values

`mysqli_get_links_stats` returns an associative array with three elements, keyed as follows:

<code>total</code>	An integer indicating the total number of open links in any state.
<code>active_plinks</code>	An integer representing the number of active persistent connections.
<code>cached_plinks</code>	An integer representing the number of inactive persistent connections.

6.3.15.14 `mysqli_get_metadata`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_get_metadata`

Alias for `mysqli_stmt_result_metadata`

Description

This function is an alias of `mysqli_stmt_result_metadata`.

Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_result_metadata`

6.3.15.15 `mysqli_master_query`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_master_query`

Enforce execution of a query on the master in a master/slave setup

Description

```
bool mysqli_master_query(
    mysqli link,
    string query);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.15.16 `mysqli_param_count`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_param_count](#)

Alias for [mysqli_stmt_param_count](#)

Description

This function is an alias of [mysqli_stmt_param_count](#).

Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

[mysqli_stmt_param_count](#)

6.3.15.17 [mysqli_report](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_report](#)

Alias of [mysqli_driver->report_mode](#)

Description

This function is an alias of: [mysqli_driver->report_mode](#)

6.3.15.18 [mysqli_rpl_parse_enabled](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_rpl_parse_enabled](#)

Check if RPL parse is enabled

Description

```
int mysqli_rpl_parse_enabled(  
    mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.15.19 [mysqli_rpl_probe](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_rpl_probe](#)

RPL probe

Description

```
bool mysqli_rpl_probe(  

```

```
mysqli link);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.15.20 `mysqli_send_long_data`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_send_long_data`

Alias for `mysqli_stmt_send_long_data`

Description

This function is an alias of `mysqli_stmt_send_long_data`.

Warning

This function has been *DEPRECATED* as of PHP 5.3.0 and *REMOVED* as of PHP 5.4.0.

See Also

`mysqli_stmt_send_long_data`

6.3.15.21 `mysqli::set_opt`, `mysqli_set_opt`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli::set_opt`

`mysqli_set_opt`

Alias of `mysqli_options`

Description

This function is an alias of `mysqli_options`.

6.3.15.22 `mysqli_slave_query`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqli_slave_query`

Force execution of a query on a slave in a master/slave setup

Description

```
bool mysqli_slave_query(  
    mysqli link,  
    string query);
```

Warning

This function is currently not documented; only its argument list is available.

Warning

This function has been *DEPRECATED* and *REMOVED* as of PHP 5.3.0.

6.3.16 Changelog

[Copyright 1997-2014 the PHP Documentation Group.](#)

The following changes have been made to classes/functions/methods of this extension.

6.4 MySQL Functions (PDO_MYSQL)

[Copyright 1997-2014 the PHP Documentation Group.](#)

PDO_MYSQL is a driver that implements the [PHP Data Objects \(PDO\) interface](#) to enable access from PHP to MySQL 3.x, 4.x and 5.x databases.

PDO_MYSQL will take advantage of native prepared statement support present in MySQL 4.1 and higher. If you're using an older version of the mysql client libraries, PDO will emulate them for you.

Warning

Beware: Some MySQL table types (storage engines) do not support transactions. When writing transactional database code using a table type that does not support transactions, MySQL will pretend that a transaction was initiated successfully. In addition, any DDL queries issued will implicitly commit any pending transactions.

The common Unix distributions include binary versions of PHP that can be installed. Although these binary versions are typically built with support for the MySQL extensions, the extension libraries themselves may need to be installed using an additional package. Check the package manager that comes with your chosen distribution for availability.

For example, on Ubuntu the [php5-mysql](#) package installs the ext/mysql, ext/mysqli, and PDO_MYSQL PHP extensions. On CentOS, the [php-mysql](#) package also installs these three PHP extensions.

Alternatively, you can compile this extension yourself. Building PHP from source allows you to specify the MySQL extensions you want to use, as well as your choice of client library for each extension.

When compiling, use `--with-pdo-mysql[=DIR]` to install the PDO MySQL extension, where the optional `[=DIR]` is the MySQL base library. As of PHP 5.4, [mysqlnd](#) is the default library. For details about choosing a library, see [Choosing a MySQL library](#).

Optionally, the `--with-mysql-sock[=DIR]` sets to location to the MySQL unix socket pointer for all MySQL extensions, including PDO_MYSQL. If unspecified, the default locations are searched.

Optionally, the `--with-zlib-dir[=DIR]` is used to set the path to the libz install prefix.

```
$ ./configure --with-pdo-mysql --with-mysql-sock=/var/mysql/mysql.sock
```

SSL support is enabled using the appropriate [PDO_MYSQL constants](#), which is equivalent to calling the [MySQL C API function `mysql\_ssl\_set\(\)`](#). Also, SSL cannot be enabled with `PDO::setAttribute` because the connection already exists. See also the MySQL documentation about [connecting to MySQL with SSL](#).

Table 6.21 Changelog

Version	Description
5.4.0	mysqlnd became the default MySQL library when compiling PDO_MYSQL. Previously, libmysqlclient was the default MySQL library.
5.4.0	MySQL client libraries 4.1 and below are no longer supported.
5.3.9	Added SSL support with mysqlnd and OpenSSL.
5.3.7	Added SSL support with libmysqlclient and OpenSSL.

The constants below are defined by this driver, and will only be available when the extension has been either compiled into PHP or dynamically loaded at runtime. In addition, these driver-specific constants should only be used if you are using this driver. Using driver-specific attributes with another driver may result in unexpected behaviour. [PDO::getAttribute](#) may be used to obtain the [PDO_ATTR_DRIVER_NAME](#) attribute to check the driver, if your code can run against multiple drivers.

[PDO::MYSQL_ATTR_USE_BUFFERED_QUERY](#) (integer) If this attribute is set to [TRUE](#) on a [PDOStatement](#), the MySQL driver will use the buffered versions of the MySQL API. If you're writing portable code, you should use [PDOStatement::fetchAll](#) instead.

Example 6.147 Forcing queries to be buffered in mysql

```
<?php
if ($db->getAttribute(PDO::ATTR_DRIVER_NAME) == 'mysql') {
    $stmt = $db->prepare('select * from foo',
        array(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY => true));
} else {
    die("my application only works with mysql; I should use \$stmt->fetchAll()");
}
?>
```

[PDO::MYSQL_ATTR_LOCAL_INFILE](#) (integer) Enable [LOAD LOCAL INFILE](#).

Note, this constant can only be used in the [driver_options](#) array when constructing a new database handle.

[PDO::MYSQL_ATTR_INIT_COMMAND](#) (integer) Command to execute when connecting to the MySQL server. Will automatically be re-executed when reconnecting.

Note, this constant can only be used in the [driver_options](#) array when constructing a new database handle.

[PDO::MYSQL_ATTR_READ_DEFAULT_FILE](#) (integer) Read options from the named option file instead of from [my.cnf](#). This option is not available if [mysqlnd](#) is used, because [mysqlnd](#) does not read the mysql configuration files.

[PDO::MYSQL_ATTR_READ_DEFAULT_GROUP](#) (integer) Read options from the named group from [my.cnf](#) or the file specified with [MYSQL_READ_DEFAULT_FILE](#). This option is not available if [mysqlnd](#) is used, because [mysqlnd](#) does not read the mysql configuration files.

[PDO::MYSQL_ATTR_MAX_BUFFER_SIZE](#) (integer) Maximum buffer size. Defaults to 1 MiB. This constant is not supported when compiled against [mysqlnd](#).

<code>PDO::MYSQL_ATTR_DIRECT_QUERY</code> (integer)	Perform direct queries, don't use prepared statements.
<code>PDO::MYSQL_ATTR_FOUND_ROWS</code> (integer)	Return the number of found (matched) rows, not the number of changed rows.
<code>PDO::MYSQL_ATTR_IGNORE_SPACE</code> (integer)	Permit spaces after function names. Makes all functions names reserved words.
<code>PDO::MYSQL_ATTR_COMPRESS</code> (integer)	Enable network communication compression. This is also supported when compiled against mysqlnd as of PHP 5.3.11.
<code>PDO::MYSQL_ATTR_SSL_CA</code> (integer)	The file path to the SSL certificate authority. This exists as of PHP 5.3.7.
<code>PDO::MYSQL_ATTR_SSL_CAPATH</code> (integer)	The file path to the directory that contains the trusted SSL CA certificates, which are stored in PEM format. This exists as of PHP 5.3.7.
<code>PDO::MYSQL_ATTR_SSL_CERT</code> (integer)	The file path to the SSL certificate. This exists as of PHP 5.3.7.
<code>PDO::MYSQL_ATTR_SSL_CIPHER</code> (integer)	A list of one or more permissible ciphers to use for SSL encryption, in a format understood by OpenSSL. For example: <code>DHE-RSA-AES256-SHA:AES128-SHA</code> This exists as of PHP 5.3.7.
<code>PDO::MYSQL_ATTR_SSL_KEY</code> (integer)	The file path to the SSL key. This exists as of PHP 5.3.7.
<code>PDO::MYSQL_ATTR_MULTI_STATEMENTS</code> (integer)	Disables multi query execution in both <code>PDO::prepare</code> and <code>PDO::query</code> when set to <code>FALSE</code> . Note, this constant can only be used in the <code>driver_options</code> array when constructing a new database handle. This exists as of PHP 5.5.21 and PHP 5.6.5.

The behaviour of these functions is affected by settings in `php.ini`.

Table 6.22 PDO_MYSQL Configuration Options

Name	Default	Changeable
<code>pdo_mysql.default_socket</code>	<code>"/tmp/mysql.sock"</code>	PHP_INI_SYSTEM
<code>pdo_mysql.debug</code>	NULL	PHP_INI_SYSTEM

For further details and definitions of the `PHP_INI_*` modes, see the <http://www.php.net/manual/en/configuration.changes.modes>.

Here's a short explanation of the configuration directives.

<code>pdo_mysql.default_socket</code> string	Sets a Unix domain socket. This value can either be set at compile time if a domain socket is found at configure. This ini setting is Unix only.

`pdo_mysql.debug` boolean

Enables debugging for PDO_MYSQL. This setting is only available when PDO_MYSQL is compiled against mysqlnd and in PDO debug mode.

6.4.1 PDO_MYSQL DSN

Copyright 1997-2014 the PHP Documentation Group.

- PDO_MYSQL DSN

Connecting to MySQL databases

Description

The PDO_MYSQL Data Source Name (DSN) is composed of the following elements:

DSN prefix	The DSN prefix is <code>mysql:</code> .
<code>host</code>	The hostname on which the database server resides.
<code>port</code>	The port number where the database server is listening.
<code>dbname</code>	The name of the database.
<code>unix_socket</code>	The MySQL Unix socket (shouldn't be used with <code>host</code> or <code>port</code>).
<code>charset</code>	The character set. See the character set concepts documentation for more information. Prior to PHP 5.3.6, this element was silently ignored. The same behaviour can be partly replicated with the <code>PDO::MYSQL_ATTR_INIT_COMMAND</code> driver option, as the following example shows.

Warning

The method in the below example can only be used with character sets that share the same lower 7 bit representation as ASCII, such as ISO-8859-1 and UTF-8. Users using character sets that have different representations (such as UTF-16 or Big5) *must* use the `charset` option provided in PHP 5.3.6 and later versions.

Example 6.148 Setting the connection character set to UTF-8 prior to PHP 5.3.6

```
<?php
$dsn = 'mysql:host=localhost;dbname=testdb';
$username = 'username';
$password = 'password';
$options = array(
    PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8',
);
$dbh = new PDO($dsn, $username, $password, $options);
?>
```

Changelog

Version	Description
5.3.6	Prior to version 5.3.6, charset was ignored.

Examples

Example 6.149 PDO_MYSQL DSN examples

The following example shows a PDO_MYSQL DSN for connecting to MySQL databases:

```
mysql:host=localhost;dbname=testdb
```

More complete examples:

```
mysql:host=localhost;port=3307;dbname=testdb  
mysql:unix_socket=/tmp/mysql.sock;dbname=testdb
```

Notes

Unix only:

When the host name is set to "[localhost](#)", then the connection to the server is made thru a domain socket. If PDO_MYSQL is compiled against libmysqlclient then the location of the socket file is at libmysqlclient's compiled in location. If PDO_MYSQL is compiled against mysqlnd a default socket can be set thru the [pdo_mysql.default_socket](#) setting.

6.5 Original MySQL API

Copyright 1997-2014 the PHP Documentation Group.

This extension is deprecated as of PHP 5.5.0, and has been removed as of PHP 7.0.0. Instead, either the [mysql](#) or [PDO_MYSQL](#) extension should be used. See also the [MySQL API Overview](#) for further help while choosing a MySQL API.

These functions allow you to access MySQL database servers. More information about MySQL can be found at <http://www.mysql.com/>.

Documentation for MySQL can be found at <http://dev.mysql.com/doc/>.

6.5.1 Installing/Configuring

Copyright 1997-2014 the PHP Documentation Group.

6.5.1.1 Requirements

Copyright 1997-2014 the PHP Documentation Group.

In order to have these functions available, you must compile PHP with MySQL support.

6.5.1.2 Installation

Copyright 1997-2014 the PHP Documentation Group.

For compiling, simply use the `--with-mysql[=DIR]` configuration option where the optional `[DIR]` points to the MySQL installation directory.

Although this MySQL extension is compatible with MySQL 4.1.0 and greater, it doesn't support the extra functionality that these versions provide. For that, use the [MySQLi](#) extension.

If you would like to install the `mysql` extension along with the `mysqli` extension you have to use the same client library to avoid any conflicts.

Installation on Linux Systems

Copyright 1997-2014 the PHP Documentation Group.

Note: `[DIR]` is the path to the MySQL client library files (*headers and libraries*), which can be downloaded from [MySQL](#).

Table 6.23 ext/mysql compile time support matrix

PHP Version	Default	Configure Options: mysqlnd	Configure Options: libmysqlclient	Changelog
4.x.x	libmysqlclient	Not Available	<code>--without-mysql</code> to disable	MySQL enabled by default, MySQL client libraries are bundled
5.0.x, 5.1.x, 5.2.x	libmysqlclient	Not Available	<code>--with-mysql=[DIR]</code>	MySQL is no longer enabled by default, and the MySQL client libraries are no longer bundled
5.3.x	libmysqlclient	<code>--with-mysql=mysqlnd</code>	<code>--with-mysql=[DIR]</code>	mysqlnd is now available
5.4.x	mysqlnd	<code>--with-mysql</code>	<code>--with-mysql=[DIR]</code>	mysqlnd is now the default

Installation on Windows Systems

Copyright 1997-2014 the PHP Documentation Group.

PHP 4

Copyright 1997-2014 the PHP Documentation Group.

The PHP MySQL extension is compiled into PHP.

PHP 5.0.x, 5.1.x, 5.2.x

Copyright 1997-2014 the PHP Documentation Group.

MySQL is no longer enabled by default, so the `php_mysql.dll` DLL must be enabled inside of `php.ini`. Also, PHP needs access to the MySQL client library. A file named `libmysql.dll` is included in the Windows PHP distribution and in order for PHP to talk to MySQL this file needs to be available to the Windows systems `PATH`. See the FAQ titled "[How do I add my PHP directory to the PATH on Windows](#)" for information on how to do this. Although copying `libmysql.dll` to the Windows system directory also works (because the system directory is by default in the system's `PATH`), it's not recommended.

As with enabling any PHP extension (such as `php_mysql.dll`), the PHP directive `extension_dir` should be set to the directory where the PHP extensions are located. See also the [Manual Windows Installation Instructions](#). An example `extension_dir` value for PHP 5 is `c:\php\ext`

Note

If when starting the web server an error similar to the following occurs: `"Unable to load dynamic library './php_mysql.dll'"`, this is because `php_mysql.dll` and/or `libmysql.dll` cannot be found by the system.

PHP 5.3.0+

Copyright 1997-2014 the PHP Documentation Group.

The [MySQL Native Driver](#) is enabled by default. Include `php_mysql.dll`, but `libmysql.dll` is no longer required or used.

MySQL Installation Notes

Copyright 1997-2014 the PHP Documentation Group.

Warning

Crashes and startup problems of PHP may be encountered when loading this extension in conjunction with the [recode](#) extension. See the [recode](#) extension for more information.

Note

If you need charsets other than *latin* (default), you have to install external (not bundled) `libmysqlclient` with compiled charset support.

6.5.1.3 Runtime Configuration

Copyright 1997-2014 the PHP Documentation Group.

The behaviour of these functions is affected by settings in `php.ini`.

Table 6.24 MySQL Configuration Options

Name	Default	Changeable	Changelog
<code>mysql.allow_local_infile</code>	"1"	PHP_INI_SYSTEM	
<code>mysql.allow_persistent</code>	"1"	PHP_INI_SYSTEM	
<code>mysql.max_persistent</code>	"-1"	PHP_INI_SYSTEM	
<code>mysql.max_links</code>	"-1"	PHP_INI_SYSTEM	
<code>mysql.trace_mode</code>	"0"	PHP_INI_ALL	Available since PHP 4.3.0.
<code>mysql.default_port</code>	NULL	PHP_INI_ALL	
<code>mysql.default_socket</code>	NULL	PHP_INI_ALL	Available since PHP 4.0.1.
<code>mysql.default_host</code>	NULL	PHP_INI_ALL	
<code>mysql.default_user</code>	NULL	PHP_INI_ALL	
<code>mysql.default_password</code>	NULL	PHP_INI_ALL	
<code>mysql.connect_timeout</code>	"60"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP <= 4.3.2. Available since PHP 4.3.0.

For further details and definitions of the `PHP_INI_*` modes, see the <http://www.php.net/manual/en/configuration.changes.modes>.

Here's a short explanation of the configuration directives.

<code>mysql.allow_local_infile</code> integer	Allow accessing, from PHP's perspective, local files with <code>LOAD DATA</code> statements
<code>mysql.allow_persistent</code> boolean	Whether to allow persistent connections to MySQL.
<code>mysql.max_persistent</code> integer	The maximum number of persistent MySQL connections per process.
<code>mysql.max_links</code> integer	The maximum number of MySQL connections per process, including persistent connections.
<code>mysql.trace_mode</code> boolean	Trace mode. When <code>mysql.trace_mode</code> is enabled, warnings for table/index scans, non free result sets, and SQL-Errors will be displayed. (Introduced in PHP 4.3.0)
<code>mysql.default_port</code> string	The default TCP port number to use when connecting to the database server if no other port is specified. If no default is specified, the port will be obtained from the <code>MYSQL_TCP_PORT</code> environment variable, the <code>mysql-tcp</code> entry in <code>/etc/services</code> or the compile-time <code>MYSQL_PORT</code> constant, in that order. Win32 will only use the <code>MYSQL_PORT</code> constant.
<code>mysql.default_socket</code> string	The default socket name to use when connecting to a local database server if no other socket name is specified.
<code>mysql.default_host</code> string	The default server host to use when connecting to the database server if no other host is specified. Doesn't apply in SQL safe mode .
<code>mysql.default_user</code> string	The default user name to use when connecting to the database server if no other name is specified. Doesn't apply in SQL safe mode .
<code>mysql.default_password</code> string	The default password to use when connecting to the database server if no other password is specified. Doesn't apply in SQL safe mode .
<code>mysql.connect_timeout</code> integer	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.

6.5.1.4 Resource Types

[Copyright 1997-2014 the PHP Documentation Group.](#)

There are two resource types used in the MySQL module. The first one is the link identifier for a database connection, the second a resource which holds the result of a query.

6.5.2 Changelog

[Copyright 1997-2014 the PHP Documentation Group.](#)

The following changes have been made to classes/functions/methods of this extension.

General Changelog for the ext/mysql extension

This changelog references the ext/mysql extension.

Global ext/mysql changes

The following is a list of changes to the entire ext/mysql extension.

Version	Description
7.0.0	This extension was removed from PHP. For details, see Section 6.2.3, “Choosing an API” .
5.5.0	This extension has been deprecated. Connecting to a MySQL database via mysql_connect , mysql_pconnect or an implicit connection via any other mysql_* function will generate an E_DEPRECATED error.
5.5.0	All of the old deprecated functions and aliases now emit E_DEPRECATED errors. These functions are: mysql(), mysql_fieldname(), mysql_fieldtable(), mysql_fieldlen(), mysql_fieldtype(), mysql_fieldflags(), mysql_selectdb(), mysql_createdb(), mysql_dropdb(), mysql_freeresult(), mysql_numfields(), mysql_numrows(), mysql_listdbs(), mysql_listtables(), mysql_listfields(), mysql_db_name(), mysql_dbname(), mysql_tablename(), and mysql_table_name().

Changes to existing functions

The following list is a compilation of changelog entries from the ext/mysql functions.

6.5.3 Predefined Constants

Copyright 1997-2014 the PHP Documentation Group.

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Since PHP 4.3.0 it is possible to specify additional client flags for the [mysql_connect](#) and [mysql_pconnect](#) functions. The following constants are defined:

Table 6.25 MySQL client constants

Constant	Description
MYSQL_CLIENT_COMPRESS	Use compression protocol
MYSQL_CLIENT_IGNORE_SPACE	Allow space after function names
MYSQL_CLIENT_INTERACTIVE	Allow interactive_timeout seconds (instead of wait_timeout) of inactivity before closing the connection.
MYSQL_CLIENT_SSL	Use SSL encryption. This flag is only available with version 4.x of the MySQL client library or newer. Version 3.23.x is bundled both with PHP 4 and Windows binaries of PHP 5.

The function [mysql_fetch_array](#) uses a constant for the different types of result arrays. The following constants are defined:

Table 6.26 MySQL fetch constants

Constant	Description
<code>MYSQL_ASSOC</code>	Columns are returned into the array having the fieldname as the array index.
<code>MYSQL_BOTH</code>	Columns are returned into the array having both a numerical index and the fieldname as the array index.
<code>MYSQL_NUM</code>	Columns are returned into the array having a numerical index to the fields. This index starts with 0, the first field in the result.

6.5.4 Examples

Copyright 1997-2014 the PHP Documentation Group.

6.5.4.1 MySQL extension overview example

Copyright 1997-2014 the PHP Documentation Group.

This simple example shows how to connect, execute a query, print resulting rows and disconnect from a MySQL database.

Example 6.150 MySQL extension overview example

```
<?php
// Connecting, selecting database
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    or die('Could not connect: ' . mysql_error());
echo 'Connected successfully';
mysql_select_db('my_database') or die('Could not select database');
// Performing SQL query
$query = 'SELECT * FROM my_table';
$result = mysql_query($query) or die('Query failed: ' . mysql_error());
// Printing results in HTML
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
        echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
}
echo "</table>\n";
// Free resultset
mysql_free_result($result);
// Closing connection
mysql_close($link);
?>
```

6.5.5 MySQL Functions

Copyright 1997-2014 the PHP Documentation Group.

Note

Most MySQL functions accept *link_identifier* as the last optional parameter. If it is not provided, last opened connection is used. If it doesn't exist,

connection is tried to establish with default parameters defined in [php.ini](#). If it is not successful, functions return [FALSE](#).

6.5.5.1 `mysql_affected_rows`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_affected_rows`

Get number of affected rows in previous MySQL operation

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_affected_rows](#)
[PDOStatement::rowCount](#)

Description

```
int mysql_affected_rows(
    resource link_identifier
    = =NULL);
```

Get the number of affected rows by the last INSERT, UPDATE, REPLACE or DELETE query associated with [link_identifier](#).

Parameters

[link_identifier](#)

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns the number of affected rows on success, and -1 if the last query failed.

If the last query was a DELETE query with no WHERE clause, all of the records will have been deleted from the table but this function will return zero with MySQL versions prior to 4.1.2.

When using UPDATE, MySQL will not update columns where the new value is the same as the old value. This creates the possibility that [mysql_affected_rows](#) may not actually equal the number of rows matched, only the number of rows that were literally affected by the query.

The REPLACE statement first deletes the record with the same primary key and then inserts the new record. This function returns the number of deleted records plus the number of inserted records.

In the case of "INSERT ... ON DUPLICATE KEY UPDATE" queries, the return value will be [1](#) if an insert was performed, or [2](#) for an update of an existing row.

Examples

Example 6.151 `mysql_affected_rows` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('mydb');
/* this should return the correct numbers of deleted records */
mysql_query('DELETE FROM mytable WHERE id < 10');
printf("Records deleted: %d\n", mysql_affected_rows());
/* with a where clause that is never true, it should return 0 */
mysql_query('DELETE FROM mytable WHERE 0');
printf("Records deleted: %d\n", mysql_affected_rows());
?>
```

The above example will output something similar to:

```
Records deleted: 10
Records deleted: 0
```

Example 6.152 `mysql_affected_rows` example using transactions

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('mydb');
/* Update records */
mysql_query("UPDATE mytable SET used=1 WHERE id < 10");
printf("Updated records: %d\n", mysql_affected_rows());
mysql_query("COMMIT");
?>
```

The above example will output something similar to:

```
Updated Records: 10
```

Notes

Transactions

If you are using transactions, you need to call `mysql_affected_rows` after your INSERT, UPDATE, or DELETE query, not after the COMMIT.

SELECT Statements

To retrieve the number of rows returned by a SELECT, it is possible to use `mysql_num_rows`.

Cascaded Foreign Keys

`mysql_affected_rows` does not count rows affected implicitly through the use of ON DELETE CASCADE and/or ON UPDATE CASCADE in foreign key constraints.

See Also

[mysql_num_rows](#)
[mysql_info](#)

6.5.5.2 [mysql_client_encoding](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_client_encoding](#)

Returns the name of the character set

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_character_set_name](#)

Description

```
string mysql_client_encoding(  
    resource link_identifier  
    = =NULL);
```

Retrieves the [character_set](#) variable from MySQL.

Parameters

[link_identifier](#)

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns the default character set name for the current connection.

Examples**Example 6.153 [mysql_client_encoding](#) example**

```
<?php  
$link    = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
$charset = mysql_client_encoding($link);  
echo "The current character set is: $charset\n";  
?>
```

The above example will output something similar to:

```
The current character set is: latin1
```

See Also

[mysql_set_charset](#)
[mysql_real_escape_string](#)

6.5.5.3 mysql_close

Copyright 1997-2014 the PHP Documentation Group.

- **mysql_close**

Close MySQL connection

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_close](#)

PDO: Assign the value of [NULL](#) to the PDO object

Description

```
bool mysql_close(  
    resource link_identifier  
    = =NULL);
```

[mysql_close](#) closes the non-persistent connection to the MySQL server that's associated with the specified link identifier. If *link_identifier* isn't specified, the last opened link is used.

Open non-persistent MySQL connections and result sets are automatically destroyed when a PHP script finishes its execution. So, while explicitly closing open connections and freeing result sets is optional, doing so is recommended. This will immediately return resources to PHP and MySQL, which can improve performance. For related information, see [freeing resources](#)

Parameters

link_identifier The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples**Example 6.154 mysql_close example**

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
if (!$link) {  
    die('Could not connect: ' . mysql_error());  
}  
echo 'Connected successfully';  
mysql_close($link);  
?>
```

The above example will output:

```
Connected successfully
```

Notes

Note

`mysql_close` will not close persistent links created by `mysql_pconnect`. For additional details, see the manual page on [persistent connections](#).

See Also

`mysql_connect`
`mysql_free_result`

6.5.5.4 `mysql_connect`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_connect`

Open a connection to a MySQL Server

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_connect`
`PDO::__construct`

Description

```
resource mysql_connect(  
    string server  
        = ini_get("mysql.default_host"),  
    string username  
        = ini_get("mysql.default_user"),  
    string password  
        = ini_get("mysql.default_password"),  
    bool new_link  
        = false,  
    int client_flags  
        = 0);
```

Opens or reuses a connection to a MySQL server.

Parameters

`server`

The MySQL server. It can also include a port number. e.g. "hostname:port" or a path to a local socket e.g. "unix:/path/to/socket" for the localhost.

If the PHP directive `mysql.default_host` is undefined (default), then the default value is 'localhost:3306'. In [SQL safe mode](#), this parameter is ignored and value 'localhost:3306' is always used.

<i>username</i>	The username. Default value is defined by mysql.default_user . In SQL safe mode , this parameter is ignored and the name of the user that owns the server process is used.
<i>password</i>	The password. Default value is defined by mysql.default_password . In SQL safe mode , this parameter is ignored and empty password is used.
<i>new_link</i>	If a second call is made to mysql_connect with the same arguments, no new link will be established, but instead, the link identifier of the already opened link will be returned. The <i>new_link</i> parameter modifies this behavior and makes mysql_connect always open a new link, even if mysql_connect was called before with the same parameters. In SQL safe mode , this parameter is ignored.
<i>client_flags</i>	The <i>client_flags</i> parameter can be a combination of the following constants: 128 (enable LOAD DATA LOCAL handling), MYSQL_CLIENT_SSL , MYSQL_CLIENT_COMPRESS , MYSQL_CLIENT_IGNORE_SPACE or MYSQL_CLIENT_INTERACTIVE . Read the section about Table 6.25, “MySQL client constants” for further information. In SQL safe mode , this parameter is ignored.

Return Values

Returns a MySQL link identifier on success or [FALSE](#) on failure.

Changelog

Version	Description
5.5.0	This function will generate an E_DEPRECATED error.

Examples

Example 6.155 [mysql_connect](#) example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Example 6.156 [mysql_connect](#) example using [hostname:port](#) syntax

```
<?php
// we connect to example.com and port 3307
$link = mysql_connect('example.com:3307', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
```



```
// we connect to localhost at port 3307
$link = mysql_connect('127.0.0.1:3307', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Example 6.157 `mysql_connect` example using `"/path/to/socket"` syntax

```
<?php
// we connect to localhost and socket e.g. /tmp/mysql.sock
// variant 1: omit localhost
$link = mysql_connect('/:tmp/mysql', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
// variant 2: with localhost
$link = mysql_connect('localhost:/tmp/mysql.sock', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
mysql_close($link);
?>
```

Notes

Note

Whenever you specify "localhost" or "localhost:port" as server, the MySQL client library will override this and try to connect to a local socket (named pipe on Windows). If you want to use TCP/IP, use "127.0.0.1" instead of "localhost". If the MySQL client library tries to connect to the wrong local socket, you should set the correct path as `mysql.default_host string` in your PHP configuration and leave the server field blank.

Note

The link to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling `mysql_close`.

Note

You can suppress the error message on failure by prepending a `@` to the function name.

Note

Error "Can't create TCP/IP socket (10106)" usually means that the `variables_order` configure directive doesn't contain character `E`. On Windows, if the environment is not copied the `SYSTEMROOT` environment variable won't be available and PHP will have problems loading Winsock.

See Also

[`mysql\_pconnect`](#)

`mysql_close`

6.5.5.5 `mysql_create_db`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_create_db`

Create a MySQL database

Warning

This function was deprecated in PHP 4.3.0, and it and the entire [original MySQL extension](#) was removed in PHP 7.0.0. Instead, use either the actively developed [MySQLi](#) or [PDO_MySQL](#) extensions. See also the [MySQL: choosing an API guide](#) and its [related FAQ entry](#) for additional information. Alternatives to this function include:

`mysqli_query`
`PDO::query`

Description

```
bool mysql_create_db(
    string database_name,
    resource link_identifier
    = =NULL);
```

`mysql_create_db` attempts to create a new database on the server associated with the specified link identifier.

Parameters

<i>database_name</i>	The name of the database being created.
<i>link_identifier</i>	The MySQL connection. If the link identifier is not specified, the last link opened by <code>mysql_connect</code> is assumed. If no such link is found, it will try to create one as if <code>mysql_connect</code> had been called with no arguments. If no connection is found or established, an E_WARNING level error is generated.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 6.158 `mysql_create_db` alternative example

The function `mysql_create_db` is deprecated. It is preferable to use `mysql_query` to issue an `sql CREATE DATABASE` statement instead.

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
$sql = 'CREATE DATABASE my_db';
if (mysql_query($sql, $link)) {
    echo "Database my_db created successfully\n";
} else {
```

```

    echo 'Error creating database: ' . mysql_error() . "\n";
}
?>

```

The above example will output something similar to:

```
Database my_db created successfully
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
[mysql_createdb](#)

Note

This function will not be available if the MySQL extension was built against a MySQL 4.x client library.

See Also

[mysql_query](#)
[mysql_select_db](#)

6.5.5.6 [mysql_data_seek](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_data_seek](#)

Move internal result pointer

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_data_seek](#)
[PDO::FETCH_ORI_ABS](#)

Description

```
bool mysql_data_seek(
    resource result,
    int row_number);
```

[mysql_data_seek](#) moves the internal row pointer of the MySQL result associated with the specified result identifier to point to the specified row number. The next call to a MySQL fetch function, such as [mysql_fetch_assoc](#), would return that row.

row_number starts at 0. The *row_number* should be a value in the range from 0 to [mysql_num_rows](#) - 1. However if the result set is empty ([mysql_num_rows](#) == 0), a seek to 0 will fail with a [E_WARNING](#) and [mysql_data_seek](#) will return [FALSE](#).

Parameters

<i>result</i>	The result resource that is being evaluated. This result comes from a call to mysql_query .
<i>row_number</i>	The desired row number of the new result pointer.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples**Example 6.159 [mysql_data_seek](#) example**

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
$db_selected = mysql_select_db('sample_db');
if (!$db_selected) {
    die('Could not select database: ' . mysql_error());
}
$query = 'SELECT last_name, first_name FROM friends';
$result = mysql_query($query);
if (!$result) {
    die('Query failed: ' . mysql_error());
}
/* fetch rows in reverse order */
for ($i = mysql_num_rows($result) - 1; $i >= 0; $i--) {
    if (!mysql_data_seek($result, $i)) {
        echo "Cannot seek to row $i: " . mysql_error() . "\n";
        continue;
    }
    if (!($row = mysql_fetch_assoc($result))) {
        continue;
    }
    echo $row['last_name'] . ' ' . $row['first_name'] . "<br />\n";
}
mysql_free_result($result);
?>
```

Notes**Note**

The function [mysql_data_seek](#) can be used in conjunction only with [mysql_query](#), not with [mysql_unbuffered_query](#).

See Also

[mysql_query](#)
[mysql_num_rows](#)
[mysql_fetch_row](#)
[mysql_fetch_assoc](#)
[mysql_fetch_array](#)
[mysql_fetch_object](#)

6.5.5.7 [mysql_db_name](#)

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_db_name`

Retrieves database name from the call to `mysql_list_dbs`

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

Query: `SELECT DATABASE()`

Description

```
string mysql_db_name(
    resource result,
    int row,
    mixed field
    = NULL);
```

Retrieve the database name from a call to `mysql_list_dbs`.

Parameters

result The result pointer from a call to `mysql_list_dbs`.

row The index into the result set.

field The field name.

Return Values

Returns the database name on success, and `FALSE` on failure. If `FALSE` is returned, use `mysql_error` to determine the nature of the error.

Changelog

Version	Description
5.5.0	The <code>mysql_list_dbs</code> function is deprecated, and emits an <code>E_DEPRECATED</code> level error.

Examples

Example 6.160 `mysql_db_name` example

```
<?php
error_reporting(E_ALL);
$link = mysql_connect('dbhost', 'username', 'password');
$db_list = mysql_list_dbs($link);
$i = 0;
$cnt = mysql_num_rows($db_list);
while ($i < $cnt) {
    echo mysql_db_name($db_list, $i) . "\n";
    $i++;
}
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
`mysql_dbname`

See Also

`mysql_list_dbs`
`mysql_tablename`

6.5.5.8 `mysql_db_query`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_db_query`

Selects a database and executes a query on it

Warning

This function was deprecated in PHP 5.3.0, and it and the entire [original MySQL extension](#) was removed in PHP 7.0.0. Instead, use either the actively developed [MySQLi](#) or [PDO_MySQL](#) extensions. See also the [MySQL: choosing an API](#) guide and its [related FAQ entry](#) for additional information. Alternatives to this function include:

`mysqli_select_db` then the query
`PDO::__construct`

Description

```
resource mysql_db_query(  
    string database,  
    string query,  
    resource link_identifier  
    = NULL);
```

`mysql_db_query` selects a database, and executes a query on it.

Parameters

database

The name of the database that will be selected.

query

The MySQL query.

Data inside the query should be [properly escaped](#).

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns a positive MySQL result resource to the query result, or [FALSE](#) on error. The function also returns [TRUE/FALSE](#) for [INSERT/UPDATE/DELETE](#) queries to indicate success/failure.

Changelog

Version	Description
5.3.0	This function now throws an E_DEPRECATED notice.

Examples

Example 6.161 `mysql_db_query` alternative example

```
<?php
if (!$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')) {
    echo 'Could not connect to mysql';
    exit;
}
if (!mysql_select_db('mysql_dbname', $link)) {
    echo 'Could not select database';
    exit;
}
$sql = 'SELECT foo FROM bar WHERE id = 42';
$result = mysql_query($sql, $link);
if (!$result) {
    echo "DB Error, could not query the database\n";
    echo 'MySQL Error: ' . mysql_error();
    exit;
}
while ($row = mysql_fetch_assoc($result)) {
    echo $row['foo'];
}
mysql_free_result($result);
?>
```

Notes

Note

Be aware that this function does *NOT* switch back to the database you were connected before. In other words, you can't use this function to *temporarily* run a sql query on another database, you would have to manually switch back. Users are strongly encouraged to use the [database.table](#) syntax in their sql queries or [mysql_select_db](#) instead of this function.

See Also

[mysql_query](#)
[mysql_select_db](#)

6.5.5.9 `mysql_drop_db`

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_drop_db](#)

Drop (delete) a MySQL database

Warning

This function was deprecated in PHP 4.3.0, and it and the entire [original MySQL extension](#) was removed in PHP 7.0.0. Instead, use either the actively developed [MySQLi](#) or [PDO_MySQL](#) extensions. See also the [MySQL: choosing an API](#) guide and its [related FAQ entry](#) for additional information. Alternatives to this function include:

Execute a [DROP DATABASE](#) query

Description

```
bool mysql_drop_db(
```

```
string database_name,  
resource link_identifier  
= =NULL);
```

[mysql_drop_db](#) attempts to drop (remove) an entire database from the server associated with the specified link identifier. This function is deprecated, it is preferable to use [mysql_query](#) to issue an `sql DROP DATABASE` statement instead.

Parameters

database_name	The name of the database that will be deleted.
link_identifier	The MySQL connection. If the link identifier is not specified, the last link opened by mysql_connect is assumed. If no such link is found, it will try to create one as if mysql_connect had been called with no arguments. If no connection is found or established, an E_WARNING level error is generated.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 6.162 [mysql_drop_db](#) alternative example

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
if (!$link) {  
    die('Could not connect: ' . mysql_error());  
}  
$sql = 'DROP DATABASE my_db';  
if (mysql_query($sql, $link)) {  
    echo "Database my_db was successfully dropped\n";  
} else {  
    echo 'Error dropping database: ' . mysql_error() . "\n";  
}  
?>
```

Notes

Warning

This function will not be available if the MySQL extension was built against a MySQL 4.x client library.

Note

For backward compatibility, the following deprecated alias may be used:
[mysql_dropdb](#)

See Also

[mysql_query](#)

6.5.5.10 [mysql_errno](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_errno](#)

Returns the numerical value of the error message from previous MySQL operation

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_errno`
`PDO::errorCode`

Description

```
int mysql_errno(
    resource link_identifier
    = =NULL);
```

Returns the error number from the last MySQL function.

Errors coming back from the MySQL database backend no longer issue warnings. Instead, use [mysql_errno](#) to retrieve the error code. Note that this function only returns the error code from the most recently executed MySQL function (not including [mysql_error](#) and [mysql_errno](#)), so if you want to use it, make sure you check the value before calling another MySQL function.

Parameters

link_identifier The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns the error number from the last MySQL function, or 0 (zero) if no error occurred.

Examples

Example 6.163 [mysql_errno](#) example

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
if (!mysql_select_db("nonexistentdb", $link)) {
    echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
}
mysql_select_db("kossu", $link);
if (!mysql_query("SELECT * FROM nonexistenttable", $link)) {
    echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
}
?>
```

The above example will output something similar to:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

See Also

[mysql_error](#)
[MySQL error codes](#)

6.5.5.11 mysql_error

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_error](#)

Returns the text of the error message from previous MySQL operation

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_error](#)
[PDO::errorInfo](#)

Description

```
string mysql_error(
    resource link_identifier
    = =NULL);
```

Returns the error text from the last MySQL function. Errors coming back from the MySQL database backend no longer issue warnings. Instead, use [mysql_error](#) to retrieve the error text. Note that this function only returns the error text from the most recently executed MySQL function (not including [mysql_error](#) and [mysql_errno](#)), so if you want to use it, make sure you check the value before calling another MySQL function.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns the error text from the last MySQL function, or '' (empty string) if no error occurred.

Examples**Example 6.164 mysql_error example**

```
<?php
$link = mysql_connect("localhost", "mysql_user", "mysql_password");
mysql_select_db("nonexistentdb", $link);
echo mysql_errno($link) . ": " . mysql_error($link). "\n";
mysql_select_db("kossu", $link);
mysql_query("SELECT * FROM nonexistenttable", $link);
echo mysql_errno($link) . ": " . mysql_error($link) . "\n";
?>
```

The above example will output something similar to:

```
1049: Unknown database 'nonexistentdb'
1146: Table 'kossu.nonexistenttable' doesn't exist
```

See Also

[mysql_errno](#)
[MySQL error codes](#)

6.5.5.12 [mysql_escape_string](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_escape_string](#)

Escapes a string for use in a `mysql_query`

Warning

This function was deprecated in PHP 4.3.0, and it and the entire [original MySQL extension](#) was removed in PHP 7.0.0. Instead, use either the actively developed [MySQLi](#) or [PDO_MySQL](#) extensions. See also the [MySQL: choosing an API](#) guide and its [related FAQ entry](#) for additional information. Alternatives to this function include:

[mysqli_escape_string](#)
[PDO::quote](#)

Description

```
string mysql_escape_string(
    string unescaped_string);
```

This function will escape the [unescaped_string](#), so that it is safe to place it in a `mysql_query`. This function is deprecated.

This function is identical to [mysql_real_escape_string](#) except that [mysql_real_escape_string](#) takes a connection handler and escapes the string according to the current character set. [mysql_escape_string](#) does not take a connection argument and does not respect the current charset setting.

Parameters

[unescaped_string](#) The string that is to be escaped.

Return Values

Returns the escaped string.

Changelog

Version	Description
5.3.0	This function now throws an E_DEPRECATED notice.

Version	Description
4.3.0	This function became deprecated, do not use this function. Instead, use mysql_real_escape_string .

Examples

Example 6.165 [mysql_escape_string](#) example

```
<?php
$item = "Zak's Laptop";
$escaped_item = mysql_escape_string($item);
printf("Escaped string: %s\n", $escaped_item);
?>
```

The above example will output:

```
Escaped string: Zak\'s Laptop
```

Notes

Note

[mysql_escape_string](#) does not escape % and _.

See Also

[mysql_real_escape_string](#)

[addslashes](#)

The [magic_quotes_gpc](#) directive.

6.5.5.13 [mysql_fetch_array](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_fetch_array](#)

Fetch a result row as an associative array, a numeric array, or both

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_fetch_array](#)
[PDOStatement::fetch](#)

Description

```
array mysql_fetch_array(
    resource result,
    int result_type
    = =MYSQL_BOTH);
```

Returns an array that corresponds to the fetched row and moves the internal data pointer ahead.

Parameters

<i>result</i>	The result resource that is being evaluated. This result comes from a call to <code>mysql_query</code> .
<i>result_type</i>	The type of array that is to be fetched. It's a constant and can take the following values: <code>MYSQL_ASSOC</code> , <code>MYSQL_NUM</code> , and <code>MYSQL_BOTH</code> .

Return Values

Returns an array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows. The type of returned array depends on how *result_type* is defined. By using `MYSQL_BOTH` (default), you'll get an array with both associative and number indices. Using `MYSQL_ASSOC`, you only get associative indices (as `mysql_fetch_assoc` works), using `MYSQL_NUM`, you only get number indices (as `mysql_fetch_row` works).

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you must use the numeric index of the column or make an alias for the column. For aliased columns, you cannot access the contents with the original column name.

Examples

Example 6.166 Query with aliased duplicate field names

```
SELECT table1.field AS foo, table2.field AS bar FROM table1, table2
```

Example 6.167 `mysql_fetch_array` with `MYSQL_NUM`

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    printf("ID: %s  Name: %s", $row[0], $row[1]);
}
mysql_free_result($result);
?>
```

Example 6.168 `mysql_fetch_array` with `MYSQL_ASSOC`

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
    printf("ID: %s  Name: %s", $row["id"], $row["name"]);
}
mysql_free_result($result);
?>
```

Example 6.169 `mysql_fetch_array` with `MYSQL_BOTH`

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password") or
    die("Could not connect: " . mysql_error());
mysql_select_db("mydb");
$result = mysql_query("SELECT id, name FROM mytable");
while ($row = mysql_fetch_array($result, MYSQL_BOTH)) {
    printf ("ID: %s  Name: %s", $row[0], $row["name"]);
}
mysql_free_result($result);
?>
```

Notes**Performance**

An important thing to note is that using `mysql_fetch_array` is *not significantly* slower than using `mysql_fetch_row`, while it provides a significant added value.

Note

Field names returned by this function are *case-sensitive*.

Note

This function sets NULL fields to the PHP `NULL` value.

See Also

`mysql_fetch_row`
`mysql_fetch_assoc`
`mysql_data_seek`
`mysql_query`

6.5.5.14 `mysql_fetch_assoc`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_fetch_assoc`

Fetch a result row as an associative array

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the `MySQLi` or `PDO_MySQL` extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_fetch_assoc`
`PDOStatement::fetch(PDO::FETCH_ASSOC)`

Description

```
array mysql_fetch_assoc(
    resource result);
```

Returns an associative array that corresponds to the fetched row and moves the internal data pointer ahead. `mysql_fetch_assoc` is equivalent to calling `mysql_fetch_array` with `MYSQL_ASSOC` for the optional second parameter. It only returns an associative array.

Parameters

result The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

Returns an associative array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you either need to access the result with numeric indices by using `mysql_fetch_row` or add alias names. See the example at the `mysql_fetch_array` description about aliases.

Examples

Example 6.170 An expanded `mysql_fetch_assoc` example

```
<?php
$conn = mysql_connect("localhost", "mysql_user", "mysql_password");
if (!$conn) {
    echo "Unable to connect to DB: " . mysql_error();
    exit;
}
if (!mysql_select_db("mydbname")) {
    echo "Unable to select mydbname: " . mysql_error();
    exit;
}
$sql = "SELECT id as userid, fullname, userstatus
        FROM   sometable
        WHERE  userstatus = 1";
$result = mysql_query($sql);
if (!$result) {
    echo "Could not successfully run query ($sql) from DB: " . mysql_error();
    exit;
}
if (mysql_num_rows($result) == 0) {
    echo "No rows found, nothing to print so am exiting";
    exit;
}
// While a row of data exists, put that row in $row as an associative array
// Note: If you're expecting just one row, no need to use a loop
// Note: If you put extract($row); inside the following loop, you'll
//       then create $userid, $fullname, and $userstatus
while ($row = mysql_fetch_assoc($result)) {
    echo $row["userid"];
    echo $row["fullname"];
    echo $row["userstatus"];
}
mysql_free_result($result);
?>
```

Notes

Performance

An important thing to note is that using `mysql_fetch_assoc` is *not significantly* slower than using `mysql_fetch_row`, while it provides a significant added value.

Note

Field names returned by this function are *case-sensitive*.

Note

This function sets NULL fields to the PHP [NULL](#) value.

See Also

[mysql_fetch_row](#)
[mysql_fetch_array](#)
[mysql_data_seek](#)
[mysql_query](#)
[mysql_error](#)

6.5.5.15 [mysql_fetch_field](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_fetch_field](#)

Get column information from a result and return as an object

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_fetch_field](#)
[PDOStatement::getColumnMeta](#)

Description

```
object mysql_fetch_field(  
    resource result,  
    int field_offset  
    = 0);
```

Returns an object containing field information. This function can be used to obtain information about fields in the provided query result.

Parameters

<i>result</i>	The result resource that is being evaluated. This result comes from a call to mysql_query .
<i>field_offset</i>	The numerical field offset. If the field offset is not specified, the next field that was not yet retrieved by this function is retrieved. The <i>field_offset</i> starts at 0.

Return Values

Returns an object containing field information. The properties of the object are:

- name - column name
- table - name of the table the column belongs to, which is the alias name if one is defined
- max_length - maximum length of the column

- `not_null` - 1 if the column cannot be `NULL`
- `primary_key` - 1 if the column is a primary key
- `unique_key` - 1 if the column is a unique key
- `multiple_key` - 1 if the column is a non-unique key
- `numeric` - 1 if the column is numeric
- `blob` - 1 if the column is a BLOB
- `type` - the type of the column
- `unsigned` - 1 if the column is unsigned
- `zerofill` - 1 if the column is zero-filled

Examples

Example 6.171 `mysql_fetch_field` example

```
<?php
$conn = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$conn) {
    die('Could not connect: ' . mysql_error());
}
mysql_select_db('database');
$result = mysql_query('select * from table');
if (!$result) {
    die('Query failed: ' . mysql_error());
}
/* get column metadata */
$i = 0;
while ($i < mysql_num_fields($result)) {
    echo "Information for column $i:<br />\n";
    $meta = mysql_fetch_field($result, $i);
    if (!$meta) {
        echo "No information available<br />\n";
    }
    echo "<pre>
blob:      $meta->blob
max_length: $meta->max_length
multiple_key: $meta->multiple_key
name:      $meta->name
not_null:   $meta->not_null
numeric:    $meta->numeric
primary_key: $meta->primary_key
table:      $meta->table
type:       $meta->type
unique_key: $meta->unique_key
unsigned:   $meta->unsigned
zerofill:   $meta->zerofill
</pre>";
    $i++;
}
mysql_free_result($result);
?>
```

Notes

Note

Field names returned by this function are *case-sensitive*.

Note

If field or tablename are aliased in the SQL query the aliased name will be returned. The original name can be retrieved for instance by using `mysqli_result::fetch_field`.

See Also

`mysql_field_seek`

6.5.5.16 `mysql_fetch_lengths`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_fetch_lengths`

Get the length of each output in a result

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_fetch_lengths`
`PDOStatement::getColumnMeta`

Description

```
array mysql_fetch_lengths(  
    resource result);
```

Returns an array that corresponds to the lengths of each field in the last row fetched by MySQL.

`mysql_fetch_lengths` stores the lengths of each result column in the last row returned by `mysql_fetch_row`, `mysql_fetch_assoc`, `mysql_fetch_array`, and `mysql_fetch_object` in an array, starting at offset 0.

Parameters

result

The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

An array of lengths on success or `FALSE` on failure.

Examples**Example 6.172 A `mysql_fetch_lengths` example**

```
<?php  
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");  
if (!$result) {  
    echo 'Could not run query: ' . mysql_error();  
    exit;  
}  
$row = mysql_fetch_assoc($result);  
$lengths = mysql_fetch_lengths($result);
```

```
print_r($row);
print_r($lengths);
?>
```

The above example will output something similar to:

```
Array
(
    [id] => 42
    [email] => user@example.com
)
Array
(
    [0] => 2
    [1] => 16
)
```

See Also

[mysql_field_len](#)
[mysql_fetch_row](#)
[strlen](#)

6.5.5.17 [mysql_fetch_object](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_fetch_object](#)

Fetch a result row as an object

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_fetch_object](#)
`PDOStatement::fetch(PDO::FETCH_OBJ)`

Description

```
object mysql_fetch_object(
    resource result,
    string class_name,
    array params);
```

Returns an object with properties that correspond to the fetched row and moves the internal data pointer ahead.

Parameters

result

The result resource that is being evaluated. This result comes from a call to [mysql_query](#).

class_name

The name of the class to instantiate, set the properties of and return. If not specified, a [stdClass](#) object is returned.

params

An optional array of parameters to pass to the constructor for *class_name* objects.

Return Values

Returns an object with string properties that correspond to the fetched row, or **FALSE** if there are no more rows.

Examples

Example 6.173 `mysql_fetch_object` example

```
<?php
mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");
$result = mysql_query("select * from mytable");
while ($row = mysql_fetch_object($result)) {
    echo $row->user_id;
    echo $row->fullname;
}
mysql_free_result($result);
?>
```

Example 6.174 `mysql_fetch_object` example

```
<?php
class foo {
    public $name;
}
mysql_connect("hostname", "user", "password");
mysql_select_db("mydb");
$result = mysql_query("select name from mytable limit 1");
$obj = mysql_fetch_object($result, 'foo');
var_dump($obj);
?>
```

Notes

Performance

Speed-wise, the function is identical to `mysql_fetch_array`, and almost as quick as `mysql_fetch_row` (the difference is insignificant).

Note

`mysql_fetch_object` is similar to `mysql_fetch_array`, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

Note

Field names returned by this function are *case-sensitive*.

Note

This function sets NULL fields to the PHP **NULL** value.

See Also

```
mysql_fetch_array
mysql_fetch_assoc
mysql_fetch_row
mysql_data_seek
mysql_query
```

6.5.5.18 `mysql_fetch_row`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_fetch_row`

Get a result row as an enumerated array

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_fetch_row
PDOStatement::fetch(PDO::FETCH_NUM)
```

Description

```
array mysql_fetch_row(
    resource result);
```

Returns a numerical array that corresponds to the fetched row and moves the internal data pointer ahead.

Parameters

result The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

Returns an numerical array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows.

`mysql_fetch_row` fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Examples

Example 6.175 Fetching one row with `mysql_fetch_row`

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
$row = mysql_fetch_row($result);
echo $row[0]; // 42
echo $row[1]; // the email value
?>
```

Notes

Note

This function sets NULL fields to the PHP [NULL](#) value.

See Also

[mysql_fetch_array](#)
[mysql_fetch_assoc](#)
[mysql_fetch_object](#)
[mysql_data_seek](#)
[mysql_fetch_lengths](#)
[mysql_result](#)

6.5.5.19 mysql_field_flags

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_field_flags](#)

Get the flags associated with the specified field in a result

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_fetch_field_direct](#) [flags]
[PDOStatement::getColumnMeta](#) [flags]

Description

```
string mysql_field_flags(
    resource result,
    int field_offset);
```

[mysql_field_flags](#) returns the field flags of the specified field. The flags are reported as a single word per flag separated by a single space, so that you can split the returned value using [explode](#).

Parameters

result

The result resource that is being evaluated. This result comes from a call to [mysql_query](#).

field_offset

The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level [E_WARNING](#) is also issued.

Return Values

Returns a string of flags associated with the result or [FALSE](#) on failure.

The following flags are reported, if your version of MySQL is current enough to support them: "not_null", "primary_key", "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment" and "timestamp".

Examples

Example 6.176 A `mysql_field_flags` example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
$flags = mysql_field_flags($result, 0);
echo $flags;
print_r(explode(' ', $flags));
?>
```

The above example will output something similar to:

```
not_null primary_key auto_increment
Array
(
    [0] => not_null
    [1] => primary_key
    [2] => auto_increment
)
```

Notes**Note**

For backward compatibility, the following deprecated alias may be used:
`mysql_fieldflags`

See Also

`mysql_field_type`
`mysql_field_len`

6.5.5.20 `mysql_field_len`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_field_len`

Returns the length of the specified field

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_fetch_field_direct` [length]
`PDOStatement::getColumnMeta` [len]

Description

```
int mysql_field_len(
    resource result,
```

```
int field_offset);
```

`mysql_field_len` returns the length of the specified field.

Parameters

<code>result</code>	The result resource that is being evaluated. This result comes from a call to <code>mysql_query</code> .
<code>field_offset</code>	The numerical field offset. The <code>field_offset</code> starts at 0. If <code>field_offset</code> does not exist, an error of level <code>E_WARNING</code> is also issued.

Return Values

The length of the specified field index on success or `FALSE` on failure.

Examples

Example 6.177 `mysql_field_len` example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
// Will get the length of the id field as specified in the database
// schema.
$length = mysql_field_len($result, 0);
echo $length;
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
`mysql_fieldlen`

See Also

`mysql_fetch_lengths`
`strlen`

6.5.5.21 `mysql_field_name`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_field_name`

Get the name of the specified field in a result

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_fetch_field_direct` [name] or [orgname]
`PDOStatement::getColumnMeta` [name]

Description

```
string mysql_field_name(
    resource result,
    int field_offset);
```

`mysql_field_name` returns the name of the specified field index.

Parameters

result

The result resource that is being evaluated. This result comes from a call to `mysql_query`.

field_offset

The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level `E_WARNING` is also issued.

Return Values

The name of the specified field index on success or `FALSE` on failure.

Examples

Example 6.178 `mysql_field_name` example

```
<?php
/* The users table consists of three fields:
 *  user_id
 *  username
 *  password.
 */
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect to MySQL server: ' . mysql_error());
}
$dbname = 'mydb';
$db_selected = mysql_select_db($dbname, $link);
if (!$db_selected) {
    die("Could not set $dbname: " . mysql_error());
}
$res = mysql_query('select * from users', $link);
echo mysql_field_name($res, 0) . "\n";
echo mysql_field_name($res, 2);
?>
```

The above example will output:

```
user_id
password
```

Notes

Note

Field names returned by this function are *case-sensitive*.

Note

For backward compatibility, the following deprecated alias may be used:
`mysql_fieldname`

See Also

`mysql_field_type`
`mysql_field_len`

6.5.5.22 `mysql_field_seek`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_field_seek`

Set result pointer to a specified field offset

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_field_seek`
`PDOStatement::fetch` using the *`cursor_orientation`* and *`offset`* parameters

Description

```
bool mysql_field_seek(  
    resource result,  
    int field_offset);
```

Seeks to the specified field offset. If the next call to `mysql_fetch_field` doesn't include a field offset, the field offset specified in `mysql_field_seek` will be returned.

Parameters

result

The result resource that is being evaluated. This result comes from a call to `mysql_query`.

field_offset

The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level `E_WARNING` is also issued.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

See Also

`mysql_fetch_field`

6.5.5.23 `mysql_field_table`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_field_table`

Get name of the table the specified field is in

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_fetch_field_direct` [table] or [orgtable]
`PDOStatement::getColumnMeta` [table]

Description

```
string mysql_field_table(  
    resource result,  
    int field_offset);
```

Returns the name of the table that the specified field is in.

Parameters

<i>result</i>	The result resource that is being evaluated. This result comes from a call to mysql_query .
<i>field_offset</i>	The numerical field offset. The <i>field_offset</i> starts at 0. If <i>field_offset</i> does not exist, an error of level E_WARNING is also issued.

Return Values

The name of the table on success.

Examples**Example 6.179 A `mysql_field_table` example**

```
<?php  
$query = "SELECT account.*, country.* FROM account, country WHERE country.name = 'Portugal' AND account  
// get the result from the DB  
$result = mysql_query($query);  
// Lists the table name and then the field name  
for ($i = 0; $i < mysql_num_fields($result); ++$i) {  
    $table = mysql_field_table($result, $i);  
    $field = mysql_field_name($result, $i);  
    echo "$table: $field\n";  
}  
?>
```

Notes**Note**

For backward compatibility, the following deprecated alias may be used:
[mysql_fielddtable](#)

See Also

[mysql_list_tables](#)

6.5.5.24 `mysql_field_type`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_field_type`

Get the type of the specified field in a result

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_fetch_field_direct` [type]
`PDOStatement::getColumnMeta` [driver:decl_type] or [pdo_type]

Description

```
string mysql_field_type(
    resource result,
    int field_offset);
```

`mysql_field_type` is similar to the `mysql_field_name` function. The arguments are identical, but the field type is returned instead.

Parameters

result

The result resource that is being evaluated. This result comes from a call to `mysql_query`.

field_offset

The numerical field offset. The *field_offset* starts at 0. If *field_offset* does not exist, an error of level `E_WARNING` is also issued.

Return Values

The returned field type will be one of `"int"`, `"real"`, `"string"`, `"blob"`, and others as detailed in the [MySQL documentation](#).

Examples

Example 6.180 `mysql_field_type` example

```
<?php
mysql_connect("localhost", "mysql_username", "mysql_password");
mysql_select_db("mysql");
$result = mysql_query("SELECT * FROM func");
$fields = mysql_num_fields($result);
$rows = mysql_num_rows($result);
$table = mysql_field_table($result, 0);
echo "Your '" . $table . "' table has " . $fields . " fields and " . $rows . " record(s)\n";
echo "The table has the following fields:\n";
for ($i=0; $i < $fields; $i++) {
    $type = mysql_field_type($result, $i);
    $name = mysql_field_name($result, $i);
    $len = mysql_field_len($result, $i);
    $flags = mysql_field_flags($result, $i);
    echo $type . " " . $name . " " . $len . " " . $flags . "\n";
}
mysql_free_result($result);
mysql_close();
```

```
?>
```

The above example will output something similar to:

```
Your 'func' table has 4 fields and 1 record(s)
The table has the following fields:
string name 64 not_null primary_key binary
int ret 1 not_null
string dl 128 not_null
string type 9 not_null enum
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
`mysql_fieldtype`

See Also

`mysql_field_name`
`mysql_field_len`

6.5.5.25 `mysql_free_result`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_free_result`

Free result memory

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_free_result`
Assign the value of `NULL` to the PDO object, or
`PDOStatement::closeCursor`

Description

```
bool mysql_free_result(
    resource result);
```

`mysql_free_result` will free all memory associated with the result identifier *result*.

`mysql_free_result` only needs to be called if you are concerned about how much memory is being used for queries that return large result sets. All associated result memory is automatically freed at the end of the script's execution.

Parameters

result The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

If a non-resource is used for the [result](#), an error of level E_WARNING will be emitted. It's worth noting that [mysql_query](#) only returns a resource for SELECT, SHOW, EXPLAIN, and DESCRIBE queries.

Examples

Example 6.181 A [mysql_free_result](#) example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
/* Use the result, assuming we're done with it afterwards */
$row = mysql_fetch_assoc($result);
/* Now we free up the result and continue on with our script */
mysql_free_result($result);
echo $row['id'];
echo $row['email'];
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
[mysql_freeresult](#)

See Also

[mysql_query](#)
[is_resource](#)

6.5.5.26 [mysql_get_client_info](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_get_client_info](#)

Get MySQL client info

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_get_client_info](#)
[PDO::getAttribute\(PDO::ATTR_CLIENT_VERSION\)](#)

Description

```
string mysql_get_client_info();
```

`mysql_get_client_info` returns a string that represents the client library version.

Return Values

The MySQL client version.

Examples

Example 6.182 `mysql_get_client_info` example

```
<?php
printf("MySQL client info: %s\n", mysql_get_client_info());
?>
```

The above example will output something similar to:

```
MySQL client info: 3.23.39
```

See Also

`mysql_get_host_info`
`mysql_get_proto_info`
`mysql_get_server_info`

6.5.5.27 `mysql_get_host_info`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_get_host_info`

Get MySQL host info

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_get_host_info`
`PDO::getAttribute(PDO::ATTR_CONNECTION_STATUS)`

Description

```
string mysql_get_host_info(
    resource link_identifier
    = =NULL);
```

Describes the type of connection in use for the connection, including the server host name.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link

is found, it will try to create one as if `mysql_connect` had been called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a string describing the type of MySQL connection in use for the connection or `FALSE` on failure.

Examples

Example 6.183 `mysql_get_host_info` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
printf("MySQL host info: %s\n", mysql_get_host_info());
?>
```

The above example will output something similar to:

```
MySQL host info: Localhost via UNIX socket
```

See Also

[mysql_get_client_info](#)
[mysql_get_proto_info](#)
[mysql_get_server_info](#)

6.5.5.28 `mysql_get_proto_info`

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_get_proto_info](#)

Get MySQL protocol info

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_get_proto_info](#)

Description

```
int mysql_get_proto_info(
    resource link_identifier
    = NULL);
```

Retrieves the MySQL protocol.

Parameters*link_identifier*

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` had been called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the MySQL protocol on success or `FALSE` on failure.

Examples**Example 6.184 `mysql_get_proto_info` example**

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
printf("MySQL protocol version: %s\n", mysql_get_proto_info());
?>
```

The above example will output something similar to:

```
MySQL protocol version: 10
```

See Also

`mysql_get_client_info`
`mysql_get_host_info`
`mysql_get_server_info`

6.5.5.29 `mysql_get_server_info`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_get_server_info`

Get MySQL server info

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the `MySQLi` or `PDO_MySQL` extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_get_server_info`
`PDO::getAttribute(PDO::ATTR_SERVER_VERSION)`

Description

```
string mysql_get_server_info(
    resource link_identifier
```

```
= =NULL);
```

Retrieves the MySQL server version.

Parameters

link_identifier The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns the MySQL server version on success or [FALSE](#) on failure.

Examples

Example 6.185 [mysql_get_server_info](#) example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
printf("MySQL server version: %s\n", mysql_get_server_info());
?>
```

The above example will output something similar to:

```
MySQL server version: 4.0.1-alpha
```

See Also

[mysql_get_client_info](#)
[mysql_get_host_info](#)
[mysql_get_proto_info](#)
[phpversion](#)

6.5.5.30 [mysql_info](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_info](#)

Get information about the most recent query

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_info](#)

Description

```
string mysql_info(
    resource link_identifier
    = =NULL);
```

Returns detailed information about the last query.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` had been called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns information about the statement on success, or `FALSE` on failure. See the example below for which statements provide information, and what the returned value may look like. Statements that are not listed will return `FALSE`.

Examples

Example 6.186 Relevant MySQL Statements

Statements that return string values. The numbers are only for illustrating purpose; their values will correspond to the query.

```
INSERT INTO ... SELECT ...
String format: Records: 23 Duplicates: 0 Warnings: 0
INSERT INTO ... VALUES (...),(...),(...)...
String format: Records: 37 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...
String format: Records: 42 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE
String format: Records: 60 Duplicates: 0 Warnings: 0
UPDATE
String format: Rows matched: 65 Changed: 65 Warnings: 0
```

Notes

Note

`mysql_info` returns a non-`FALSE` value for the `INSERT ... VALUES` statement only if multiple value lists are specified in the statement.

See Also

`mysql_affected_rows`
`mysql_insert_id`
`mysql_stat`

6.5.5.31 `mysql_insert_id`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_insert_id`

Get the ID generated in the last query

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_insert_id](#)
[PDO::lastInsertId](#)

Description

```
int mysql_insert_id(  
    resource link_identifier  
    = =NULL);
```

Retrieves the ID generated for an AUTO_INCREMENT column by the previous query (usually INSERT).

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

The ID generated for an AUTO_INCREMENT column by the previous query on success, [0](#) if the previous query does not generate an AUTO_INCREMENT value, or [FALSE](#) if no MySQL connection was established.

Examples**Example 6.187 [mysql_insert_id](#) example**

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
if (!$link) {  
    die('Could not connect: ' . mysql_error());  
}  
mysql_select_db('mydb');  
mysql_query("INSERT INTO mytable (product) values ('kossu')");  
printf("Last inserted record has id %d\n", mysql_insert_id());  
?>
```

Notes**Caution**

[mysql_insert_id](#) will convert the return type of the native MySQL C API function [mysql_insert_id\(\)](#) to a type of [long](#) (named int in PHP). If your AUTO_INCREMENT column has a column type of BIGINT (64 bits) the conversion may result in an incorrect value. Instead, use the internal MySQL SQL function LAST_INSERT_ID() in an SQL query. For more information about PHP's maximum integer values, please see the [integer](#) documentation.

Note

Because `mysql_insert_id` acts on the last performed query, be sure to call `mysql_insert_id` immediately after the query that generates the value.

Note

The value of the MySQL SQL function `LAST_INSERT_ID()` always contains the most recently generated `AUTO_INCREMENT` value, and is not reset between queries.

See Also

`mysql_query`
`mysql_info`

6.5.5.32 `mysql_list_dbs`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_list_dbs`

List databases available on a MySQL server

Warning

This function was deprecated in PHP 5.4.0, and it and the entire [original MySQL extension](#) was removed in PHP 7.0.0. Instead, use either the actively developed [MySQLi](#) or [PDO_MySQL](#) extensions. See also the [MySQL: choosing an API](#) guide and its [related FAQ entry](#) for additional information. Alternatives to this function include:

SQL Query: `SHOW DATABASES`

Description

```
resource mysql_list_dbs(  
    resource link_identifier  
    = NULL);
```

Returns a result pointer containing the databases available from the current mysql daemon.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` had been called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a result pointer resource on success, or `FALSE` on failure. Use the `mysql_tablename` function to traverse this result pointer, or any function for result tables, such as `mysql_fetch_array`.

Examples**Example 6.188 `mysql_list_dbs` example**

```
<?php
```

```
// Usage without mysql_list_dbs()
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$res = mysql_query("SHOW DATABASES");
while ($row = mysql_fetch_assoc($res)) {
    echo $row['Database'] . "\n";
}
// Deprecated as of PHP 5.4.0
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$db_list = mysql_list_dbs($link);
while ($row = mysql_fetch_object($db_list)) {
    echo $row->Database . "\n";
}
?>
```

The above example will output something similar to:

```
database1
database2
database3
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
[mysql_listdbs](#)

See Also

[mysql_db_name](#)
[mysql_select_db](#)

6.5.5.33 [mysql_list_fields](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_list_fields](#)

List MySQL table fields

Warning

This function was deprecated in PHP 5.4.0, and it and the entire [original MySQL extension](#) was removed in PHP 7.0.0. Instead, use either the actively developed [MySQLi](#) or [PDO_MySQL](#) extensions. See also the [MySQL: choosing an API](#) guide and its [related FAQ entry](#) for additional information. Alternatives to this function include:

SQL Query: [SHOW COLUMNS FROM sometable](#)

Description

```
resource mysql_list_fields(
    string database_name,
    string table_name,
    resource link_identifier
    = NULL);
```

Retrieves information about the given table name.

This function is deprecated. It is preferable to use `mysql_query` to issue an SQL `SHOW COLUMNS FROM table [LIKE 'name']` statement instead.

Parameters

<i>database_name</i>	The name of the database that's being queried.
<i>table_name</i>	The name of the table that's being queried.
<i>link_identifier</i>	The MySQL connection. If the link identifier is not specified, the last link opened by <code>mysql_connect</code> is assumed. If no such link is found, it will try to create one as if <code>mysql_connect</code> had been called with no arguments. If no connection is found or established, an <code>E_WARNING</code> level error is generated.

Return Values

A result pointer resource on success, or `FALSE` on failure.

The returned result can be used with `mysql_field_flags`, `mysql_field_len`, `mysql_field_name` and `mysql_field_type`.

Examples

Example 6.189 Alternate to deprecated `mysql_list_fields`

```
<?php
$result = mysql_query("SHOW COLUMNS FROM sometable");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
if (mysql_num_rows($result) > 0) {
    while ($row = mysql_fetch_assoc($result)) {
        print_r($row);
    }
}
?>
```

The above example will output something similar to:

```
Array
(
    [Field] => id
    [Type] => int(7)
    [Null] =>
    [Key] => PRI
    [Default] =>
    [Extra] => auto_increment
)
Array
(
    [Field] => email
    [Type] => varchar(100)
    [Null] =>
    [Key] =>
    [Default] =>
    [Extra] =>
)
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
`mysql_listfields`

See Also

`mysql_field_flags`
`mysql_info`

6.5.5.34 `mysql_list_processes`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_list_processes`

List MySQL processes

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_thread_id`

Description

```
resource mysql_list_processes(  
    resource link_identifier  
    = =NULL);
```

Retrieves the current MySQL server threads.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

A result pointer resource on success or [FALSE](#) on failure.

Examples

Example 6.190 `mysql_list_processes` example

```
<?php  
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');  
$result = mysql_list_processes($link);  
while ($row = mysql_fetch_assoc($result)){  
    printf("%s %s %s %s %s\n", $row["Id"], $row["Host"], $row["db"],  
        $row["Command"], $row["Time"]);  
}
```



```
mysql_free_result($result);  
?>
```

The above example will output something similar to:

```
1 localhost test Processlist 0  
4 localhost mysql sleep 5
```

See Also

[mysql_thread_id](#)
[mysql_stat](#)

6.5.5.35 `mysql_list_tables`

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_list_tables](#)

List tables in a MySQL database

Warning

This function was deprecated in PHP 4.3.0, and it and the entire [original MySQL extension](#) was removed in PHP 7.0.0. Instead, use either the actively developed [MySQLi](#) or [PDO_MySQL](#) extensions. See also the [MySQL: choosing an API](#) guide and its [related FAQ entry](#) for additional information. Alternatives to this function include:

SQL Query: `SHOW TABLES FROM dbname`

Description

```
resource mysql_list_tables(  
    string database,  
    resource link_identifier  
    = NULL);
```

Retrieves a list of table names from a MySQL database.

This function is deprecated. It is preferable to use [mysql_query](#) to issue an SQL `SHOW TABLES [FROM db_name] [LIKE 'pattern']` statement instead.

Parameters

database

The name of the database

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

A result pointer resource on success or [FALSE](#) on failure.

Use the [mysql_tablename](#) function to traverse this result pointer, or any function for result tables, such as [mysql_fetch_array](#).

Changelog

Version	Description
4.3.7	This function became deprecated.

Examples

Example 6.191 [mysql_list_tables](#) alternative example

```
<?php
$dbname = 'mysql_dbname';
if (!mysql_connect('mysql_host', 'mysql_user', 'mysql_password')) {
    echo 'Could not connect to mysql';
    exit;
}
$sql = "SHOW TABLES FROM $dbname";
$result = mysql_query($sql);
if (!$result) {
    echo "DB Error, could not list tables\n";
    echo 'MySQL Error: ' . mysql_error();
    exit;
}
while ($row = mysql_fetch_row($result)) {
    echo "Table: {$row[0]}\n";
}
mysql_free_result($result);
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
[mysql_listtables](#)

See Also

[mysql_list_dbs](#)
[mysql_tablename](#)

6.5.5.36 [mysql_num_fields](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_num_fields](#)

Get number of fields in result

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_num_fields](#)
[PDOStatement::columnCount](#)

Description

```
int mysql_num_fields(
    resource result);
```

Retrieves the number of fields from a query.

Parameters

result The result resource that is being evaluated. This result comes from a call to [mysql_query](#).

Return Values

Returns the number of fields in the result set resource on success or [FALSE](#) on failure.

Examples

Example 6.192 A [mysql_num_fields](#) example

```
<?php
$result = mysql_query("SELECT id,email FROM people WHERE id = '42'");
if (!$result) {
    echo 'Could not run query: ' . mysql_error();
    exit;
}
/* returns 2 because id,email === two fields */
echo mysql_num_fields($result);
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
[mysql_numfields](#)

See Also

[mysql_select_db](#)
[mysql_query](#)
[mysql_fetch_field](#)
[mysql_num_rows](#)

6.5.5.37 [mysql_num_rows](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_num_rows](#)

Get number of rows in result

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_num_rows](#)

`mysqli_stmt_num_rows`
`PDOStatement::rowCount`

Description

```
int mysqli_num_rows(  
    resource result);
```

Retrieves the number of rows from a result set. This command is only valid for statements like `SELECT` or `SHOW` that return an actual result set. To retrieve the number of rows affected by a `INSERT`, `UPDATE`, `REPLACE` or `DELETE` query, use [mysqli_affected_rows](#).

Parameters

result The result resource that is being evaluated. This result comes from a call to [mysqli_query](#).

Return Values

The number of rows in a result set on success or `FALSE` on failure.

Examples

Example 6.193 `mysqli_num_rows` example

```
<?php  
$link = mysqli_connect("localhost", "mysql_user", "mysql_password");  
mysqli_select_db("database", $link);  
$result = mysqli_query("SELECT * FROM table1", $link);  
$num_rows = mysqli_num_rows($result);  
echo "$num_rows Rows\n";  
?>
```

Notes

Note

If you use [mysqli_unbuffered_query](#), [mysqli_num_rows](#) will not return the correct value until all the rows in the result set have been retrieved.

Note

For backward compatibility, the following deprecated alias may be used:
[mysqli_numrows](#)

See Also

[mysqli_affected_rows](#)
[mysqli_connect](#)
[mysqli_data_seek](#)
[mysqli_select_db](#)
[mysqli_query](#)

6.5.5.38 `mysqli_pconnect`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqli_pconnect](#)

Open a persistent connection to a MySQL server

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_connect](#) with `p`: host prefix
[PDO::__construct](#) with `PDO::ATTR_PERSISTENT` as a driver option

Description

```
resource mysql_pconnect(  
    string server  
        = =ini_get("mysql.default_host"),  
    string username  
        = =ini_get("mysql.default_user"),  
    string password  
        = =ini_get("mysql.default_password"),  
    int client_flags  
        = =0);
```

Establishes a persistent connection to a MySQL server.

[mysql_pconnect](#) acts very much like [mysql_connect](#) with two major differences.

First, when connecting, the function would first try to find a (persistent) link that's already open with the same host, username and password. If one is found, an identifier for it will be returned instead of opening a new connection.

Second, the connection to the SQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use ([mysql_close](#) will not close links established by [mysql_pconnect](#)).

This type of link is therefore called 'persistent'.

Parameters

<i>server</i>	The MySQL server. It can also include a port number. e.g. "hostname:port" or a path to a local socket e.g. ":/path/to/socket" for the localhost. If the PHP directive mysql.default_host is undefined (default), then the default value is 'localhost:3306'
<i>username</i>	The username. Default value is the name of the user that owns the server process.
<i>password</i>	The password. Default value is an empty password.
<i>client_flags</i>	The <i>client_flags</i> parameter can be a combination of the following constants: 128 (enable LOAD DATA LOCAL handling), MYSQL_CLIENT_SSL , MYSQL_CLIENT_COMPRESS , MYSQL_CLIENT_IGNORE_SPACE or MYSQL_CLIENT_INTERACTIVE .

Return Values

Returns a MySQL persistent link identifier on success, or [FALSE](#) on failure.

Changelog

Version	Description
5.5.0	This function will generate an E_DEPRECATED error.

Notes

Note

Note, that these kind of links only work if you are using a module version of PHP. See the [Persistent Database Connections](#) section for more information.

Warning

Using persistent connections can require a bit of tuning of your Apache and MySQL configurations to ensure that you do not exceed the number of connections allowed by MySQL.

Note

You can suppress the error message on failure by prepending a [@](#) to the function name.

See Also

[mysql_connect](#)
[Persistent Database Connections](#)

6.5.5.39 [mysql_ping](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_ping](#)

Ping a server connection or reconnect if there is no connection

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_ping](#)

Description

```
bool mysql_ping(  
    resource link_identifier  
    = =NULL);
```

Checks whether or not the connection to the server is working. If it has gone down, an automatic reconnection is attempted. This function can be used by scripts that remain idle for a long while, to check whether or not the server has closed the connection and reconnect if necessary.

Note

Automatic reconnection is disabled by default in versions of MySQL $\geq$ 5.0.3.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` had been called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns `TRUE` if the connection to the server MySQL server is working, otherwise `FALSE`.

Examples

Example 6.194 A `mysql_ping` example

```
<?php
set_time_limit(0);
$conn = mysql_connect('localhost', 'mysqluser', 'mypass');
$db    = mysql_select_db('mydb');
/* Assuming this query will take a long time */
$result = mysql_query($sql);
if (!$result) {
    echo 'Query #1 failed, exiting.';
    exit;
}
/* Make sure the connection is still alive, if not, try to reconnect */
if (!mysql_ping($conn)) {
    echo 'Lost connection, exiting after query #1';
    exit;
}
mysql_free_result($result);
/* So the connection is still alive, let's run another query */
$result2 = mysql_query($sql2);
?>
```

See Also

`mysql_thread_id`
`mysql_list_processes`

6.5.5.40 `mysql_query`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_query`

Send a MySQL query

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the `MySQLi` or `PDO_MySQL` extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_query`
`PDO::query`

Description

```
mixed mysql_query(
```

```
string query,  
resource link_identifier  
    = NULL);
```

`mysql_query` sends a unique query (multiple queries are not supported) to the currently active database on the server that's associated with the specified *link_identifier*.

Parameters

<i>query</i>	An SQL query The query string should not end with a semicolon. Data inside the query should be properly escaped .
<i>link_identifier</i>	The MySQL connection. If the link identifier is not specified, the last link opened by <code>mysql_connect</code> is assumed. If no such link is found, it will try to create one as if <code>mysql_connect</code> had been called with no arguments. If no connection is found or established, an E_WARNING level error is generated.

Return Values

For SELECT, SHOW, DESCRIBE, EXPLAIN and other statements returning resultset, `mysql_query` returns a resource on success, or [FALSE](#) on error.

For other type of SQL statements, INSERT, UPDATE, DELETE, DROP, etc, `mysql_query` returns [TRUE](#) on success or [FALSE](#) on error.

The returned result resource should be passed to `mysql_fetch_array`, and other functions for dealing with result tables, to access the returned data.

Use `mysql_num_rows` to find out how many rows were returned for a SELECT statement or `mysql_affected_rows` to find out how many rows were affected by a DELETE, INSERT, REPLACE, or UPDATE statement.

`mysql_query` will also fail and return [FALSE](#) if the user does not have permission to access the table(s) referenced by the query.

Examples

Example 6.195 Invalid Query

The following query is syntactically invalid, so `mysql_query` fails and returns [FALSE](#).

```
<?php  
$result = mysql_query('SELECT * WHERE 1=1');  
if (!$result) {  
    die('Invalid query: ' . mysql_error());  
}  
?>
```

Example 6.196 Valid Query

The following query is valid, so `mysql_query` returns a resource.

```
<?php  
// This could be supplied by a user, for example
```



```

$firstname = 'fred';
$lastname = 'fox';
// Formulate Query
// This is the best way to perform an SQL query
// For more examples, see mysql_real_escape_string()
$query = sprintf("SELECT firstname, lastname, address, age FROM friends
    WHERE firstname='%s' AND lastname='%s'",
        mysql_real_escape_string($firstname),
        mysql_real_escape_string($lastname));
// Perform Query
$result = mysql_query($query);
// Check result
// This shows the actual query sent to MySQL, and the error. Useful for debugging.
if (!$result) {
    $message = 'Invalid query: ' . mysql_error() . "\n";
    $message .= 'Whole query: ' . $query;
    die($message);
}
// Use result
// Attempting to print $result won't allow access to information in the resource
// One of the mysql result functions must be used
// See also mysql_result(), mysql_fetch_array(), mysql_fetch_row(), etc.
while ($row = mysql_fetch_assoc($result)) {
    echo $row['firstname'];
    echo $row['lastname'];
    echo $row['address'];
    echo $row['age'];
}
// Free the resources associated with the result set
// This is done automatically at the end of the script
mysql_free_result($result);
?>

```

See Also

[mysql_connect](#)
[mysql_error](#)
[mysql_real_escape_string](#)
[mysql_result](#)
[mysql_fetch_assoc](#)
[mysql_unbuffered_query](#)

6.5.5.41 [mysql_real_escape_string](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_real_escape_string](#)

Escapes special characters in a string for use in an SQL statement

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_real_escape_string](#)
[PDO::quote](#)

Description

```

string mysql_real_escape_string(
    string unescaped_string,

```

```
resource link_identifier
= =NULL);
```

Escapes special characters in the *unescaped_string*, taking into account the current character set of the connection so that it is safe to place it in a *mysql_query*. If binary data is to be inserted, this function must be used.

mysql_real_escape_string calls MySQL's library function *mysql_real_escape_string*, which prepends backslashes to the following characters: `\x00`, `\n`, `\r`, `\`, `'`, `"` and `\x1a`.

This function must always (with few exceptions) be used to make data safe before sending a query to MySQL.

Security: the default character set

The character set must be set either at the server level, or with the API function *mysql_set_charset* for it to affect *mysql_real_escape_string*. See the concepts section on *character sets* for more information.

Parameters

<i>unescaped_string</i>	The string that is to be escaped.
<i>link_identifier</i>	The MySQL connection. If the link identifier is not specified, the last link opened by <i>mysql_connect</i> is assumed. If no such link is found, it will try to create one as if <i>mysql_connect</i> had been called with no arguments. If no connection is found or established, an <i>E_WARNING</i> level error is generated.

Return Values

Returns the escaped string, or *FALSE* on error.

Errors/Exceptions

Executing this function without a MySQL connection present will also emit *E_WARNING* level PHP errors. Only execute this function with a valid MySQL connection present.

Examples

Example 6.197 Simple *mysql_real_escape_string* example

```
<?php
// Connect
$link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
    OR die(mysql_error());
// Query
$query = sprintf("SELECT * FROM users WHERE user='%s' AND password='%s'",
    mysql_real_escape_string($user),
    mysql_real_escape_string($password));
?>
```

Example 6.198 *mysql_real_escape_string* requires a connection example

This example demonstrates what happens if a MySQL connection is not present when calling this function.

```
<?php
// We have not connected to MySQL
$lastname = "O'Reilly";
$_lastname = mysql_real_escape_string($lastname);
$query = "SELECT * FROM actors WHERE last_name = '$_lastname'";
var_dump($_lastname);
var_dump($query);
?>
```

The above example will output something similar to:

```
Warning: mysql_real_escape_string(): No such file or directory in /this/test/script.php on line 5
Warning: mysql_real_escape_string(): A link to the server could not be established in /this/test/script.php on line 5
bool(false)
string(41) "SELECT * FROM actors WHERE last_name = ''"
```

Example 6.199 An example SQL Injection Attack

```
<?php
// We didn't check $_POST['password'], it could be anything the user wanted! For example:
$_POST['username'] = 'aidan';
$_POST['password'] = "' OR ''='";
// Query database to check if there are any matching users
$query = "SELECT * FROM users WHERE user='{$_POST['username']}' AND password='{$_POST['password']}'";
mysql_query($query);
// This means the query sent to MySQL would be:
echo $query;
?>
```

The query sent to MySQL:

```
SELECT * FROM users WHERE user='aidan' AND password='' OR ''=''
```

This would allow anyone to log in without a valid password.

Notes

Note

A MySQL connection is required before using `mysql_real_escape_string` otherwise an error of level `E_WARNING` is generated, and `FALSE` is returned. If `link_identifier` isn't defined, the last MySQL connection is used.

Note

If `magic_quotes_gpc` is enabled, first apply `stripslashes` to the data. Using this function on data which has already been escaped will escape the data twice.

Note

If this function is not used to escape data, the query is vulnerable to [SQL Injection Attacks](#).

Note

`mysql_real_escape_string` does not escape `%` and `_`. These are wildcards in MySQL if combined with [LIKE](#), [GRANT](#), or [REVOKE](#).

See Also

[mysql_set_charset](#)

[mysql_client_encoding](#)

[addslashes](#)

[stripslashes](#)

The [magic_quotes_gpc](#) directive

The [magic_quotes_runtime](#) directive

6.5.5.42 `mysql_result`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_result`

Get result data

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_data_seek](#) in conjunction with [mysqli_field_seek](#) and [mysqli_fetch_field](#)
[PDOStatement::fetchColumn](#)

Description

```
string mysql_result(  
    resource result,  
    int row,  
    mixed field  
    = 0);
```

Retrieves the contents of one cell from a MySQL result set.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than `mysql_result`. Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Parameters

result

The result resource that is being evaluated. This result comes from a call to [mysql_query](#).

row

The row number from the result that's being retrieved. Row numbers start at 0.

field

The name or offset of the field being retrieved.

It can be the field's offset, the field's name, or the field's table dot field name (tablename.fieldname). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name. If undefined, the first field is retrieved.

Return Values

The contents of one cell from a MySQL result set on success, or **FALSE** on failure.

Examples

Example 6.200 `mysql_result` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . mysql_error());
}
if (!mysql_select_db('database_name')) {
    die('Could not select database: ' . mysql_error());
}
$result = mysql_query('SELECT name FROM work.employee');
if (!$result) {
    die('Could not query: ' . mysql_error());
}
echo mysql_result($result, 2); // outputs third employee's name
mysql_close($link);
?>
```

Notes

Note

Calls to `mysql_result` should not be mixed with calls to other functions that deal with the result set.

See Also

`mysql_fetch_row`
`mysql_fetch_array`
`mysql_fetch_assoc`
`mysql_fetch_object`

6.5.5.43 `mysql_select_db`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_select_db`

Select a MySQL database

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

`mysqli_select_db`
`PDO::__construct` (part of dsn)

Description

```
bool mysql_select_db(
    string database_name,
    resource link_identifier
```

```
= =NULL);
```

Sets the current active database on the server that's associated with the specified link identifier. Every subsequent call to `mysql_query` will be made on the active database.

Parameters

database_name

The name of the database that is to be selected.

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysql_connect` is assumed. If no such link is found, it will try to create one as if `mysql_connect` had been called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 6.201 `mysql_select_db` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Not connected : ' . mysql_error());
}
// make foo the current db
$db_selected = mysql_select_db('foo', $link);
if (!$db_selected) {
    die ('Can\'t use foo : ' . mysql_error());
}
?>
```

Notes

Note

For backward compatibility, the following deprecated alias may be used:
`mysql_selectdb`

See Also

`mysql_connect`
`mysql_pconnect`
`mysql_query`

6.5.5.44 `mysql_set_charset`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_set_charset`

Sets the client character set

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the `MySQLi` or `PDO_MySQL` extension should be used. See

also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

[mysqli_set_charset](#)

PDO: Add [charset](#) to the connection string, such as [charset=utf8](#)

Description

```
bool mysql_set_charset(  
    string charset,  
    resource link_identifier  
    = NULL);
```

Sets the default character set for the current connection.

Parameters

charset

A valid character set name.

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Notes

Note

This function requires MySQL 5.0.7 or later.

Note

This is the preferred way to change the charset. Using [mysql_query](#) to set it (such as [SET NAMES utf8](#)) is not recommended. See the [MySQL character set concepts](#) section for more information.

See Also

[Setting character sets in MySQL](#)

[List of character sets that MySQL supports](#)

[mysql_client_encoding](#)

6.5.5.45 [mysql_stat](#)

[Copyright 1997-2014 the PHP Documentation Group.](#)

- [mysql_stat](#)

Get current system status

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

```
mysqli_stat
PDO::getAttribute(PDO::ATTR_SERVER_INFO)
```

Description

```
string mysqli_stat(
    resource link_identifier
    = =NULL);
```

`mysqli_stat` returns the current server status.

Parameters

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by `mysqli_connect` is assumed. If no such link is found, it will try to create one as if `mysqli_connect` had been called with no arguments. If no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns a string with the status for uptime, threads, queries, open tables, flush tables and queries per second. For a complete list of other status variables, you have to use the `SHOW STATUS` SQL command. If *link_identifier* is invalid, `NULL` is returned.

Examples

Example 6.202 `mysqli_stat` example

```
<?php
$link = mysqli_connect('localhost', 'mysql_user', 'mysql_password');
$status = explode(' ', mysqli_stat($link));
print_r($status);
?>
```

The above example will output something similar to:

```
Array
(
    [0] => Uptime: 5380
    [1] => Threads: 2
    [2] => Questions: 1321299
    [3] => Slow queries: 0
    [4] => Opens: 26
    [5] => Flush tables: 1
    [6] => Open tables: 17
    [7] => Queries per second avg: 245.595
)
```

Example 6.203 Alternative `mysqli_stat` example

```
<?php
$link = mysqli_connect('localhost', 'mysql_user', 'mysql_password');
$result = mysqli_query('SHOW STATUS', $link);
while ($row = mysqli_fetch_assoc($result)) {
```



```

        echo $row['Variable_name'] . ' = ' . $row['Value'] . "\n";
    }
    ?>

```

The above example will output something similar to:

```

back_log = 50
basedir = /usr/local/
bdb_cache_size = 8388600
bdb_log_buffer_size = 32768
bdb_home = /var/db/mysql/
bdb_max_lock = 10000
bdb_logdir =
bdb_shared_data = OFF
bdb_tmpdir = /var/tmp/
...

```

See Also

[mysql_get_server_info](#)
[mysql_list_processes](#)

6.5.5.46 `mysql_tablename`

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_tablename](#)

Get table name of field

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

SQL Query: [SHOW TABLES](#)

Description

```

string mysql_tablename(
    resource result,
    int i);

```

Retrieves the table name from a *result*.

This function is deprecated. It is preferable to use [mysql_query](#) to issue an SQL [SHOW TABLES \[FROM db_name\] \[LIKE 'pattern'\]](#) statement instead.

Parameters

result A result pointer resource that's returned from [mysql_list_tables](#).

i The integer index (row/table number)

Return Values

The name of the table on success or **FALSE** on failure.

Use the `mysql_tablename` function to traverse this result pointer, or any function for result tables, such as `mysql_fetch_array`.

Changelog

Version	Description
5.5.0	The <code>mysql_tablename</code> function is deprecated, and emits an E_DEPRECATED level error.

Examples

Example 6.204 `mysql_tablename` example

```
<?php
mysql_connect("localhost", "mysql_user", "mysql_password");
$result = mysql_list_tables("mydb");
$num_rows = mysql_num_rows($result);
for ($i = 0; $i < $num_rows; $i++) {
    echo "Table: ", mysql_tablename($result, $i), "\n";
}
mysql_free_result($result);
?>
```

Notes

Note

The `mysql_num_rows` function may be used to determine the number of tables in the result pointer.

See Also

`mysql_list_tables`
`mysql_field_table`
`mysql_db_name`

6.5.5.47 `mysql_thread_id`

Copyright 1997-2014 the PHP Documentation Group.

- `mysql_thread_id`

Return the current thread ID

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the **MySQLi** or **PDO_MySQL** extension should be used. See also **MySQL: choosing an API** guide and **related FAQ** for more information. Alternatives to this function include:

`mysqli_thread_id`

Description

```
int mysql_thread_id(
```

```
resource link_identifier
    = =NULL);
```

Retrieves the current thread ID. If the connection is lost, and a reconnect with [mysql_ping](#) is executed, the thread ID will change. This means only retrieve the thread ID when needed.

Parameters

link_identifier The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

The thread ID on success or [FALSE](#) on failure.

Examples

Example 6.205 [mysql_thread_id](#) example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$thread_id = mysql_thread_id($link);
if ($thread_id){
    printf("current thread id is %d\n", $thread_id);
}
?>
```

The above example will output something similar to:

```
current thread id is 73
```

See Also

[mysql_ping](#)
[mysql_list_processes](#)

6.5.5.48 [mysql_unbuffered_query](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysql_unbuffered_query](#)

Send an SQL query to MySQL without fetching and buffering the result rows.

Warning

This extension was deprecated in PHP 5.5.0, and it was removed in PHP 7.0.0. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

See: [Buffered and Unbuffered queries](#)

Description

```
resource mysql_unbuffered_query(  
    string query,  
    resource link_identifier  
    = =NULL);
```

[mysql_unbuffered_query](#) sends the SQL query *query* to MySQL without automatically fetching and buffering the result rows as [mysql_query](#) does. This saves a considerable amount of memory with SQL queries that produce large result sets, and you can start working on the result set immediately after the first row has been retrieved as you don't have to wait until the complete SQL query has been performed. To use [mysql_unbuffered_query](#) while multiple database connections are open, you must specify the optional parameter *link_identifier* to identify which connection you want to use.

Parameters

query

The SQL query to execute.

Data inside the query should be [properly escaped](#).

link_identifier

The MySQL connection. If the link identifier is not specified, the last link opened by [mysql_connect](#) is assumed. If no such link is found, it will try to create one as if [mysql_connect](#) had been called with no arguments. If no connection is found or established, an [E_WARNING](#) level error is generated.

Return Values

For SELECT, SHOW, DESCRIBE or EXPLAIN statements, [mysql_unbuffered_query](#) returns a resource on success, or [FALSE](#) on error.

For other type of SQL statements, UPDATE, DELETE, DROP, etc, [mysql_unbuffered_query](#) returns [TRUE](#) on success or [FALSE](#) on error.

Notes

Note

The benefits of [mysql_unbuffered_query](#) come at a cost: you cannot use [mysql_num_rows](#) and [mysql_data_seek](#) on a result set returned from [mysql_unbuffered_query](#), until all rows are fetched. You also have to fetch all result rows from an unbuffered SQL query before you can send a new SQL query to MySQL, using the same *link_identifier*.

See Also

[mysql_query](#)

6.6 MySQL Native Driver

Copyright 1997-2014 the PHP Documentation Group.

MySQL Native Driver is a replacement for the MySQL Client Library (libmysqlclient). MySQL Native Driver is part of the official PHP sources as of PHP 5.3.0.

The MySQL database extensions MySQL extension, [mysqli](#) and PDO MySQL all communicate with the MySQL server. In the past, this was done by the extension using the services provided by the MySQL Client Library. The extensions were compiled against the MySQL Client Library in order to use its client-server protocol.

With MySQL Native Driver there is now an alternative, as the MySQL database extensions can be compiled to use MySQL Native Driver instead of the MySQL Client Library.

MySQL Native Driver is written in C as a PHP extension.

6.6.1 Overview

[Copyright 1997-2014 the PHP Documentation Group.](#)

What it is not

Although MySQL Native Driver is written as a PHP extension, it is important to note that it does not provide a new API to the PHP programmer. The programmer APIs for MySQL database connectivity are provided by the MySQL extension, `mysqli` and PDO MySQL. These extensions can now use the services of MySQL Native Driver to communicate with the MySQL Server. Therefore, you should not think of MySQL Native Driver as an API.

Why use it?

Using the MySQL Native Driver offers a number of advantages over using the MySQL Client Library.

The older MySQL Client Library was written by MySQL AB (now Oracle Corporation) and so was released under the MySQL license. This ultimately led to MySQL support being disabled by default in PHP. However, the MySQL Native Driver has been developed as part of the PHP project, and is therefore released under the PHP license. This removes licensing issues that have been problematic in the past.

Also, in the past, you needed to build the MySQL database extensions against a copy of the MySQL Client Library. This typically meant you needed to have MySQL installed on a machine where you were building the PHP source code. Also, when your PHP application was running, the MySQL database extensions would call down to the MySQL Client library file at run time, so the file needed to be installed on your system. With MySQL Native Driver that is no longer the case as it is included as part of the standard distribution. So you do not need MySQL installed in order to build PHP or run PHP database applications.

Because MySQL Native Driver is written as a PHP extension, it is tightly coupled to the workings of PHP. This leads to gains in efficiency, especially when it comes to memory usage, as the driver uses the PHP memory management system. It also supports the PHP memory limit. Using MySQL Native Driver leads to comparable or better performance than using MySQL Client Library, it always ensures the most efficient use of memory. One example of the memory efficiency is the fact that when using the MySQL Client Library, each row is stored in memory twice, whereas with the MySQL Native Driver each row is only stored once in memory.

Reporting memory usage

Because MySQL Native Driver uses the PHP memory management system, its memory usage can be tracked with `memory_get_usage`. This is not possible with `libmysqlclient` because it uses the C function `malloc()` instead.

Special features

MySQL Native Driver also provides some special features not available when the MySQL database extensions use MySQL Client Library. These special features are listed below:

- Improved persistent connections
- The special function `mysqli_fetch_all`
- Performance statistics calls: `mysqli_get_cache_stats`, `mysqli_get_client_stats`, `mysqli_get_connection_stats`

The performance statistics facility can prove to be very useful in identifying performance bottlenecks.

MySQL Native Driver also allows for persistent connections when used with the [mysqli](#) extension.

SSL Support

MySQL Native Driver has supported SSL since PHP version 5.3.3

Compressed Protocol Support

As of PHP 5.3.2 MySQL Native Driver supports the compressed client server protocol. MySQL Native Driver did not support this in 5.3.0 and 5.3.1. Extensions such as [ext/mysql](#), [ext/mysqli](#), that are configured to use MySQL Native Driver, can also take advantage of this feature. Note that [PDO_MYSQL](#) does *NOT* support compression when used together with [mysqlnd](#).

Named Pipes Support

Named pipes support for Windows was added in PHP version 5.4.0.

6.6.2 Installation

Copyright 1997-2014 the PHP Documentation Group.

Changelog

Table 6.27 Changelog

Version	Description
5.3.0	The MySQL Native Driver was added, with support for all MySQL extensions (i.e., mysql , mysqli and PDO_MYSQL). Passing in mysqlnd to the appropriate configure switch enables this support.
5.4.0	The MySQL Native Driver is now the default for all MySQL extensions (i.e., mysql , mysqli and PDO_MYSQL). Passing in mysqlnd to configure is now optional.
5.5.0	SHA-256 Authentication Plugin support was added

Installation on Unix

The MySQL database extensions must be configured to use the MySQL Client Library. In order to use the MySQL Native Driver, PHP needs to be built specifying that the MySQL database extensions are compiled with MySQL Native Driver support. This is done through configuration options prior to building the PHP source code.

For example, to build the MySQL extension, [mysqli](#) and PDO MYSQL using the MySQL Native Driver, the following command would be given:

```
./configure --with-mysql=mysqlnd \
--with-mysqli=mysqlnd \
--with-pdo-mysql=mysqlnd \
[other options]
```

Installation on Windows

In the official PHP Windows distributions from 5.3 onwards, MySQL Native Driver is enabled by default, so no additional configuration is required to use it. All MySQL database extensions will use MySQL Native Driver in this case.

SHA-256 Authentication Plugin support

The MySQL Native Driver requires the OpenSSL functionality of PHP to be loaded and enabled to connect to MySQL through accounts that use the MySQL SHA-256 Authentication Plugin. For example, PHP could be configured using:

```
./configure --with-mysql=mysqlnd \
--with-mysqli=mysqlnd \
--with-pdo-mysql=mysqlnd \
--with-openssl
[other options]
```

6.6.3 Runtime Configuration

Copyright 1997-2014 the PHP Documentation Group.

The behaviour of these functions is affected by settings in [php.ini](#).

Table 6.28 MySQL Native Driver Configuration Options

Name	Default	Changeable	Changelog
mysqlnd.collect_statistics	"1"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.collect_memory_statistics	"0"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.debug	""	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.log_mask	0	PHP_INI_ALL	Available since PHP 5.3.0
mysqlnd.mempool_default_size	16000	PHP_INI_ALL	Available since PHP 5.3.3
mysqlnd.net_read_timeout	"31536000"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.net_cmd_buffer_size	512 - "2048", 5.3.1 - "4096"	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.net_read_buffer_size	32768	PHP_INI_SYSTEM	Available since PHP 5.3.0.
mysqlnd.sha256_server_public_key		PHP_INI_PERDIR	Available since PHP 5.5.0.
mysqlnd.fetch_data_copy	0	PHP_INI_ALL	Available since PHP 5.6.0.

For further details and definitions of the PHP_INI_* modes, see the <http://www.php.net/manual/en/configuration.changes.modes>.

Here's a short explanation of the configuration directives.

[mysqlnd.collect_statistics](#) Enables the collection of various client statistics which can be accessed through [mysqli_get_client_stats](#),
boolean

`mysqli_get_connection_stats`, `mysqli_get_cache_stats` and are shown in `mysqli` section of the output of the `phpinfo` function as well.

This configuration setting enables all [MySQL Native Driver statistics](#) except those relating to memory management.

`mysqli.collect_memory_stats`
boolean

Enable the collection of various memory statistics which can be accessed through `mysqli_get_client_stats`, `mysqli_get_connection_stats`, `mysqli_get_cache_stats` and are shown in `mysqli` section of the output of the `phpinfo` function as well.

This configuration setting enables the memory management statistics within the overall set of [MySQL Native Driver statistics](#).

`mysqli.debug` string

Records communication from all extensions using `mysqli` to the specified log file.

The format of the directive is `mysqli.debug = "option1[,parameter_option1][:option2[,parameter_option2]]"`.

The options for the format string are as follows:

- `A[,file]` - Appends trace output to specified file. Also ensures that data is written after each write. This is done by closing and reopening the trace file (this is slow). It helps ensure a complete log file should the application crash.
- `a[,file]` - Appends trace output to the specified file.
- `d` - Enables output from `DEBUG_<N>` macros for the current state. May be followed by a list of keywords which selects output only for the `DEBUG` macros with that keyword. An empty list of keywords implies output for all macros.
- `f[,functions]` - Limits debugger actions to the specified list of functions. An empty list of functions implies that all functions are selected.
- `F` - Marks each debugger output line with the name of the source file containing the macro causing the output.
- `i` - Marks each debugger output line with the PID of the current process.
- `L` - Marks each debugger output line with the name of the source file line number of the macro causing the output.
- `n` - Marks each debugger output line with the current function nesting depth
- `o[,file]` - Similar to `a[,file]` but overwrites old file, and does not append.
- `O[,file]` - Similar to `A[,file]` but overwrites old file, and does not append.
- `t[,N]` - Enables function control flow tracing. The maximum nesting depth is specified by `N`, and defaults to 200.

- x - This option activates profiling.
- m - Trace memory allocation and deallocation related calls.

Example:

```
d:t:x:0,/tmp/mysqlnd.trace
```

Note

This feature is only available with a debug build of PHP. Works on Microsoft Windows if using a debug build of PHP and PHP was built using Microsoft Visual C version 9 and above.

mysqlnd.log_mask integer

Defines which queries will be logged. The default 0, which disables logging. Define using an integer, and not with PHP constants. For example, a value of 48 (16 + 32) will log slow queries which either use 'no good index' (SERVER_QUERY_NO_GOOD_INDEX_USED = 16) or no index at all (SERVER_QUERY_NO_INDEX_USED = 32). A value of 2043 (1 + 2 + 8 + ... + 1024) will log all slow query types.

The types are as follows: SERVER_STATUS_IN_TRANS=1, SERVER_STATUS_AUTOCOMMIT=2, SERVER_MORE_RESULTS_EXISTS=8, SERVER_QUERY_NO_GOOD_INDEX_USED=16, SERVER_QUERY_NO_INDEX_USED=32, SERVER_STATUS_CURSOR_EXISTS=64, SERVER_STATUS_LAST_ROW_SENT=128, SERVER_STATUS_DB_DROPPED=256, SERVER_STATUS_NO_BACKSLASH_ESCAPES=512, and SERVER_QUERY_WAS_SLOW=1024.

mysqlnd.mempool_default_size integer

Default size of the mysqlnd memory pool, which is used by result sets.

mysqlnd.net_read_timeout integer

mysqlnd and the MySQL Client Library, *libmysqlclient* use different networking APIs. *mysqlnd* uses PHP streams, whereas *libmysqlclient* uses its own wrapper around the operating level network calls. PHP, by default, sets a read timeout of 60s for streams. This is set via *php.ini*, *default_socket_timeout*. This default applies to all streams that set no other timeout value. *mysqlnd* does not set any other value and therefore connections of long running queries can be disconnected after *default_socket_timeout* seconds resulting in an error message "2006 - MySQL Server has gone away". The MySQL Client Library sets a default timeout of 365 * 24 * 3600 seconds (1 year) and waits for other timeouts to occur, such as TCP/IP timeouts. *mysqlnd* now uses the same very long timeout. The value is configurable through a new *php.ini* setting: *mysqlnd.net_read_timeout*. *mysqlnd.net_read_timeout* gets used by any extension (*ext/mysql*, *ext/mysql_i*, *PDO_MySQL*) that uses *mysqlnd*. *mysqlnd* tells PHP Streams to use *mysqlnd.net_read_timeout*. Please note that there

may be subtle differences between `MYSQL_OPT_READ_TIMEOUT` from the MySQL Client Library and PHP Streams, for example `MYSQL_OPT_READ_TIMEOUT` is documented to work only for TCP/IP connections and, prior to MySQL 5.1.2, only for Windows. PHP streams may not have this limitation. Please check the streams documentation, if in doubt.

`mysqlnd.net_cmd_buffer_size`
long
`mysqlnd` allocates an internal command/network buffer of `mysqlnd.net_cmd_buffer_size` (in `php.ini`) bytes for every connection. If a MySQL Client Server protocol command, for example, `COM_QUERY` ("normal" query), does not fit into the buffer, `mysqlnd` will grow the buffer to the size required for sending the command. Whenever the buffer gets extended for one connection, `command_buffer_too_small` will be incremented by one.

If `mysqlnd` has to grow the buffer beyond its initial size of `mysqlnd.net_cmd_buffer_size` bytes for almost every connection, you should consider increasing the default size to avoid re-allocations.

The default buffer size is 2048 bytes in PHP 5.3.0. In later versions the default is 4096 bytes.

It is recommended that the buffer size be set to no less than 4096 bytes because `mysqlnd` also uses it when reading certain communication packet from MySQL. In PHP 5.3.0, `mysqlnd` will not grow the buffer if MySQL sends a packet that is larger than the current size of the buffer. As a consequence, `mysqlnd` is unable to decode the packet and the client application will get an error. There are only two situations when the packet can be larger than the 2048 bytes default of `mysqlnd.net_cmd_buffer_size` in PHP 5.3.0: the packet transports a very long error message, or the packet holds column meta data from `COM_LIST_FIELD` (`mysql_list_fields()`) and the meta data come from a string column with a very long default value (>1900 bytes).

As of PHP 5.3.2 `mysqlnd` does not allow setting buffers smaller than 4096 bytes.

The value can also be set using `mysqli_options(link, MYSQLI_OPT_NET_CMD_BUFFER_SIZE, size)`.

`mysqlnd.net_read_buffer_size`
long
Maximum read chunk size in bytes when reading the body of a MySQL command packet. The MySQL client server protocol encapsulates all its commands in packets. The packets consist of a small header and a body with the actual payload. The size of the body is encoded in the header. `mysqlnd` reads the body in chunks of `MIN(header.size, mysqlnd.net_read_buffer_size)` bytes. If a packet body is larger than `mysqlnd.net_read_buffer_size` bytes, `mysqlnd` has to call `read()` multiple times.

The value can also be set using `mysqli_options(link, MYSQLI_OPT_NET_READ_BUFFER_SIZE, size)`.

`mysqlnd.sha256_server_public_key`
string
SHA256 Authentication Plugin related. File with the MySQL server public RSA key.

Clients can either omit setting a public RSA key, specify the key through this PHP configuration setting or set the key at runtime

`mysqlnd.fetch_data_copy`
long

using `mysqli_options`. If not public RSA key file is given by the client, then the key will be exchanged as part of the standard SHA-256 Authentication Plugin authentication procedure.

Enforce copying result sets from the internal result set buffers into PHP variables instead of using the default reference and copy-on-write logic. Please, see the [memory management implementation notes](#) for further details.

Copying result sets instead of having PHP variables reference them allows releasing the memory occupied for the PHP variables earlier. Depending on the user API code, the actual database queries and the size of their result sets this may reduce the memory footprint of `mysqlnd`.

Do not set if using PDO_MySQL. PDO_MySQL has not yet been updated to support the new fetch mode.

6.6.4 Incompatibilities

[Copyright 1997-2014 the PHP Documentation Group.](#)

MySQL Native Driver is in most cases compatible with MySQL Client Library (`libmysql`). This section documents incompatibilities between these libraries.

- Values of `bit` data type are returned as binary strings (e.g. `"\0"` or `"\x1F"`) with `libmysql` and as decimal strings (e.g. `"0"` or `"31"`) with `mysqlnd`. If you want the code to be compatible with both libraries then always return bit fields as numbers from MySQL with a query like this: `SELECT bit + 0 FROM table`.

6.6.5 Persistent Connections

[Copyright 1997-2014 the PHP Documentation Group.](#)

Using Persistent Connections

If `mysqli` is used with `mysqlnd`, when a persistent connection is created it generates a `COM_CHANGE_USER` (`mysqli_change_user()`) call on the server. This ensures that re-authentication of the connection takes place.

As there is some overhead associated with the `COM_CHANGE_USER` call, it is possible to switch this off at compile time. Reusing a persistent connection will then generate a `COM_PING` (`mysqli_ping`) call to simply test the connection is reusable.

Generation of `COM_CHANGE_USER` can be switched off with the compile flag `MYSQLI_NO_CHANGE_USER_ON_PCONNECT`. For example:

```
shell# CFLAGS="-DMYSQLI_NO_CHANGE_USER_ON_PCONNECT" ./configure --with-mysql=/usr/local/mysql/ --with-m
```

Or alternatively:

```
shell# export CFLAGS="-DMYSQLI_NO_CHANGE_USER_ON_PCONNECT"
shell# configure --whatever-option
shell# make clean
shell# make
```

Note that only `mysqli` on `mysqlnd` uses `COM_CHANGE_USER`. Other extension-driver combinations use `COM_PING` on initial use of a persistent connection.

6.6.6 Statistics

Copyright 1997-2014 the PHP Documentation Group.

Using Statistical Data

MySQL Native Driver contains support for gathering statistics on the communication between the client and the server. The statistics gathered are of two main types:

- Client statistics
- Connection statistics

If you are using the `mysqli` extension, these statistics can be obtained through two API calls:

- `mysqli_get_client_stats`
- `mysqli_get_connection_stats`

Note

Statistics are aggregated among all extensions that use MySQL Native Driver. For example, when compiling both `ext/mysql` and `ext/mysqli` against MySQL Native Driver, both function calls of `ext/mysql` and `ext/mysqli` will change the statistics. There is no way to find out how much a certain API call of any extension that has been compiled against MySQL Native Driver has impacted a certain statistic. You can configure the PDO MySQL Driver, `ext/mysql` and `ext/mysqli` to optionally use the MySQL Native Driver. When doing so, all three extensions will change the statistics.

Accessing Client Statistics

To access client statistics, you need to call `mysqli_get_client_stats`. The function call does not require any parameters.

The function returns an associative array that contains the name of the statistic as the key and the statistical data as the value.

Client statistics can also be accessed by calling the `phpinfo` function.

Accessing Connection Statistics

To access connection statistics call `mysqli_get_connection_stats`. This takes the database connection handle as the parameter.

The function returns an associative array that contains the name of the statistic as the key and the statistical data as the value.

Buffered and Unbuffered Result Sets

Result sets can be buffered or unbuffered. Using default settings, `ext/mysql` and `ext/mysqli` work with buffered result sets for normal (non prepared statement) queries. Buffered result sets are cached on the client. After the query execution all results are fetched from the MySQL Server and stored in a cache on the client. The big advantage of buffered result sets is that they allow the server to free all resources allocated to a result set, once the results have been fetched by the client.

Unbuffered result sets on the other hand are kept much longer on the server. If you want to reduce memory consumption on the client, but increase load on the server, use unbuffered results. If you experience a high server load and the figures for unbuffered result sets are high, you should consider moving the load to the clients. Clients typically scale better than servers. "Load" does not only refer to memory buffers - the server also needs to keep other resources open, for example file handles and threads, before a result set can be freed.

Prepared Statements use unbuffered result sets by default. However, you can use `mysqli_stmt_store_result` to enable buffered result sets.

Statistics returned by MySQL Native Driver

The following tables show a list of statistics returned by the `mysqli_get_client_stats` and `mysqli_get_connection_stats` functions.

Table 6.29 Returned mysqli statistics: Network

Statistic	Scope	Description	Notes
<code>bytes_sent</code>	Connection	Number of bytes sent from PHP to the MySQL server	Can be used to check the efficiency of the compression protocol
<code>bytes_received</code>	Connection	Number of bytes received from MySQL server	Can be used to check the efficiency of the compression protocol
<code>packets_sent</code>	Connection	Number of MySQL Client Server protocol packets sent	Used for debugging Client Server protocol implementation
<code>packets_received</code>	Connection	Number of MySQL Client Server protocol packets received	Used for debugging Client Server protocol implementation
<code>protocol_overhead_in</code>	Connection	MySQL Client Server protocol overhead in bytes for incoming traffic. Currently only the Packet Header (4 bytes) is considered as overhead. $\text{protocol\_overhead\_in} = \text{packets\_received} * 4$	Used for debugging Client Server protocol implementation
<code>protocol_overhead_out</code>	Connection	MySQL Client Server protocol overhead in bytes for outgoing traffic. Currently only the Packet Header (4 bytes) is considered as overhead. $\text{protocol\_overhead\_out} = \text{packets\_sent} * 4$	Used for debugging Client Server protocol implementation
<code>bytes_received_ok</code>	Connection	Total size in bytes of MySQL Client Server protocol OK packets received. OK packets can contain a status message. The length of the status message can vary and thus the size of an OK packet is not fixed.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received_ok</code>	Connection	Number of MySQL Client Server protocol OK packets received.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>bytes_received_eof</code>	Connection	Total size in bytes of MySQL Client Server protocol EOF packets received. EOF can vary in size depending on the server version. Also, EOF can transport an error message.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>packets_received_eof</code>	Connection	Number of MySQL Client Server protocol EOF packets. Like with other packet statistics the number of packets will be increased even if PHP does not receive the expected packet but, for example, an error message.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
<code>bytes_received_result_set_header</code>	Connection	Total size in bytes of MySQL Client Server protocol result set header	Used for debugging CS protocol implementation. Note that the total

Statistic	Scope	Description	Notes
		packets. The size of the packets varies depending on the payload (LOAD LOCAL INFILE , INSERT , UPDATE , SELECT , error message).	size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received	Connection	Number of MySQL Client Server protocol result set header packets.	Used for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
bytes_received	Connection	Total size in bytes of MySQL Client Server protocol result set meta data (field information) packets. Of course the size varies with the fields in the result set. The packet may also transport an error or an EOF packet in case of COM_LIST_FIELDS.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received	Connection	Number of MySQL Client Server protocol result set meta data (field information) packets.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
bytes_received	Connection	Total size in bytes of MySQL Client Server protocol result set row data packets. The packet may also transport an error or an EOF packet. You can reverse engineer the number of error and EOF packets by subtracting rows_fetched_from_server_normal and rows_fetched_from_server_ps from bytes_received_rset_row_packet .	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_received	Connection	Number of MySQL Client Server protocol result set row data packets and their total size in bytes.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
bytes_received	Connection	Total size in bytes of MySQL Client Server protocol OK for Prepared Statement Initialization packets (prepared statement init packets). The packet may also transport an error. The packet size depends on the MySQL version: 9 bytes with MySQL 4.1 and 12 bytes from MySQL 5.0 on. There is no safe way to know how many errors happened. You may be able to guess that an error has occurred if, for example, you always connect to MySQL 5.0 or newer and, bytes_received_prepare_response_packet != packets_received_prepare_response * 12. See also	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).

Statistic	Scope	Description	Notes
		ps_prepared_never_executed , ps_prepared_once_executed .	
packets_sent	Connection	Number of MySQL Client Server protocol OK for Prepared Statement Initialization packets (prepared statement init packets).	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
bytes_received	Connection	Total size in bytes of MySQL Client Server protocol COM_CHANGE_USER packets. The packet may also transport an error or EOF.	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_sent	Connection	Number of MySQL Client Server protocol COM_CHANGE_USER packets	Only useful for debugging CS protocol implementation. Note that the total size in bytes includes the size of the header packet (4 bytes, see protocol overhead).
packets_sent	Connection	Number of MySQL Client Server protocol commands sent from PHP to MySQL. There is no way to know which specific commands and how many of them have been sent. At its best you can use it to check if PHP has sent any commands to MySQL to know if you can consider to disable MySQL support in your PHP binary. There is also no way to reverse engineer the number of errors that may have occurred while sending data to MySQL. The only error that is recorded is <code>command_buffer_too_small</code> (see below).	Only useful for debugging CS protocol implementation.
bytes_received	Connection	Number of bytes of payload fetched by the PHP client from mysqlnd using the text protocol.	This is the size of the actual data contained in result sets that do not originate from prepared statements and which have been fetched by the PHP client. Note that although a full result set may have been pulled from MySQL by mysqlnd, this statistic only counts actual data pulled from mysqlnd by the PHP client. An example of a code sequence that will increase the value is as follows: <pre>\$mysqli = new mysqli(); \$res = \$mysqli->query("SELECT 'abc'"); \$res->fetch_assoc(); \$res->close();</pre> Every fetch operation will increase the value.

Statistic	Scope	Description	Notes
			The statistic will not be increased if the result set is only buffered on the client, but not fetched, such as in the following example: <pre>\$mysqli = new mysqli(); \$res = \$mysqli->query("SELECT 'abc'"); \$res->close();</pre> This statistic is available as of PHP version 5.3.4.
bytes_received	Connection	Number of bytes of the payload fetched by the PHP client from mysqli using the prepared statement protocol.	This is the size of the actual data contained in result sets that originate from prepared statements and which has been fetched by the PHP client. The value will not be increased if the result set is not subsequently read by the PHP client. Note that although a full result set may have been pulled from MySQL by mysqli, this statistic only counts actual data pulled from mysqli by the PHP client. See also bytes_received_real_data_normal. This statistic is available as of PHP version 5.3.4.

Result Set

Table 6.30 Returned mysqli statistics: Result Set

Statistic	Scope	Description	Notes
result_set_queries	Connection	Number of queries that have generated a result set. Examples of queries that generate a result set: SELECT , SHOW . The statistic will not be incremented if there is an error reading the result set header packet from the line.	You may use it as an indirect measure for the number of queries PHP has sent to MySQL, for example, to identify a client that causes a high database load.
non_result_set_queries	Connection	Number of queries that did not generate a result set. Examples of queries that do not generate a result set: INSERT , UPDATE , LOAD DATA , SHOW . The statistic will not be incremented if there is an error reading the result set header packet from the line.	You may use it as an indirect measure for the number of queries PHP has sent to MySQL, for example, to identify a client that causes a high database load.
no_index_queries	Connection	Number of queries that have generated a result set but did not use an index (see also <code>mysqld</code> start option <code>--log-queries-not-using-indexes</code>). If you want these queries to be reported you can use <code>mysqli_report(MYSQLI_REPORT_INDEX)</code> to make <code>ext/mysqli</code> throw an exception. If you prefer a warning instead of an exception use	

Statistic	Scope	Description	Notes
		<code>mysqli_report(MYSQLI_REPORT_INDEX ^ MYSQLI_REPORT_STRICT)</code> .	
bad_index	Connection	Number of queries that have generated a result set and did not use a good index (see also <code>mysqld</code> start option – <code>log-slow-queries</code>).	If you want these queries to be reported you can use <code>mysqli_report(MYSQLI_REPORT_INDEX)</code> to make ext/mysqli throw an exception. If you prefer a warning instead of an exception use <code>mysqli_report(MYSQLI_REPORT_INDEX ^ MYSQLI_REPORT_STRICT)</code>
slow_queries	Connection	SQL statements that took more than long_query_time seconds to execute and required at least min_examined_row_limit rows to be examined.	Not reported through mysqli_report
buffered_queries	Connection	Number of buffered result sets returned by “normal” queries. “Normal” means “not prepared statement” in the following notes.	Examples of API calls that will buffer result sets on the client: mysql_query , mysqli_query , mysqli_store_result , mysqli_stmt_get_result . Buffering result sets on the client ensures that server resources are freed as soon as possible and it makes result set scrolling easier. The downside is the additional memory consumption on the client for buffering data. Note that <code>mysqlnd</code> (unlike the MySQL Client Library) respects the PHP memory limit because it uses PHP internal memory management functions to allocate memory. This is also the reason why memory_get_usage reports a higher memory consumption when using <code>mysqlnd</code> instead of the MySQL Client Library. memory_get_usage does not measure the memory consumption of the MySQL Client Library at all because the MySQL Client Library does not use PHP internal memory management functions monitored by the function!
unbuffered_queries	Connection	Number of unbuffered result sets returned by normal (non prepared statement) queries.	Examples of API calls that will not buffer result sets on the client: mysqli_use_result
ps_buffered_queries	Connection	Number of buffered result sets returned by prepared statements. By default prepared statements are unbuffered.	Examples of API calls that will not buffer result sets on the client: mysqli_stmt_store_result
ps_unbuffered_queries	Connection	Number of unbuffered result sets returned by prepared statements.	By default prepared statements are unbuffered.
flushed_queries	Connection	Number of result sets from normal (non prepared statement) queries with unread data which have been flushed silently for you. Flushing happens only with unbuffered result sets.	Unbuffered result sets must be fetched completely before a new query can be run on the connection otherwise MySQL will throw an error. If the application does not fetch all rows from an unbuffered

Statistic	Scope	Description	Notes
			result set, mysqlnd does implicitly fetch the result set to clear the line. See also rows_skipped_normal, rows_skipped_ps. Some possible causes for an implicit flush: - Faulty client application - Client stopped reading after it found what it was looking for but has made MySQL calculate more records than needed - Client application has stopped unexpectedly
flushed_connection	Connection	Number of result sets from prepared statements with unread data which have been flushed silently for you. Flushing happens only with unbuffered result sets.	Unbuffered result sets must be fetched completely before a new query can be run on the connection otherwise MySQL will throw an error. If the application does not fetch all rows from an unbuffered result set, mysqlnd does implicitly fetch the result set to clear the line. See also rows_skipped_normal, rows_skipped_ps. Some possible causes for an implicit flush: - Faulty client application - Client stopped reading after it found what it was looking for but has made MySQL calculate more records than needed - Client application has stopped unexpectedly
ps_prepared_statements	Connection	Number of statements prepared but never executed.	Prepared statements occupy server resources. You should not prepare a statement if you do not plan to execute it.
ps_prepared_statements_executed	Connection	Number of prepared statements executed only one.	One of the ideas behind prepared statements is that the same query gets executed over and over again (with different parameters) and some parsing and other preparation work can be saved, if statement execution is split up in separate prepare and execute stages. The idea is to prepare once and “cache” results, for example, the parse tree to be reused during multiple statement executions. If you execute a prepared statement only once the two stage processing can be inefficient compared to “normal” queries because all the caching means extra work and it takes (limited) server resources to hold

Statistic	Scope	Description	Notes
			the cached information. Consequently, prepared statements that are executed only once may cause performance hurts.
rows_fetched rows_fetched_from_mysql	Connection	Total number of result set rows successfully fetched from MySQL regardless if the client application has consumed them or not. Some of the rows may not have been fetched by the client application but have been flushed implicitly.	See also packets_received_rset_row
rows_buffered rows_buffered_from_mysql	Connection	Total number of successfully buffered rows originating from a "normal" query or a prepared statement. This is the number of rows that have been fetched from MySQL and buffered on client. Note that there are two distinct statistics on rows that have been buffered (MySQL to mysqlnd internal buffer) and buffered rows that have been fetched by the client application (mysqlnd internal buffer to client application). If the number of buffered rows is higher than the number of fetched buffered rows it can mean that the client application runs queries that cause larger result sets than needed resulting in rows not read by the client.	Examples of queries that will buffer results: mysqli_query , mysqli_store_result
rows_fetched_from_buffered	Connection	Total number of rows fetched by the client from a buffered result set created by a normal query or a prepared statement.	
rows_fetched_from_unbuffered	Connection	Total number of rows fetched by the client from a unbuffered result set created by a "normal" query or a prepared statement.	
rows_fetched_from_cursor	Connection	Total number of rows fetched by the client from a cursor created by a prepared statement.	
rows_skipped rows_skipped_ps	Connection	Reserved for future use (currently not supported)	
copy_on_write_extensions copy_on_write_extensions	Process	With mysqlnd, variables returned by the extensions point into mysqlnd internal network result buffers. If you do not change the variables, fetched data will be kept only once in memory. If you change the variables, mysqlnd has to perform a copy-on-write to protect the internal network result buffers from being changed. With the MySQL Client Library you always hold fetched data twice in memory. Once in the internal MySQL Client Library buffers and once in the variables returned by	

Statistic	Scope	Description	Notes
		the extensions. In theory mysqld can save up to 40% memory. However, note that the memory saving cannot be measured using <code>memory_get_usage</code> .	
<code>explicit_connection_result</code> <code>implicit_connection_result</code>	Connection (only during prepared statement cleanup)	Total number of freed result sets.	The free is always considered explicit but for result sets created by an init command, for example, <code>mysql_options(MYSQLI_INIT_COMMAND, ..</code>
<code>proto_text_fetched</code> <code>proto_text_fetched_int</code> <code>proto_text_fetched_int24</code> <code>proto_text_fetched_short</code> <code>proto_text_fetched_int</code> <code>proto_text_fetched_bigint</code> <code>proto_text_fetched_decimal</code> <code>proto_text_fetched_float</code> <code>proto_text_fetched_double</code> <code>proto_text_fetched_date</code> <code>proto_text_fetched_year</code> <code>proto_text_fetched_time</code> <code>proto_text_fetched_datetime</code> <code>proto_text_fetched_timestamp</code> <code>proto_text_fetched_string</code> <code>proto_text_fetched_blob</code> <code>proto_text_fetched_enum</code> <code>proto_text_fetched_set</code> <code>proto_text_fetched_geometry</code> <code>proto_text_fetched_other</code>	Connection	Total number of columns of a certain type fetched from a normal query (MySQL text protocol).	Mapping from C API / MySQL meta data type to statistics name: • <code>MYSQL_TYPE_NULL</code> - <code>proto_text_fetched_null</code> • <code>MYSQL_TYPE_BIT</code> - <code>proto_text_fetched_bit</code> • <code>MYSQL_TYPE_TINY</code> - <code>proto_text_fetched_tinyint</code> • <code>MYSQL_TYPE_SHORT</code> - <code>proto_text_fetched_short</code> • <code>MYSQL_TYPE_INT24</code> - <code>proto_text_fetched_int24</code> • <code>MYSQL_TYPE_LONG</code> - <code>proto_text_fetched_int</code> • <code>MYSQL_TYPE_LONGLONG</code> - <code>proto_text_fetched_bigint</code> • <code>MYSQL_TYPE_DECIMAL</code>, <code>MYSQL_TYPE_NEWDECIMAL</code> - <code>proto_text_fetched_decimal</code> • <code>MYSQL_TYPE_FLOAT</code> - <code>proto_text_fetched_float</code> • <code>MYSQL_TYPE_DOUBLE</code> - <code>proto_text_fetched_double</code> • <code>MYSQL_TYPE_DATE</code>, <code>MYSQL_TYPE_NEWDATE</code> - <code>proto_text_fetched_date</code> • <code>MYSQL_TYPE_YEAR</code> - <code>proto_text_fetched_year</code> • <code>MYSQL_TYPE_TIME</code> - <code>proto_text_fetched_time</code> • <code>MYSQL_TYPE_DATETIME</code> - <code>proto_text_fetched_datetime</code>

Statistic	Scope	Description	Notes
			MYSQL_TYPE_TIMESTAMP - <code>proto_text_fetched_timestamp</code> MYSQL_TYPE_STRING, MYSQL_TYPE_VARSTRING, MYSQL_TYPE_VARCHAR - <code>proto_text_fetched_string</code> MYSQL_TYPE_TINY_BLOB, MYSQL_TYPE_MEDIUM_BLOB, MYSQL_TYPE_LONG_BLOB, MYSQL_TYPE_BLOB - <code>proto_text_fetched_blob</code> MYSQL_TYPE_ENUM - <code>proto_text_fetched_enum</code> MYSQL_TYPE_SET - <code>proto_text_fetched_set</code> MYSQL_TYPE_GEOMETRY - <code>proto_text_fetched_geometry</code> - Any MYSQL_TYPE_* not listed before (there should be none) - <code>proto_text_fetched_other</code> Note that the MYSQL_*-type constants may not be associated with the very same SQL column types in every version of MySQL.
proto_binary_fetched_short , proto_binary_fetched_int24 , proto_binary_fetched_int , proto_binary_fetched_bigint , proto_binary_fetched_decimal , proto_binary_fetched_float , proto_binary_fetched_double , proto_binary_fetched_date , proto_binary_fetched_year , proto_binary_fetched_time , proto_binary_fetched_datetime , proto_binary_fetched_timestamp , proto_binary_fetched_string , proto_binary_fetched_blob , proto_binary_fetched_enum , proto_binary_fetched_set , proto_binary_fetched_geometry , proto_binary_fetched_other	Connection	Total number of columns of a certain type fetched from a prepared statement (MySQL binary protocol).	For type mapping see proto_text_* described in the preceding text.

Table 6.31 Returned mysqlnd statistics: Connection

Statistic	Scope	Description	Notes
<code>connect_success</code> <code>connect_failure</code>	Connection	Total number of successful / failed connection attempt.	Reused connections and all other kinds of connections are included.
<code>reconnect</code>	Process	Total number of (real_)connect attempts made on an already opened connection handle.	The code sequence <code>\$link = new mysqli(...); \$link->real_connect(...)</code> will cause a reconnect. But <code>\$link = new mysqli(...); \$link->connect(...)</code> will not because <code>\$link->connect(...)</code> will explicitly close the existing connection before a new connection is established.
<code>pconnect_success</code>	Connection	Total number of successful persistent connection attempts.	Note that <code>connect_success</code> holds the sum of successful persistent and non-persistent connection attempts. The number of successful non-persistent connection attempts is <code>connect_success - pconnect_success</code> .
<code>active_connections</code>	Connection	Total number of active persistent and non-persistent connections.	
<code>active_persistent_connections</code>	Connection	Total number of active persistent connections.	The total number of active non-persistent connections is <code>active_connections - active_persistent_connections</code> .
<code>explicit_close</code>	Connection	Total number of explicitly closed connections (ext/mysqli only).	Examples of code snippets that cause an explicit close : <pre>\$link = new mysqli(...); \$link->close(...); \$link = new mysqli(...); \$link->connect(...);</pre>
<code>implicit_close</code>	Connection	Total number of implicitly closed connections (ext/mysqli only).	Examples of code snippets that cause an implicit close : <code>\$link = new mysqli(...); \$link->real_connect(...)</code> <code>unset(\$link)</code> - Persistent connection: pooled connection has been created with <code>real_connect</code> and there may be unknown options set - close implicitly to avoid returning a connection with unknown options - Persistent connection: <code>ping/change_user</code> fails and ext/mysqli closes the connection - end of script execution: close connections that have not been closed by the user

Statistic	Scope	Description	Notes
<code>disconnect</code>	Connection	Connection failures indicated by the C API call <code>mysql_real_connect</code> during an attempt to establish a connection.	It is called <code>disconnect_close</code> because the connection handle passed to the C API call will be closed.
<code>in_middle</code>	Process	A connection has been closed in the middle of a command execution (outstanding result sets not fetched, after sending a query and before retrieving an answer, while fetching data, while transferring data with LOAD DATA).	Unless you use asynchronous queries this should only happen if your script stops unexpectedly and PHP shuts down the connections for you.
<code>init_command</code>	Connection	Total number of init command executions, for example, <code>mysql_options(MYSQLI_INIT_COMMAND, ...)</code>	The number of successful executions is <code>init_command_executed_count - init_command_failed_count</code> .
<code>init_command_failed</code>	Connection	Total number of failed init commands.	

Table 6.32 Returned mysqlnd statistics: COM_* Command

Statistic	Scope	Description	Notes
<code>com_quit</code> , <code>com_init_db</code> , <code>com_query</code> , <code>com_field_list</code> , <code>com_create_db</code> , <code>com_drop_db</code> , <code>com_refresh</code> , <code>com_shutdown</code> , <code>com_statistics</code> , <code>com_process_info</code> , <code>com_connect</code> , <code>com_process_kill</code> , <code>com_debug</code> , <code>com_ping</code> , <code>com_time</code> , <code>com_delayed_insert</code> , <code>com_change_user</code> , <code>com_binlog_dump</code> , <code>com_table_dump</code> , <code>com_connect_out</code> , <code>com_register_slave</code> , <code>com_stmt_prepare</code> , <code>com_stmt_execute</code> , <code>com_stmt_send_long_data</code> , <code>com_stmt_close</code> , <code>com_stmt_reset</code> , <code>com_stmt_set_option</code> , <code>com_stmt_fetch</code> , <code>com_daemon</code>	Connection	Total number of attempts to send a certain COM_* command from PHP to MySQL.	The statistics are incremented after checking the line and immediately before sending the corresponding MySQL client server protocol packet. If mysqlnd fails to send the packet over the wire the statistics will not be decremented. In case of a failure mysqlnd emits a PHP warning "Error while sending %s packet. PID=%d." Usage examples: • Check if PHP sends certain commands to MySQL, for example, check if a client sends <code>COM_PROCESS_KILL</code> • Calculate the average number of prepared statement executions by comparing <code>COM_EXECUTE</code> with <code>COM_PREPARE</code> • Check if PHP has run any non-prepared SQL statements by checking if <code>COM_QUERY</code> is zero • Identify PHP scripts that run an excessive number of SQL statements by checking <code>COM_QUERY</code> and <code>COM_EXECUTE</code>

Miscellaneous

Table 6.33 Returned mysqlnd statistics: Miscellaneous

Statistic	Scope	Description	Notes
<code>explicit_stmt_close</code> , <code>implicit_stmt_close</code>	Process	Total number of close prepared statements.	A close is always considered explicit but for a failed prepare.

830

Statistic	Scope	Description	Notes
			will get an error. There are only two situations when the packet can be larger than the 2048 bytes default of <code>mysqlnd.net_cmd_buffer_size</code> in PHP 5.3.0: the packet transports a very long error message or the packet holds column meta data from <code>COM_LIST_FIELD</code> (<code>mysql_list_fields</code>) and the meta data comes from a string column with a very long default value (>1900 bytes). No bug report on this exists - it should happen rarely. As of PHP 5.3.2 <code>mysqlnd</code> does not allow setting buffers smaller than 4096 bytes.
<code>connection_reused</code>			

6.6.7 Notes

Copyright 1997-2014 the PHP Documentation Group.

This section provides a collection of miscellaneous notes on MySQL Native Driver usage.

- Using `mysqlnd` means using PHP streams for underlying connectivity. For `mysqlnd`, the PHP streams documentation (<http://www.php.net/manual/en/book.stream>) should be consulted on such details as timeout settings, not the documentation for the MySQL Client Library.

6.6.8 Memory management

Copyright 1997-2014 the PHP Documentation Group.

Introduction

The MySQL Native Driver manages memory different than the MySQL Client Library. The libraries differ in the way memory is allocated and released, how memory is allocated in chunks while reading results from MySQL, which debug and development options exist, and how results read from MySQL are linked to PHP user variables.

The following notes are intended as an introduction and summary to users interested at understanding the MySQL Native Driver at the C code level.

Memory management functions used

All memory allocation and deallocation is done using the PHP memory management functions. Therefore, the memory consumption of `mysqlnd` can be tracked using PHP API calls, such as `memory_get_usage`. Because memory is allocated and released using the PHP memory management, the changes may not immediately become visible at the operating system level. The PHP memory management acts as a proxy which may delay releasing memory towards the system. Due to this, comparing the memory usage of the MySQL Native Driver and the MySQL Client Library is difficult. The MySQL Client Library is using the operating system memory management calls directly, hence the effects can be observed immediately at the operating system level.

Any memory limit enforced by PHP also affects the MySQL Native Driver. This may cause out of memory errors when fetching large result sets that exceed the size of the remaining memory made available by PHP. Because the MySQL Client Library is not using PHP memory management functions, it does not comply to any PHP memory limit set. If using the MySQL Client Library, depending on the deployment model, the memory footprint of the PHP process may grow beyond

the PHP memory limit. But also PHP scripts may be able to process larger result sets as parts of the memory allocated to hold the result sets are beyond the control of the PHP engine.

PHP memory management functions are invoked by the MySQL Native Driver through a lightweight wrapper. Among others, the wrapper makes debugging easier.

Handling of result sets

The various MySQL Server and the various client APIs differentiate between [buffered and unbuffered](#) result sets. Unbuffered result sets are transferred row-by-row from MySQL to the client as the client iterates over the results. Buffered results are fetched in their entirety by the client library before passing them on to the client.

The MySQL Native Driver is using PHP Streams for the network communication with the MySQL Server. Results sent by MySQL are fetched from the PHP Streams network buffers into the result buffer of `mysqlnd`. The result buffer is made of `zvals`. In a second step the results are made available to the PHP script. This final transfer from the result buffer into PHP variables impacts the memory consumption and is mostly noticeable when using buffered result sets.

By default the MySQL Native Driver tries to avoid holding buffered results twice in memory. Results are kept only once in the internal result buffers and their `zvals`. When results are fetched into PHP variables by the PHP script, the variables will reference the internal result buffers. Database query results are not copied and kept in memory only once. Should the user modify the contents of a variable holding the database results a copy-on-write must be performed to avoid changing the referenced internal result buffer. The contents of the buffer must not be modified because the user may decide to read the result set a second time. The copy-on-write mechanism is implemented using an additional reference management list and the use of standard `zval` reference counters. Copy-on-write must also be done if the user reads a result set into PHP variables and frees a result set before the variables are unset.

Generally speaking, this pattern works well for scripts that read a result set once and do not modify variables holding results. Its major drawback is the memory overhead caused by the additional reference management which comes primarily from the fact that user variables holding results cannot be entirely released until the `mysqlnd` reference management stops referencing them. The MySQL Native driver removes the reference to the user variables when the result set is freed or a copy-on-write is performed. An observer will see the total memory consumption grow until the result set is released. Use the [statistics](#) to check whether a script does release result sets explicitly or the driver does implicit releases and thus memory is used for a time longer than necessary. Statistics also help to see how many copy-on-write operations happened.

A PHP script reading many small rows of a buffered result set using a code snippet equal or equivalent to `while ($row = $res->fetch_assoc()) { ... }` may optimize memory consumption by requesting copies instead of references. Albeit requesting copies means keeping results twice in memory, it allows PHP to free the copy contained in `$row` as the result set is being iterated and prior to releasing the result set itself. On a loaded server optimizing peak memory usage may help improving the overall system performance although for an individual script the copy approach may be slower due to additional allocations and memory copy operations.

The copy mode can be enforced by setting `mysqlnd.fetch_data_copy=1`.

Monitoring and debugging

There are multiple ways of tracking the memory usage of the MySQL Native Driver. If the goal is to get a quick high level overview or to verify the memory efficiency of PHP scripts, then check the [statistics](#) collected by the library. The statistics allow you, for example, to catch SQL statements which generate more results than are processed by a PHP script.

The [debug](#) trace log can be configured to record memory management calls. This helps to see when memory is allocated or free'd. However, the size of the requested memory chunks may not be listed.

Some, recent versions of the MySQL Native Driver feature the emulation of random out of memory situations. This feature is meant to be used by the C developers of the library or `mysqlnd` [plugin](#)

authors only. Please, search the source code for corresponding PHP configuration settings and further details. The feature is considered private and may be modified at any time without prior notice.

6.6.9 MySQL Native Driver Plugin API

Copyright 1997-2014 the PHP Documentation Group.

The MySQL Native Driver Plugin API is a feature of MySQL Native Driver, or `mysqlnd`. `mysqlnd` plugins operate in the layer between PHP applications and the MySQL server. This is comparable to MySQL Proxy. MySQL Proxy operates on a layer between any MySQL client application, for example, a PHP application and, the MySQL server. `mysqlnd` plugins can undertake typical MySQL Proxy tasks such as load balancing, monitoring and performance optimizations. Due to the different architecture and location, `mysqlnd` plugins do not have some of MySQL Proxy's disadvantages. For example, with plugins, there is no single point of failure, no dedicated proxy server to deploy, and no new programming language to learn (Lua).

A `mysqlnd` plugin can be thought of as an extension to `mysqlnd`. Plugins can intercept the majority of `mysqlnd` functions. The `mysqlnd` functions are called by the PHP MySQL extensions such as `ext/mysql`, `ext/mysqli`, and `PDO_MYSQL`. As a result, it is possible for a `mysqlnd` plugin to intercept all calls made to these extensions from the client application.

Internal `mysqlnd` function calls can also be intercepted, or replaced. There are no restrictions on manipulating `mysqlnd` internal function tables. It is possible to set things up so that when certain `mysqlnd` functions are called by the extensions that use `mysqlnd`, the call is directed to the appropriate function in the `mysqlnd` plugin. The ability to manipulate `mysqlnd` internal function tables in this way allows maximum flexibility for plugins.

`mysqlnd` plugins are in fact PHP Extensions, written in C, that use the `mysqlnd` plugin API (which is built into MySQL Native Driver, `mysqlnd`). Plugins can be made 100% transparent to PHP applications. No application changes are needed because plugins operate on a different layer. The `mysqlnd` plugin can be thought of as operating in a layer below `mysqlnd`.

The following list represents some possible applications of `mysqlnd` plugins.

- Load Balancing
 - Read/Write Splitting. An example of this is the PECL/mysqlnd_ms (Master Slave) extension. This extension splits read/write queries for a replication setup.
 - Failover
 - Round-Robin, least loaded
- Monitoring
 - Query Logging
 - Query Analysis
 - Query Auditing. An example of this is the PECL/mysqlnd_sip (SQL Injection Protection) extension. This extension inspects queries and executes only those that are allowed according to a ruleset.
- Performance
 - Caching. An example of this is the PECL/mysqlnd_qc (Query Cache) extension.
 - Throttling
 - Sharding. An example of this is the PECL/mysqlnd_mc (Multi Connect) extension. This extension will attempt to split a SELECT statement into n-parts, using SELECT ... LIMIT part_1, SELECT LIMIT part_n. It sends the queries to distinct MySQL servers and merges the result at the client.

MySQL Native Driver Plugins Available

There are a number of `mysqlnd` plugins already available. These include:

- `PECL/mysqlnd_mc` - Multi Connect plugin.
- `PECL/mysqlnd_ms` - Master Slave plugin.
- `PECL/mysqlnd_qc` - Query Cache plugin.
- `PECL/mysqlnd_pscache` - Prepared Statement Handle Cache plugin.
- `PECL/mysqlnd_sip` - SQL Injection Protection plugin.
- `PECL/mysqlnd_uh` - User Handler plugin.

6.6.9.1 A comparison of `mysqlnd` plugins with MySQL Proxy

[Copyright 1997-2014 the PHP Documentation Group.](#)

`mysqlnd` plugins and MySQL Proxy are different technologies using different approaches. Both are valid tools for solving a variety of common tasks such as load balancing, monitoring, and performance enhancements. An important difference is that MySQL Proxy works with all MySQL clients, whereas `mysqlnd` plugins are specific to PHP applications.

As a PHP Extension, a `mysqlnd` plugin gets installed on the PHP application server, along with the rest of PHP. MySQL Proxy can either be run on the PHP application server or can be installed on a dedicated machine to handle multiple PHP application servers.

Deploying MySQL Proxy on the application server has two advantages:

1. No single point of failure
2. Easy to scale out (horizontal scale out, scale by client)

MySQL Proxy (and `mysqlnd` plugins) can solve problems easily which otherwise would have required changes to existing applications.

However, MySQL Proxy does have some disadvantages:

- MySQL Proxy is a new component and technology to master and deploy.
- MySQL Proxy requires knowledge of the Lua scripting language.

MySQL Proxy can be customized with C and Lua programming. Lua is the preferred scripting language of MySQL Proxy. For most PHP experts Lua is a new language to learn. A `mysqlnd` plugin can be written in C. It is also possible to write plugins in PHP using `PECL/mysqlnd_uh`.

MySQL Proxy runs as a daemon - a background process. MySQL Proxy can recall earlier decisions, as all state can be retained. However, a `mysqlnd` plugin is bound to the request-based lifecycle of PHP. MySQL Proxy can also share one-time computed results among multiple application servers. A `mysqlnd` plugin would need to store data in a persistent medium to be able to do this. Another daemon would need to be used for this purpose, such as Memcache. This gives MySQL Proxy an advantage in this case.

MySQL Proxy works on top of the wire protocol. With MySQL Proxy you have to parse and reverse engineer the MySQL Client Server Protocol. Actions are limited to those that can be achieved by manipulating the communication protocol. If the wire protocol changes (which happens very rarely) MySQL Proxy scripts would need to be changed as well.

`mysqlnd` plugins work on top of the C API, which mirrors the `libmysqlclient` client and Connector/C APIs. This C API is basically a wrapper around the MySQL Client Server protocol, or wire protocol,

as it is sometimes called. You can intercept all C API calls. PHP makes use of the C API, therefore you can hook all PHP calls, without the need to program at the level of the wire protocol.

[mysqlnd](#) implements the wire protocol. Plugins can therefore parse, reverse engineer, manipulate and even replace the communication protocol. However, this is usually not required.

As plugins allow you to create implementations that use two levels (C API and wire protocol), they have greater flexibility than MySQL Proxy. If a [mysqlnd](#) plugin is implemented using the C API, any subsequent changes to the wire protocol do not require changes to the plugin itself.

6.6.9.2 Obtaining the mysqlnd plugin API

Copyright 1997-2014 the PHP Documentation Group.

The [mysqlnd](#) plugin API is simply part of the MySQL Native Driver PHP extension, [ext/mysqlnd](#). Development started on the [mysqlnd](#) plugin API in December 2009. It is developed as part of the PHP source repository, and as such is available to the public either via Git, or through source snapshot downloads.

The following table shows PHP versions and the corresponding [mysqlnd](#) version contained within.

Table 6.34 The bundled mysqlnd version per PHP release

PHP Version	MySQL Native Driver version
5.3.0	5.0.5
5.3.1	5.0.5
5.3.2	5.0.7
5.3.3	5.0.7
5.3.4	5.0.7

Plugin developers can determine the [mysqlnd](#) version through accessing [MYSQLND_VERSION](#), which is a string of the format “mysqlnd 5.0.7-dev - 091210 - \$Revision: 300535”, or through [MYSQLND_VERSION_ID](#), which is an integer such as 50007. Developers can calculate the version number as follows:

Table 6.35 MYSQLND_VERSION_ID calculation table

Version (part)	Example
Major*10000	5*10000 = 50000
Minor*100	0*100 = 0
Patch	7 = 7
MYSQLND_VERSION_ID	50007

During development, developers should refer to the [mysqlnd](#) version number for compatibility and version tests, as several iterations of [mysqlnd](#) could occur during the lifetime of a PHP development branch with a single PHP version number.

6.6.9.3 MySQL Native Driver Plugin Architecture

Copyright 1997-2014 the PHP Documentation Group.

This section provides an overview of the [mysqlnd](#) plugin architecture.

MySQL Native Driver Overview

Before developing [mysqlnd](#) plugins, it is useful to know a little of how [mysqlnd](#) itself is organized. [mysqlnd](#) consists of the following modules:

Table 6.36 The mysqlnd organization chart, per module

Modules Statistics	mysqlnd_statistics.c
Connection	mysqlnd.c
Resultset	mysqlnd_result.c
Resultset Metadata	mysqlnd_result_meta.c
Statement	mysqlnd_ps.c
Network	mysqlnd_net.c
Wire protocol	mysqlnd_wireprotocol.c

C Object Oriented Paradigm

At the code level, `mysqlnd` uses a C pattern for implementing object orientation.

In C you use a `struct` to represent an object. Members of the struct represent object properties. Struct members pointing to functions represent methods.

Unlike with other languages such as C++ or Java, there are no fixed rules on inheritance in the C object oriented paradigm. However, there are some conventions that need to be followed that will be discussed later.

The PHP Life Cycle

When considering the PHP life cycle there are two basic cycles:

- PHP engine startup and shutdown cycle
- Request cycle

When the PHP engine starts up it will call the module initialization (MINIT) function of each registered extension. This allows each module to setup variables and allocate resources that will exist for the lifetime of the PHP engine process. When the PHP engine shuts down it will call the module shutdown (MSHUTDOWN) function of each extension.

During the lifetime of the PHP engine it will receive a number of requests. Each request constitutes another life cycle. On each request the PHP engine will call the request initialization function of each extension. The extension can perform any variable setup and resource allocation required for request processing. As the request cycle ends the engine calls the request shutdown (RSHUTDOWN) function of each extension so the extension can perform any cleanup required.

How a plugin works

A `mysqlnd` plugin works by intercepting calls made to `mysqlnd` by extensions that use `mysqlnd`. This is achieved by obtaining the `mysqlnd` function table, backing it up, and replacing it by a custom function table, which calls the functions of the plugin as required.

The following code shows how the `mysqlnd` function table is replaced:

```
/* a place to store original function table */
struct st_mysqlnd_conn_methods org_methods;
void minit_register_hooks(TSRMLS_D) {
    /* active function table */
    struct st_mysqlnd_conn_methods * current_methods
        = mysqlnd_conn_get_methods();
    /* backup original function table */
    memcpy(&org_methods, current_methods,
        sizeof(struct st_mysqlnd_conn_methods));
    /* install new methods */
    current_methods->query = MYSQLND_METHOD(my_conn_class, query);
}
```

```
}
```

Connection function table manipulations must be done during Module Initialization (MINIT). The function table is a global shared resource. In an multi-threaded environment, with a TSRM build, the manipulation of a global shared resource during the request processing will almost certainly result in conflicts.

Note

Do not use any fixed-size logic when manipulating the `mysqlnd` function table: new methods may be added at the end of the function table. The function table may change at any time in the future.

Calling parent methods

If the original function table entries are backed up, it is still possible to call the original function table entries - the parent methods.

In some cases, such as for `Connection::stmt_init()`, it is vital to call the parent method prior to any other activity in the derived method.

```
MYSQLND_METHOD(my_conn_class, query)(MYSQLND *conn,
    const char *query, unsigned int query_len TSRMLS_DC) {
    php_printf("my_conn_class::query(query = %s)\n", query);
    query = "SELECT 'query rewritten' FROM DUAL";
    query_len = strlen(query);
    return org_methods.query(conn, query, query_len); /* return with call to parent */
}
```

Extending properties

A `mysqlnd` object is represented by a C struct. It is not possible to add a member to a C struct at run time. Users of `mysqlnd` objects cannot simply add properties to the objects.

Arbitrary data (properties) can be added to a `mysqlnd` objects using an appropriate function of the `mysqlnd_plugin_get_plugin_<object>_data()` family. When allocating an object `mysqlnd` reserves space at the end of the object to hold a `void *` pointer to arbitrary data. `mysqlnd` reserves space for one `void *` pointer per plugin.

The following table shows how to calculate the position of the pointer for a specific plugin:

Table 6.37 Pointer calculations for mysqlnd

Memory address	Contents
0	Beginning of the <code>mysqlnd</code> object C struct
n	End of the <code>mysqlnd</code> object C struct
n + (m x sizeof(void*))	<code>void*</code> to object data of the m-th plugin

If you plan to subclass any of the `mysqlnd` object constructors, which is allowed, you must keep this in mind!

The following code shows extending properties:

```
/* any data we want to associate */
typedef struct my_conn_properties {
    unsigned long query_counter;
} MY_CONN_PROPERTIES;
/* plugin id */
```



```

unsigned int my_plugin_id;
void minit_register_hooks(TSRMLS_D) {
    /* obtain unique plugin ID */
    my_plugin_id = mysqlnd_plugin_register();
    /* snip - see Extending Connection: methods */
}
static MY_CONN_PROPERTIES** get_conn_properties(const MYSQLND *conn TSRMLS_DC) {
    MY_CONN_PROPERTIES** props;
    props = (MY_CONN_PROPERTIES**)mysqlnd_plugin_get_plugin_connection_data(
        conn, my_plugin_id);
    if (!props || !(*props)) {
        *props = mnd_pecalloc(1, sizeof(MY_CONN_PROPERTIES), conn->persistent);
        (*props)->query_counter = 0;
    }
    return props;
}

```

The plugin developer is responsible for the management of plugin data memory.

Use of the `mysqlnd` memory allocator is recommended for plugin data. These functions are named using the convention: `mnd_*loc()`. The `mysqlnd` allocator has some useful features, such as the ability to use a debug allocator in a non-debug build.

Table 6.38 When and how to subclass

	When to subclass?	Each instance has its own private function table?	How to subclass?
Connection (MYSQLND)	MINIT	No	<code>mysqlnd_conn_get_methods()</code>
Resultset (MYSQLND_RES)	MINIT or later	Yes	<code>mysqlnd_result_get_methods()</code> or object method function table manipulation
Resultset Meta (MYSQLND_RES_METADATA)	MINIT	No	<code>mysqlnd_result_metadata_get_methods()</code>
Statement (MYSQLND_STMT)	MINIT	No	<code>mysqlnd_stmt_get_methods()</code>
Network (MYSQLND_NET)	MINIT or later	Yes	<code>mysqlnd_net_get_methods()</code> or object method function table manipulation
Wire protocol (MYSQLND_PROTOCOL)	MINIT or later	Yes	<code>mysqlnd_protocol_get_methods()</code> or object method function table manipulation

You must not manipulate function tables at any time later than MINIT if it is not allowed according to the above table.

Some classes contain a pointer to the method function table. All instances of such a class will share the same function table. To avoid chaos, in particular in threaded environments, such function tables must only be manipulated during MINIT.

Other classes use copies of a globally shared function table. The class function table copy is created together with the object. Each object uses its own function table. This gives you two options: you can manipulate the default function table of an object at MINIT, and you can additionally refine methods of an object without impacting other instances of the same class.

The advantage of the shared function table approach is performance. There is no need to copy a function table for each and every object.

Table 6.39 Constructor status

	Allocation, construction, reset	Can be modified?	Caller
Connection (MYSQLND)	mysqlnd_init()	No	mysqlnd_connect()
Resultset (MYSQLND_RESULT)	Allocation: • Connection::result_init() Reset and re-initialized during: • Result::use_result() • Result::store_result	Yes, but call parent!	• Connection::list_fields() • Statement::get_result() • Statement::prepare() (Metadata only) • Statement::resultMetadata()
Resultset Meta (MYSQLND_RES_METADATA)	Connection::result_meta_init()	Yes, but call parent!	Result::read_result_metadata()
Statement (MYSQLND_STMT)	Connection::stmt_init()	Yes, but call parent!	Connection::stmt_init()
Network (MYSQLND_NET)	mysqlnd_net_init()	No	Connection::init()
Wire protocol (MYSQLND_PROTOCOL)	mysqlnd_protocol_init()	No	Connection::init()

It is strongly recommended that you do not entirely replace a constructor. The constructors perform memory allocations. The memory allocations are vital for the `mysqlnd` plugin API and the object logic of `mysqlnd`. If you do not care about warnings and insist on hooking the constructors, you should at least call the parent constructor before doing anything in your constructor.

Regardless of all warnings, it can be useful to subclass constructors. Constructors are the perfect place for modifying the function tables of objects with non-shared object tables, such as Resultset, Network, Wire Protocol.

Table 6.40 Destruction status

	Derived method must call parent?	Destructor
Connection	yes, after method execution	free_contents(), end_psession()
Resultset	yes, after method execution	free_result()
Resultset Meta	yes, after method execution	free()
Statement	yes, after method execution	dtor(), free_stmt_content()
Network	yes, after method execution	free()
Wire protocol	yes, after method execution	free()

The destructors are the appropriate place to free properties, `mysqlnd_plugin_get_plugin_<object>_data()`.

The listed destructors may not be equivalent to the actual `mysqlnd` method freeing the object itself. However, they are the best possible place for you to hook in and free your plugin data. As with constructors you may replace the methods entirely but this is not recommended. If multiple methods are listed in the above table you will need to hook all of the listed methods and free your plugin data in whichever method is called first by `mysqlnd`.

The recommended method for plugins is to simply hook the methods, free your memory and call the parent implementation immediately following this.

Caution

Due to a bug in PHP versions 5.3.0 to 5.3.3, plugins do not associate plugin data with a persistent connection. This is because `ext/mysql` and `ext/mysqli` do not trigger all the necessary `mysqlnd_end_psession()` method calls and the plugin may therefore leak memory. This has been fixed in PHP 5.3.4.

6.6.9.4 The `mysqlnd` plugin API

Copyright 1997-2014 the PHP Documentation Group.

The following is a list of functions provided in the `mysqlnd` plugin API:

- `mysqlnd_plugin_register()`
- `mysqlnd_plugin_count()`
- `mysqlnd_plugin_get_plugin_connection_data()`
- `mysqlnd_plugin_get_plugin_result_data()`
- `mysqlnd_plugin_get_plugin_stmt_data()`
- `mysqlnd_plugin_get_plugin_net_data()`
- `mysqlnd_plugin_get_plugin_protocol_data()`
- `mysqlnd_conn_get_methods()`
- `mysqlnd_result_get_methods()`
- `mysqlnd_result_meta_get_methods()`
- `mysqlnd_stmt_get_methods()`
- `mysqlnd_net_get_methods()`
- `mysqlnd_protocol_get_methods()`

There is no formal definition of what a plugin is and how a plugin mechanism works.

Components often found in plugins mechanisms are:

- A plugin manager
- A plugin API
- Application services (or modules)
- Application service APIs (or module APIs)

The `mysqlnd` plugin concept employs these features, and additionally enjoys an open architecture.

No Restrictions

A plugin has full access to the inner workings of `mysqlnd`. There are no security limits or restrictions. Everything can be overwritten to implement friendly or hostile algorithms. It is recommended you only deploy plugins from a trusted source.

As discussed previously, plugins can use pointers freely. These pointers are not restricted in any way, and can point into another plugin's data. Simple offset arithmetic can be used to read another plugin's data.

It is recommended that you write cooperative plugins, and that you always call the parent method. The plugins should always cooperate with `mysqlnd` itself.

Table 6.41 Issues: an example of chaining and cooperation

Extension	<code>mysqlnd.query()</code> pointer	call stack if calling parent
<code>ext/mysqlnd</code>	<code>mysqlnd.query()</code>	<code>mysqlnd.query</code>
<code>ext/mysqlnd_cache</code>	<code>mysqlnd_cache.query()</code>	1. <code>mysqlnd_cache.query()</code> 2. <code>mysqlnd.query</code>
<code>ext/mysqlnd_monitor</code>	<code>mysqlnd_monitor.query()</code>	1. <code>mysqlnd_monitor.query()</code> 2. <code>mysqlnd_cache.query()</code> 3. <code>mysqlnd.query</code>

In this scenario, a cache (`ext/mysqlnd_cache`) and a monitor (`ext/mysqlnd_monitor`) plugin are loaded. Both subclass `Connection::query()`. Plugin registration happens at `MINIT` using the logic shown previously. PHP calls extensions in alphabetical order by default. Plugins are not aware of each other and do not set extension dependencies.

By default the plugins call the parent implementation of the query method in their derived version of the method.

PHP Extension Recap

This is a recap of what happens when using an example plugin, `ext/mysqlnd_plugin`, which exposes the `mysqlnd` C plugin API to PHP:

- Any PHP MySQL application tries to establish a connection to 192.168.2.29
- The PHP application will either use `ext/mysql`, `ext/mysqli` or `PDO_MYSQL`. All three PHP MySQL extensions use `mysqlnd` to establish the connection to 192.168.2.29.
- `MySQLnd` calls its connect method, which has been subclassed by `ext/mysqlnd_plugin`.
- `ext/mysqlnd_plugin` calls the userspace hook `proxy::connect()` registered by the user.
- The userspace hook changes the connection host IP from 192.168.2.29 to 127.0.0.1 and returns the connection established by `parent::connect()`.
- `ext/mysqlnd_plugin` performs the equivalent of `parent::connect(127.0.0.1)` by calling the original `mysqlnd` method for establishing a connection.
- `ext/mysqlnd` establishes a connection and returns to `ext/mysqlnd_plugin`. `ext/mysqlnd_plugin` returns as well.
- Whatever PHP MySQL extension had been used by the application, it receives a connection to 127.0.0.1. The PHP MySQL extension itself returns to the PHP application. The circle is closed.

6.6.9.5 Getting started building a `mysqlnd` plugin

Copyright 1997-2014 the PHP Documentation Group.

It is important to remember that a `mysqlnd` plugin is itself a PHP extension.

The following code shows the basic structure of the `MINIT` function that will be used in the typical `mysqlnd` plugin:

```
/* my_php_mysqlnd_plugin.c */
```

```
static PHP_MINIT_FUNCTION(mysqlnd_plugin) {
    /* globals, ini entries, resources, classes */
    /* register mysqlnd plugin */
    mysqlnd_plugin_id = mysqlnd_plugin_register();
    conn_m = mysqlnd_get_conn_methods();
    memcpy(org_conn_m, conn_m,
           sizeof(struct st_mysqlnd_conn_methods));
    conn_m->query = MYSQLND_METHOD(mysqlnd_plugin_conn, query);
    conn_m->connect = MYSQLND_METHOD(mysqlnd_plugin_conn, connect);
}
```

```
/* my_mysqlnd_plugin.c */
enum_func_status MYSQLND_METHOD(mysqlnd_plugin_conn, query)(/* ... */) {
    /* ... */
}
enum_func_status MYSQLND_METHOD(mysqlnd_plugin_conn, connect)(/* ... */) {
    /* ... */
}
```

Task analysis: from C to userspace

```
class proxy extends mysqlnd_plugin_connection {
    public function connect($host, ...) { .. }
}
mysqlnd_plugin_set_conn_proxy(new proxy());
```

Process:

1. PHP: user registers plugin callback
2. PHP: user calls any PHP MySQL API to connect to MySQL
3. C: ext/*mysql* calls mysqlnd method
4. C: mysqlnd ends up in ext/mysqlnd_plugin
5. C: ext/mysqlnd_plugin
 - a. Calls userspace callback
 - b. Or original `mysqlnd` method, if userspace callback not set

You need to carry out the following:

1. Write a class "mysqlnd_plugin_connection" in C
2. Accept and register proxy object through "mysqlnd_plugin_set_conn_proxy()"
3. Call userspace proxy methods from C (optimization - zend_interfaces.h)

Userspace object methods can either be called using `call_user_function()` or you can operate at a level closer to the Zend Engine and use `zend_call_method()`.

Optimization: calling methods from C using zend_call_method

The following code snippet shows the prototype for the `zend_call_method` function, taken from `zend_interfaces.h`.

```
ZEND_API zval* zend_call_method(
```

```

zval **object_pp, zend_class_entry *obj_ce,
zend_function **fn_proxy, char *function_name,
int function_name_len, zval **retval_ptr_ptr,
int param_count, zval* arg1, zval* arg2 TSRMLS_DC
);

```

Zend API supports only two arguments. You may need more, for example:

```

enum_func_status (*func_mysqlnd_conn__connect)(
    MYSQLND *conn, const char *host,
    const char * user, const char * passwd,
    unsigned int passwd_len, const char * db,
    unsigned int db_len, unsigned int port,
    const char * socket, unsigned int mysql_flags TSRMLS_DC
);

```

To get around this problem you will need to make a copy of [zend_call_method\(\)](#) and add a facility for additional parameters. You can do this by creating a set of [MY_ZEND_CALL_METHOD_WRAPPER](#) macros.

Calling PHP userspace

This code snippet shows the optimized method for calling a userspace function from C:

```

/* my_mysqlnd_plugin.c */
MYSQLND_METHOD(my_conn_class,connect)(
    MYSQLND *conn, const char *host /* ... */ TSRMLS_DC) {
    enum_func_status ret = FAIL;
    zval * global_user_conn_proxy = fetch_userspace_proxy();
    if (global_user_conn_proxy) {
        /* call userspace proxy */
        ret = MY_ZEND_CALL_METHOD_WRAPPER(global_user_conn_proxy, host, /*...*/);
    } else {
        /* or original mysqlnd method = do nothing, be transparent */
        ret = org_methods.connect(conn, host, user, passwd,
            passwd_len, db, db_len, port,
            socket, mysql_flags TSRMLS_CC);
    }
    return ret;
}

```

Calling userspace: simple arguments

```

/* my_mysqlnd_plugin.c */
MYSQLND_METHOD(my_conn_class,connect)(
    /* ... */ /*, const char *host, /* ... */ */ {
    /* ... */
    if (global_user_conn_proxy) {
        /* ... */
        zval* zv_host;
        MAKE_STD_ZVAL(zv_host);
        ZVAL_STRING(zv_host, host, 1);
        MY_ZEND_CALL_METHOD_WRAPPER(global_user_conn_proxy, zv_retval, zv_host /*, ... */);
        zval_ptr_dtor(&zv_host);
        /* ... */
    }
    /* ... */
}

```

Calling userspace: structs as arguments

```

/* my_mysqlnd_plugin.c */
MYSQLND_METHOD(my_conn_class, connect)(
    MYSQLND *conn, /* ... */) {
    /* ... */
    if (global_user_conn_proxy) {
        /* ... */
        zval* zv_conn;
        ZEND_REGISTER_RESOURCE(zv_conn, (void *)conn, le_mysqlnd_plugin_conn);
        MY_ZEND_CALL_METHOD_WRAPPER(global_user_conn_proxy, zv_retval, zv_conn, zv_host /*, ... */);
        zval_ptr_dtor(&zv_conn);
        /* ... */
    }
    /* ... */
}

```

The first argument of many [mysqlnd](#) methods is a C "object". For example, the first argument of the `connect()` method is a pointer to [MYSQLND](#). The struct [MYSQLND](#) represents a [mysqlnd](#) connection object.

The [mysqlnd](#) connection object pointer can be compared to a standard I/O file handle. Like a standard I/O file handle a [mysqlnd](#) connection object shall be linked to the userspace using the PHP resource variable type.

From C to userspace and back

```

class proxy extends mysqlnd_plugin_connection {
    public function connect($conn, $host, ...) {
        /* "pre" hook */
        printf("Connecting to host = '%s'\n", $host);
        debug_print_backtrace();
        return parent::connect($conn);
    }
    public function query($conn, $query) {
        /* "post" hook */
        $ret = parent::query($conn, $query);
        printf("Query = '%s'\n", $query);
        return $ret;
    }
}
mysqlnd_plugin_set_conn_proxy(new proxy());

```

PHP users must be able to call the parent implementation of an overwritten method.

As a result of subclassing it is possible to refine only selected methods and you can choose to have "pre" or "post" hooks.

Buildin class: `mysqlnd_plugin_connection::connect()`

```

/* my_mysqlnd_plugin_classes.c */
PHP_METHOD("mysqlnd_plugin_connection", connect) {
    /* ... simplified! ... */
    zval* mysqlnd_rsrc;
    MYSQLND* conn;
    char* host; int host_len;
    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "rs",
        &mysqlnd_rsrc, &host, &host_len) == FAILURE) {
        RETURN_NULL();
    }
    ZEND_FETCH_RESOURCE(conn, MYSQLND* conn, &mysqlnd_rsrc, -1,
        "Mysqlnd Connection", le_mysqlnd_plugin_conn);
    if (PASS == org_methods.connect(conn, host, /* simplified! */ TSRMLS_CC))
        RETVAL_TRUE;
}

```

```
else
    RETVAL_FALSE;
}
```

6.7 Mysqlnd replication and load balancing plugin

Copyright 1997-2014 the PHP Documentation Group.

The mysqlnd replication and load balancing plugin ([mysqlnd_ms](#)) adds easy to use MySQL replication support to all PHP MySQL extensions that use [mysqlnd](#).

As of version PHP 5.3.3 the MySQL native driver for PHP ([mysqlnd](#)) features an internal plugin C API. C plugins, such as the replication and load balancing plugin, can extend the functionality of [mysqlnd](#).

The MySQL native driver for PHP is a C library that ships together with PHP as of PHP 5.3.0. It serves as a drop-in replacement for the MySQL Client Library (libmysqlclient). Using mysqlnd has several advantages: no extra downloads are required because it's bundled with PHP, it's under the PHP license, there is lower memory consumption in certain cases, and it contains new functionality such as asynchronous queries.

Mysqlnd plugins like [mysqlnd_ms](#) operate, for the most part, transparently from a user perspective. The replication and load balancing plugin supports all PHP applications, and all MySQL PHP extensions. It does not change existing APIs. Therefore, it can easily be used with existing PHP applications.

6.7.1 Key Features

Copyright 1997-2014 the PHP Documentation Group.

The key features of PECL/mysqlnd_ms are as follows.

- Transparent and therefore easy to use.
 - Supports all of the PHP MySQL extensions.
 - SSL support.
 - A consistent API.
 - Little to no application changes required, dependent on the required usage scenario.
 - Lazy connections: connections to master and slave servers are not opened before a SQL statement is executed.
 - Optional: automatic use of master after the first write in a web request, to lower the possible impact of replication lag.
- Can be used with any MySQL clustering solution.
 - MySQL Replication: Read-write splitting is done by the plugin. Primary focus of the plugin.
 - MySQL Cluster: Read-write splitting can be disabled. Configuration of multiple masters possible
 - Third-party solutions: the plugin is optimized for MySQL Replication but can be used with any other kind of MySQL clustering solution.
- Featured read-write split strategies
 - Automatic detection of SELECT.
 - Supports SQL hints to overrule automatism.

- User-defined.
- Can be disabled for, for example, when using synchronous clusters such as MySQL Cluster.
- Featured load balancing strategies
 - Round Robin: choose a different slave in round-robin fashion for every slave request.
 - Random: choose a random slave for every slave request.
 - Random once (sticky): choose a random slave once to run all slave requests for the duration of a web request.
 - User-defined. The application can register callbacks with `mysqlnd_ms`.
 - PHP 5.4.0 or newer: transaction aware when using API calls only to control transactions.
 - Weighted load balancing: servers can be assigned different priorities, for example, to direct more requests to a powerful machine than to another less powerful machine. Or, to prefer nearby machines to reduce latency.
- Global transaction ID
 - Client-side emulation. Makes manual master server failover and slave promotion easier with asynchronous clusters, such as MySQL Replication.
 - Support for built-in global transaction identifier feature of MySQL 5.6.5 or newer.
 - Supports using transaction ids to identify up-to-date asynchronous slaves for reading when session consistency is required. Please, note the restrictions mentioned in the manual.
 - Throttling: optionally, the plugin can wait for a slave to become "synchronous" before continuing.
- Service and consistency levels
 - Applications can request eventual, session and strong consistency service levels for connections. Appropriate cluster nodes will be searched automatically.
 - Eventual consistent MySQL Replication slave accesses can be replaced with fast local cache accesses transparently to reduce server load.
- Partitioning and sharding
 - Servers of a replication cluster can be organized into groups. SQL hints can be used to manually direct queries to a specific group. Grouping can be used to partition (shard) the data, or to cure the issue of hotspots with updates.
 - MySQL Replication filters are supported through the table filter.
- MySQL Fabric
 - [Experimental support](#) for MySQL Fabric is included.

6.7.2 Limitations

Copyright 1997-2014 the PHP Documentation Group.

The built-in read-write-split mechanism is very basic. Every query which starts with `SELECT` is considered a read request to be sent to a MySQL slave server. All other queries (such as `SHOW` statements) are considered as write requests that are sent to the MySQL master server. The build-in behavior can be overruled using [SQL hints](#), or a user-defined [callback function](#).

The read-write splitter is not aware of multi-statements. Multi-statements are considered as one statement. The decision of where to run the statement will be based on the beginning of the statement string. For example, if using `mysqli_multi_query` to execute the multi-statement `SELECT id FROM test ; INSERT INTO test(id) VALUES (1)`, the statement will be redirected to a slave server because it begins with `SELECT`. The `INSERT` statement, which is also part of the multi-statement, will not be redirected to a master server.

Note

Applications must be aware of the consequences of connection switches that are performed for load balancing purposes. Please check the documentation on [connection pooling and switching](#), [transaction handling](#), [failover load balancing](#) and [read-write splitting](#).

6.7.3 On the name

Copyright 1997-2014 the PHP Documentation Group.

The shortcut `mysqlnd_ms` stands for `mysqlnd master slave plugin`. The name was chosen for a quick-and-dirty proof-of-concept. In the beginning the developers did not expect to continue using the code base.

6.7.4 Quickstart and Examples

Copyright 1997-2014 the PHP Documentation Group.

The `mysqlnd` replication load balancing plugin is easy to use. This quickstart will demo typical use-cases, and provide practical advice on getting started.

It is strongly recommended to read the reference sections in addition to the quickstart. The quickstart tries to avoid discussing theoretical concepts and limitations. Instead, it will link to the reference sections. It is safe to begin with the quickstart. However, before using the plugin in mission critical environments we urge you to read additionally the background information from the reference sections.

The focus is on using PECL `mysqlnd_ms` for work with an asynchronous MySQL cluster, namely MySQL replication. Generally speaking an asynchronous cluster is more difficult to use than a synchronous one. Thus, users of, for example, MySQL Cluster will find more information than needed.

6.7.4.1 Setup

Copyright 1997-2014 the PHP Documentation Group.

The plugin is implemented as a PHP extension. See also the [installation instructions](#) to install the `PECL/mysqlnd_ms` extension.

Compile or configure the PHP MySQL extension (API) (`mysqli`, `PDO_MYSQL`, `mysql`) that you plan to use with support for the `mysqlnd` library. PECL/`mysqlnd_ms` is a plugin for the `mysqlnd` library. To use the plugin with any of the PHP MySQL extensions, the extension has to use the `mysqlnd` library.

Then, load the extension into PHP and activate the plugin in the PHP configuration file using the PHP configuration directive named `mysqlnd_ms.enable`.

Example 6.206 Enabling the plugin (php.ini)

```
mysqlnd_ms.enable=1
mysqlnd_ms.config_file=/path/to/mysqlnd_ms_plugin.ini
```

The plugin uses its own configuration file. Use the PHP configuration directive `mysqlnd_ms.config_file` to set the full file path to the plugin-specific configuration file. This file must be readable by PHP (e.g.,

the web server user). Please note, the configuration directive [mysqlnd_ms.config_file](#) supersedes [mysqlnd_ms.ini_file](#) since 1.4.0. It is a common pitfall to use the old, no longer available configuration directive.

Create a plugin-specific configuration file. Save the file to the path set by the PHP configuration directive [mysqlnd_ms.config_file](#).

The plugins [configuration file](#) is JSON based. It is divided into one or more sections. Each section has a name, for example, [myapp](#). Every section makes its own set of configuration settings.

A section must, at a minimum, list the MySQL replication master server, and set a list of slaves. The plugin supports using only one master server per section. Multi-master MySQL replication setups are not yet fully supported. Use the configuration setting [master](#) to set the hostname, and the port or socket of the MySQL master server. MySQL slave servers are configured using the [slave](#) keyword.

Example 6.207 Minimal plugin-specific configuration file (mysqlnd_ms_plugin.ini)

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": [
    ]
  }
}
```

Configuring a MySQL slave server list is required, although it may contain an empty list. It is recommended to always configure at least one slave server.

Server lists can use [anonymous or non-anonymous syntax](#). Non-anonymous lists include alias names for the servers, such as [master_0](#) for the master in the above example. The quickstart uses the more verbose non-anonymous syntax.

Example 6.208 Recommended minimal plugin-specific config (mysqlnd_ms_plugin.ini)

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      }
    }
  }
}
```

If there are at least two servers in total, the plugin can start to load balance and switch connections. Switching connections is not always transparent and can cause issues in certain cases. The reference sections about [connection pooling and switching](#), [transaction handling](#), [fail over load balancing](#) and [read-write splitting](#) all provide more details. And potential pitfalls are described later in this guide.

It is the responsibility of the application to handle potential issues caused by connection switches, by configuring a master with at least one slave server, which allows switching to work therefore related problems can be found.

The MySQL master and MySQL slave servers, which you configure, do not need to be part of MySQL replication setup. For testing purpose you can use single MySQL server and make it known to the plugin as a master and slave server as shown below. This could help you to detect many potential issues with connection switches. However, such a setup will not be prone to the issues caused by replication lag.

Example 6.209 Using one server as a master and as a slave (testing only!)

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    }
  }
}
```

The plugin attempts to notify you of invalid configurations. Since 1.5.0 it will throw a warning during PHP startup if the configuration file cannot be read, is empty or parsing the JSON failed. Depending on your PHP settings those errors may appear in some log files only. Further validation is done when a connection is to be established and the configuration file is searched for valid sections. Setting [mysqlnd_ms.force_config_usage](#) may help debugging a faulty setup. Please, see also [configuration file debugging notes](#).

6.7.4.2 Running statements

Copyright 1997-2014 the PHP Documentation Group.

The plugin can be used with any PHP MySQL extension ([mysqli](#), [mysql](#), and [PDO_MYSQL](#)) that is compiled to use the [mysqlnd](#) library. [PECL/mysqlnd_ms](#) plugs into the [mysqlnd](#) library. It does not change the API or behavior of those extensions.

Whenever a connection to MySQL is being opened, the plugin compares the host parameter value of the connect call, with the section names from the plugin specific configuration file. If, for example, the plugin specific configuration file has a section [myapp](#) then the section should be referenced by opening a MySQL connection to the host [myapp](#)

Example 6.210 Plugin specific configuration file (mysqlnd_ms_plugin.ini)

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
```

```

        "slave_0": {
            "host": "192.168.2.27",
            "port": "3306"
        }
    }
}
}

```

Example 6.211 Opening a load balanced connection

```

<?php
/* Load balanced following "myapp" section rules from the plugins config file */
$mysqli = new mysqli("myapp", "username", "password", "database");
$pdo = new PDO('mysql:host=myapp;dbname=database', 'username', 'password');
$mysql = mysqli_connect("myapp", "username", "password");
?>

```

The connection examples above will be load balanced. The plugin will send read-only statements to the MySQL slave server with the IP [192.168.2.27](#) and will listen on port [3306](#) for the MySQL client connection. All other statements will be directed to the MySQL master server running on the host [localhost](#). If on Unix like operating systems, the master on [localhost](#) will be accepting MySQL client connections on the Unix domain socket [/tmp/mysql.sock](#), while TCP/IP is the default port on Windows. The plugin will use the user name [username](#) and the password [password](#) to connect to any of the MySQL servers listed in the section [myapp](#) of the plugins configuration file. Upon connect, the plugin will select [database](#) as the current schemata.

The username, password and schema name are taken from the connect API calls and used for all servers. In other words: you must use the same username and password for every MySQL server listed in a plugin configuration file section. This is not a general limitation. As of [PECL/mysqlnd_ms 1.1.0](#), it is possible to set the [username](#) and [password](#) for any server in the plugins configuration file, to be used instead of the credentials passed to the API call.

The plugin does not change the API for running statements. [Read-write splitting](#) works out of the box. The following example assumes that there is no significant replication lag between the master and the slave.

Example 6.212 Executing statements

```

<?php
/* Load balanced following "myapp" section rules from the plugins config file */
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno()) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Statements will be run on the master */
if (!$mysqli->query("DROP TABLE IF EXISTS test")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
if (!$mysqli->query("CREATE TABLE test(id INT)")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
if (!$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* read-only: statement will be run on a slave */
if (!$res = $mysqli->query("SELECT id FROM test")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
} else {
    $row = $res->fetch_assoc();
}

```

```

    $res->close();
    printf("Slave returns id = '%s'\n", $row['id']);
}
$mysqli->close();
?>

```

The above example will output something similar to:

```
Slave returns id = '1'
```

6.7.4.3 Connection state

Copyright 1997-2014 the PHP Documentation Group.

The plugin changes the semantics of a PHP MySQL connection handle. A new connection handle represents a connection pool, instead of a single MySQL client-server network connection. The connection pool consists of a master connection, and optionally any number of slave connections.

Every connection from the connection pool has its own state. For example, SQL user variables, temporary tables and transactions are part of the state. For a complete list of items that belong to the state of a connection, see the [connection pooling and switching](#) concepts documentation. If the plugin decides to switch connections for load balancing, the application could be given a connection which has a different state. Applications must be made aware of this.

Example 6.213 Plugin config with one slave and one master

```

{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      }
    }
  }
}

```

Example 6.214 Pitfall: connection state and SQL user variables

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Connection 1, connection bound SQL user variable, no SELECT thus run on master */
if (!$mysqli->query("SET @myrole='master'")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* Connection 2, run on slave because SELECT */
if (!($res = $mysqli->query("SELECT @myrole AS _role"))) {

```

```

    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
} else {
    $row = $res->fetch_assoc();
    $res->close();
    printf("@myrole = '%s'\n", $row['_role']);
}
$mysqli->close();
?>

```

The above example will output:

```
@myrole = ''
```

The example opens a load balanced connection and executes two statements. The first statement `SET @myrole='master'` does not begin with the string `SELECT`. Therefore the plugin does not recognize it as a read-only query which shall be run on a slave. The plugin runs the statement on the connection to the master. The statement sets a SQL user variable which is bound to the master connection. The state of the master connection has been changed.

The next statement is `SELECT @myrole AS _role`. The plugin does recognize it as a read-only query and sends it to the slave. The statement is run on a connection to the slave. This second connection does not have any SQL user variables bound to it. It has a different state than the first connection to the master. The requested SQL user variable is not set. The example script prints `@myrole = ''`.

It is the responsibility of the application developer to take care of the connection state. The plugin does not monitor all connection state changing activities. Monitoring all possible cases would be a very CPU intensive task, if it could be done at all.

The pitfalls can easily be worked around using SQL hints.

6.7.4.4 SQL Hints

Copyright 1997-2014 the PHP Documentation Group.

SQL hints can force a query to choose a specific server from the connection pool. It gives the plugin a hint to use a designated server, which can solve issues caused by connection switches and connection state.

SQL hints are standard compliant SQL comments. Because SQL comments are supposed to be ignored by SQL processing systems, they do not interfere with other programs such as the MySQL Server, the MySQL Proxy, or a firewall.

Three SQL hints are supported by the plugin: The `MYSQLND_MS_MASTER_SWITCH` hint makes the plugin run a statement on the master, `MYSQLND_MS_SLAVE_SWITCH` enforces the use of the slave, and `MYSQLND_MS_LAST_USED_SWITCH` will run a statement on the same server that was used for the previous statement.

The plugin scans the beginning of a statement for the existence of an SQL hint. SQL hints are only recognized if they appear at the beginning of the statement.

Example 6.215 Plugin config with one slave and one master

```

{
    "myapp": {
        "master": {

```

```

        "master_0": {
            "host": "localhost",
            "socket": "\tmp\mysql.sock"
        },
        "slave": {
            "slave_0": {
                "host": "192.168.2.27",
                "port": "3306"
            }
        }
    }
}

```

Example 6.216 SQL hints to prevent connection switches

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno()) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Connection 1, connection bound SQL user variable, no SELECT thus run on master */
if (!$mysqli->query("SET @myrole='master'")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* Connection 1, run on master because of SQL hint */
if (!($res = $mysqli->query(sprintf("/**%s*/SELECT @myrole AS _role", MYSQLND_MS_LAST_USED_SWITCH)))) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
} else {
    $row = $res->fetch_assoc();
    $res->close();
    printf("@myrole = '%s'\n", $row['_role']);
}
$mysqli->close();
?>

```

The above example will output:

```
@myrole = 'master'
```

In the above example, using [MYSQLND_MS_LAST_USED_SWITCH](#) prevents session switching from the master to a slave when running the [SELECT](#) statement.

SQL hints can also be used to run [SELECT](#) statements on the MySQL master server. This may be desired if the MySQL slave servers are typically behind the master, but you need current data from the cluster.

In version 1.2.0 the concept of a service level has been introduced to address cases when current data is required. Using a service level requires less attention and removes the need of using SQL hints for this use case. Please, find more information below in the service level and consistency section.

Example 6.217 Fighting replication lag

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
}

```

```

        die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
    }
    /* Force use of master, master has always fresh and current data */
    if (!$mysqli->query(sprintf("/*%s*/SELECT critical_data FROM important_table", MYSQLND_MS_MASTER_SWITCH))) {
        printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
    }
    ?>

```

A use case may include the creation of tables on a slave. If an SQL hint is not given, then the plugin will send [CREATE](#) and [INSERT](#) statements to the master. Use the SQL hint [MYSQLND_MS_SLAVE_SWITCH](#) if you want to run any such statement on a slave, for example, to build temporary reporting tables.

Example 6.218 Table creation on a slave

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Force use of slave */
if (!$mysqli->query(sprintf("/*%s*/CREATE TABLE slave_reporting(id INT)", MYSQLND_MS_SLAVE_SWITCH))) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* Continue using this particular slave connection */
if (!$mysqli->query(sprintf("/*%s*/INSERT INTO slave_reporting(id) VALUES (1), (2), (3)", MYSQLND_MS_LAST_USED))) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* Don't use MYSQLND_MS_SLAVE_SWITCH which would allow switching to another slave! */
if ($res = $mysqli->query(sprintf("/*%s*/SELECT COUNT(*) AS _num FROM slave_reporting", MYSQLND_MS_LAST_USED))) {
    $row = $res->fetch_assoc();
    $res->close();
    printf("There are %d rows in the table 'slave_reporting'", $row['_num']);
} else {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
$mysqli->close();
?>

```

The SQL hint [MYSQLND_MS_LAST_USED](#) forbids switching a connection, and forces use of the previously used connection.

6.7.4.5 Local transactions

Copyright 1997-2014 the PHP Documentation Group.

The current version of the plugin is not transaction safe by default, because it is not aware of running transactions in all cases. SQL transactions are units of work to be run on a single server. The plugin does not always know when the unit of work starts and when it ends. Therefore, the plugin may decide to switch connections in the middle of a transaction.

No kind of MySQL load balancer can detect transaction boundaries without any kind of hint from the application.

You can either use SQL hints to work around this limitation. Alternatively, you can activate transaction API call monitoring. In the latter case you must use API calls only to control transactions, see below.

Example 6.219 Plugin config with one slave and one master


```
[myapp]
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      }
    }
  }
}
```

Example 6.220 Using SQL hints for transactions

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Not a SELECT, will use master */
if (!$mysqli->query("START TRANSACTION")) {
    /* Please use better error handling in your code */
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Prevent connection switch! */
if (!$mysqli->query(sprintf("/*%s*/INSERT INTO test(id) VALUES (1)", MYSQLND_MS_LAST_USED_SWITCH))) {
    /* Please do proper ROLLBACK in your code, don't just die */
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if ($res = $mysqli->query(sprintf("/*%s*/SELECT COUNT(*) AS _num FROM test", MYSQLND_MS_LAST_USED_SWITCH))) {
    $row = $res->fetch_assoc();
    $res->close();
    if ($row['_num'] > 1000) {
        if (!$mysqli->query(sprintf("/*%s*/INSERT INTO events(task) VALUES ('cleanup')", MYSQLND_MS_LAST_USED_SWITCH))) {
            die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
        }
    }
} else {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if (!$mysqli->query(sprintf("/*%s*/UPDATE log SET last_update = NOW()", MYSQLND_MS_LAST_USED_SWITCH))) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if (!$mysqli->query(sprintf("/*%s*/COMMIT", MYSQLND_MS_LAST_USED_SWITCH))) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
$mysqli->close();
?>
```

Starting with PHP 5.4.0, the [mysqlnd](#) library allows the plugin to monitor the status of the [autocommit](#) mode, if the mode is set by API calls instead of using SQL statements such as [SET AUTOCOMMIT=0](#). This makes it possible for the plugin to become transaction aware. In this case, you do not need to use SQL hints.

If using PHP 5.4.0 or newer, API calls that enable [autocommit](#) mode, and when setting the plugin configuration option [trx_stickiness=master](#), the plugin can automatically disable load balancing and connection switches for SQL transactions. In this configuration, the plugin stops load balancing if

`autocommit` is disabled and directs all statements to the master. This prevents connection switches in the middle of a transaction. Once `autocommit` is re-enabled, the plugin starts to load balance statements again.

API based transaction boundary detection has been improved with PHP 5.5.0 and `PECL/mysqlnd_ms` 1.5.0 to cover not only calls to `mysqli_autocommit` but also `mysqli_begin`, `mysqli_commit` and `mysqli_rollback`.

Example 6.221 Transaction aware load balancing: `trx_stickiness` setting

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    },
    "trx_stickiness": "master"
  }
}
```

Example 6.222 Transaction aware

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Disable autocommit, plugin will run all statements on the master */
$mysqli->autocommit(false);
if (!$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    /* Please do proper ROLLBACK in your code, don't just die */
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if ($res = $mysqli->query("SELECT COUNT(*) AS _num FROM test")) {
    $row = $res->fetch_assoc();
    $res->close();
    if ($row['_num'] > 1000) {
        if (!$mysqli->query("INSERT INTO events(task) VALUES ('cleanup')")) {
            die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
        }
    }
} else {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if (!$mysqli->query("UPDATE log SET last_update = NOW()")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
if (!$mysqli->commit()) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Plugin assumes that the transaction has ended and starts load balancing again */
$mysqli->autocommit(true);
$mysqli->close();
?>
```

Version requirement

The plugin configuration option `trx_stickiness=master` requires PHP 5.4.0 or newer.

Please note the restrictions outlined in the [transaction handling](#) concepts section.

6.7.4.6 XA/Distributed Transactions

Copyright 1997-2014 the PHP Documentation Group.

Version requirement

XA related functions have been introduced in PECL mysqlnd_ms version 1.6.0-alpha.

Early adaptors wanted

The feature is currently under development. There may be issues and/or feature limitations. Do not use in production environments, although early lab tests indicate reasonable quality.

Please, contact the development team if you are interested in this feature. We are looking for real life feedback to complement the feature.

XA transactions are a standardized method for executing transactions across multiple resources. Those resources can be databases or other transactional systems. The MySQL server supports XA SQL statements which allows users to carry out a distributed SQL transaction that spawns multiple database servers or any kind as long as they support the SQL statements too. In such a scenario it is in the responsibility of the user to coordinate the participating servers.

[PECL/mysqlnd_ms](#) can act as a transaction coordinator for a global (distributed, XA) transaction carried out on MySQL servers only. As a transaction coordinator, the plugin tracks all servers involved in a global transaction and transparently issues appropriate SQL statements on the participants. The global transactions are controlled with [mysqlnd_ms_xa_begin](#), [mysqlnd_ms_xa_commit](#) and [mysqlnd_ms_xa_rollback](#). SQL details are mostly hidden from the application as is the need to track and coordinate participants.

Example 6.223 General pattern for XA transactions

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* start a global transaction */
$gtrid_id = "12345";
if (!$mysqli->query("INSERT INTO orders(order_id, item) VALUES (1, 'christmas tree, 1.8m')")) {
    /* Either INSERT failed or the injected XA BEGIN failed */
    if ('XA' == substr($mysqli->sqlstate, 0, 2)) {
        printf("Global transaction/XA related failure, [%d] %s\n", $mysqli->errno, $mysqli->error);
    } else {
        printf("INSERT failed, [%d] %s\n", $mysqli->errno, $mysqli->error);
    }
}
/* rollback global transaction */
```

```

mysqlnd_ms_xa_rollback($mysqli, $xid);
die("Stopping.");
}
/* continue carrying out queries on other servers, e.g. other shards */
/* commit the global transaction */
if (!mysqlnd_ms_xa_commit($mysqli, $xa_id)) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
?>

```

Unlike with local transactions, which are carried out on a single server, XA transactions have an identifier (xid) associated with them. The XA transaction identifier is composed of a global transaction identifier (gtrid), a branch qualifier (bqual) a format identifier (formatID). Only the global transaction identifier can and must be given when calling any of the plugins XA functions.

Once a global transaction has been started, the plugin begins tracking servers until the global transaction ends. When a server is picked for query execution, the plugin injects the SQL statement `XA BEGIN` prior to executing the actual SQL statement on the server. `XA BEGIN` makes the server participate in the global transaction. If the injected SQL statement fails, the plugin will report the issue in reply to the query execution function that was used. In the above example, `$mysqli->query("INSERT INTO orders(order_id, item) VALUES (1, 'christmas tree, 1.8m')")` would indicate such an error. You may want to check the errors SQL state code to determine whether the actual query (here: `INSERT`) has failed or the error is related to the global transaction. It is up to you to ignore the failure to start the global transaction on a server and continue execution without having the server participate in the global transaction.

Example 6.224 Local and global transactions are mutually exclusive

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* start a local transaction */
if (!$mysqli->begin_transaction()) {
    die(sprintf("[%d/%s] %s\n", $mysqli->errno, $mysqli->sqlstate, $mysqli->error));
}
/* cannot start global transaction now - must end local transaction first */
$gtrid_id = "12345";
if (!mysqlnd_ms_xa_begin($mysqli, $gtrid_id)) {
    die(sprintf("[%d/%s] %s\n", $mysqli->errno, $mysqli->sqlstate, $mysqli->error));
}
?>

```

The above example will output:

```

Warning: mysqlnd_ms_xa_begin(): (mysqlnd_ms) Some work is done outside global transaction. You must end the
[1400/XAE09] (mysqlnd_ms) Some work is done outside global transaction. You must end the active local trans

```

A global transaction cannot be started when a local transaction is active. The plugin tries to detect this situation as early as possible, that is when `mysqlnd_ms_xa_begin` is called. If using API calls only to control transactions, the plugin will know that a local transaction is open and return an error for `mysqlnd_ms_xa_begin`. However, note the plugins [limitations on detecting transaction boundaries](#).. In the worst case, if using direct SQL for local transactions (`BEGIN`, `COMMIT`, ...), it may happen that an error is delayed until some SQL is executed on a server.

To end a global transaction invoke `mysqlnd_ms_xa_commit` or `mysqlnd_ms_xa_rollback`. When a global transaction is ended all participants must be informed of the end. Therefore, PECL/`mysqlnd_ms` transparently issues appropriate XA related SQL statements on some or all of them. Any failure during this phase will cause an implicit rollback. The XA related API is intentionally kept simple here. A more complex API that gave more control would bare few, if any, advantages over a user implementation that issues all lower level XA SQL statements itself.

XA transactions use the two-phase commit protocol. The two-phase commit protocol is a blocking protocol. There are cases when no progress can be made, not even when using timeouts. Transaction coordinators should survive their own failure, be able to detect blockades and break ties. PECL/`mysqlnd_ms` takes the role of a transaction coordinator and can be configured to survive its own crash to avoid issues with blocked MySQL servers. Therefore, the plugin can and should be configured to use a persistent and crash-safe state to allow garbage collection of unfinished, aborted global transactions. A global transaction can be aborted in an open state if either the plugin fails (crashes) or a connection from the plugin to a global transaction participant fails.

Example 6.225 Transaction coordinator state store

```
{
  "myapp": {
    "xa": {
      "state_store": {
        "participant_localhost_ip": "192.168.2.12",
        "mysql": {
          "host": "192.168.2.13",
          "user": "root",
          "password": "",
          "db": "test",
          "port": "3312",
          "socket": null
        }
      }
    },
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.14",
        "port": "3306"
      }
    }
  }
}
```

Currently, PECL/`mysqlnd_ms` supports only using MySQL database tables as a state store. The SQL definitions of the tables are given in the [plugin configuration section](#). Please, make sure to use a transactional and crash-safe storage engine for the tables, such as InnoDB. InnoDB is the default table engine in recent versions of the MySQL server. Make also sure the database server itself is highly available.

If a state store has been configured, the plugin can perform a garbage collection. During garbage collection it may be necessary to connect to a participant of a failed global transaction. Thus, the state store holds a list of participants and, among others, their host names. If the garbage collection is run on another host but the one that has written a participant entry with the host name `localhost`, then `localhost` resolves to different machines. There are two solutions to the problem. Either you do not configure any servers with the host name `localhost` but configure an IP address (and port) or, you hint the garbage collection. In the above example, `localhost` is used for `master_0`, hence it may not

resolve to the correct host during garbage collection. However, `participant_localhost_ip` is also set to hint the garbage collection that `localhost` stands for the IP `192.168.2.12`.

6.7.4.7 Service level and consistency

Copyright 1997-2014 the PHP Documentation Group.

Version requirement

Service levels have been introduced in PECL `mysqlnd_ms` version 1.2.0-alpha. `mysqlnd_ms_set_qos` is available with PHP 5.4.0 or newer.

Different types of MySQL cluster solutions offer different service and data consistency levels to their users. An asynchronous MySQL replication cluster offers eventual consistency by default. A read executed on an asynchronous slave may return current, stale or no data at all, depending on whether the slave has replayed all changesets from the master or not.

Applications using an MySQL replication cluster need to be designed to work correctly with eventual consistent data. In some cases, however, stale data is not acceptable. In those cases only certain slaves or even only master accesses are allowed to achieve the required quality of service from the cluster.

As of PECL `mysqlnd_ms` 1.2.0 the plugin is capable of selecting MySQL replication nodes automatically that deliver session consistency or strong consistency. Session consistency means that one client can read its writes. Other clients may or may not see the clients' write. Strong consistency means that all clients will see all writes from the client.

Example 6.226 Session consistency: read your writes

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    }
  }
}
```

Example 6.227 Requesting session consistency

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* read-write splitting: master used */
if (!$mysqli->query("INSERT INTO orders(order_id, item) VALUES (1, 'christmas tree, 1.8m')")) {
    /* Please use better error handling in your code */
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Request session consistency: read your writes */
if (!mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION)) {
```

```

        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    /* Plugin picks a node which has the changes, here: master */
    if (!$res = $mysqli->query("SELECT item FROM orders WHERE order_id = 1")) {
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    var_dump($res->fetch_assoc());
    /* Back to eventual consistency: stale data allowed */
    if (!$mysqli->set_charset($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL)) {
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    /* Plugin picks any slave, stale data is allowed */
    if (!$res = $mysqli->query("SELECT item, price FROM specials")) {
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    ?>

```

Service levels can be set in the plugins configuration file and at runtime using `mysqli_ms_set_qos`. In the example the function is used to enforce session consistency (read your writes) for all future statements until further notice. The `SELECT` statement on the `orders` table is run on the master to ensure the previous write can be seen by the client. Read-write splitting logic has been adapted to fulfill the service level.

After the application has read its changes from the `orders` table it returns to the default service level, which is eventual consistency. Eventual consistency puts no restrictions on choosing a node for statement execution. Thus, the `SELECT` statement on the `specials` table is executed on a slave.

The new functionality supersedes the use of SQL hints and the `master_on_write` configuration option. In many cases `mysqli_ms_set_qos` is easier to use, more powerful improves portability.

Example 6.228 Maximum age/slave lag

```

{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    },
    "failover" : "master"
  }
}

```

Example 6.229 Limiting slave lag

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Read from slaves lagging no more than four seconds */
$ret = mysqli_ms_set_qos(
    $mysqli,

```

```

MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL,
MYSQLND_MS_QOS_OPTION_AGE,
4
);
if (!$ret) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Plugin picks any slave, which may or may not have the changes */
if (!$res = $mysqli->query("SELECT item, price FROM daytrade")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Back to default: use of all slaves and masters permitted */
if (!mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL)) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
?>

```

The eventual consistency service level can be used with an optional parameter to set a maximum slave lag for choosing slaves. If set, the plugin checks **SHOW SLAVE STATUS** for all configured slaves. In case of the example, only slaves for which **Slave_IO_Running=Yes**, **Slave_SQL_Running=Yes** and **Seconds_Behind_Master <= 4** is true are considered for executing the statement **SELECT item, price FROM daytrade**.

Checking **SHOW SLAVE STATUS** is done transparently from an applications perspective. Errors, if any, are reported as warnings. No error will be set on the connection handle. Even if all **SHOW SLAVE STATUS** SQL statements executed by the plugin fail, the execution of the users statement is not stopped, given that master fail over is enabled. Thus, no application changes are required.

Expensive and slow operation

Checking **SHOW SLAVE STATUS** for all slaves adds overhead to the application. It is an expensive and slow background operation. Try to minimize the use of it. Unfortunately, a MySQL replication cluster does not give clients the possibility to request a list of candidates from a central instance. Thus, a more efficient way of checking the slaves lag is not available.

Please, note the limitations and properties of **SHOW SLAVE STATUS** as explained in the MySQL reference manual.

To prevent **mysqlnd_ms** from emitting a warning if no slaves can be found that lag no more than the defined number of seconds behind the master, it is necessary to enable master fail over in the plugins configuration file. If no slaves can be found and fail over is turned on, the plugin picks a master for executing the statement.

If no slave can be found and fail over is turned off, the plugin emits a warning, it does not execute the statement and it sets an error on the connection.

Example 6.230 Fail over not set

```

{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost",
                "socket": "\\tmp\\mysql.sock"
            }
        },
        "slave": {
            "slave_0": {
                "host": "127.0.0.1",
                "port": "3306"
            }
        }
    }
}

```



```

    }
}

```

Example 6.231 No slave within time limit

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Read from slaves lagging no more than four seconds */
$ret = mysqlnd_ms_set_qos(
    $mysqli,
    MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL,
    MYSQLND_MS_QOS_OPTION_AGE,
    4
);
if (!$ret) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Plugin picks any slave, which may or may not have the changes */
if (!$res = $mysqli->query("SELECT item, price FROM daytrade")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Back to default: use of all slaves and masters permitted */
if (!mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL)) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
?>

```

The above example will output:

```

PHP Warning: mysqli::query(): (mysqlnd_ms) Couldn't find the appropriate slave connection. 0 slaves to
PHP Warning: mysqli::query(): (mysqlnd_ms) No connection selected by the last filter in %s on line %d
[2000] (mysqlnd_ms) No connection selected by the last filter

```

6.7.4.8 Global transaction IDs

Copyright 1997-2014 the PHP Documentation Group.

Version requirement

A client-side global transaction ID injection has been introduced in `mysqlnd_ms` version 1.2.0-alpha. The feature is not required for synchronous clusters, such as MySQL Cluster. Use it with asynchronous clusters such as classical MySQL replication.

As of MySQL 5.6.5-m8 release candidate the MySQL server features built-in global transaction identifiers. The MySQL built-in global transaction ID feature is supported by [PECL/mysqlnd_ms](#) 1.3.0-alpha or later. However, the final feature set found in MySQL 5.6 production releases to date is not sufficient to support the ideas discussed below in all cases. Please, see also the [concepts section](#).

[PECL/mysqlnd_ms](#) can either use its own global transaction ID emulation or the global transaction ID feature built-in to MySQL 5.6.5-m8 or later. From a developer perspective the client-side and server-

side approach offer the same features with regards to service levels provided by PECL/mysqlnd_ms. Their differences are discussed in the [concepts section](#).

The quickstart first demonstrates the use of the client-side global transaction ID emulation built-in to [PECL/mysqlnd_ms](#) before it shows how to use the server-side counterpart. The order ensures that the underlying idea is discussed first.

Idea and client-side emulation

In its most basic form a global transaction ID (GTID) is a counter in a table on the master. The counter is incremented whenever a transaction is committed on the master. Slaves replicate the table. The counter serves two purposes. In case of a master failure, it helps the database administrator to identify the most recent slave for promoting it to the new master. The most recent slave is the one with the highest counter value. Applications can use the global transaction ID to search for slaves which have replicated a certain write (identified by a global transaction ID) already.

[PECL/mysqlnd_ms](#) can inject SQL for every committed transaction to increment a GTID counter. The so created GTID is accessible by the application to identify an application's write operation. This enables the plugin to deliver session consistency (read your writes) service level by not only querying masters but also slaves which have replicated the change already. Read load is taken away from the master.

Client-side global transaction ID emulation has some limitations. Please, read the [concepts section](#) carefully to fully understand the principles and ideas behind it, before using in production environments. The background knowledge is not required to continue with the quickstart.

First, create a counter table on your master server and insert a record into it. The plugin does not assist creating the table. Database administrators must make sure it exists. Depending on the error reporting mode, the plugin will silently ignore the lack of the table or bail out.

Example 6.232 Create counter table on master

```
CREATE TABLE `trx` (
  `trx_id` int(11) DEFAULT NULL,
  `last_update` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1
INSERT INTO `trx`(`trx_id`) VALUES (1);
```

In the plugin's configuration file set the SQL to update the global transaction ID table using [on_commit](#) from the [global_transaction_id_injection](#) section. Make sure the table name used for the [UPDATE](#) statement is fully qualified. In the example, `test.trx` is used to refer to table `trx` in the schema (database) `test`. Use the table that was created in the previous step. It is important to set the fully qualified table name because the connection on which the injection is done may use a different default database. Make sure the user that opens the connection is allowed to execute the [UPDATE](#).

Enable reporting of errors that may occur when `mysqlnd_ms` does global transaction ID injection.

Example 6.233 Plugin config: SQL for client-side GTID injection

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    }
  },
}
```

```

    "slave": {
        "slave_0": {
            "host": "127.0.0.1",
            "port": "3306"
        }
    },
    "global_transaction_id_injection": {
        "on_commit": "UPDATE test.trx SET trx_id = trx_id + 1",
        "report_error": true
    }
}
}
}

```

Example 6.234 Transparent global transaction ID injection

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("DROP TABLE IF EXISTS test")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("CREATE TABLE test(id INT)")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* auto commit mode, read on slave, no increment */
if (!$res = $mysqli->query("SELECT id FROM test")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
var_dump($res->fetch_assoc());
?>

```

The above example will output:

```

array(1) {
    ["id"]=>
    string(1) "1"
}

```

The example runs three statements in auto commit mode on the master, causing three transactions on the master. For every such statement, the plugin will inject the configured [UPDATE](#) transparently before executing the users SQL statement. When the script ends the global transaction ID counter on the master has been incremented by three.

The fourth SQL statement executed in the example, a [SELECT](#), does not trigger an increment. Only transactions (writes) executed on a master shall increment the GTID counter.

SQL for global transaction ID: efficient solution wanted!

The SQL used for the client-side global transaction ID emulation is inefficient. It is optimized for clarity not for performance. Do not use it for production

environments. Please, help finding an efficient solution for inclusion in the manual. We appreciate your input.

Example 6.235 Plugin config: SQL for fetching GTID

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    },
    "global_transaction_id_injection": {
      "on_commit": "UPDATE test.trx SET trx_id = trx_id + 1",
      "fetch_last_gtid" : "SELECT MAX(trx_id) FROM test.trx",
      "report_error": true
    }
  }
}
```

Example 6.236 Obtaining GTID after injection

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("DROP TABLE IF EXISTS test")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
printf("GTID after transaction %s\n", mysqlnd_ms_get_last_gtid($mysqli));
/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("CREATE TABLE test(id INT)")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
printf("GTID after transaction %s\n", mysqlnd_ms_get_last_gtid($mysqli));
?>
```

The above example will output:

```
GTID after transaction 7
GTID after transaction 8
```

Applications can ask PECL `mysqlnd_ms` for a global transaction ID which belongs to the last write operation performed by the application. The function `mysqlnd_ms_get_last_gtid` returns the GTID obtained when executing the SQL statement from the `fetch_last_gtid` entry of the `global_transaction_id_injection` section from the plugins configuration file. The function may be called after the GTID has been incremented.

Applications are advised not to run the SQL statement themselves as this bares the risk of accidentally causing an implicit GTID increment. Also, if the function is used, it is easy to migrate an application from one SQL statement for fetching a transaction ID to another, for example, if any MySQL server ever features built-in global transaction ID support.

The quickstart shows a SQL statement which will return a GTID equal or greater to that created for the previous statement. It is exactly the GTID created for the previous statement if no other clients have incremented the GTID in the time span between the statement execution and the [SELECT](#) to fetch the GTID. Otherwise, it is greater.

Example 6.237 Plugin config: Checking for a certain GTID

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    },
    "global_transaction_id_injection": {
      "on_commit": "UPDATE test.trx SET trx_id = trx_id + 1",
      "fetch_last_gtid" : "SELECT MAX(trx_id) FROM test.trx",
      "check_for_gtid" : "SELECT trx_id FROM test.trx WHERE trx_id >= #GTID",
      "report_error": true
    }
  }
}
```

Example 6.238 Session consistency service level and GTID combined

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* auto commit mode, transaction on master, GTID must be incremented */
if ( !$mysqli->query("DROP TABLE IF EXISTS test")
    || !$mysqli->query("CREATE TABLE test(id INT)")
    || !$mysqli->query("INSERT INTO test(id) VALUES (1)")
) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* GTID as an identifier for the last write */
$gtid = mysqlnd_ms_get_last_gtid($mysqli);
/* Session consistency (read your writes): try to read from slaves not only master */
if (false == mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION, MYSQLND_MS_QOS_OPTION_GTID
    die(sprintf("[%006] [%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Either run on master or a slave which has replicated the INSERT */
if (!($res = $mysqli->query("SELECT id FROM test"))) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
var_dump($res->fetch_assoc());
?>
```

A GTID returned from `mysqlnd_ms_get_last_gtid` can be used as an option for the session consistency service level. Session consistency delivers read your writes. Session consistency can be requested by calling `mysqlnd_ms_set_qos`. In the example, the plugin will execute the `SELECT` statement either on the master or on a slave which has replicated the previous `INSERT` already.

PECL `mysqlnd_ms` will transparently check every configured slave if it has replicated the `INSERT` by checking the slaves GTID table. The check is done running the SQL set with the `check_for_gtid` option from the `global_transaction_id_injection` section of the plugins configuration file. Please note, that this is a slow and expensive procedure. Applications should try to use it sparsely and only if read load on the master becomes too high otherwise.

Use of the server-side global transaction ID feature

Insufficient server support in MySQL 5.6

The plugin has been developed against a pre-production version of MySQL 5.6. It turns out that all released production versions of MySQL 5.6 do not provide clients with enough information to enforce session consistency based on GTIDs. Please, read the [concepts section](#) for details.

Starting with MySQL 5.6.5-m8 the MySQL Replication system features server-side global transaction IDs. Transaction identifiers are automatically generated and maintained by the server. Users do not need to take care of maintaining them. There is no need to setup any tables in advance, or for setting `on_commit`. A client-side emulation is no longer needed.

Clients can continue to use global transaction identifier to achieve session consistency when reading from MySQL Replication slaves in some cases but not all! The algorithm works as described above. Different SQL statements must be configured for `fetch_last_gtid` and `check_for_gtid`. The statements are given below. Please note, MySQL 5.6.5-m8 is a development version. Details of the server implementation may change in the future and require adoption of the SQL statements shown.

Using the following configuration any of the above described functionality can be used together with the server-side global transaction ID feature. `mysqlnd_ms_get_last_gtid` and `mysqlnd_ms_set_qos` continue to work as described above. The only difference is that the server does not use a simple sequence number but a string containing of a server identifier and a sequence number. Thus, users cannot easily derive an order from GTIDs returned by `mysqlnd_ms_get_last_gtid`.

Example 6.239 Plugin config: using MySQL 5.6.5-m8 built-in GTID feature

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    },
    "global_transaction_id_injection": {
      "fetch_last_gtid" : "SELECT @@GLOBAL.GTID_DONE AS trx_id FROM DUAL",
      "check_for_gtid" : "SELECT GTID_SUBSET('#GTID', @@GLOBAL.GTID_DONE) AS trx_id FROM DUAL",
      "report_error": true
    }
  }
}
```

6.7.4.9 Cache integration

Copyright 1997-2014 the PHP Documentation Group.

Version requirement, dependencies and status

Please, find more about version requirements, extension load order dependencies and the current status in the [concepts section](#)!

Databases clusters can deliver different levels of consistency. As of [PECL/mysqlnd_ms](#) 1.2.0 it is possible to advice the plugin to consider only cluster nodes that can deliver the consistency level requested. For example, if using asynchronous MySQL Replication with its cluster-wide eventual consistency, it is possible to request session consistency (read your writes) at any time using [mysqlnd_ms_set_quos](#). Please, see also the [service level and consistency](#) introduction.

Example 6.240 Recap: quality of service to request read your writes

```
/* Request session consistency: read your writes */
if (!mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
```

Assuming PECL/mysqlnd has been explicitly told to deliver no consistency level higher than eventual consistency, it is possible to replace a database node read access with a client-side cache using time-to-live (TTL) as its invalidation strategy. Both the database node and the cache may or may not serve current data as this is what eventual consistency defines.

Replacing a database node read access with a local cache access can improve overall performance and lower the database load. If the cache entry is every reused by other clients than the one creating the cache entry, a database access is saved and thus database load is lowered. Furthermore, system performance can become better if computation and delivery of a database query is slower than a local cache access.

Example 6.241 Plugin config: no special entries for caching

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": "3306"
      }
    }
  }
}
```

Example 6.242 Caching a slave request

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
}
```

```

        die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
    }
    if (    !$mysqli->query("DROP TABLE IF EXISTS test")
        || !$mysqli->query("CREATE TABLE test(id INT)")
        || !$mysqli->query("INSERT INTO test(id) VALUES (1)")
    ) {
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    /* Explicitly allow eventual consistency and caching (TTL <= 60 seconds) */
    if (false == mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL, MYSQLND_MS_QOS_OPTION_CACHE,
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    /* To make this example work, we must wait for a slave to catch up. Brute force style. */
    $attempts = 0;
    do {
        /* check if slave has the table */
        if ($res = $mysqli->query("SELECT id FROM test")) {
            break;
        } else if ($mysqli->errno) {
            die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
        }
        /* wait for slave to catch up */
        usleep(200000);
    } while ($attempts++ < 10);
    /* Query has been run on a slave, result is in the cache */
    assert($res);
    var_dump($res->fetch_assoc());
    /* Served from cache */
    $res = $mysqli->query("SELECT id FROM test");
    ?>

```

The example shows how to use the cache feature. First, you have to set the quality of service to eventual consistency and explicitly allow for caching. This is done by calling `mysqlnd_ms_set_qos`. Then, the result set of every read-only statement is cached for upto that many seconds as allowed with `mysqlnd_ms_set_qos`.

The actual TTL is lower or equal to the value set with `mysqlnd_ms_set_qos`. The value passed to the function sets the maximum age (seconds) of the data delivered. To calculate the actual TTL value the replication lag on a slave is checked and subtracted from the given value. If, for example, the maximum age is set to 60 seconds and the slave reports a lag of 10 seconds the resulting TTL is 50 seconds. The TTL is calculated individually for every cached query.

Example 6.243 Read your writes and caching combined

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
if (    !$mysqli->query("DROP TABLE IF EXISTS test")
    || !$mysqli->query("CREATE TABLE test(id INT)")
    || !$mysqli->query("INSERT INTO test(id) VALUES (1)")
) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Explicitly allow eventual consistency and caching (TTL <= 60 seconds) */
if (false == mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL, MYSQLND_MS_QOS_OPTION_CACHE,
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
)
/* To make this example work, we must wait for a slave to catch up. Brute force style. */
$attempts = 0;
do {
    /* check if slave has the table */
    if ($res = $mysqli->query("SELECT id FROM test")) {
        break;
    }
} while ($attempts++ < 10);
/* Query has been run on a slave, result is in the cache */
assert($res);
var_dump($res->fetch_assoc());
/* Served from cache */
$res = $mysqli->query("SELECT id FROM test");
?>

```



```

    } else if ($mysqli->errno) {
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    }
    /* wait for slave to catch up */
    usleep(200000);
} while ($attempts++ < 10);
assert($res);
/* Query has been run on a slave, result is in the cache */
var_dump($res->fetch_assoc());
/* Served from cache */
if (!($res = $mysqli->query("SELECT id FROM test"))) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
var_dump($res->fetch_assoc());
/* Update on master */
if (!$mysqli->query("UPDATE test SET id = 2")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Read your writes */
if (false == mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION)) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Fetch latest data */
if (!($res = $mysqli->query("SELECT id FROM test"))) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
var_dump($res->fetch_assoc());
?>

```

The quality of service can be changed at any time to avoid further cache usage. If needed, you can switch to read your writes (session consistency). In that case, the cache will not be used and fresh data is read.

6.7.4.10 Failover

Copyright 1997-2014 the PHP Documentation Group.

By default, the plugin does not attempt to fail over if connecting to a host fails. This prevents pitfalls related to [connection state](#). It is recommended to manually handle connection errors in a way similar to a failed transaction. You should catch the error, rebuild the connection state and rerun your query as shown below.

If connection state is no issue to you, you can alternatively enable automatic and silent failover. Depending on the configuration, the automatic and silent failover will either attempt to fail over to the master before issuing an error or, try to connect to other slaves, given the query allows for it, before attempting to connect to a master. Because [automatic failover](#) is not fool-proof, it is not discussed in the quickstart. Instead, details are given in the concepts section below.

Example 6.244 Manual failover, automatic optional

```

{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost",
                "socket": "\tmp\mysql.sock"
            }
        },
        "slave": {
            "slave_0": {
                "host": "simulate_slave_failure",
                "port": "0"
            },
            "slave_1": {

```

```
        "host": "127.0.0.1",
        "port": 3311
    },
    "filters": { "roundrobin": [] }
}
```

Example 6.245 Manual failover

```
<?php
$link = new mysqli("myapp", "username", "password", "database");
if (!$link) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
$sql = "SELECT 1 FROM DUAL";
/* error handling as it should be done regardless of the plugin */
if (!$res = $link->query($sql)) {
    /* plugin specific: check for connection error */
    switch ($link->errno) {
        case 2002:
        case 2003:
        case 2005:
            printf("Connection error - trying next slave!\n");
            /* load balancer will pick next slave */
            $res = $link->query($sql);
            break;
        default:
            /* no connection error, failover is unlikely to help */
            die(sprintf("SQL error: [%d] %s", $link->errno, $link->error));
            break;
    }
}
if ($res) {
    var_dump($res->fetch_assoc());
}
?>
```

6.7.4.11 Partitioning and Sharding

[Copyright 1997-2014 the PHP Documentation Group.](#)

Database clustering is done for various reasons. Clusters can improve availability, fault tolerance, and increase performance by applying a divide and conquer approach as work is distributed over many machines. Clustering is sometimes combined with partitioning and sharding to further break up a large complex task into smaller, more manageable units.

The `mysqlnd_ms` plugin aims to support a wide variety of MySQL database clusters. Some flavors of MySQL database clusters have built-in methods for partitioning and sharding, which could be transparent to use. The plugin supports the two most common approaches: MySQL Replication table filtering, and Sharding (application based partitioning).

MySQL Replication supports partitioning as filters that allow you to create slaves that replicate all or specific databases of the master, or tables. It is then in the responsibility of the application to choose a slave according to the filter rules. You can either use the `mysqlnd_ms` [node_groups](#) filter to manually support this, or use the experimental table filter.

Manual partitioning or sharding is supported through the node grouping filter, and SQL hints as of 1.5.0. The `node_groups` filter lets you assign a symbolic name to a group of master and slave servers. In the example, the master `master_0` and `slave_0` form a group with the name `Partition_A`. It

is entirely up to you to decide what makes up a group. For example, you may use node groups for sharding, and use the group names to address shards like [Shard_A_Range_0_100](#).

Example 6.246 Cluster node groups

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\tmp\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "simulate_slave_failure",
        "port": "0"
      },
      "slave_1": {
        "host": "127.0.0.1",
        "port": 3311
      }
    },
    "filters": {
      "node_groups": {
        "Partition_A": {
          "master": ["master_0"],
          "slave": ["slave_0"]
        }
      },
      "roundrobin": []
    }
  }
}
```

Example 6.247 Manual partitioning using SQL hints

```
<?php
function select($mysqli, $msg, $hint = '')
{
    /* Note: weak test, two connections to two servers may have the same thread id */
    $sql = sprintf("SELECT CONNECTION_ID() AS _thread, '%s' AS _hint FROM DUAL", $msg);
    if ($hint) {
        $sql = $hint . $sql;
    }
    if (!$res = $mysqli->query($sql)) {
        printf("[%d] %s", $mysqli->errno, $mysqli->error);
        return false;
    }
    $row = $res->fetch_assoc();
    printf("%d - %s - %s\n", $row['_thread'], $row['_hint'], $sql);
    return true;
}

$mysqli = new mysqli("myapp", "user", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}

/* All slaves allowed */
select($mysqli, "slave_0");
select($mysqli, "slave_1");
/* only servers of node group "Partition_A" allowed */
select($mysqli, "slave_1", "/*Partition_A*/");
select($mysqli, "slave_1", "/*Partition_A*/");
?>
```

```

6804 - slave_0 - SELECT CONNECTION_ID() AS _thread, 'slave1' AS _hint FROM DUAL
2442 - slave_1 - SELECT CONNECTION_ID() AS _thread, 'slave2' AS _hint FROM DUAL
6804 - slave_0 - /*Partition_A*/SELECT CONNECTION_ID() AS _thread, 'slave1' AS _hint FROM DUAL
6804 - slave_0 - /*Partition_A*/SELECT CONNECTION_ID() AS _thread, 'slave1' AS _hint FROM DUAL

```

By default, the plugin will use all configured master and slave servers for query execution. But if a query begins with a SQL hint like `/*node_group*/`, the plugin will only consider the servers listed in the `node_group` for query execution. Thus, `SELECT` queries prefixed with `/*Partition_A*/` will only be executed on `slave_0`.

6.7.4.12 MySQL Fabric

Copyright 1997-2014 the PHP Documentation Group.

Version requirement and status

Work on supporting MySQL Fabric started in version 1.6. Please, consider the support to be of pre-alpha quality. The manual may not list all features or feature limitations. This is work in progress.

Sharding is the only use case supported by the plugin to date.

MySQL Fabric concepts

Please, check the MySQL reference manual for more information about MySQL Fabric and how to set it up. The PHP manual assumes that you are familiar with the basic concepts and ideas of MySQL Fabric.

MySQL Fabric is a system for managing farms of MySQL servers to achieve High Availability and optionally support sharding. Technically, it is a middleware to manage and monitor MySQL servers.

Clients query MySQL Fabric to obtain lists of MySQL servers, their state and their roles. For example, clients can request a list of slaves for a MySQL Replication group and whether they are ready to handle SQL requests. Another example is a cluster of sharded MySQL servers where the client seeks to know which shard to query for a given table and shard key. If configured to use Fabric, the plugin uses XML RCP over HTTP to obtain the list at runtime from a MySQL Fabric host. The XML remote procedure call itself is done in the background and transparent from a developers point of view.

Instead of listing MySQL servers directly in the plugins configuration file it contains a list of one or more MySQL Fabric hosts

Example 6.248 Plugin config: Fabric hosts instead of MySQL servers

```

{
  "myapp": {
    "fabric": {
      "hosts": [
        {
          "host" : "127.0.0.1",
          "port" : 8080
        }
      ]
    }
  }
}

```

Users utilize the new functions `mysqlnd_ms_fabric_select_shard` and `mysqlnd_ms_fabric_select_global` to switch to the set of servers responsible for a given shard key. Then, the plugin picks an appropriate server for running queries on. When doing so, the plugin takes care of additional load balancing rules set.

The below example assumes that MySQL Fabric has been setup to shard the table `test.fabrictest` using the `id` column of the table as a shard key.

Example 6.249 Manual partitioning using SQL hints

```
<?php
$mysqli = new mysqli("myapp", "user", "password", "database");
if (!$mysqli) {
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
}
/* Create a global table - a table available on all shards */
mysqlnd_ms_fabric_select_global($mysqli, "test.fabrictest");
if (!$mysqli->query("CREATE TABLE test.fabrictest(id INT NOT NULL PRIMARY KEY)")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Switch connection to appropriate shard and insert record */
mysqlnd_ms_fabric_select_shard($mysqli, "test.fabrictest", 10);
if (!$res = $mysqli->query("INSERT INTO fabrictest(id) VALUES (10)")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
/* Try to read newly inserted record */
mysqlnd_ms_fabric_select_shard($mysqli, "test.fabrictest", 10);
if (!$res = $mysqli->query("SELECT id FROM test WHERE id = 10")) {
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
}
?>
```

The example creates the sharded table, inserts a record and reads the record thereafter. All SQL data definition language (DDL) operations on a sharded table must be applied to the so called global server group. Prior to creating or altering a sharded table, `mysqlnd_ms_fabric_select_global` is called to switch the given connection to the corresponding servers of the global group. Data manipulation (DML) SQL statements must be sent to the shards directly. The `mysqlnd_ms_fabric_select_shard` switches a connection to shards handling a certain shard key.

6.7.5 Concepts

Copyright 1997-2014 the PHP Documentation Group.

This explains the architecture and related concepts for this plugin, and describes the impact that MySQL replication and this plugin have on developmental tasks while using a database cluster. Reading and understanding these concepts is required, in order to use this plugin with success.

6.7.5.1 Architecture

Copyright 1997-2014 the PHP Documentation Group.

The `mysqlnd` replication and load balancing plugin is implemented as a PHP extension. It is written in C and operates under the hood of PHP. During the startup of the PHP interpreter, in the module init phase of the PHP engine, it gets registered as a `mysqlnd` plugin to replace selected `mysqlnd` C methods.

At PHP runtime, it inspects queries sent from `mysqlnd` (PHP) to the MySQL server. If a query is recognized as read-only, it will be sent to one of the configured slave servers. Statements are considered read-only if they either start with `SELECT`, the SQL hint `/*ms=slave*/` or a

slave had been chosen for running the previous query, and the query started with the SQL hint `/*ms=last_used*/`. In all other cases, the query will be sent to the MySQL replication master server.

For better portability, applications should use the `MYSQLND_MS_MASTER_SWITCH`, `MYSQLND_MS_SLAVE_SWITCH`, and `MYSQLND_MS_LAST_USED_SWITCH` predefined `mysqlnd_ms` constants, instead of their literal values, such as `/*ms=slave*/`.

The plugin handles the opening and closing of database connections to both master and slave servers. From an application point of view, there continues to be only one connection handle. However, internally, this one public connection handle represents a pool of network connections that are managed by the plugin. The plugin proxies queries to the master server, and to the slaves using multiple connections.

Database connections have a state consisting of, for example, transaction status, transaction settings, character set settings, and temporary tables. The plugin will try to maintain the same state among all internal connections, whenever this can be done in an automatic and transparent way. In cases where it is not easily possible to maintain state among all connections, such as when using `BEGIN TRANSACTION`, the plugin leaves it to the user to handle.

6.7.5.2 Connection pooling and switching

Copyright 1997-2014 the PHP Documentation Group.

The replication and load balancing plugin changes the semantics of a PHP MySQL connection handle. The existing API of the PHP MySQL extensions (`mysqli`, `mysql`, and `PDO_MYSQL`) are not changed in a way that functions are added or removed. But their behavior changes when using the plugin. Existing applications do not need to be adapted to a new API, but they may need to be modified because of the behavior changes.

The plugin breaks the one-by-one relationship between a `mysqli`, `mysql`, and `PDO_MYSQL` connection handle and a MySQL network connection. And a `mysqli`, `mysql`, and `PDO_MYSQL` connection handle represents a local pool of connections to the configured MySQL replication master and MySQL replication slave servers. The plugin redirects queries to the master and slave servers. At some point in time one and the same PHP connection handle may point to the MySQL master server. Later on, it may point to one of the slave servers or still the master. Manipulating and replacing the network connection referenced by a PHP MySQL connection handle is not a transparent operation.

Every MySQL connection has a state. The state of the connections in the connection pool of the plugin can differ. Whenever the plugin switches from one wire connection to another, the current state of the user connection may change. The applications must be aware of this.

The following list shows what the connection state consists of. The list may not be complete.

- Transaction status
- Temporary tables
- Table locks
- Session system variables and session user variables
- The current database set using `USE` and other state chaining SQL commands
- Prepared statements
- `HANDLER` variables
- Locks acquired with `GET_LOCK()`

Connection switches happen right before queries are executed. The plugin does not switch the current connection until the next statement is executed.

Replication issues

See also the MySQL reference manual chapter about [replication features](#) and related issues. Some restrictions may not be related to the PHP plugin, but are properties of the MySQL replication system.

Broadcasted messages

The plugins philosophy is to align the state of connections in the pool only if the state is under full control of the plugin, or if it is necessary for security reasons. Just a few actions that change the state of the connection fall into this category.

The following is a list of connection client library calls that change state, and are broadcasted to all open connections in the connection pool.

If any of the listed calls below are to be executed, the plugin loops over all open master and slave connections. The loop continues until all servers have been contacted, and the loop does not break if a server indicates a failure. If possible, the failure will propagate to the called user API function, which may be detected depending on which underlying library function was triggered.

Library call	Notes	Version
change_user()	Called by the mysqli_change_user user API call. Also triggered upon reuse of a persistent mysqli connection.	Since 1.0.0.
select_db()	Called by the following user API calls: mysql_select_db , mysqli_list_tables , mysql_db_query , mysqli_list_fields , mysqli_select_db . Note, that SQL USE is not monitored.	Since 1.0.0.
set_charset()	Called by the following user API calls: mysql_set_charset , mysqli_set_charset . Note, that SQL SET NAMES is not monitored.	Since 1.0.0.
set_server_options()	Called by the following user API calls: mysqli_multi_query , mysqli_real_query , mysqli_query , mysql_query .	Since 1.0.0.
set_client_options()	Called by the following user API calls: mysqli_options , mysqli_ssl_set , mysqli_connect , mysql_connect , mysql_pconnect .	Since 1.0.0.
set_autocommit()	Called by the following user API calls: mysqli_autocommit , PDO::setAttribute(PDO::ATTR_AUTOCOMMIT) .	Since 1.0.0. PHP >= 5.4.0.
ssl_set()	Called by the following user API calls: mysqli_ssl_set .	Since 1.1.0.

Broadcasting and lazy connections

The plugin does not proxy or “remember” all settings to apply them on connections opened in the future. This is important to remember, if using [lazy connections](#). Lazy connections are connections which are not opened before the client sends the first connection. Use of lazy connections is the default plugin action.

The following connection library calls each changed state, and their execution is recorded for later use when lazy connections are opened. This helps ensure that the connection state of all connections in the connection pool are comparable.

Library call	Notes	Version
change_user()	User, password and database recorded for future use.	Since 1.1.0.
select_db()	Database recorded for future use.	Since 1.1.0.
set_charset()	Calls (set_client_option(MYSQL_SET_CHARSET_NAME, charset)) on lazy connection to ensure charset will be used upon opening the lazy connection.	Since 1.1.0.

Library call	Notes	Version
<code>set_autocommit()</code>	Adds <code>SET AUTOCOMMIT=0 1</code> to the list of init commands of a lazy connection using <code>set_client_option(MYSQL_INIT_COMMAND, "SET AUTOCOMMIT=...%quot;)</code> .	Since 1.1.0. PHP >= 5.4.0.

Connection state

The connection state is not only changed by API calls. Thus, even if PECL `mysqlnd_ms` monitors all API calls, the application must still be aware. Ultimately, it is the applications responsibility to maintain the connection state, if needed.

Charsets and string escaping

Due to the use of lazy connections, which are a default, it can happen that an application tries to escape a string for use within SQL statements before a connection has been established. In this case string escaping is not possible. The string escape function does not know what charset to use before a connection has been established.

To overcome the problem a new configuration setting `server_charset` has been introduced in version 1.4.0.

Attention has to be paid on escaping strings with a certain charset but using the result on a connection that uses a different charset. Please note, that PECL/`mysqlnd_ms` manipulates connections and one application level connection represents a pool of multiple connections that all may have different default charsets. It is recommended to configure the servers involved to use the same default charsets. The configuration setting `server_charset` does help with this situation as well. If using `server_charset`, the plugin will set the given charset on all newly opened connections.

6.7.5.3 Local transaction handling

Copyright 1997-2014 the PHP Documentation Group.

Transaction handling is fundamentally changed. An SQL transaction is a unit of work that is run on one database server. The unit of work consists of one or more SQL statements.

By default the plugin is not aware of SQL transactions. The plugin may switch connections for load balancing at any point in time. Connection switches may happen in the middle of a transaction. This is against the nature of an SQL transaction. By default, the plugin is not transaction safe.

Any kind of MySQL load balancer must be hinted about the begin and end of a transaction. Hinting can either be done implicitly by monitoring API calls or using SQL hints. Both options are supported by the plugin, depending on your PHP version. API monitoring requires PHP 5.4.0 or newer. The plugin, like any other MySQL load balancer, cannot detect transaction boundaries based on the MySQL Client Server Protocol. Thus, entirely transparent transaction aware load balancing is not possible. The least intrusive option is API monitoring, which requires little to no application changes, depending on your application.

Please, find examples of using SQL hints or the API monitoring in the [examples section](#). The details behind the API monitoring, which makes the plugin transaction aware, are described below.

Beginning with PHP 5.4.0, the `mysqlnd` library allows this plugin to subclass the library C API call `set_autocommit()`, to detect the status of `autocommit` mode.

The PHP MySQL extensions either issue a query (such as `SET AUTOCOMMIT=0|1`), or use the `mysqlnd` library call `set_autocommit()` to control the `autocommit` setting. If an extension makes use of `set_autocommit()`, the plugin can be made transaction aware. Transaction awareness cannot be achieved if using SQL to set the `autocommit`

mode. The library function `set_autocommit()` is called by the `mysqli_autocommit` and `PDO::setAttribute(PDO::ATTR_AUTOCOMMIT)` user API calls.

The plugin configuration option `trx_stickiness=master` can be used to make the plugin transactional aware. In this mode, the plugin stops load balancing if autocommit becomes disabled, and directs all statements to the master until autocommit gets enabled.

An application that does not want to set SQL hints for transactions but wants to use the transparent API monitoring to avoid application changes must make sure that the autocommit settings is changed exclusively through the listed API calls.

API based transaction boundary detection has been improved with PHP 5.5.0 and PECL/mysqldb_ms 1.5.0 to cover not only calls to `mysqli_autocommit` but also `mysqli_begin`, `mysqli_commit` and `mysqli_rollback`.

6.7.5.4 Error handling

Copyright 1997-2014 the PHP Documentation Group.

Applications using PECL/mysqldb_ms should implement proper error handling for all user API calls. And because the plugin changes the semantics of a connection handle, API calls may return unexpected errors. If using the plugin on a connection handle that no longer represents an individual network connection, but a connection pool, an error code and error message will be set on the connection handle whenever an error occurs on any of the network connections behind.

If using lazy connections, which is the default, connections are not opened until they are needed for query execution. Therefore, an API call for a statement execution may return a connection error. In the example below, an error is provoked when trying to run a statement on a slave. Opening a slave connection fails because the plugin configuration file lists an invalid host name for the slave.

Example 6.250 Provoking a connection error

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost",
        "socket": "\\tmp\\mysql.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "invalid_host_name",
      }
    },
    "lazy_connections": 1
  }
}
```

The explicit activation of lazy connections is for demonstration purpose only.

Example 6.251 Connection error on query execution

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
/* Connection 1, connection bound SQL user variable, no SELECT thus run on master */
if (!$mysqli->query("SET @myrole='master'")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
```

```

}
/* Connection 2, run on slave because SELECT, provoke connection error */
if (!$res = $mysqli->query("SELECT @myrole AS _role")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
} else {
    $row = $res->fetch_assoc();
    $res->close();
    printf("@myrole = '%s'\n", $row['_role']);
}
$mysqli->close();
?>

```

The above example will output something similar to:

```

PHP Warning: mysqli::query(): php_network_getaddresses: getaddrinfo failed: Name or service not known in %
PHP Warning: mysqli::query(): [2002] php_network_getaddresses: getaddrinfo failed: Name or service not kno
[2002] php_network_getaddresses: getaddrinfo failed: Name or service not known

```

Applications are expected to handle possible connection errors by implementing proper error handling.

Depending on the use case, applications may want to handle connection errors differently from other errors. Typical connection errors are [2002 \(CR_CONNECTION_ERROR\) - Can't connect to local MySQL server through socket '%s' \(%d\)](#), [2003 \(CR_CONN_HOST_ERROR\) - Can't connect to MySQL server on '%s' \(%d\)](#) and [2005 \(CR_UNKNOWN_HOST\) - Unknown MySQL server host '%s' \(%d\)](#). For example, the application may test for the error codes and manually perform a fail over. The plugins philosophy is not to offer automatic fail over, beyond master fail over, because fail over is not a transparent operation.

Example 6.252 Provoking a connection error

```

{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost"
            }
        },
        "slave": {
            "slave_0": {
                "host": "invalid_host_name"
            },
            "slave_1": {
                "host": "192.168.78.136"
            }
        },
        "lazy_connections": 1,
        "filters": {
            "roundrobin": [
            ]
        }
    }
}

```

Explicitly activating lazy connections is done for demonstration purposes, as is round robin load balancing as opposed to the default [random once](#) type.

Example 6.253 Most basic failover

```

<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
/* Connection 1, connection bound SQL user variable, no SELECT thus run on master */
if (!$mysqli->query("SET @myrole='master'")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* Connection 2, first slave */
$res = $mysqli->query("SELECT VERSION() AS _version");
/* Hackish manual fail over */
if (2002 == $mysqli->errno || 2003 == $mysqli->errno || 2004 == $mysqli->errno) {
    /* Connection 3, first slave connection failed, trying next slave */
    $res = $mysqli->query("SELECT VERSION() AS _version");
}
if (!$res) {
    printf("ERROR, [%d] '%s'\n", $mysqli->errno, $mysqli->error);
} else {
    /* Error messages are taken from connection 3, thus no error */
    printf("SUCCESS, [%d] '%s'\n", $mysqli->errno, $mysqli->error);
    $row = $res->fetch_assoc();
    $res->close();
    printf("version = %s\n", $row['_version']);
}
$mysqli->close();
?>

```

The above example will output something similar to:

```

[1045] Access denied for user 'username'@'localhost' (using password: YES)
PHP Warning: mysqli::query(): php_network_getaddresses: getaddrinfo failed: Name or service not known
PHP Warning: mysqli::query(): [2002] php_network_getaddresses: getaddrinfo failed: Name or service not known
SUCCESS, [0] ''
version = 5.6.2-m5-log

```

In some cases, it may not be easily possible to retrieve all errors that occur on all network connections through a connection handle. For example, let's assume a connection handle represents a pool of three open connections. One connection to a master and two connections to the slaves. The application changes the current database using the user API call `mysqli_select_db`, which then calls the `mysqlnd` library function to change the schemata. `mysqlnd_ms` monitors the function, and tries to change the current database on all connections to harmonize their state. Now, assume the master succeeds in changing the database, and both slaves fail. Upon the initial error from the first slave, the plugin will set an appropriate error on the connection handle. The same is done when the second slave fails to change the database. The error message from the first slave is lost.

Such cases can be debugged by either checking for errors of the type `E_WARNING` (see above) or, if no other option, investigation of the [mysqlnd_ms debug and trace log](#).

6.7.5.5 Transient errors

Copyright 1997-2014 the PHP Documentation Group.

Some distributed database clusters make use of transient errors. A transient error is a temporary error that is likely to disappear soon. By definition it is safe for a client to ignore a transient error and retry the failed operation on the same database server. The retry is free of side effects. Clients are not forced to abort their work or to fail over to another database server immediately. They may enter a retry loop before to wait for the error to disappear before giving up on the database server. Transient errors can be seen, for example, when using MySQL Cluster. But they are not bound to any specific clustering solution per se.

[PECL/mysqlnd_ms](#) can perform an automatic retry loop in case of a transient error. This increases distribution transparency and thus makes it easier to migrate an application running on a single database server to run on a cluster of database servers without having to change the source of the application.

The automatic retry loop will repeat the requested operation up to a user configurable number of times and pause between the attempts for a configurable amount of time. If the error disappears during the loop, the application will never see it. If not, the error is forwarded to the application for handling.

In the example below a duplicate key error is provoked to make the plugin retry the failing query two times before the error is passed to the application. Between the two attempts the plugin sleeps for 100 milliseconds.

Example 6.254 Provoking a transient error

```
mysqlnd_ms.enable=1
mysqlnd_ms.collect_statistics=1
```

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "transient_error": {
      "mysql_error_codes": [
        1062
      ],
      "max_retries": 2,
      "usleep_retry": 100
    }
  }
}
```

Example 6.255 Transient error retry loop

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT PRIMARY KEY)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
/* Retry loop is completely transparent. Checking statistics is
   the only way to know about implicit retries */
$stats = mysqlnd_ms_get_stats();
printf("Transient error retries before error: %d\n", $stats['transient_error_retries']);
/* Provoking duplicate key error to see statistics change */
if (!$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
```

```

    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
}
$stmts = mysqlnd_ms_get_stats();
printf("Transient error retries after error: %d\n", $stats['transient_error_retries']);
$mysqli->close();
?>

```

The above example will output something similar to:

```

Transient error retries before error: 0
[1062] Duplicate entry '1' for key 'PRIMARY'
Transient error retries before error: 2

```

Because the execution of the retry loop is transparent from a users point of view, the example checks the [statistics](#) provided by the plugin to learn about it.

As the example shows, the plugin can be instructed to consider any error transient regardless of the database servers error semantics. The only error that a stock MySQL server considers temporary has the error code [1297](#). When configuring other error codes but [1297](#) make sure your configuration reflects the semantics of your clusters error codes.

The following mysqlnd C API calls are monitored by the plugin to check for transient errors: [query\(\)](#), [change_user\(\)](#), [select_db\(\)](#), [set_charset\(\)](#), [set_server_option\(\)](#), [prepare\(\)](#), [execute\(\)](#), [set_autocommit\(\)](#), [tx_begin\(\)](#), [tx_commit\(\)](#), [tx_rollback\(\)](#), [tx_commit_or_rollback\(\)](#). The corresponding user API calls have similar names.

The maximum time the plugin may sleep during the retry loop depends on the function in question. The a retry loop for [query\(\)](#), [prepare\(\)](#) or [execute\(\)](#) will sleep for up to [max_retries](#) * [usleep_retry](#) milliseconds.

However, functions that [control connection state](#) are dispatched to all connections. The retry loop settings are applied to every connection on which the command is to be run. Thus, such a function may interrupt program execution for longer than a function that is run on one server only. For example, [set_autocommit\(\)](#) is dispatched to connections and may sleep up to ([max_retries](#) * [usleep_retry](#)) * [number_of_open_connections](#)) milliseconds. Please, keep this in mind when setting long sleep times and large retry numbers. Using the default settings of [max_retries=1](#), [usleep_retry=100](#) and [lazy_connections=1](#) it is unlikely that you will ever see a delay of more than 1 second.

6.7.5.6 Failover

Copyright 1997-2014 the PHP Documentation Group.

By default, connection failover handling is left to the user. The application is responsible for checking return values of the database functions it calls and reacting to possible errors. If, for example, the plugin recognizes a query as a read-only query to be sent to the slave servers, and the slave server selected by the plugin is not available, the plugin will raise an error after not executing the statement.

Default: manual failover

It is up to the application to handle the error and, if required, re-issue the query to trigger the selection of another slave server for statement execution. The plugin will make no attempts to failover automatically, because the plugin cannot ensure that an automatic failover will not change the state of the connection. For example, the application may have issued a query which depends on SQL user variables which are bound to a specific connection. Such a query might return incorrect results if the plugin would switch the connection implicitly as part of automatic failover. To ensure correct results,

the application must take care of the failover, and rebuild the required connection state. Therefore, by default, no automatic failover is performed by the plugin.

A user that does not change the connection state after opening a connection may activate automatic failover. Please note, that automatic failover logic is limited to connection attempts. Automatic failover is not used for already established connections. There is no way to instruct the plugin to attempt failover on a connection that has been connected to MySQL already in the past.

Automatic failover

The failover policy is configured in the plugins configuration file, by using the [failover](#) configuration directive.

Automatic and silent failover can be enabled through the [failover](#) configuration directive. Automatic failover can either be configured to try exactly one master after a slave failure or, alternatively, loop over slaves and masters before returning an error to the user. The number of connection attempts can be limited and failed hosts can be excluded from future load balancing attempts. Limiting the number of retries and remembering failed hosts are considered experimental features, albeit being reasonable stable. Syntax and semantics may change in future versions.

Please note, since version 1.5.0 automatic failover is disabled for the duration of a transaction if transaction stickiness is enabled and transaction boundaries have been detected. The plugin will not switch connections for the duration of a transaction. It will also not perform automatic and silent failover. Instead an error will be thrown. It is then left to the user to handle the failure of the transaction. Please check, the [trx_stickiness](#) documentation how to do this.

A basic manual failover example is provided within the [error handling](#) section.

Standby servers

Using [weighted load balancing](#), introduced in PECL/mysqlnd 1.4.0, it is possible to configure standby servers that are sparsely used during normal operations. A standby server that is primarily used as a worst-case standby failover target can be assigned a very low weight/priority in relation to all other servers. As long as all servers are up and running the majority of the workload is assigned to the servers which have high weight values. Few requests will be directed to the standby system which has a very low weight value.

Upon failure of the servers with a high priority, you can still failover to the standby, which has been given a low load balancing priority by assigning a low weight to it. Failover can be some manually or automatically. If done automatically, you may want to combine it with the [remember_failed](#) option.

At this point, it is not possible to instruct the load balancer to direct no requests at all to a standby. This may not be much of a limitation given that the highest weight you can assign to a server is 65535. Given two slaves, of which one shall act as a standby and has been assigned a weight of 1, the standby will have to handle far less than one percent of the overall workload.

Failover and primary copy

Please note, if using a primary copy cluster, such as MySQL Replication, it is difficult to do connection failover in case of a master failure. At any time there is only one master in the cluster for a given dataset. The master is a single point of failure. If the master fails, clients have no target to fail over write requests. In case of a master outage the database administrator must take care of the situation and update the client configurations, if need be.

6.7.5.7 Load balancing

Copyright 1997-2014 the PHP Documentation Group.

Four load balancing strategies are supported to distribute statements over the configured MySQL slave servers:

random	Chooses a random server whenever a statement is executed.
random once (default)	Chooses a random server after the first statement is executed, and uses the decision for the rest of the PHP request. It is the default, and the lowest impact on the connection state.
round robin	Iterates over the list of configured servers.
user-defined via callback	Is used to implement any other strategy.

The load balancing policy is configured in the plugins configuration file using the [random](#), [roundrobin](#), and [user filters](#).

Servers can be prioritized assigning a weight. A server that has been given a weight of two will get twice as many requests as a server that has been given the default weight of one. Prioritization can be handy in heterogenous environments. For example, you may want to assign more requests to a powerful machine than to a less powerful. Or, you may have configured servers that are close or far from the client, thus expose different latencies.

6.7.5.8 Read-write splitting

Copyright 1997-2014 the PHP Documentation Group.

The plugin executes read-only statements on the configured MySQL slaves, and all other queries on the MySQL master. Statements are considered read-only if they either start with [SELECT](#), the SQL hint `/*ms=slave*/`, or if a slave had been chosen for running the previous query and the query starts with the SQL hint `/*ms=last_used*/`. In all other cases, the query will be sent to the MySQL replication master server. It is recommended to use the constants [MYSQLND_MS_SLAVE_SWITCH](#), [MYSQLND_MS_MASTER_SWITCH](#) and [MYSQLND_MS_LAST_USED_SWITCH](#) instead of `/*ms=slave*/`. See also the [list of mysqlnd_ms constants](#).

SQL hints are a special kind of standard compliant SQL comments. The plugin does check every statement for certain SQL hints. The SQL hints are described within the [mysqlnd_ms constants](#) documentation, constants that are exported by the extension. Other systems involved with the statement processing, such as the MySQL server, SQL firewalls, and SQL proxies, are unaffected by the SQL hints, because those systems are designed to ignore SQL comments.

The built-in read-write splitter can be replaced by a user-defined filter, see also the [user filter](#) documentation.

A user-defined read-write splitter can request the built-in logic to send a statement to a specific location, by invoking [mysqlnd_ms_is_select](#).

Note

The built-in read-write splitter is not aware of multi-statements. Multi-statements are seen as one statement. The splitter will check the beginning of the statement to decide where to run the statement. If, for example, a multi-statement begins with `SELECT 1 FROM DUAL; INSERT INTO test(id) VALUES (1); ...` the plugin will run it on a slave although the statement is not read-only.

6.7.5.9 Filter

Copyright 1997-2014 the PHP Documentation Group.

Version requirement

Filters exist as of [mysqlnd_ms](#) version 1.1.0-beta.

filters. PHP applications that implement a MySQL replication cluster must first identify a group of servers in the cluster which could execute a statement before the statement is executed by one of the candidates. In other words: a defined list of servers must be filtered until only one server is available.

The process of filtering may include using one or more filters, and filters can be chained. And they are executed in the order they are defined in the plugins configuration file.

Explanation: comparing filter chaining to pipes

The concept of chained filters can be compared to using pipes to connect command line utilities on an operating system command shell. For example, an input stream is passed to a processor, filtered, and then transferred to be output. Then, the output is passed as input to the next command, which is connected to the previous using the pipe operator.

Available filters:

- Load balancing filters: `random` and `roundrobin`.
- Selection filter: `user`, `user_multi`, `quality_of_service`.

The `random` filter implements the 'random' and 'random once' load balancing policies. The 'round robin' load balancing can be configured through the `roundrobin` filter. Setting a 'user defined callback' for server selection is possible with the `user` filter. The `quality_of_service` filter finds cluster nodes capable of delivering a certain service, for example, read-your-writes or, not lagging more seconds behind the master than allowed.

Filters can accept parameters to change their behavior. The `random` filter accepts an optional `sticky` parameter. If set to true, the filter changes load balancing from random to random once. Random picks a random server every time a statement is to be executed. Random once picks a random server when the first statement is to be executed and uses the same server for the rest of the PHP request.

One of the biggest strength of the filter concept is the possibility to chain filters. This strength does not become immediately visible because the `random`, `roundrobin` and `user` filters are supposed to output no more than one server. If a filter reduces the list of candidates for running a statement to only one server, it makes little sense to use that one server as input for another filter for further reduction of the list of candidates.

An example filter sequence that will fail:

- Statement to be executed: `SELECT 1 FROM DUAL`. Passed to all filters.
- All configured nodes are passed as input to the first filter. Master nodes: `master_0`. Slave nodes: `slave_0`, `slave_1`
- Filter: `random`, argument `sticky=1`. Picks a random slave once to be used for the rest of the PHP request. Output: `slave_0`.
- Output of `slave_0` and the statement to be executed is passed as input to the next filter. Here: `roundrobin`, server list passed to filter is: `slave_0`.
- Filter: `roundrobin`. Server list consists of one server only, round robin will always return the same server.

If trying to use such a filter sequence, the plugin may emit a warning like `(mysqlnd_ms) Error while creating filter '%s' . Non-multi filter '%s' already created. Stopping in %s on line %d`. Furthermore, an appropriate error on the connection handle may be set.

A second type of filter exists: multi filter. A multi filter emits zero, one or multiple servers after processing. The `quality_of_service` filter is an example. If the service quality requested sets an upper limit for the slave lag and more than one slave is lagging behind less than the allowed number of

seconds, the filter returns more than one cluster node. A multi filter must be followed by other to further reduce the list of candidates for statement execution until a candidate is found.

A filter sequence with the `quality_of_service` multi filter followed by a load balancing filter.

- Statement to be executed: `SELECT sum(price) FROM orders WHERE order_id = 1`. Passed to all filters.
- All configured nodes are passed as input to the first filter. Master nodes: `master_0`. Slave nodes: `slave_0`, `slave_1`, `slave_2`, `slave_3`
- Filter: `quality_of_service`, rule set: `session_consistency` (read-your-writes) Output: `master_0`
- Output of `master_0` and the statement to be executed is passed as input to the next filter, which is `roundrobin`.
- Filter: `roundrobin`. Server list consists of one server. Round robin selects `master_0`.

A filter sequence must not end with a multi filter. If trying to use a filter sequence which ends with a multi filter the plugin may emit a warning like `(mysqlnd_ms) Error in configuration. Last filter is multi filter. Needs to be non-multi one. Stopping in %s on line %d`. Furthermore, an appropriate error on the connection handle may be set.

Speculation towards the future: MySQL replication filtering

In future versions, there may be additional multi filters. For example, there may be a `table` filter to support MySQL replication filtering. This would allow you to define rules for which database or table is to be replicated to which node of a replication cluster. Assume your replication cluster consists of four slaves (`slave_0`, `slave_1`, `slave_2`, `slave_3`) two of which replicate a database named `sales` (`slave_0`, `slave_1`). If the application queries the database `sales`, the hypothetical `table` filter reduces the list of possible servers to `slave_0` and `slave_1`. Because the output and list of candidates consists of more than one server, it is necessary and possible to add additional filters to the candidate list, for example, using a load balancing filter to identify a server for statement execution.

6.7.5.10 Service level and consistency

Copyright 1997-2014 the PHP Documentation Group.

Version requirement

Service levels have been introduced in `mysqlnd_ms` version 1.2.0-alpha. `mysqlnd_ms_set_qos` requires PHP 5.4.0 or newer.

The plugin can be used with different kinds of MySQL database clusters. Different clusters can deliver different levels of service to applications. The service levels can be grouped by the data consistency levels that can be achieved. The plugin knows about:

- eventual consistency
- session consistency
- strong consistency

Depending how a cluster is used it may be possible to achieve higher service levels than the default one. For example, a read from an asynchronous MySQL replication slave is eventual consistent. Thus, one may say the default consistency level of a MySQL replication cluster is eventual consistency. However, if the master only is used by a client for reading and writing during a session, session consistency (read your writes) is given. PECL `mysqlnd` 1.2.0 abstracts the details of choosing an appropriate node for any of the above service levels from the user.

Service levels can be set through the qualify-of-service filter in the [plugins configuration file](#) and at runtime using the function `mysqlnd_ms_set_qos`.

The plugin defines the different service levels as follows.

Eventual consistency is the default service provided by an asynchronous cluster, such as classical MySQL replication. A read operation executed on an arbitrary node may or may not return stale data. The applications view of the data is eventual consistent.

Session consistency is given if a client can always read its own writes. An asynchronous MySQL replication cluster can deliver session consistency if clients always use the master after the first write or never query a slave which has not yet replicated the clients write operation.

The plugins understanding of strong consistency is that all clients always see the committed writes of all other clients. This is the default when using MySQL Cluster or any other cluster offering synchronous data distribution.

Service level parameters

Eventual consistency and session consistency service level accept parameters.

Eventual consistency is the service provided by classical MySQL replication. By default, all nodes qualify for read requests. An optional `age` parameter can be given to filter out nodes which lag more than a certain number of seconds behind the master. The plugin is using `SHOW SLAVE STATUS` to measure the lag. Please, see the MySQL reference manual to learn about accuracy and reliability of the `SHOW SLAVE STATUS` command.

Session consistency (read your writes) accepts an optional `GTID` parameter to consider reading not only from the master but also from slaves which already have replicated a certain write described by its transaction identifier. This way, when using asynchronous MySQL replication, read requests may be load balanced over slaves while still ensuring session consistency.

The latter requires the use of [client-side global transaction id injection](#).

Advantages of the new approach

The new approach supersedes the use of SQL hints and the configuration option `master_on_write` in some respects. If an application running on top of an asynchronous MySQL replication cluster cannot accept stale data for certain reads, it is easier to tell the plugin to choose appropriate nodes than prefixing all read statements in question with the SQL hint to enforce the use of the master. Furthermore, the plugin may be able to use selected slaves for reading.

The `master_on_write` configuration option makes the plugin use the master after the first write (session consistency, read your writes). In some cases, session consistency may not be needed for the rest of the session but only for some, few read operations. Thus, `master_on_write` may result in more read load on the master than necessary. In those cases it is better to request a higher than default service level only for those reads that actually need it. Once the reads are done, the application can return to default service level. Switching between service levels is only possible using `mysqlnd_ms_set_qos`.

Performance considerations

A MySQL replication cluster cannot tell clients which slaves are capable of delivering which level of service. Thus, in some cases, clients need to query the slaves to check their status. PECL `mysqlnd_ms` transparently runs the necessary SQL in the background. However, this is an expensive and slow operation. SQL statements are run if eventual consistency is combined with an `age` (slave lag) limit and if session consistency is combined with a global transaction ID.

If eventual consistency is combined with an maximum `age` (slave lag), the plugin selects candidates for statement execution and load balancing for each statement as follows. If the statement is a write

all masters are considered as candidates. Slaves are not checked and not considered as candidates. If the statement is a read, the plugin transparently executes `SHOW SLAVE STATUS` on every slaves connection. It will loop over all connections, send the statement and then start checking for results. Usually, this is slightly faster than a loop over all connections in which for every connection a query is send and the plugin waits for its results. A slave is considered a candidate if `SHOW SLAVE STATUS` reports `Slave_IO_Running=Yes`, `Slave_SQL_Running=Yes` and `Seconds_Behind_Master` is less or equal than the allowed maximum age. In case of an SQL error, the plugin emits a warning but does not set an error on the connection. The error is not set to make it possible to use the plugin as a drop-in.

If session consistency is combined with a global transaction ID, the plugin executes the SQL statement set with the `fetch_last_gtid` entry of the `global_transaction_id_injection` section from the plugins configuration file. Further details are identical to those described above.

In version 1.2.0 no additional optimizations are done for executing background queries. Future versions may contain optimizations, depending on user demand.

If no parameters and options are set, no SQL is needed. In that case, the plugin consider all nodes of the type shown below.

- eventual consistency, no further options set: all masters, all slaves
- session consistency, no further options set: all masters
- strong consistency (no options allowed): all masters

Throttling

The quality of service filter can be combined with [Global transaction IDs](#) to throttle clients. Throttling does reduce the write load on the master by slowing down clients. If session consistency is requested and global transactions identifier are used to check the status of a slave, the check can be done in two ways. By default a slave is checked and skipped immediately if it does not match the criteria for session consistency. Alternatively, the plugin can wait for a slave to catch up to the master until session consistency is possible. To enable the throttling, you have to set `wait_for_gtid_timeout` configuration option.

6.7.5.11 Global transaction IDs

[Copyright 1997-2014 the PHP Documentation Group.](#)

Version requirement

Client side global transaction ID injection exists as of `mysqlnd_ms` version 1.2.0-alpha. Transaction boundaries are detected by monitoring API calls. This is possible as of PHP 5.4.0. Please, see also [Transaction handling](#).

As of MySQL 5.6.5-m8 the MySQL server features built-in global transaction identifiers. The MySQL built-in global transaction ID feature is supported by [PECL/mysqlnd_ms](#) 1.3.0-alpha or later. Neither are client-side transaction boundary monitoring nor any setup activities required if using the server feature.

Please note, all MySQL 5.6 production versions do not provide clients with enough information to use GTIDs for enforcing session consistency. In the worst case, the plugin will choose the master only.

Idea and client-side emulation

[PECL/mysqlnd_ms](#) can do client-side transparent global transaction ID injection. In its most basic form, a global transaction identifier is a counter which is incremented for every transaction executed on the master. The counter is held in a table on the master. Slaves replicate the counter table.

In case of a master failure a database administrator can easily identify the most recent slave for promoting it as a new master. The most recent slave has the highest transaction identifier.

Application developers can ask the plugin for the global transaction identifier (GTID) for their last successful write operation. The plugin will return an identifier that refers to a transaction no older than that of the clients last write operation. Then, the GTID can be passed as a parameter to the quality of service (QoS) filter as an option for session consistency. Session consistency ensures read your writes. The filter ensures that all reads are either directed to a master or a slave which has replicated the write referenced by the GTID.

When injection is done

The plugin transparently maintains the GTID table on the master. In autocommit mode the plugin injects an `UPDATE` statement before executing the users statement for every master use. In manual transaction mode, the injection is done before the application calls `commit()` to close a transaction. The configuration option `report_error` of the GTID section in the plugins configuration file is used to control whether a failed injection shall abort the current operation or be ignored silently (default).

Please note, the PHP version requirements for [transaction boundary monitoring](#) and their limits.

Limitations

Client-side global transaction ID injection has shortcomings. The potential issues are not specific to [PECL/mysqlnd_ms](#) but are rather of general nature.

- Global transaction ID tables must be deployed on all masters and replicas.
- The GTID can have holes. Only PHP clients using the plugin will maintain the table. Other clients will not.
- Client-side transaction boundary detection is based on API calls only.
- Client-side transaction boundary detection does not take implicit commit into account. Some MySQL SQL statements cause an implicit commit and cannot be rolled back.

Using server-side global transaction identifier

Starting with [PECL/mysqlnd_ms](#) 1.3.0-alpha the MySQL 5.6.5-m8 or newer built-in global transaction identifier feature is supported. Use of the server feature lifts all of the above listed limitations. Please, see the MySQL Reference Manual for limitations and preconditions for using server built-in global transaction identifiers.

Whether to use the client-side emulation or the server built-in functionality is a question not directly related to the plugin, thus it is not discussed in depth. There are no plans to remove the client-side emulation and you can continue to use it, if the server-side solution is no option. This may be the case in heterogenous environments with old MySQL server or, if any of the server-side solution limitations is not acceptable.

From an applications perspective there is hardly a difference in using one or the other approach. The following properties differ.

- Client-side emulation, as shown in the manual, is using an easy to compare sequence number for global transactions. Multi-master is not handled to keep the manual examples easy.

Server-side built-in feature is using a combination of a server identifier and a sequence number as a global transaction identifier. Comparison cannot use numeric algebra. Instead a SQL function must be used. Please, see the MySQL Reference Manual for details.

Server-side built-in feature of MySQL 5.6 cannot be used to ensure session consistency under all circumstances. Do not use it for the quality-of-service feature. Here is a simple example why it will not give reliable results. There are more edge cases that cannot be covered with limited

functionality exported by the server. Currently, clients can ask a MySQL replication master for a list of all executed global transaction IDs only. If a slave is configured not to replicate all transactions, for example, because replication filters are set, then the slave will never show the same set of executed global transaction IDs. Albeit the slave may have replicated a clients writes and it may be a candidate for a consistent read, it will never be considered by the plugin. Upon write the plugin learns from the master that the servers complete transaction history consists of GTID=1..3. There is no way for the plugin to ask for the GTID of the write transaction itself, say GTID=3. Assume that a slave does not replicate the transactions GTID=1..2 but only GTID=3 because of a replication feature. Then, the slaves transaction history is GTID=3. However, the plugin tries to find a node which has a transaction history of GITD=1...3. Albeit the slave has replicated the clients write and session consistency may be achieved when reading from the slave, it will not be considered by the plugin. This is not a fault of the plugin implementation but a feature gap on the server side. Please note, this is a trivial case to illustrate the issue there are other issues. In sum you are asked not to attempt using MySQL 5.6 built-in GTIDs for enforcing session consistency. Sooner or later the load balancing will stop working properly and the plugin will direct all session consistency requests to the master.

- Plugin global transaction ID statistics are only available with client-side emulation because they monitor the emulation.

Global transaction identifiers in distributed systems

Global transaction identifiers can serve multiple purposes in the context of distributed systems, such as a database cluster. Global transaction identifiers can be used for, for example, system wide identification of transactions, global ordering of transactions, heartbeat mechanism and for checking the replication status of replicas. [PECL/mysqlnd_ms](#), a clientside driver based software, does focus on using GTIDs for tasks that can be handled at the client, such as checking the replication status of replicas for asynchronous replication setups.

6.7.5.12 Cache integration

Copyright 1997-2014 the PHP Documentation Group.

Version requirement

The feature requires use of [PECL/mysqlnd_ms](#) 1.3.0-beta or later, and [PECL/mysqlnd_qc](#) 1.1.0-alpha or newer. [PECL/mysqlnd_ms](#) must be compiled to support the feature. PHP 5.4.0 or newer is required.

Setup: extension load order

[PECL/mysqlnd_ms](#) must be loaded before [PECL/mysqlnd_qc](#), when using shared extensions.

Feature stability

The cache integration is of beta quality.

Suitable MySQL clusters

The feature is targeted for use with MySQL Replication (primary copy). Currently, no other kinds of MySQL clusters are supported. Users of such cluster must control [PECL/mysqlnd_qc](#) manually if they are interested in client-side query caching.

Support for MySQL replication clusters (asynchronous primary copy) is the main focus of [PECL/mysqlnd_ms](#). The slaves of a MySQL replication cluster may or may not reflect the latest updates from the master. Slaves are asynchronous and can lag behind the master. A read from a slave is eventual consistent from a cluster-wide perspective.

The same level of consistency is offered by a local cache using time-to-live (TTL) invalidation strategy. Current data or stale data may be served. Eventually, data searched for in the cache is not available and the source of the cache needs to be accessed.

Given that both a MySQL Replication slave (asynchronous secondary) and a local TTL-driven cache deliver the same level of service it is possible to transparently replace a remote database access with a local cache access to gain better possibility.

As of [PECL/mysqlnd_ms](#) 1.3.0-beta the plugin is capable of transparently controlling [PECL/mysqlnd_ms](#) 1.1.0-alpha or newer to cache a read-only query if explicitly allowed by setting an appropriate quality of service through [mysqlnd_ms_set_qos](#). Please, see the [quickstart](#) for a code example. Both plugins must be installed, [PECL/mysqlnd_ms](#) must be compiled to support the cache feature and PHP 5.4.0 or newer has to be used.

Applications have full control of cache usage and can request fresh data at any time, if need be. The cache usage can be enabled and disabled time during the execution of a script. The cache will be used if [mysqlnd_ms_set_qos](#) sets the quality of service to eventual consistency and enables cache usage. Cache usage is disabled by requesting higher consistency levels, for example, session consistency (read your writes). Once the quality of service has been relaxed to eventual consistency the cache can be used again.

If caching is enabled for a read-only statement, [PECL/mysqlnd_ms](#) may inject [SQL hints to control caching](#) by [PECL/mysqlnd_qc](#). It may modify the SQL statement it got from the application. Subsequent SQL processors are supposed to ignore the SQL hints. A SQL hint is a SQL comment. Comments must not be ignored, for example, by the database server.

The TTL of a cache entry is computed on a per statement basis. Applications set an maximum age for the data they want to retrieve using [mysqlnd_ms_set_qos](#). The age sets an approximate upper limit of how many seconds the data returned may lag behind the master.

The following logic is used to compute the actual TTL if caching is enabled. The logic takes the estimated slave lag into account for choosing a TTL. If, for example, there are two slaves lagging 5 and 10 seconds behind and the maximum age allowed is 60 seconds, the TTL is set to 50 seconds. Please note, the age setting is no more than an estimated guess.

- Check whether the statement is read-only. If not, don't cache.
- If caching is enabled, check the slave lag of all configured slaves. Establish slave connections if none exist so far and lazy connections are used.
- Send [SHOW SLAVE STATUS](#) to all slaves. Do not wait for the first slave to reply before sending to the second slave. Clients often wait long for replies, thus we send out all requests in a burst before fetching in a second stage.
- Loop over all slaves. For every slave wait for its reply. Do not start checking another slave before the currently waited for slave has replied. Check for [Slave_IO_Running=Yes](#) and [Slave_SQL_Running=Yes](#). If both conditions hold true, fetch the value of [Seconds_Behind_Master](#). In case of any errors or if conditions fail, set an error on the slave connection. Skip any such slave connection for the rest of connection filtering.
- Search for the maximum value of [Seconds_Behind_Master](#) from all slaves that passed the previous conditions. Subtract the value from the maximum age provided by the user with [mysqlnd_ms_set_qos](#). Use the result as a TTL.
- The filtering may sort out all slaves. If so, the maximum age is used as TTL, because the maximum lag found equals zero. It is perfectly valid to sort out all slaves. In the following it is up to subsequent filter to decide what to do. The built-in load balancing filter will pick the master.
- Inject the appropriate SQL hints to enable caching by [PECL/mysqlnd_qc](#).
- Proceed with the connection filtering, e.g. apply load balancing rules to pick a slave.

- [PECL/mysqlnd_qc](#) is loaded after [PECL/mysqlnd_ms](#) by PHP. Thus, it will see all query modifications of [PECL/mysqlnd_ms](#) and cache the query if instructed to do so.

The algorithm may seem expensive. `SHOW SLAVE STATUS` is a very fast operation. Given a sufficient number of requests and cache hits per second the cost of checking the slaves lag can easily outweigh the costs of the cache decision.

Suggestions on a better algorithm are always welcome.

6.7.5.13 Supported clusters

Copyright 1997-2014 the PHP Documentation Group.

Any application using any kind of MySQL cluster is faced with the same tasks:

- Identify nodes capable of executing a given statement with the required service level
- Load balance requests within the list of candidates
- Automatic fail over within candidates, if needed

The plugin is optimized for fulfilling these tasks in the context of a classical asynchronous MySQL replication cluster consisting of a single master and many slaves (primary copy). When using classical, asynchronous MySQL replication all of the above listed tasks need to be mastered at the client side.

Other types of MySQL cluster may have lower requirements on the application side. For example, if all nodes in the cluster can answer read and write requests, no read-write splitting needs to be done (multi-master, update-all). If all nodes in the cluster are synchronous, they automatically provide the highest possible quality of service which makes choosing a node easier. In this case, the plugin may serve the application after some reconfiguration to disable certain features, such as built-in read-write splitting.

Documentation focus

The documentation focusses describing the use of the plugin with classical asynchronous MySQL replication clusters (primary copy). Support for this kind of cluster has been the original development goal. Use of other clusters is briefly described below. Please note, that this is still work in progress.

Primary copy (MySQL Replication)

This is the primary use case of the plugin. Follow the hints given in the descriptions of each feature.

- Configure one master and one or more slaves. [Server configuration details](#) are given in the setup section.
- Use random load balancing policy together with the [sticky](#) flag.
- If you do not plan to use the [service level](#) API calls, add the [master on write](#) flag.
- Please, make yourself aware of the properties of automatic failover before adding a [failover](#) directive.
- Consider the use of [trx_stickiness](#) to execute transactions on the primary only. Please, read carefully how it works before you rely on it.

Example 6.256 Enabling the plugin (php.ini)

```
mysqlnd_ms.enable=1
mysqlnd_ms.config_file=/path/to/mysqlnd_ms_plugin.ini
```

Example 6.257 Basic plugin configuration (mysqlnd_ms_plugin.ini) for MySQL Replication

```
{
  "myapp": {
    "master": {
      "master_1": {
        "host": "localhost",
        "socket": "\tmp\mysql57.sock"
      }
    },
    "slave": {
      "slave_0": {
        "host": "127.0.0.1",
        "port": 3308
      },
      "slave_1": {
        "host": "192.168.2.28",
        "port": 3306
      }
    },
    "filters": {
      "random": {
        "sticky": "1"
      }
    }
  }
}
```

Primary copy with multi primaries (MMM - MySQL Multi Master)

MySQL Replication allows you to create cluster topologies with multiple masters (primaries). Write-write conflicts are not handled by the replication system. This is no update anywhere setup. Thus, data must be partitioned manually and clients must be redirected in accordance to the partitioning rules. The recommended setup is equal to the sharding setup below.

Manual sharding, possibly combined with primary copy and multiple primaries

Use SQL hints and the node group filter for clusters that use data partitioning but leave query redirection to the client. The example configuration shows a multi master setup with two shards.

Example 6.258 Multiple primaries - multi master (php.ini)

```
mysqlnd_ms.enable=1
mysqlnd_ms.config_file=/path/to/mysqlnd_ms_plugin.ini
mysqlnd_ms.multi_master=1
```

Example 6.259 Primary copy with multiple primaries and partitioning

```
{
  "myapp": {
    "master": {
      "master_1": {
        "host": "localhost",
        "socket": "\tmp\mysql57.sock"
      }
    },
    "master_2": {
      "host": "192.168.2.27",
      "socket": "3306"
    }
  },
}
```



```

"slave": {
  "slave_1": {
    "host": "127.0.0.1",
    "port": 3308
  },
  "slave_2": {
    "host": "192.168.2.28",
    "port": 3306
  }
},
"filters": {
  "node_groups": {
    "Partition_A": {
      "master": ["master_1"],
      "slave": ["slave_1"]
    },
    "Partition_B": {
      "master": ["master_2"],
      "slave": ["slave_2"]
    }
  },
  "roundrobin": []
}
}

```

The plugin can also be used with a loose collection of unrelated shards. For such a cluster, configure masters only and disable read write splitting. The nodes of such a cluster are called masters in the plugin configuration as they accept both reads and writes for their partition.

Using synchronous update everywhere clusters such as MySQL Cluster

MySQL Cluster is a synchronous cluster solution. All cluster nodes accept read and write requests. In the context of the plugin, all nodes shall be considered as masters.

Use the load balancing and fail over features only.

- Disable the plugins [built-in read-write splitting](#).
- Configure masters only.
- Consider random once load balancing strategy, which is the plugins default. If random once is used, only masters are configured and no SQL hints are used to force using a certain node, no connection switches will happen for the duration of a web request. Thus, no special handling is required for transactions. The plugin will pick one master at the beginning of the PHP script and use it until the script terminates.
- Do not set the quality of service. All nodes have all the data. This automatically gives you the highest possible service quality (strong consistency).
- Do not enable client-side global transaction injection. It is neither required to help with server-side fail over nor to assist the quality of service filter choosing an appropriate node.

Disabling built-in read-write splitting.

- Set `mysqlnd_ms.disable_rw_split=1`
- Do not use [SQL hints](#) to enforce the use of slaves

Configure masters only.

- Set `mysqlnd_ms.multi_master=1`.
- Do not configure any slaves.

- Set `failover=loop_before_master` in the plugins configuration file to avoid warnings about the empty slave list and to make the failover logic loop over all configured masters before emitting an error.

Please, note the warnings about automatic failover given in the previous sections.

Example 6.260 Multiple primaries - multi master (php.ini)

```
mysqlnd_ms.enable=1
mysqlnd_ms.config_file=/path/to/mysqlnd_ms_plugin.ini
mysqlnd_ms.multi_master=1
mysqlnd_ms.disable_rw_split=1
```

Example 6.261 Synchronous update anywhere cluster

```
"myapp": {
  "master": {
    "master_1": {
      "host": "localhost",
      "socket": "\\tmp\\mysql57.sock"
    },
    "master_2": {
      "host": "192.168.2.28",
      "port": 3306
    }
  },
  "slave": {
  },
  "filters": {
    "roundrobin": {
    }
  },
  "failover": {
    "strategy": "loop_before_master",
    "remember_failed": true
  }
}
```

If running an update everywhere cluster that has no built-in partitioning to avoid hot spots and high collision rates, consider using the node groups filter to keep updates on a frequently accessed table on one of the nodes. This may help to reduce collision rates and thus improve performance.

6.7.5.14 XA/Distributed transactions

Copyright 1997-2014 the PHP Documentation Group.

Version requirement

XA related functions have been introduced in [PECL/mysqlnd_ms](#) version 1.6.0-alpha.

Early adaptors wanted

The feature is currently under development. There may be issues and/or feature limitations. Do not use in production environments, although early lab tests indicate reasonable quality.

Please, contact the development team if you are interested in this feature. We are looking for real life feedback to complement the feature.

Below is a list of some feature restrictions.

- The feature is not yet compatible with the MySQL Fabric support . This limitation is soon to be lifted.

XA transaction identifier are currently restricted to numbers. This limitation will be lifted upon request, it is a simplification used during the initial implementation.

MySQL server restrictions

The XA support by the MySQL server has some restrictions. Most notably, the servers binary log may lack changes made by XA transactions in case of certain errors. Please, see the MySQL manual for details.

XA/Distributed transactions can spawn multiple MySQL servers. Thus, they may seem like a perfect tool for sharded MySQL clusters, for example, clusters managed with MySQL Fabric. [PECL/mysqldb_ms](#) hides most of the SQL commands to control XA transactions and performs automatic administrative tasks in cases of errors, to provide the user with a comprehensive API. Users should setup the plugin carefully and be well aware of server restrictions prior to using the feature.

Example 6.262 General pattern for XA transactions

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
/* BEGIN */
mysqlnd_ms_xa_begin($mysqli, 1 /* xa id */);
/* run queries on various servers */
$mysqli->query("UPDATE some_table SET col_a = 1");
...
/* COMMIT */
mysqlnd_ms_xa_commit($link, 1);
?>
```

XA transactions use the two-phase commit protocol. The two-phase commit protocol is a blocking protocol. During the first phase participating servers begin a transaction and the client carries out its work. This phase is followed by a second voting phase. During voting, the servers first make a firm promise that they are ready to commit the work even in case of their possible unexpected failure. Should a server crash in this phase, it will still recall the aborted transaction after recover and wait for the client to decide on whether it shall be committed or rolled back.

Should a client that has initiated a global transaction crash after all the participating servers gave their promise to be ready to commit, then the servers must wait for a decision. The servers are not allowed to unilaterally decide on the transaction.

A client crash or disconnect from a participant, a server crash or server error during the first phase of the protocol is uncritical. In most cases, the server will forget about the XA transaction and its work is rolled back. Additionally, the plugin tries to reach out to as many participants as it can to instruct the server to roll back the work immediately. It is not possible to disable this implicit rollback carried out by [PECL/mysqldb_ms](#) in case of errors during the first phase of the protocol. This design decision has been made to keep the implementation simple.

An error during the second phase of the commit protocol can develop into a more severe situation. The servers will not forget about prepared but unfinished transactions in all cases. The plugin will not attempt to solve these cases immediately but waits for optional background garbage collection to ensure progress of the commit protocol. It is assumed that a solution will take significant time as it may include waiting for a participating server to recover from a crash. This time span may be longer than a developer and end user expects when trying to commit a global transaction with

`mysqlnd_ms_xa_commit`. Thus, the function returns with the unfinished global transaction still requiring attention. Please, be warned that at this point, it is not yet clear whether the global transaction will be committed or rolled back later on.

Errors during the second phase can be ignored, handled by yourself or solved by the build-in garbage collection logic. Ignoring them is not recommended as you may experience unfinished global transactions on your servers that block resources virtually indefinitely. Handling the errors requires knowing the participants, checking their state and issuing appropriate SQL commands on them. There are no user API calls to expose this very information. You will have to configure a state store and make the plugin record its actions in it to receive the desired facts.

Please, see the [quickstart](#) and related [plugin configuration file settings](#) for an example how to configure a state. In addition to configuring a state store, you have to setup some SQL tables. The table definitions are given in the description of the plugin configuration settings.

Setting up and configuring a state store is also a precondition for using the built-in garbage collection for XA transactions that fail during the second commit phase. Recording information about ongoing XA transactions is an unavoidable extra task. The extra task consists of updating the state store after each and every operation that changes the state of the global transaction itself (started, committed, rolled back, errors and aborts), the addition of participants (host, optionally user and password required to connect) and any changes to a participants state. Please note, depending on configuration and your security policies, these recordings may be considered sensitive. It is therefore recommended to restrict access to the state store. Unless the state store itself becomes overloaded, writing the state information may contribute noteworthy to the runtime but should overall be only a minor factor.

It is possible that the effort it takes to implement your own routines for handling XA transactions that failed during the second commit phase exceeds the benefits of using the XA feature of [PECL/mysqlnd_ms](#) in the first place. Thus, the manual focussed on using the built-on garbage collection only.

Garbage collection can be triggered manually or automatically in the background. You may want to call `mysqlnd_ms_xa_gc` immediately after a commit failure to attempt to solve any failed but still open global transactions as soon as possible. You may also decide to disable the automatic background garbage collection, implement your own rule set for invoking the built-in garbage collection and trigger it when desired.

By default the plugin will start the garbage collection with a certain probability in the extensions internal `RSHUTDOWN` method. The request shutdown is called after your script finished. Whether the garbage collection will be triggered is determined by computing a random value between `1..1000` and comparing it with the configuration setting `probability` (default: 5). If the setting is greater or equal to the random value, the garbage collection will be triggered.

Once started, the garbage collection acts upon up to `max_transactions_per_run` (default: 100) global transactions recorded. Records include successfully finished but also unfinished XA transactions. Records for successful transactions are removed and unfinished transactions are attempted to be solved. There are no statistics that help you finding the right balance between keeping garbage collection runs short by limiting the number of transactions considered per run and preventing the garbage collection to fall behind, resulting in many records.

For each failed XA transaction the garbage collection makes `max_retries` (default: 5) attempts to finish it. After that [PECL/mysqlnd_ms](#) gives up. There are two possible reasons for this. Either a participating server crashed and has not become accessible again within `max_retries` invocations of the garbage collection, or there is a situation that the built-in garbage collection cannot cope with. Likely, the latter would be considered a bug. However, you can manually force more garbage collection runs calling `mysqlnd_ms_xa_gc` with the appropriate parameter set. Should even those function runs fail to solve the situation, then the problem must be solved by an operator.

The function `mysqlnd_ms_get_stats` provides some statistics on how many XA transactions have been started, committed, failed or rolled back.

6.7.6 Installing/Configuring

Copyright 1997-2014 the PHP Documentation Group.

6.7.6.1 Requirements

Copyright 1997-2014 the PHP Documentation Group.

PHP 5.3.6 or newer. Some advanced functionality requires PHP 5.4.0 or newer.

The `mysqlnd_ms` replication and load balancing plugin supports all PHP applications and all available PHP MySQL extensions (`mysqli`, `mysql`, `PDO_MYSQL`). The PHP MySQL extension must be configured to use `mysqlnd` in order to be able to use the `mysqlnd_ms` plugin for `mysqlnd`.

6.7.6.2 Installation

Copyright 1997-2014 the PHP Documentation Group.

This PECL extension is not bundled with PHP.

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: http://pecl.php.net/package/mysqlnd_ms

A DLL for this PECL extension is currently unavailable. See also the [building on Windows](#) section.

6.7.6.3 Runtime Configuration

Copyright 1997-2014 the PHP Documentation Group.

The behaviour of these functions is affected by settings in `php.ini`.

Table 6.42 Mysqlnd_ms Configure Options

Name	Default	Changeable	Changelog
<code>mysqlnd_ms.enable</code>	0	PHP_INI_SYSTEM	
<code>mysqlnd_ms.force_config_usage</code>	0	PHP_INI_SYSTEM	
<code>mysqlnd_ms.ini_file</code>	""	PHP_INI_SYSTEM	
<code>mysqlnd_ms.config_file</code>	""	PHP_INI_SYSTEM	
<code>mysqlnd_ms.collect_statistics</code>	0	PHP_INI_SYSTEM	
<code>mysqlnd_ms.multi_master</code>	0	PHP_INI_SYSTEM	
<code>mysqlnd_ms.disable_rw_split</code>	0	PHP_INI_SYSTEM	

Here's a short explanation of the configuration directives.

`mysqlnd_ms.enable` integer Enables or disables the plugin. If disabled, the extension will not plug into `mysqlnd` to proxy internal `mysqlnd` C API calls.

`mysqlnd_ms.force_config_usage` integer If enabled, the plugin checks if the host (server) parameters value of any MySQL connection attempt, matches a section name from the plugin configuration file. If not, the connection attempt is blocked.

This setting is not only useful to restrict PHP to certain servers but also to debug configuration file problems. The configuration file validity is checked at two different stages. The first check is performed when PHP begins to handle a web request. At this point the plugin reads and decodes the configuration file. Errors thrown at this early stage in an extensions life cycle may not be shown properly to the user. Thus, the plugin buffers the errors, if any, and additionally displays them when establishing a connection to

MySQL. By default a buffered startup error will emit an error of type `E_WARNING`. If `force_config_usage` is set, the error type used is `E_RECOVERABLE_ERROR`.

Please, see also [configuration file debugging notes](#).

<code>mysqlnd_ms.ini_file</code>	string	Plugin specific configuration file. This setting has been renamed to <code>mysqlnd_ms.config_file</code> in version 1.4.0.
<code>mysqlnd_ms.config_file</code>	string	Plugin specific configuration file. This setting superseeds <code>mysqlnd_ms.ini_file</code> since 1.4.0.
<code>mysqlnd_ms.collect_statistics</code>	integer	Enables or disables the collection of statistics. The collection of statistics is disabled by default for performance reasons. Statistics are returned by the function <code>mysqlnd_ms_get_stats</code> .
<code>mysqlnd_ms.multi_master</code>	integer	Enables or disables support of MySQL multi master replication setups. Please, see also supported clusters .
<code>mysqlnd_ms.disable_rw_splitting</code>	integer	Enables or disables built-in read write splitting. Controls whether load balancing and lazy connection functionality can be used independently of read write splitting. If read write splitting is disabled, only servers from the master list will be used for statement execution. All configured slave servers will be ignored. The SQL hint <code>MYSQLND_MS_USE_SLAVE</code> will not be recognized. If found, the statement will be redirected to a master. Disabling read write splitting impacts the return value of <code>mysqlnd_ms_query_is_select</code> . The function will no longer propose query execution on slave servers.

Multiple master servers

Setting `mysqlnd_ms.multi_master=1` allows the plugin to use multiple master servers, instead of only the first master server of the master list.

Please, see also [supported clusters](#).

6.7.6.4 Plugin configuration file (>=1.1.x)

[Copyright 1997-2014 the PHP Documentation Group](#).

The following documentation applies to PECL/mysqlnd_ms >= 1.1.0-beta. It is not valid for prior versions. For documentation covering earlier versions, see the configuration documentation for [mysqlnd_ms 1.0.x and below](#).

Introduction

[Copyright 1997-2014 the PHP Documentation Group](#).

Changelog: Feature was added in PECL/mysqlnd_ms 1.1.0-beta

The below description applies to PECL/mysqlnd_ms >= 1.1.0-beta. It is not valid for prior versions.

The plugin uses its own configuration file. The configuration file holds information about the MySQL replication master server, the MySQL replication slave servers, the server pick (load balancing) policy, the failover strategy, and the use of lazy connections.

The plugin loads its configuration file at the beginning of a web request. It is then cached in memory and used for the duration of the web request. This way, there is no need to restart PHP after deploying the configuration file. Configuration file changes will become active almost instantly.

The PHP configuration directive `mysqlnd_ms.config_file` is used to set the plugins configuration file. Please note, that the PHP configuration directive may not be evaluated for every web request. Therefore, changing the plugins configuration file name or location may require a PHP restart. However, no restart is required to read changes if an already existing plugin configuration file is updated.

Using and parsing JSON is efficient, and using JSON makes it easier to express hierarchical data structures than the standard `php.ini` format.

Example 6.263 Converting a PHP array (hash) into JSON format

Or alternatively, a developer may be more familiar with the PHP array syntax, and prefer it. This example demonstrates how a developer might convert a PHP array to JSON.

```
<?php
$config = array(
    "myapp" => array(
        "master" => array(
            "master_0" => array(
                "host" => "localhost",
                "socket" => "/tmp/mysql.sock",
            ),
        ),
        "slave" => array(),
    ),
);
file_put_contents("mysqlnd_ms.ini", json_encode($config, JSON_PRETTY_PRINT));
printf("mysqlnd_ms.ini file created...\n");
printf("Dumping file contents...\n");
printf("%s\n", str_repeat("-", 80));
echo file_get_contents("mysqlnd_ms.ini");
printf("\n%s\n", str_repeat("-", 80));
?>
```

The above example will output:

```
mysqlnd_ms.ini file created...
Dumping file contents...
-----
{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost",
                "socket": "\tmp\mysql.sock"
            }
        },
        "slave": [
        ]
    }
}
-----
```

A plugin configuration file consists of one or more sections. Sections are represented by the top-level object properties of the object encoded in the JSON file. Sections could also be called *configuration names*.

Applications reference sections by their name. Applications use section names as the host (server) parameter to the various connect methods of the [mysqli](#), [mysql](#) and [PDO_MYSQL](#) extensions. Upon connect, the [mysqli](#) plugin compares the hostname with all of the section names from the plugin configuration file. If the hostname and section name match, then the plugin will load the settings for that section.

Example 6.264 Using section names example

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27"
      },
      "slave_1": {
        "host": "192.168.2.27",
        "port": 3306
      }
    }
  },
  "localhost": {
    "master": [
      {
        "host": "localhost",
        "socket": "\\path\\to\\mysql.sock"
      }
    ],
    "slave": [
      {
        "host": "192.168.3.24",
        "port": "3305"
      },
      {
        "host": "192.168.3.65",
        "port": "3309"
      }
    ]
  }
}
```

```
<?php
/* All of the following connections will be load balanced */
$mysqli = new mysqli("myapp", "username", "password", "database");
$pdo = new PDO('mysql:host=myapp;dbname=database', 'username', 'password');
$mysql = mysql_connect("myapp", "username", "password");
$mysqli = new mysqli("localhost", "username", "password", "database");
?>
```

Section names are strings. It is valid to use a section name such as [192.168.2.1](#), [127.0.0.1](#) or [localhost](#). If, for example, an application connects to [localhost](#) and a plugin configuration section [localhost](#) exists, the semantics of the connect operation are changed. The application will no longer only use the MySQL server running on the host [localhost](#), but the plugin will start to load balance MySQL queries following the rules from the [localhost](#) configuration section. This way you can load balance queries from an application without changing the applications source code. Please keep in mind, that such a configuration may not contribute to overall readability of your applications source code. Using section names that can be mixed up with host names should be seen as a last resort.

Each configuration section contains, at a minimum, a list of master servers and a list of slave servers. The master list is configured with the keyword `master`, while the slave list is configured with the `slave` keyword. Failing to provide a slave list will result in a fatal `E_ERROR` level error, although a slave list may be empty. It is possible to allow no slaves. However, this is only recommended with synchronous clusters, please see also [supported clusters](#). The main part of the documentation focusses on the use of asynchronous MySQL replication clusters.

The master and slave server lists can be optionally indexed by symbolic names for the servers they describe. Alternatively, an array of descriptions for slave and master servers may be used.

Example 6.265 List of anonymous slaves

```
"slave": [
  {
    "host": "192.168.3.24",
    "port": "3305"
  },
  {
    "host": "192.168.3.65",
    "port": "3309"
  }
]
```

An anonymous server list is encoded by the `JSON array` type. Optionally, symbolic names may be used for indexing the slave or master servers of a server list, and done so using the `JSON object` type.

Example 6.266 Master list using symbolic names

```
"master": {
  "master_0": {
    "host": "localhost"
  }
}
```

It is recommended to index the server lists with symbolic server names. The alias names will be shown in error messages.

The order of servers is preserved and taken into account by `mysqlnd_ms`. If, for example, you configure round robin load balancing strategy, the first `SELECT` statement will be executed on the slave that appears first in the slave server list.

A configured server can be described with the `host`, `port`, `socket`, `db`, `user`, `password` and `connect_flags`. It is mandatory to set the database server host using the `host` keyword. All other settings are optional.

Example 6.267 Keywords to configure a server

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "db_server_host",
        "port": "db_server_port",
        "socket": "db_server_socket",
        "db": "database_resp_schema",
        "user": "user",
        "password": "password",
```

```

        "connect_flags": 0
    }
},
"slave": {
    "slave_0": {
        "host": "db_server_host",
        "port": "db_server_port",
        "socket": "db_server_socket"
    }
}
}
}
}

```

If a setting is omitted, the plugin will use the value provided by the user API call used to open a connection. Please, see the [using section names example](#) above.

The configuration file format has been changed in version 1.1.0-beta to allow for chained filters. Filters are responsible for filtering the configured list of servers to identify a server for execution of a given statement. Filters are configured with the [filter](#) keyword. Filters are executed by `mysqlnd_ms` in the order of their appearance. Defining filters is optional. A configuration section in the plugins configuration file does not need to have a [filters](#) entry.

Filters replace the [pick\[\]](#) setting from prior versions. The new [random](#) and [roundrobin](#) provide the same functionality.

Example 6.268 New [roundrobin](#) filter, old functionality

```

{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      },
      "slave_1": {
        "host": "192.168.78.137",
        "port": "3306"
      }
    },
    "filters": {
      "roundrobin": [
      ]
    }
  }
}

```

The function `mysqlnd_ms_set_user_pick_server` has been removed. Setting a callback is now done with the [user](#) filter. Some filters accept parameters. The [user](#) filter requires and accepts a mandatory [callback](#) parameter to set the callback previously set through the function `mysqlnd_ms_set_user_pick_server`.

Example 6.269 The [user](#) filter replaces `mysqlnd_ms_set_user_pick_server`

```

"filters": {
  "user": {
    "callback": "pick_server"
  }
}

```

```
}
}
```

The validity of the configuration file is checked both when reading the configuration file and later when establishing a connection. The configuration file is read during PHP request startup. At this early stage a PHP extension may not display error messages properly. In the worst case, no error is shown and a connection attempt fails without an adequate error message. This problem has been cured in version 1.5.0.

Example 6.270 Common error message in case of configuration file issues (upto version 1.5.0)

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
?>
```

The above example will output:

```
Warning: mysqli::mysqli(): (mysqlnd_ms) (mysqlnd_ms) Failed to parse config file [s1.json]. Please, ver
Warning: mysqli::mysqli(): (HY000/2002): php_network_getaddresses: getaddrinfo failed: Name or service
Warning: mysqli::query(): Couldn't fetch mysqli in Command line code on line 1
Fatal error: Call to a member function fetch_assoc() on a non-object in Command line code on line 1
```

Since version 1.5.0 startup errors are additionally buffered and emitted when a connection attempt is made. Use the configuration directive `mysqlnd_ms.force_config_usage` to set the error type used to display buffered errors. By default an error of type `E_WARNING` will be emitted.

Example 6.271 Improved configuration file validation since 1.5.0

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
?>
```

The above example will output:

```
Warning: mysqli::mysqli(): (mysqlnd_ms) (mysqlnd_ms) Failed to parse config file [s1.json]. Please, ver
```

It can be useful to set `mysqlnd_ms.force_config_usage = 1` when debugging potential configuration file errors. This will not only turn the type of buffered startup errors into `E_RECOVERABLE_ERROR` but also help detecting misspelled section names.

Example 6.272 Possibly more precise error due to `mysqlnd_ms.force_config_usage=1`

```
mysqlnd_ms.force_config_usage=1
```

```
<?php
$mysqli = new mysqli("invalid_section", "username", "password", "database");
```

```
?>
```

The above example will output:

```
Warning: mysqli::mysqli(): (mysqlnd_ms) Exclusive usage of configuration enforced but did not find the cor
```

Configuration Directives

Copyright 1997-2014 the PHP Documentation Group.

Here is a short explanation of the configuration directives that can be used.

master array or object

List of MySQL replication master servers. The list of either of the [JSON type array](#) to declare an anonymous list of servers or of the [JSON type object](#). Please, see [above](#) for examples.

Setting at least one master server is mandatory. The plugin will issue an error of type [E_ERROR](#) if the user has failed to provide a master server list for a configuration section. The fatal error may read [\(mysqlnd_ms\) Section \[master\] doesn't exist for host \[name_of_a_config_section\] in %s on line %d](#).

A server is described with the [host](#), [port](#), [socket](#), [db](#), [user](#), [password](#) and [connect_flags](#). It is mandatory to provide at a value for [host](#). If any of the other values is not given, it will be taken from the user API connect call, please, see also: [using section names example](#).

Table of server configuration keywords.

Keyword	Description	Version
host	Database server host. This is a mandatory setting. Failing to provide, will cause an error of type E_RECOVERABLE_ERROR when the plugin tries to connect to the server. The error message may read (mysqlnd_ms) Cannot find [host] in [%s] section in config in %s on line %d .	Since 1.1.0.
port	Database server TCP/IP port.	Since 1.1.0.
socket	Database server Unix domain socket.	Since 1.1.0.
db	Database (schemata).	Since 1.1.0.
user	MySQL database user.	Since 1.1.0.
password	MySQL database user password.	Since 1.1.0.
connect_flags	Connection flags.	Since 1.1.0.

The plugin supports using only one master server. An experimental setting exists to enable multi-master support. The details are not documented. The setting is meant for development only.

slave array or object

List of one or more MySQL replication slave servers. The syntax is identical to setting master servers, please, see [master](#) above for details.

The plugin supports using one or more slave servers.

Setting a list of slave servers is mandatory. The plugin will report an error of the type `E_ERROR` if `slave` is not given for a configuration section. The fatal error message may read `(mysqlnd_ms) Section [slave] doesn't exist for host [%s] in %s on line %d`. Note, that it is valid to use an empty slave server list. The error has been introduced to prevent accidentally setting no slaves by forgetting about the `slave` setting. A master-only setup is still possible using an empty slave server list.

If an empty slave list is configured and an attempt is made to execute a statement on a slave the plugin may emit a warning like `(mysqlnd_ms) Couldn't find the appropriate slave connection. 0 slaves to choose from`, upon statement execution. It is possible that another warning follows such as `(mysqlnd_ms) No connection selected by the last filter`.

`global_transaction_id_injection` Global transaction identifier configuration related to both the use of array or object the server built-in global transaction ID feature and the client-side emulation.

Keyword	Description	Version
<code>fetch_sql_statement</code>	SQL statement for accessing the latest global transaction identifier. The SQL statement is run if the plugin needs to know the most recent global transaction identifier. This can be the case, for example, when checking MySQL Replication slave status. Also used with <code>mysqlnd_ms_get_last_gtid</code> .	Since 1.2.0.
<code>check_sql_statement</code>	SQL statement for checking if a replica has replicated all transactions up to and including ones searched for. The SQL statement is run when searching for replicas which can offer a higher level of consistency than eventual consistency. The statement must contain a placeholder <code>#GTID</code> which is to be replaced with the global transaction identifier searched for by the plugin. Please, check the quickstart for examples.	Since 1.2.0.
<code>report_warnings</code>	Whether to emit an error of type warning if an issue occurs while executing any of the configured SQL statements.	Since 1.2.0.
<code>on_client_side_emulation</code>	Client-side global transaction ID emulation only. SQL statement to run when a transaction finished to update the global transaction identifier sequence number on the master. Please, see the quickstart for examples.	Since 1.2.0.
<code>wait_for_gtid_timeout</code>	Instructs the plugin to wait up to <code>wait_for_gtid_timeout</code> seconds for a slave to catch up when searching for slaves that can deliver session consistency. The setting limits the time spend for polling the slave status. If polling the status takes very long, the total clock time spend waiting may	Since 1.4.0.

Keyword	Description	Version
	exceed <code>wait_for_gtid_timeout</code>. The plugin calls <code>sleep(1)</code> to sleep one second between each two polls. The setting can be used both with the plugins client-side emulation and the server-side global transaction identifier feature of MySQL 5.6. Waiting for a slave to replicate a certain GTID needed for session consistency also means throttling the client. By throttling the client the write load on the master is reduced indirectly. A primary copy based replication system, such as MySQL Replication, is given more time to reach a consistent state. This can be desired, for example, to increase the number of data copies for high availability considerations or to prevent the master from being overloaded.	

fabric object

MySQL Fabric related settings. If the plugin is used together with MySQL Fabric, then the plugins configuration file no longer contains lists of MySQL servers. Instead, the plugin will ask MySQL Fabric which list of servers to use to perform a certain task.

A minimum plugin configuration for use with MySQL Fabric contains a list of one or more MySQL Fabric hosts that the plugin can query. If more than one MySQL Fabric host is configured, the plugin will use a roundrobin strategy to choose among them. Other strategies are currently not available.

Example 6.273 Minimum plugging configuration for use with MySQL Fabric

```
{
  "myapp": {
    "fabric": {
      "hosts": [
        {
          "host" : "127.0.0.1",
          "port" : 8080
        }
      ]
    }
  }
}
```

Each MySQL Fabric host is described using a JSON object with the following members.

Keyword	Description	Version
<code>host</code>	Host name of the MySQL Fabric host.	Since 1.6.0.
<code>port</code>	The TCP/IP port on which the MySQL Fabric host listens for remote procedure calls sent by clients such as the plugin.	Since 1.6.0.

The plugin is using PHP streams to communicate with MySQL Fabric through XML RPC over HTTP. By default no timeouts are set for the network communication. Thus, the plugin defaults to PHP stream default timeouts. Those defaults are out of control of the plugin itself.

An optional timeout value can be set to overrule the PHP streams default timeout setting. Setting the timeout in the plugins configuration file has the same effect as setting a timeout for a PHP user space HTTP connection established through PHP streams.

The plugins Fabric timeout value unit is seconds. The allowed value range is from 0 to 65535. The setting exists since version 1.6.

Example 6.274 Optional timeout for communication with Fabric

```
{
  "myapp": {
    "fabric": {
      "hosts": [
        {
          "host" : "127.0.0.1",
          "port" : 8080
        }
      ],
      "timeout": 2
    }
  }
}
```

[Transaction stickiness](#) and MySQL Fabric logic can collide. The stickiness option disables switching between servers for the duration of a transaction. When using Fabric and sharding the user may (erroneously) start a local transaction on one share and then attempt to switch to a different shard using either [mysqlnd_ms_fabric_select_shard](#) or [mysqlnd_ms_fabric_select_global](#). In this case, the plugin will not reject the request to switch servers in the middle of a transaction but allow the user to switch to another server regardless of the transaction stickiness setting used. It is clearly a user error to write such code.

If transaction stickiness is enabled and you would like to get an error of type warning when calling [mysqlnd_ms_fabric_select_shard](#) or [mysqlnd_ms_fabric_select_global](#), set the boolean flag [trx_warn_server_list_changes](#).

Example 6.275 Warnings about the violation of transaction boundaries

```
{
  "myapp": {
    "fabric": {
      "hosts": [
        {
          "host" : "127.0.0.1",
```

```

        "port" : 8080
    }
    },
    "trx_warn_serverlist_changes": 1
},
"trx_stickiness": "on"
}
}

```

```

<?php
$link = new mysqli("myapp", "root", "", "test");
/*
   For the demo the call may fail.
   Failed or not we get into the state
   needed for the example.
*/
@mysqlnd_ms_fabric_select_global($link, 1);
$link->begin_transaction();
@$link->query("DROP TABLE IF EXISTS test");
/*
   Switching servers/shards is a mistake due to open
   local transaction!
*/
mysqlnd_ms_select_global($link, 1);
?>

```

The above example will output:

```
PHP Warning: mysqlnd_ms_fabric_select_global(): (mysqlnd_ms) Fabric server
```

Please, consider the feature experimental. Changes to syntax and semantics may happen.

filters object

List of filters. A filter is responsible to filter the list of available servers for executing a given statement. Filters can be chained. The [random](#) and [roundrobin](#) filter replace the [pick\[\]](#) directive used in prior version to select a load balancing policy. The [user](#) filter replaces the [mysqlnd_ms_set_user_pick_server](#) function.

Filters may accept parameters to refine their actions.

If no load balancing policy is set, the plugin will default to [random_once](#). The [random_once](#) policy picks a random slave server when running the first read-only statement. The slave server will be used for all read-only statements until the PHP script execution ends. No load balancing policy is set and thus, defaulting takes place, if neither the [random](#) nor the [roundrobin](#) are part of a configuration section.

If a filter chain is configured so that a filter which output no more than once server is used as input for a filter which should be given more than one server as input, the plugin may emit a warning upon opening a connection. The warning may read: [\(mysqlnd_ms\) Error while creating filter '%s' . Non-multi filter '%s' already created. Stopping in %s on line %d](#). Furthermore, an error of the error code [2000](#), the sql

state `HY000` and an error message similar to the warning may be set on the connection handle.

Example 6.276 Invalid filter sequence

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "filters": [
      "roundrobin",
      "random"
    ]
  }
}
```

```
<?php
$link = new mysqli("myapp", "root", "", "test");
printf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error());
$link->query("SELECT 1 FROM DUAL");
?>
```

The above example will output:

```
PHP Warning: mysqli::mysqli(): (HY000/2000): (mysqlnd_ms) Error while
[2000] (mysqlnd_ms) Error while creating filter 'random' . Non-multi fi
PHP Warning: mysqli::query(): Couldn't fetch mysqli in filter_warning.
```

Filter: *random* object

The *random* filter features the random and random once load balancing policies, set through the `pick[]` directive in older versions.

The random policy will pick a random server whenever a read-only statement is to be executed. The random once strategy picks a random slave server once and continues using the slave for the rest of the PHP web request. Random once is a default, if load balancing is not configured through a filter.

If the *random* filter is not given any arguments, it stands for random load balancing policy.

Example 6.277 Random load balancing with *random* filter

```
{
  "myapp": {
```

```

    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      },
      "slave_1": {
        "host": "192.168.78.137",
        "port": "3306"
      }
    },
    "filters": [
      "random"
    ]
  }
}

```

Optionally, the `sticky` argument can be passed to the filter. If the parameter `sticky` is set to the string `1`, the filter follows the random once load balancing strategy.

Example 6.278 Random once load balancing with `random` filter

```

{
  "filters": {
    "random": {
      "sticky": "1"
    }
  }
}

```

Both the `random` and `roundrobin` filters support setting a priority, a weight for a server, since PECL/mysqlnd_ms 1.4.0. If the `weight` argument is passed to the filter, it must assign a weight for all servers. Servers must be given an alias name in the `slave` respectively `master` server lists. The alias must be used to reference servers for assigning a priority with `weight`.

Example 6.279 Referencing error

```
[E_RECOVERABLE_ERROR] mysqli_real_connect(): (mysqlnd_ms) Unknown server 's
```

Using a wrong alias name with `weight` may result in an error similar to the shown above.

If `weight` is omitted, the default weight of all servers is one.

Example 6.280 Assigning a `weight` for load balancing

```

{
  "myapp": {
    "master": {

```

```

        "master1": {
            "host": "localhost",
            "socket": "\\var\\run\\mysql\\mysql.sock"
        },
    },
    "slave": {
        "slave1": {
            "host": "192.168.2.28",
            "port": 3306
        },
        "slave2": {
            "host": "192.168.2.29",
            "port": 3306
        },
        "slave3": {
            "host": "192.0.43.10",
            "port": 3306
        },
    },
    "filters": {
        "random": {
            "weights": {
                "slave1": 8,
                "slave2": 4,
                "slave3": 1,
                "master1": 1
            }
        }
    }
}

```

At the average a server assigned a weight of two will be selected twice as often as a server assigned a weight of one. Different weights can be assigned to reflect differently sized machines, to prefer co-located slaves which have a low network latency or, to configure a standby failover server. In the latter case, you may want to assign the standby server a very low weight in relation to the other servers. For example, given the configuration above [slave3](#) will get only some eight percent of the requests in the average. As long as [slave1](#) and [slave2](#) are running, it will be used sparsely, similar to a standby failover server. Upon failure of [slave1](#) and [slave2](#), the usage of [slave3](#) increases. Please, check the notes on failover before using [weight](#) this way.

Valid weight values range from 1 to 65535.

Unknown arguments are ignored by the filter. No warning or error is given.

The filter expects one or more servers as input. Outputs one server. A filter sequence such as [random](#), [roundrobin](#) may cause a warning and an error message to be set on the connection handle when executing a statement.

List of filter arguments.

Keyword	Description	Version
sticky	Enables or disabled random once load balancing policy. See above.	Since 1.2.0.
weight	Assigns a load balancing weight/priority to a server. Please, see above for a description.	Since 1.4.0.

Filter: *roundrobin* object

If using the *roundrobin* filter, the plugin iterates over the list of configured slave servers to pick a server for statement execution. If the plugin reaches the end of the list, it wraps around to the beginning of the list and picks the first configured slave server.

Example 6.281 *roundrobin* filter

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "filters": [
      "roundrobin"
    ]
  }
}
```

Expects one or more servers as input. Outputs one server. A filter sequence such as *roundrobin*, *random* may cause a warning and an error message to be set on the connection handle when executing a statement.

List of filter arguments.

Keyword	Description	Version
<i>weight</i>	Assigns a load balancing weight/priority to a server. Please, find a description above .	Since 1.4.0.

Filter: *user* object

The *user* replaces `mysqlnd_ms_set_user_pick_server` function, which was removed in 1.1.0-beta. The filter sets a callback for user-defined read/write splitting and server selection.

The plugin's built-in read/write query split mechanism decisions can be overwritten in two ways. The easiest way is to prepend a query string with the SQL hints `MYSQLND_MS_MASTER_SWITCH`, `MYSQLND_MS_SLAVE_SWITCH` or `MYSQLND_MS_LAST_USED_SWITCH`. Using SQL hints one can control, for example, whether a query shall be sent to the MySQL replication master server or one of the slave servers. By help of SQL hints it is not possible to pick a certain slave server for query execution.

Full control on server selection can be gained using a callback function. Use of a callback is recommended to expert users only because the callback has to cover all cases otherwise handled by the plugin.

The plugin will invoke the callback function for selecting a server from the lists of configured master and slave servers. The callback function inspects the query to run and picks a server for query

execution by returning the hosts URI, as found in the master and slave list.

If the lazy connections are enabled and the callback chooses a slave server for which no connection has been established so far and establishing the connection to the slave fails, the plugin will return an error upon the next action on the failed connection, for example, when running a query. It is the responsibility of the application developer to handle the error. For example, the application can re-run the query to trigger a new server selection and callback invocation. If so, the callback must make sure to select a different slave, or check slave availability, before returning to the plugin to prevent an endless loop.

Example 6.282 Setting a callback

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "filters": {
      "user": {
        "callback": "pick_server"
      }
    }
  }
}
```

The callback is supposed to return a host to run the query on. The host URI is to be taken from the master and slave connection lists passed to the callback function. If callback returns a value neither found in the master nor in the slave connection lists the plugin will emit an error of the type `E_RECOVERABLE_ERROR`. The error may read like `(mysqlnd_ms) User filter callback has returned an unknown server. The server 'server that is not in master or slave list' can neither be found in the master list nor in the slave list.` If the application catches the error to ignore it, follow up errors may be set on the connection handle, for example, `(mysqlnd_ms) No connection selected by the last filter` with the error code `2000` and the sqlstate `HY000`. Furthermore a warning may be emitted.

Referencing a non-existing function as a callback will result in any error of the type `E_RECOVERABLE_ERROR` whenever the plugin tries to callback function. The error message may reads like: `(mysqlnd_ms) Specified callback (pick_server) is not a valid callback.` If the application catches the error to ignore it, follow up errors may be set on the connection handle, for example, `(mysqlnd_ms) Specified callback`

([pick_server](#)) is not a valid callback with the error code 2000 and the sqlstate HY000. Furthermore a warning may be emitted.

The following parameters are passed from the plugin to the callback.

Parameter	Description	Version
connection	URI of the currently connected database server.	Since 1.1.0.
query	Query string of the statement for which a server needs to be picked.	Since 1.1.0.
master_list	List of master servers to choose from. Note, that the list of master servers may not be identical to the list of configured master servers if the filter is not the first in the filter chain. Previously run filters may have reduced the master list already.	Since 1.1.0.
slave_list	List of slave servers to choose from. Note, that the list of master servers may not be identical to the list of configured master servers if the filter is not the first in the filter chain. Previously run filters may have reduced the master list already.	Since 1.1.0.
last_used	URI of the server of the connection used to execute the previous statement on.	Since 1.1.0.
in_transaction	Boolean flag indicating whether the statement is part of an open transaction. If autocommit mode is turned off, this will be set to TRUE . Otherwise it is set to FALSE . Transaction detection is based on monitoring the mysqlnd library call set_autocommit . Monitoring is not possible before PHP 5.4.0. Please, see connection pooling and switching concepts discussion for further details.	Since 1.1.0.

Example 6.283 Using a callback

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.2.27",
        "port": "3306"
      },
      "slave_1": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "filters": {
      "user": {
        "callback": "pick_server"
      }
    }
  }
}
```

```

    }
}
}

```

```

<?php
function pick_server($connected, $query, $masters, $slaves, $last_used)
{
    static $slave_idx = 0;
    static $num_slaves = NULL;
    if (is_null($num_slaves))
        $num_slaves = count($slaves);
    /* default: fallback to the plugins build-in logic */
    $ret = NULL;
    printf("User has connected to '%s'...\n", $connected);
    printf("... deciding where to run '%s'\n", $query);
    $where = mysqlnd_ms_query_is_select($query);
    switch ($where)
    {
        case MYSQLND_MS_QUERY_USE_MASTER:
            printf("... using master\n");
            $ret = $masters[0];
            break;
        case MYSQLND_MS_QUERY_USE_SLAVE:
            /* SELECT or SQL hint for using slave */
            if (strstr($query, "FROM table_on_slave_a_only"))
            {
                /* a table which is only on the first configured slave */
                printf("... access to table available only on slave A detected\n");
                $ret = $slaves[0];
            }
            else
            {
                /* round robin */
                printf("... some read-only query for a slave\n");
                $ret = $slaves[$slave_idx++ % $num_slaves];
            }
            break;
        case MYSQLND_MS_QUERY_LAST_USED:
            printf("... using last used server\n");
            $ret = $last_used_connection;
            break;
    }
    printf("... ret = '%s'\n", $ret);
    return $ret;
}
$mysqli = new mysqli("myapp", "root", "", "test");
if (!($res = $mysqli->query("SELECT 1 FROM DUAL")))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();
if (!($res = $mysqli->query("SELECT 2 FROM DUAL")))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();
if (!($res = $mysqli->query("SELECT * FROM table_on_slave_a_only")))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();
$mysqli->close();
?>

```

The above example will output:

```
User has connected to 'myapp'...
```

```

... deciding where to run 'SELECT 1 FROM DUAL'
... some read-only query for a slave
... ret = 'tcp://192.168.2.27:3306'
User has connected to 'myapp'...
... deciding where to run 'SELECT 2 FROM DUAL'
... some read-only query for a slave
... ret = 'tcp://192.168.78.136:3306'
User has connected to 'myapp'...
... deciding where to run 'SELECT * FROM table_on_slave_a_only'
... access to table available only on slave A detected
... ret = 'tcp://192.168.2.27:3306'

```

Filter: `user_multi` object

The `user_multi` differs from the `user` only in one aspect. Otherwise, their syntax is identical. The `user` filter must pick and return exactly one node for statement execution. A filter chain usually ends with a filter that emits only one node. The filter chain shall reduce the list of candidates for statement execution down to one. This, only one node left, is the case after the `user` filter has been run.

The `user_multi` filter is a multi filter. It returns a list of slave and a list of master servers. This list needs further filtering to identify exactly one node for statement execution. A multi filter is typically placed at the top of the filter chain. The `quality_of_service` filter is another example of a multi filter.

The return value of the callback set for `user_multi` must be an array with two elements. The first element holds a list of selected master servers. The second element contains a list of selected slave servers. The lists shall contain the keys of the slave and master servers as found in the slave and master lists passed to the callback. The below example returns random master and slave lists extracted from the functions input.

Example 6.284 Returning random masters and slaves

```

<?php
function pick_server($connected, $query, $masters, $slaves, $last_used_conn)
{
    $picked_masters = array()
    foreach ($masters as $key => $value) {
        if (mt_rand(0, 2) > 1)
            $picked_masters[] = $key;
    }
    $picked_slaves = array()
    foreach ($slaves as $key => $value) {
        if (mt_rand(0, 2) > 1)
            $picked_slaves[] = $key;
    }
    return array($picked_masters, $picked_slaves);
}
?>

```

The plugin will issue an error of type `E_RECOVERABLE` if the callback fails to return a server list. The error may read `(mysqlnd_ms) User multi filter callback has not returned a list of servers to use. The callback must return an array in %s on line %d`. In case the server list is not empty but has invalid servers key/ids in it, an error

of type `E_RECOVERABLE` will be thrown with an error message like `(mysqlnd_ms) User multi filter callback has returned an invalid list of servers to use. Server id is negative in %s on line %d`, or similar.

Whether an error is emitted in case of an empty slave or master list depends on the configuration. If an empty master list is returned for a write operation, it is likely that the plugin will emit a warning that may read `(mysqlnd_ms) Couldn't find the appropriate master connection. 0 masters to choose from. Something is wrong in %s on line %d`. Typically a follow up error of type `E_ERROR` will happen. In case of a read operation and an empty slave list the behavior depends on the fail over configuration. If fail over to master is enabled, no error should appear. If fail over to master is deactivated the plugin will emit a warning that may read `(mysqlnd_ms) Couldn't find the appropriate slave connection. 0 slaves to choose from. Something is wrong in %s on line %d`.

Filter: `node_groups` object

The `node_groups` filter lets you group cluster nodes and query selected groups, for example, to support data partitioning. Data partitioning can be required for manual sharding, primary copy based clusters running multiple masters, or to avoid hot spots in update everywhere clusters that have no built-in partitioning. The filter is a multi filter which returns zero, one or multiple of its input servers. Thus, it must be followed by other filters to reduce the number of candidates down to one for statement execution.

Keyword	Description	Version
<code>user defined node group name</code>	One or more node groups must be defined. A node group can have an arbitrary user defined name. The name is used in combination with a SQL hint to restrict query execution to the nodes listed for the node group. To run a query on any of the servers of a node group, the query must begin with the SQL hint <code>/*user defined node group name*/</code>. Please note, no white space is allowed around <code>user defined node group name</code>. Because <code>user defined node group name</code> is used as-is as part of a SQL hint, you should choose the name that is compliant with the SQL language. Each node group entry must contain a list of <code>master</code> servers. Additional <code>slave</code> servers are allowed. Failing to provide a list of <code>master</code> for a node group <code>name_of_group</code> may cause an error of type <code>E_RECOVERABLE_ERROR</code> like <code>(mysqlnd_ms) No masters configured in node group 'name_of_group' for 'node_groups' filter</code>. The list of master and slave servers must reference corresponding entries in the <code>global master</code> respectively <code>slave</code> server list. Referencing an unknown server in either of the both server lists may cause	Since 1.5.0.

Keywords	Description	Version
	an <code>E_RECOVERABLE_ERROR</code> error like (mysqlnd_ms) Unknown master 'server_alias_name' (section 'name_of_group') in 'node_groups' filter configuration. Example 6.285 Manual partitioning <pre>{ "myapp": { "master": { "master_0": { "host": "localhost", "socket": "\\tmp\\mysql.sock" } }, "slave": { "slave_0": { "host": "192.168.2.28", "port": 3306 }, "slave_1": { "host": "127.0.0.1", "port": 3311 } }, "filters": { "node_groups": { "Partition_A": { "master": ["master_0"], "slave": ["slave_0"] } }, "roundrobin": [] } } }</pre> Please note, if a filter chain generates an empty slave list and the PHP configuration directive <code>mysqlnd_ms.multi_master=0</code> is used, the plugin may emit a warning.	

Filter: `quality_of_service` object

The `quality_of_service` identifies cluster nodes capable of delivering a certain quality of service. It is a multi filter which returns zero, one or multiple of its input servers. Thus, it must be followed by other filters to reduce the number of candidates down to one for statement execution.

The `quality_of_service` filter has been introduced in 1.2.0-alpha. In the 1.2 series the filters focus is on the consistency aspect of service quality. Different types of clusters offer different default data consistencies. For example, an asynchronous MySQL replication slave offers eventual consistency. The slave may not be able to deliver requested data because it has not replicated the write, it may serve stale database because its lagging behind or it may serve current information. Often, this is acceptable. In some cases higher consistency levels are needed for the application to work correct. In those cases, the `quality_of_service` can filter

out cluster nodes which cannot deliver the necessary quality of service.

The `quality_of_service` filter can be replaced or created at runtime. A successful call to `mysqlnd_ms_set_qos` removes all existing `qos` filter entries from the filter list and installs a new one at the very beginning. All settings that can be made through `mysqlnd_ms_set_qos` can also be in the plugins configuration file. However, use of the function is by far the most common use case. Instead of setting session consistency and strong consistency service levels in the plugins configuration file it is recommended to define only masters and no slaves. Both service levels will force the use of masters only. Using an empty slave list shortens the configuration file, thus improving readability. The only service level for which there is a case of defining in the plugins configuration file is the combination of eventual consistency and maximum slave lag.

Keyword	Description	Version
<code>eventual_consistency</code>	Request eventual consistency. Allows the use of all master and slave servers. Data returned may or may not be current. Eventual consistency accepts an optional <code>age</code> parameter. If <code>age</code> is given the plugin considers only slaves for reading for which MySQL replication reports a slave lag less or equal to <code>age</code>. The replication lag is measure using <code>SHOW SLAVE STATUS</code>. If the plugin fails to fetch the replication lag, the slave tested is skipped. Implementation details and tips are given in the quality of service concepts section. Please note, if a filter chain generates an empty slave list and the PHP configuration directive <code>mysqlnd_ms.multi_master=0</code> is used, the plugin may emit a warning. Example 6.286 Global limit on slave lag <pre>{ "myapp": { "master": { "master_0": { "host": "localhost" } }, "slave": { "slave_0": { "host": "192.168.2.27", "port": "3306" }, "slave_1": { "host": "192.168.78.136", "port": "3306" } }, "filters": { "quality_of_service": { "eventual_consistency": { "age": 123 } } } } }</pre>	Since 1.2.0.

Keyword	Description	Version
	<pre> } } } </pre>	
<code>session</code>	Request session consistency (read your writes). Allows use of all masters and all slaves which are in sync with the master. If no further parameters are given slaves are filtered out as there is no reliable way to test if a slave has caught up to the master or is lagging behind. Please note, if a filter chain generates an empty slave list and the PHP configuration directive <code>mysqlnd_ms.multi_master=0</code> is used, the plugin may emit a warning. Session consistency temporarily requested using <code>mysqlnd_ms_set_qos</code> is a valuable alternative to using <code>master_on_write</code>. <code>master_on_write</code> is likely to send more statements to the master than needed. The application may be able to continue operation at a lower consistency level after it has done some critical reads.	Since 1.1.0.
<code>strong</code>	Request strong consistency. Only masters will be used.	Since 1.2.0.

`failover` Up to and including 1.3.x: string. Since 1.4.0: object.

Failover policy. Supported policies: `disabled` (default), `master`, `loop_before_master` (Since 1.4.0).

If no failover policy is set, the plugin will not do any automatic failover (`failover=disabled`). Whenever the plugin fails to connect a server it will emit a warning and set the connections error code and message. Thereafter it is up to the application to handle the error and, for example, resent the last statement to trigger the selection of another server.

Please note, the automatic failover logic is applied when opening connections only. Once a connection has been opened no automatic attempts are made to reopen it in case of an error. If, for example, the server a connection is connected to is shut down and the user attempts to run a statement on the connection, no automatic failover will be tried. Instead, an error will be reported.

If using `failover=master` the plugin will implicitly failover to a master, if available. Please check the concepts documentation to learn about potential pitfalls and risks of using `failover=master`.

Example 6.287 Optional master failover when failing to connect to slave (PECL/mysqlnd_ms < 1.4.0)

```

{
  "myapp": {
    "master": {
      "master_0": {

```

```

        "host": "localhost"
    },
    "slave": {
        "slave_0": {
            "host": "192.168.78.136",
            "port": "3306"
        }
    },
    "failover": "master"
}

```

Since PECL/mysqlnd_ms 1.4.0 the failover configuration keyword refers to an object.

Example 6.288 New syntax since 1.4.0

```

{
    "myapp": {
        "master": {
            "master_0": {
                "host": "localhost"
            }
        },
        "slave": {
            "slave_0": {
                "host": "192.168.78.136",
                "port": "3306"
            }
        },
        "failover": {"strategy": "master" }
    }
}

```

Keyword	Description	Version
<code>strategy</code>	Failover policy. Possible values: <code>disabled</code> (default), <code>master</code>, <code>loop_before_master</code> A value of <code>disabled</code> disables automatic failover. Setting <code>master</code> instructs the plugin to try to connect to a master in case of a slave connection error. If the master connection attempt fails, the plugin enters the failover loop and returns an error to the user. If using <code>loop_before_master</code> and a slave request is made, the plugin tries to connect to other slaves before failing over to a master. If multiple master are given and multi master is enabled, the plugin also loops over the list of masters and attempts to connect before returning an error to the user.	Since 1.4.0.
<code>remember_failures</code>	Remember failures for the duration of a web request. Default: <code>false</code>.	Since 1.4.0. The feature is only available

Keyword	Description	Version
	If set to <code>true</code> the plugin will remember failed hosts and skip the hosts in all future load balancing made for the duration of the current web request.	together with the <code>random</code> and <code>roundrobin</code> load balancing filter. Use of the setting is recommended.
<code>max_retries</code>	Maximum number of connection attempts before skipping host. Default: <code>0</code> (no limit). The setting is used to prevent hosts from being dropped of the host list upon the first failure. If set to <code>n > 0</code> , the plugin will keep the node in the node list even after a failed connection attempt. The node will not be removed immediately from the slave respectively master lists after the first connection failure but instead be tried to connect to up to <code>n</code> times in future load balancing rounds before being removed.	Since 1.4.0. The feature is only available together with the <code>random</code> and <code>roundrobin</code> load balancing filter.

Setting `failover` to any other value but `disabled`, `master` or `loop_before_master` will not emit any warning or error.

`lazy_connections` bool

Controls the use of lazy connections. Lazy connections are connections which are not opened before the client sends the first connection. Lazy connections are a default.

It is strongly recommended to use lazy connections. Lazy connections help to keep the number of open connections low. If you disable lazy connections and, for example, configure one MySQL replication master server and two MySQL replication slaves, the plugin will open three connections upon the first call to a connect function although the application might use the master connection only.

Lazy connections bare a risk if you make heavy use of actions which change the state of a connection. The plugin does not dispatch all state changing actions to all connections from the connection pool. The few dispatched actions are applied to already opened connections only. Lazy connections opened in the future are not affected. Only some settings are "remembered" and applied when lazy connections are opened.

Example 6.289 Disabling lazy connection

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
```

```
        "port": "3306"
      },
    },
    "lazy_connections": 0
  }
}
```

Please, see also [server_charset](#) to overcome potential problems with string escaping and servers using different default charsets.

[server_charset](#) string

The setting has been introduced in 1.4.0. It is recommended to set it if using lazy connections.

The [server_charset](#) setting serves two purposes. It acts as a fallback charset to be used for string escaping done before a connection has been established and it helps to avoid escaping pitfalls in heterogeneous environments which servers using different default charsets.

String escaping takes a connections charset into account. String escaping is not possible before a connection has been opened and the connections charset is known. The use of lazy connections delays the actual opening of connections until a statement is send.

An application using lazy connections may attempt to escape a string before sending a statement. In fact, this should be a common case as the statement string may contain the string that is to be escaped. However, due to the lazy connection feature no connection has been opened yet and escaping fails. The plugin may report an error of the type [E_WARNING](#) and a message like [\(mysqlnd_ms\) string escaping doesn't work without established connection. Possible solution is to add server_charset to your configuration](#) to inform you of the pitfall.

Setting [server_charset](#) makes the plugin use the given charset for string escaping done on lazy connection handles before establishing a network connection to MySQL. Furthermore, the plugin will enforce the use of the charset when the connection is established.

Enforcing the use of the configured charset used for escaping is done to prevent tapping into the pitfall of using a different charset for escaping than used later for the connection. This has the additional benefit of removing the need to align the charset configuration of all servers used. No matter what the default charset on any of the servers is, the plugin will set the configured one as a default.

The plugin does not stop the user from changing the charset at any time using the [set_charset](#) call or corresponding SQL statements. Please, note that the use of SQL is not recommended as it cannot be monitored by the plugin. The user can, for example, change the charset on a lazy connection handle after escaping a string and before the actual connection is opened. The charset set by the user will be used for any subsequent escaping before the connection is established. The connection will be established using the configured charset, no matter what the server charset is or what the user has set before. Once a connection has been opened, [set_charset](#) is of no meaning anymore.

Example 6.290 String escaping on a lazy connection handle

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "lazy_connections": 1,
    "server_charset" : "utf8"
  }
}
```

```
<?php
$mysqli = new mysqli("myapp", "username", "password", "database");
$mysqli->real_escape("this will be escaped using the server_charset setting");
$mysqli->set_charset("latin1");
$mysqli->real_escape("this will be escaped using latin1");
/* server_charset implicitly set - utf8 connection */
$mysqli->query("SELECT 'This connection will be set to server_charset upon
/* latin1 used from now on */
$mysqli->set_charset("latin1");
?>
```

master_on_write bool

If set, the plugin will use the master server only after the first statement has been executed on the master. Applications can still send statements to the slaves using SQL hints to overrule the automatic decision.

The setting may help with replication lag. If an application runs an **INSERT** the plugin will, by default, use the master to execute all following statements, including **SELECT** statements. This helps to avoid problems with reads from slaves which have not replicated the **INSERT** yet.

Example 6.291 Master on write for consistent reads

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "master_on_write": 1
  }
}
```



```
}
}
```

Please, note the `quality_of_service` filter introduced in version 1.2.0-alpha. It gives finer control, for example, for achieving read-your-writes and, it offers additional functionality introducing `service levels`.

All `transaction stickiness` settings, including `trx_stickiness=on`, are overruled by `master_on_write=1`.

`trx_stickiness` string

Transaction stickiness policy. Supported policies: `disabled` (default), `master`.

The setting requires 5.4.0 or newer. If used with PHP older than 5.4.0, the plugin will emit a warning like `(mysqlnd_ms) trx_stickiness strategy is not supported before PHP 5.3.99`.

If no transaction stickiness policy is set or, if setting `trx_stickiness=disabled`, the plugin is not transaction aware. Thus, the plugin may load balance connections and switch connections in the middle of a transaction. The plugin is not transaction safe. SQL hints must be used avoid connection switches during a transaction.

As of PHP 5.4.0 the `mysqlnd` library allows the plugin to monitor the `autocommit` mode set by calls to the libraries `set_autocommit()` function. If setting `set_stickiness=master` and `autocommit` gets disabled by a PHP MySQL extension invoking the `mysqlnd` library internal function call `set_autocommit()`, the plugin is made aware of the begin of a transaction. Then, the plugin stops load balancing and directs all statements to the master server until `autocommit` is enabled. Thus, no SQL hints are required.

An example of a PHP MySQL API function calling the `mysqlnd` library internal function call `set_autocommit()` is `mysqli_autocommit`.

Although setting `trx_stickiness=master`, the plugin cannot be made aware of `autocommit` mode changes caused by SQL statements such as `SET AUTOCOMMIT=0` or `BEGIN`.

As of PHP 5.5.0, the `mysqlnd` library features additional C API calls to control transactions. The level of control matches the one offered by SQL statements. The `mysqli` API has been modified to use these calls. Since version 1.5.0, `PECL/mysqlnd_ms` can monitor not only `mysqli_autocommit`, but also `mysqli_begin`, `mysqli_commit` and `mysqli_rollback` to detect transaction boundaries and stop load balancing for the duration of a transaction.

Example 6.292 Using master to execute transactions

```
{
  "myapp": {
    "master": {
      "master_0": {
```

```

        "host": "localhost"
    },
    "slave": {
        "slave_0": {
            "host": "192.168.78.136",
            "port": "3306"
        }
    },
    "trx_stickiness": "master"
}
}

```

Since version 1.5.0 automatic and silent failover is disabled for the duration of a transaction. If the boundaries of a transaction have been properly detected, transaction stickiness is enabled and a server fails, the plugin will not attempt to fail over to the next server, if any, regardless of the failover policy configured. The user must handle the error manually. Depending on the configuration, the plugin may emit an error of type `E_WARNING` reading like `(mysqlnd_ms) Automatic failover is not permitted in the middle of a transaction`. This error may then be overwritten by follow up errors such as `(mysqlnd_ms) No connection selected by the last filter`. Those errors will be generated by the failing query function.

Example 6.293 No automatic failover, error handling pitfall

```

<?php
/* assumption: automatic failover configured */
$mysqli = new mysqli("myapp", "username", "password", "database");
/* sets plugin internal state in_trx = 1 */
$mysqli->autocommit(false);
/* assumption: server fails */
if (!($res = $mysqli->query("SELECT 'Assume this query fails' AS _msg FROM
/* handle failure of transaction, plugin internal state is still in_trx =
printf("%d] %s", $mysqli->errno, $mysqli->error);
/*
    If using autocommit() based transaction detection it is a
    MUST to call autocommit(true). Otherwise the plugin assumes
    the current transaction continues and connection
    changes remain forbidden.
*/
$mysqli->autocommit(true);
/* Likewise, you'll want to start a new transaction */
$mysqli->autocommit(false);
}
/* latin1 used from now on */
$mysqli->set_charset("latin1");
?>

```

If a server fails in the middle of a transaction the plugin continues to refuse to switch connections until the current transaction has been finished. Recall that the plugin monitors API calls to detect transaction boundaries. Thus, you have to, for example, enable auto commit mode to end the current transaction before the plugin continues load balancing and switches the server. Likewise, you will want to start a new transaction immediately thereafter and disable auto commit mode again.

Not handling failed queries and not ending a failed transaction using API calls may cause all following commands emit errors such as `Commands out of sync; you can't run this command now`. Thus, it is important to handle all errors.

`transient_error` object

The setting has been introduced in 1.6.0.

A database cluster node may reply a transient error to a client. The client can then repeat the operation on the same node, fail over to a different node or abort the operation. Per definition is it safe for a client to retry the same operation on the same node before giving up.

`PECL/mysqlnd_ms` can perform the retry loop on behalf of the application. By configuring `transient_error` the plugin can be instructed to repeat operations failing with a certain error code for a certain maximum number of times with a pause between the retries. If the transient error disappears during loop execution, it is hidden from the application. Otherwise, the error is forwarded to the application by the end of the loop.

Example 6.294 Retry loop for transient errors

```
{
  "myapp": {
    "master": {
      "master_0": {
        "host": "localhost"
      }
    },
    "slave": {
      "slave_0": {
        "host": "192.168.78.136",
        "port": "3306"
      }
    },
    "transient_error": {
      "mysql_error_codes": [
        1297
      ],
      "max_retries": 2,
      "usleep_retry": 100
    }
  }
}
```

Keyword	Description	Version
<code>mysqlnd_ms.transient_error.mysql_error_codes</code>	List of transient error codes. You may add any MySQL error code to the list. It is possible to consider any error as transient not only <code>1297</code> (<code>HY000 (ER_GET_TEMPORARY_ERRMSG), Message: Got temporary error %d '%s' from %s</code>). Before adding other codes but <code>1297</code> to the list, make sure your cluster supports a new attempt without impacting the state of your application.	Since 1.6.0.

Keyword	Description	Version
<code>max_retries</code>	How often to retry an operation which fails with a transient error before forwarding the failure to the user. Default: <code>1</code>	Since 1.6.0.
<code>usleep</code>	Milliseconds to sleep between transient error retries. The value is passed to the C function <code>usleep</code> , hence the name. Default: <code>100</code>	Since 1.6.0.

`xa` object

The setting has been introduced in 1.6.0.

Experimental

The feature is currently under development. There may be issues and/or feature limitations. Do not use in production environments.

`state_store`

`record_participant_credentials`

Whether to store the username and password of a global transaction participant in the participants table. If disabled, the garbage collection will use the default username and password when connecting to the participants. Unless you are using

a
different
userna
and
passwo
for
each
of
your
MySQL
servers
you
can
use
the
default
and
avoid
storing
the
sensibl
informa
in
state
store.

Please
note,
userna
and
passwo
are
stored
in
clear
text
when
using
the
MySQL
state
store,
which
is
the
only
one
availab
It
is
in
your
respons
to
protect
this
sensibl
informa

	Default: <code>false</code>
participant_localhost_ip	During XA garbage collection the plugin may find a participant server for which the host <code>localhost</code> has been recorded. If the garbage collection takes place on another host but the host that has written the participant record to the state store, the host name <code>localhost</code> now resolves to a different host. Therefore, when recording

a
particip
servers
host
name
in
the
state
store,
a
value
of
[local](#)
must
be
replace
with
the
actual
IP
address
of
[local](#)

Setting
[parti](#)
should
be
conside
only
if
using
[local](#)
cannot
be
avoided
From
a
garbag
collecti
point
of
view
only,
it
is
preferr
not
to
configu
any
socket
connec
but
to
provide
an
IP

address
and
port
for
a
node.

mysql

The
MySQL
state
store
is
the
only
state
store
available.

global_trx_

particip

garbage_co

host

user

password

db

port

socket

rollback_on_close

Whether to automatically rollback an open global transaction when a connection is closed. If enabled, it mimics the default behaviour of local transactions. Should a client disconnect, the server rolls back any open and unfinished transactions.

Default: `true`

garbage_collection

max_retries

Maximum number of garbage collection runs before giving up. Allowed values are from `0` to `100`. A setting of `0` means no limit, unless the state store enforces a limit. Should the state store

enforce
a
limit,
it
can
be
suppos
to
be
signific
higher
than
[100](#).
Availab
since
1.6.0.

Please
note,
it
is
importa
to
end
failed
XA
transac
within
reason
time
to
make
particip
servers
free
resourc
bound
to
the
transac
The
built-
in
garbag
collecti
is
not
expecte
to
fail
for
a
long
period
as
long
as
crashe

servers become available again quickly. Still, a situation may arise where a human is required to act because the built-in garbage collection stopped or failed. In this case, you may first want to check if the transaction still cannot be fixed by forcing `mysqlnd_u` to ignore the setting, prior to handling it manually.

Default:
`5`

probability

Garbage
collecti
probab
Allowed
values
are
from
[0](#)
to
[1000](#).
A
setting
of
[0](#)
disables
automa
backgre
garbag
collecti
Despite
a
setting
of
[0](#)
it
is
still
possibl
to
trigger
garbag
collecti
by
calling
[mysql](#)
Availab
since
1.6.0.

The
automa
garbag
collecti
of
stalled
XA
transac
is
only
availab
if
a
state
store
have
been
configu

The state store is responsible to keep track of XA transaction Based on its recordings it can find blocked XA transaction where the client has crashed, connect to the participants and rollback the unfinished transaction

The garbage collection is triggered as part of PHP's request shutdown procedure at the end of a web request. That is after

your
PHP
script
has
finished
working.
Do
decide
whether
to
run
the
garbag
collecti
a
random
value
between
0
and
1000
is
comput
If
the
the
probal
value
is
higher
or
equal
to
the
random
value,
the
state
stores
garbag
collecti
routine
are
invoked

Default
5

max_transactions_per_run

Maximu
numbe
of
unfinis
XA
transac
conside
by
the
garbag
collecti

during
one
run.
Allowed
values
are
from
[1](#)
to
[32768](#).
Available
since
1.6.0.

Cleaning
up
an
unfinished
XA
transaction
takes
considerable
amounts
of
time
and
resources.
The
garbage
collection
routine
may
have
to
connect
to
several
participants
of
a
failed
global
transaction
to
issue
the
SQL
commands
for
rolling
back
the
unfinished
transaction.

Default:
[100](#)

Plugin configuration file (<= 1.0.x)

Copyright 1997-2014 the PHP Documentation Group.

Note

The below description applies to PECL/mysqlnd_ms < 1.1.0-beta. It is not valid for later versions.

The plugin is using its own configuration file. The configuration file holds information on the MySQL replication master server, the MySQL replication slave servers, the server pick (load balancing) policy, the failover strategy and the use of lazy connections.

The PHP configuration directive `mysqlnd_ms.ini_file` is used to set the plugins configuration file.

The configuration file mimics standard the `php.ini` format. It consists of one or more sections. Every section defines its own unit of settings. There is no global section for setting defaults.

Applications reference sections by their name. Applications use section names as the host (server) parameter to the various connect methods of the `mysqli`, `mysql` and `PDO_MYSQL` extensions. Upon connect the `mysqlnd` plugin compares the hostname with all section names from the plugin configuration file. If hostname and section name match, the plugin will load the sections settings.

Example 6.298 Using section names example

```
[myapp]
master[] = localhost
slave[] = 192.168.2.27
slave[] = 192.168.2.28:3306
[localhost]
master[] = localhost:/tmp/mysql/mysql.sock
slave[] = 192.168.3.24:3305
slave[] = 192.168.3.65:3309
```

```
<?php
/* All of the following connections will be load balanced */
$mysqli = new mysqli("myapp", "username", "password", "database");
$pdo = new PDO('mysql:host=myapp;dbname=database', 'username', 'password');
$mysql = mysql_connect("myapp", "username", "password");
$mysqli = new mysqli("localhost", "username", "password", "database");
?>
```

Section names are strings. It is valid to use a section name such as `192.168.2.1`, `127.0.0.1` or `localhost`. If, for example, an application connects to `localhost` and a plugin configuration section `[localhost]` exists, the semantics of the connect operation are changed. The application will no longer only use the MySQL server running on the host `localhost` but the plugin will start to load balance MySQL queries following the rules from the `[localhost]` configuration section. This way you can load balance queries from an application without changing the applications source code.

The `master[]`, `slave[]` and `pick[]` configuration directives use a list-like syntax. Configuration directives supporting list-like syntax may appear multiple times in a configuration section. The plugin maintains the order in which entries appear when interpreting them. For example, the below example shows two `slave[]` configuration directives in the configuration section `[myapp]`. If doing round-robin load balancing for read-only queries, the plugin will send the first read-only query to the MySQL server `mysql_slave_1` because it is the first in the list. The second read-only query will be send to the MySQL server `mysql_slave_2` because it is the second in the list. Configuration directives supporting list-like syntax result are ordered from top to bottom in accordance to their appearance within a configuration section.

Example 6.299 List-like syntax

```
[myapp]
master[] = mysql_master_server
slave[] = mysql_slave_1
slave[] = mysql_slave_2
```

Here is a short explanation of the configuration directives that can be used.

master[] string

URI of a MySQL replication master server. The URI follows the syntax `hostname[:port|unix_domain_socket]`.

The plugin supports using only one master server.

Setting a master server is mandatory. The plugin will report a warning upon connect if the user has failed to provide a master server for a configuration section. The warning may read `(mysqlnd_ms) Cannot find master section in config`. Furthermore the plugin may set an error code for the connection handle such as `HY000/2000 (CR_UNKNOWN_ERROR)`. The corresponding error message depends on your language settings.

slave[] string

URI of one or more MySQL replication slave servers. The URI follows the syntax `hostname[:port|unix_domain_socket]`.

The plugin supports using one or more slave servers.

Setting a slave server is mandatory. The plugin will report a warning upon connect if the user has failed to provide at least one slave server for a configuration section. The warning may read `(mysqlnd_ms) Cannot find slaves section in config`. Furthermore the plugin may set an error code for the connection handle such as `HY000/2000 (CR_UNKNOWN_ERROR)`. The corresponding error message depends on your language settings.

pick[] string

Load balancing (server picking) policy. Supported policies: `random`, `random_once` (default), `roundrobin`, `user`.

If no load balancing policy is set, the plugin will default to `random_once`. The `random_once` policy picks a random slave server when running the first read-only statement. The slave server will be used for all read-only statements until the PHP script execution ends.

The `random` policy will pick a random server whenever a read-only statement is to be executed.

If using `roundrobin` the plugin iterates over the list of configured slave servers to pick a server for statement execution. If the plugin reaches the end of the list, it wraps around to the beginning of the list and picks the first configured slave server.

Setting more than one load balancing policy for a configuration section makes only sense in conjunction with `user` and `mysqlnd_ms_set_user_pick_server`. If the user defined callback fails to pick a server, the plugin falls back to the second configured load balancing policy.

failover string

Failover policy. Supported policies: `disabled` (default), `master`.

If no failover policy is set, the plugin will not do any automatic failover (`failover=disabled`). Whenever the plugin fails to connect a server it will emit a warning and set the connections error code and message. Thereafter it is up to the application to handle the error and, for example, resent the last statement to trigger the selection of another server.

If using `failover=master` the plugin will implicitly failover to a slave, if available. Please check the concepts documentation to learn about potential pitfalls and risks of using `failover=master`.

`lazy_connections` bool

Controls the use of lazy connections. Lazy connections are connections which are not opened before the client sends the first connection.

It is strongly recommended to use lazy connections. Lazy connections help to keep the number of open connections low. If you disable lazy connections and, for example, configure one MySQL replication master server and two MySQL replication slaves, the plugin will open three connections upon the first call to a connect function although the application might use the master connection only.

Lazy connections bare a risk if you make heavy use of actions which change the state of a connection. The plugin does not dispatch all state changing actions to all connections from the connection pool. The few dispatched actions are applied to already opened connections only. Lazy connections opened in the future are not affected. If, for example, the connection character set is changed using a PHP MySQL API call, the plugin will change the character set of all currently opened connection. It will not remember the character set change to apply it on lazy connections opened in the future. As a result the internal connection pool would hold connections using different character sets. This is not desired. Remember that character sets are taken into account for escaping.

`master_on_write` bool

If set, the plugin will use the master server only after the first statement has been executed on the master. Applications can still send statements to the slaves using SQL hints to overrule the automatic decision.

The setting may help with replication lag. If an application runs an `INSERT` the plugin will, by default, use the master to execute all following statements, including `SELECT` statements. This helps to avoid problems with reads from slaves which have not replicated the `INSERT` yet.

`trx_stickiness` string

Transaction stickiness policy. Supported policies: `disabled` (default), `master`.

Experimental feature.

The setting requires 5.4.0 or newer. If used with PHP older than 5.4.0, the plugin will emit a warning like `(mysqlnd_ms) trx_stickiness strategy is not supported before PHP 5.3.99`.

If no transaction stickiness policy is set or, if setting `trx_stickiness=disabled`, the plugin is not transaction

aware. Thus, the plugin may load balance connections and switch connections in the middle of a transaction. The plugin is not transaction safe. SQL hints must be used avoid connection switches during a transaction.

As of PHP 5.4.0 the `mysqlnd` library allows the plugin to monitor the `autocommit` mode set by calls to the libraries `trx_autocommit()` function. If setting `trx_stickiness=master` and `autocommit` gets disabled by a PHP MySQL extension invoking the `mysqlnd` library internal function call `trx_autocommit()`, the plugin is made aware of the begin of a transaction. Then, the plugin stops load balancing and directs all statements to the master server until `autocommit` is enabled. Thus, no SQL hints are required.

An example of a PHP MySQL API function calling the `mysqlnd` library internal function call `trx_autocommit()` is `mysqli_autocommit`.

Although setting `trx_stickiness=master`, the plugin cannot be made aware of `autocommit` mode changes caused by SQL statements such as `SET AUTOCOMMIT=0`.

Testing

Copyright 1997-2014 the PHP Documentation Group.

Note

The section applies to `mysqlnd_ms` 1.1.0 or newer, not the 1.0 series.

The PECL/`mysqlnd_ms` test suite is in the `tests/` directory of the source distribution. The test suite consists of standard phpt tests, which are described on the PHP Quality Assurance Teams website.

Running the tests requires setting up one to four MySQL servers. Some tests don't connect to MySQL at all. Others require one server for testing. Some require two distinct servers. In some cases two servers are used to emulate a replication setup. In other cases a master and a slave of an existing MySQL replication setup are required for testing. The tests will try to detect how many servers and what kind of servers are given. If the required servers are not found, the test will be skipped automatically.

Before running the tests, edit `tests/config.inc` to configure the MySQL servers to be used for testing.

The most basic configuration is as follows.

```
putenv("MYSQL_TEST_HOST=localhost");
putenv("MYSQL_TEST_PORT=3306");
putenv("MYSQL_TEST_USER=root");
putenv("MYSQL_TEST_PASSWD=");
putenv("MYSQL_TEST_DB=test");
putenv("MYSQL_TEST_ENGINE=MyISAM");
putenv("MYSQL_TEST_SOCKET=");
putenv("MYSQL_TEST_SKIP_CONNECT_FAILURE=1");
putenv("MYSQL_TEST_CONNECT_FLAGS=0");
putenv("MYSQL_TEST_EXPERIMENTAL=0");
/* replication cluster emulation */
putenv("MYSQL_TEST_EMULATED_MASTER_HOST=". getenv("MYSQL_TEST_HOST"));
putenv("MYSQL_TEST_EMULATED_SLAVE_HOST=". getenv("MYSQL_TEST_HOST"));
/* real replication cluster */
putenv("MYSQL_TEST_MASTER_HOST=". getenv("MYSQL_TEST_EMULATED_MASTER_HOST"));
```



```
putenv("MYSQL_TEST_SLAVE_HOST=". getenv("MYSQL_TEST_EMULATED_SLAVE_HOST"));
```

`MYSQL_TEST_HOST`, `MYSQL_TEST_PORT` and `MYSQL_TEST_SOCKET` define the hostname, TCP/IP port and Unix domain socket of the default database server. `MYSQL_TEST_USER` and `MYSQL_TEST_PASSWD` contain the user and password needed to connect to the database/schema configured with `MYSQL_TEST_DB`. All configured servers must have the same database user configured to give access to the test database.

Using `host`, `host:port` or `host:/path/to/socket` syntax one can set an alternate host, host and port or host and socket for any of the servers.

```
putenv("MYSQL_TEST_SLAVE_HOST=192.168.78.136:3307");
putenv("MYSQL_TEST_MASTER_HOST=myserver_hostname:/path/to/socket");
```

Debugging and Tracing

Copyright 1997-2014 the PHP Documentation Group.

The `mysqlnd` debug log can be used to debug and trace the activities of `PECL/mysqlnd_ms`. As a `mysqlnd` `PECL/mysqlnd_ms` adds trace information to the `mysqlnd` library debug file. Please, see the [mysqlnd.debug](#) PHP configuration directive documentation for a detailed description on how to configure the debug log.

Configuration setting example to activate the debug log:

```
mysqlnd.debug=d:t:x:0,/tmp/mysqlnd.trace
```

Note

This feature is only available with a debug build of PHP. Works on Microsoft Windows if using a debug build of PHP and PHP was built using Microsoft Visual C version 9 and above.

The debug log shows `mysqlnd` library and `PECL/mysqlnd_ms` plugin function calls, similar to a trace log. `mysqlnd` library calls are usually prefixed with `mysqlnd_`. `PECL/mysqlnd` internal calls begin with `mysqlnd_ms`.

Example excerpt from the debug log (connect):

```
[...]
>mysqlnd_connect
| info : host=myapp user=root db=test port=3306 flags=131072
| >mysqlnd_ms::connect
| | >mysqlnd_ms_config_json_section_exists
| | | info : section=[myapp] len=[5]
| | | >mysqlnd_ms_config_json_sub_section_exists
| | | | info : section=[myapp] len=[5]
| | | | info : ret=1
| | | <mysqlnd_ms_config_json_sub_section_exists
| | | info : ret=1
| | <mysqlnd_ms_config_json_section_exists
[...]
```

The debug log is not only useful for plugin developers but also to find the cause of user errors. For example, if your application does not do proper error handling and fails to record error messages, checking the debug and trace log may help finding the cause. Use of the debug log to debug application issues should be considered only if no other option is available. Writing the debug log to disk is a slow operation and may have negative impact on the application performance.

Example excerpt from the debug log (connection failure):

```
[...]
| | | | | info : adding error [Access denied for user 'root'@'localhost' (using password: YES)] to the
| | | | | info : PACKET_FREE(0)
| | | | | info : PACKET_FREE(0x7f3ef6323f50)
| | | | | info : PACKET_FREE(0x7f3ef6324080)
| | | | | <mysqlnd_auth_handshake
| | | | | info : switch_to_auth_protocol=n/a
| | | | | info : conn->error_info.error_no = 1045
| | | | | <mysqlnd_connect_run_authentication
| | | | | info : PACKET_FREE(0x7f3ef63236d8)
| | | | | >mysqlnd_conn::free_contents
| | | | | >mysqlnd_net::free_contents
| | | | | <mysqlnd_net::free_contents
| | | | | info : Freeing memory of members
| | | | | info : scheme=unix:///tmp/mysql.sock
| | | | | >mysqlnd_error_list_pdtor
| | | | | <mysqlnd_error_list_pdtor
| | | | | <mysqlnd_conn::free_contents
| | | | | <mysqlnd_conn::connect
[...]
```

The trace log can also be used to verify correct behaviour of PECL/mysqlnd_ms itself, for example, to check which server has been selected for query execution and why.

Example excerpt from the debug log (plugin decision):

```
[...]
>mysqlnd_ms::query
| info : query=DROP TABLE IF EXISTS test
| >mysqlnd_plugin_get_plugin_connection_data
| | info : plugin_id=5
| <mysqlnd_plugin_get_plugin_connection_data
| >mysqlnd_ms_pick_server_ex
| | info : conn_data=0x7fb6a7d3e5a0 *conn_data=0x7fb6a7d410d0
| | >mysqlnd_ms_select_servers_all
| | <mysqlnd_ms_select_servers_all
| | >mysqlnd_ms_choose_connection_rr
| | >mysqlnd_ms_query_is_select
[...]
```

```
| | <mysqlnd_ms_query_is_select
[...]
```

```
| | info : Init the master context
| | info : list(0x7fb6a7d3f598) has 1
| | info : Using master connection
| | >mysqlnd_ms_advanced_connect
| | | >mysqlnd_conn::connect
| | | | info : host=localhost user=root db=test port=3306 flags=131072 persistent=0 state=0
```

In this case the statement `DROP TABLE IF EXISTS test` has been executed. Note that the statement string is shown in the log file. You may want to take measures to restrict access to the log for security considerations.

The statement has been load balanced using round robin policy, as you can easily guess from the functions name `>mysqlnd_ms_choose_connection_rr`. It has been sent to a master

server running on `host=localhost user=root db=test port=3306 flags=131072 persistent=0 state=0`.

Monitoring

Copyright 1997-2014 the PHP Documentation Group.

Plugin activity can be monitored using the `mysqlnd` trace log, `mysqlnd` statistics, `mysqlnd_ms` plugin statistics and external PHP debugging tools. Use of the trace log should be limited to debugging. It is recommended to use the plugins statistics for monitoring.

Writing a trace log is a slow operation. If using an external PHP debugging tool, please refer to the vendors manual about its performance impact and the type of information collected. In many cases, external debugging tools will provide call stacks. Often, a call stack or a trace log is more difficult to interpret than the statistics provided by the plugin.

Plugin statistics tell how often which kind of cluster node has been used (slave or master), why the node was used, if lazy connections have been used and if global transaction ID injection has been performed. The monitoring information provided enables user to verify plugin decisions and to plan their cluster resources based on usage pattern. The function `mysqlnd_ms_get_stats` is used to access the statistics. Please, see the functions description for a list of available statistics.

Statistics are collected on a per PHP process basis. Their scope is a PHP process. Depending on the PHP deployment model a process may serve one or multiple web requests. If using CGI model, a PHP process serves one web request. If using FastCGI or pre-fork web server models, a PHP process usually serves multiple web requests. The same is the case with a threaded web server. Please, note that threads running in parallel can update the statistics in parallel. Thus, if using a threaded PHP deployment model, statistics can be changed by more than one script at a time. A script cannot rely on the fact that it sees only its own changes to statistics.

Example 6.300 Verify plugin activity in a non-threaded deployment model

```
mysqlnd_ms.enable=1
mysqlnd_ms.collect_statistics=1
```

```
<?php
/* Load balanced following "myapp" section rules from the plugins config file (not shown) */
$mysqli = new mysqli("myapp", "username", "password", "database");
if (mysqli_connect_errno())
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
$stats_before = mysqlnd_ms_get_stats();
if ($res = $mysqli->query("SELECT 'Read request' FROM DUAL")) {
    var_dump($res->fetch_all());
}
$stats_after = mysqlnd_ms_get_stats();
if ($stats_after['use_slave'] <= $stats_before['use_slave']) {
    echo "According to the statistics the read request has not been run on a slave!";
}
?>
```

Statistics are aggregated for all plugin activities and all connections handled by the plugin. It is not possible to tell how much a certain connection handle has contributed to the overall statistics.

Utilizing PHPs `register_shutdown_function` function or the `auto_append_file` PHP configuration directive it is easily possible to dump statistics into, for example, a log file when a script finishes. Instead of using a log file it is also possible to send the statistics to an external monitoring tool for recording and display.

Example 6.301 Recording statistics during shutdown

```
mysqlnd_ms.enable=1
mysqlnd_ms.collect_statistics=1
error_log=/tmp/php_errors.log
```

```
<?php
function check_stats() {
    $msg = str_repeat("-", 80) . "\n";
    $msg .= var_export(mysqlnd_ms_get_stats(), true) . "\n";
    $msg .= str_repeat("-", 80) . "\n";
    error_log($msg);
}
register_shutdown_function("check_stats");
?>
```

6.7.7 Predefined Constants

[Copyright 1997-2014 the PHP Documentation Group.](#)

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

SQL hint related

Example 6.302 Example demonstrating the usage of mysqlnd_ms constants

The mysqlnd replication and load balancing plugin ([mysqlnd_ms](#)) performs read/write splitting. This directs write queries to a MySQL master server, and read-only queries to the MySQL slave servers. The plugin has a built-in read/write split logic. All queries which start with [SELECT](#) are considered read-only queries, which are then sent to a MySQL slave server that is listed in the plugin configuration file. All other queries are directed to the MySQL master server that is also specified in the plugin configuration file.

User supplied SQL hints can be used to overrule automatic read/write splitting, to gain full control on the process. SQL hints are standards compliant SQL comments. The plugin will scan the beginning of a query string for an SQL comment for certain commands, which then control query redirection. Other systems involved in the query processing are unaffected by the SQL hints because other systems will ignore the SQL comments.

The plugin supports three SQL hints to direct queries to either the MySQL slave servers, the MySQL master server, or the last used MySQL server. SQL hints must be placed at the beginning of a query to be recognized by the plugin.

For better portability, it is recommended to use the string constants [MYSQLND_MS_MASTER_SWITCH](#), [MYSQLND_MS_SLAVE_SWITCH](#) and [MYSQLND_MS_LAST_USED_SWITCH](#) instead of their literal values.

```
<?php
/* Use constants for maximum portability */
$master_query = "/*" . MYSQLND_MS_MASTER_SWITCH . "*/SELECT id FROM test";
/* Valid but less portable: using literal instead of constant */
$slave_query = "/*ms=slave*/SHOW TABLES";
printf("master_query = '%s'\n", $master_query);
printf("slave_query = '%s'\n", $slave_query);
?>
```

The above examples will output:

```
master_query = /*ms=master*/SELECT id FROM test
slave_query = /*ms=slave*/SHOW TABLES
```

- MYSQLND_MS_MASTER_SWITCH** (string) SQL hint used to send a query to the MySQL replication master server.
- MYSQLND_MS_SLAVE_SWITCH** (string) SQL hint used to send a query to one of the MySQL replication slave servers.
- MYSQLND_MS_LAST_USED_SWITCH** (string) SQL hint used to send a query to the last used MySQL server. The last used MySQL server can either be a master or a slave server in a MySQL replication setup.
- mysqlnd_ms_query_is_select** related
- MYSQLND_MS_QUERY_USE_MASTER** (integer) If **mysqlnd_ms_is_select** returns **MYSQLND_MS_QUERY_USE_MASTER** for a given query, the built-in read/write split mechanism recommends sending the query to a MySQL replication master server.
- MYSQLND_MS_QUERY_USE_SLAVE** (integer) If **mysqlnd_ms_is_select** returns **MYSQLND_MS_QUERY_USE_SLAVE** for a given query, the built-in read/write split mechanism recommends sending the query to a MySQL replication slave server.
- MYSQLND_MS_QUERY_USE_LAST_USED** (integer) If **mysqlnd_ms_is_select** returns **MYSQLND_MS_QUERY_USE_LAST_USED** for a given query, the built-in read/write split mechanism recommends sending the query to the last used server.
- mysqlnd_ms_set_qos**, quality of service filter and service level related
- MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL** (integer) Use to request the service level eventual consistency from the **mysqlnd_ms_set_qos**. Eventual consistency is the default quality of service when reading from an asynchronous MySQL replication slave. Data returned in this service level may or may not be stale, depending on whether the selected slaves happen to have replicated the latest changes from the MySQL replication master or not.
- MYSQLND_MS_QOS_CONSISTENCY_SESSION** (integer) Use to request the service level session consistency from the **mysqlnd_ms_set_qos**. Session consistency is defined as read your writes. The client is guaranteed to see his latest changes.
- MYSQLND_MS_QOS_CONSISTENCY_STRONG** (integer) Use to request the service level strong consistency from the **mysqlnd_ms_set_qos**. Strong consistency is used to ensure all clients see each others changes.
- MYSQLND_MS_QOS_OPTION_GTID** (integer) Used as a service level option with **mysqlnd_ms_set_qos** to parameterize session consistency.
- MYSQLND_MS_QOS_OPTION_AGE** (integer) Used as a service level option with **mysqlnd_ms_set_qos** to parameterize eventual consistency.

Other

The plugins version number can be obtained using [MYSQLND_MS_VERSION](#) or [MYSQLND_MS_VERSION_ID](#). [MYSQLND_MS_VERSION](#) is the string representation of the numerical version number [MYSQLND_MS_VERSION_ID](#), which is an integer such as 10000. Developers can calculate the version number as follows.

Version (part)	Example
Major*10000	1*10000 = 10000
Minor*100	0*100 = 0
Patch	0 = 0
MYSQLND_MS_VERSION_ID	10000

[MYSQLND_MS_VERSION](#) (string) Plugin version string, for example, "1.0.0-prototype".

[MYSQLND_MS_VERSION_ID](#) (integer) Plugin version number, for example, 10000.

6.7.8 Mysqlnd_ms Functions

Copyright 1997-2014 the PHP Documentation Group.

6.7.8.1 mysqlnd_ms_dump_servers

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_ms_dump_servers](#)

Returns a list of currently configured servers

Description

```
array mysqlnd_ms_dump_servers(  
    mixed connection);
```

Returns a list of currently configured servers.

Parameters

[connection](#) A MySQL connection handle obtained from any of the connect functions of the [mysqli](#), [mysql](#) or [PDO_MYSQL](#) extensions.

Return Values

[FALSE](#) on error. Otherwise, returns an array with two entries [masters](#) and [slaves](#) each of which contains an array listing all corresponding servers.

The function can be used to check and debug the list of servers currently used by the plugin. It is mostly useful when the list of servers changes at runtime, for example, when using MySQL Fabric.

[masters](#) and [slaves](#) server entries

Key	Description	Version
name_from_config	Server entry name from config, if applicable. NULL if no configuration name is available.	Since 1.6.0.
hostname	Host name of the server.	Since 1.6.0.
user	Database user used to authenticate against the server.	Since 1.6.0.
port	TCP/IP port of the server.	Since 1.6.0.
socket	Unix domain socket of the server.	Since 1.6.0.

Notes

Note

`mysqlnd_ms_dump_servers` requires PECL mysqlnd_ms >> 1.6.0.

Examples**Example 6.303 `mysqlnd_ms_dump_servers` example**

```
{
  "myapp": {
    "master": {
      "master1": {
        "host": "master1_host",
        "port": "master1_port",
        "socket": "master1_socket",
        "db": "master1_db",
        "user": "master1_user",
        "password": "master1_pw"
      }
    },
    "slave": {
      "slave_0": {
        "host": "slave0_host",
        "port": "slave0_port",
        "socket": "slave0_socket",
        "db": "slave0_db",
        "user": "slave0_user",
        "password": "slave0_pw"
      },
      "slave_1": {
        "host": "slave1_host"
      }
    }
  }
}
```

```
<?php
$link = mysqli_connect("myapp", "global_user", "global_pass", "global_db", 1234, "global_socket");
var_dump(mysqlnd_ms_dump_servers($link);
?>
```

The above example will output:

```
array(2) {
  ["masters"]=>
  array(1) {
    [0]=>
    array(5) {
      ["name_from_config"]=>
      string(7) "master1"
      ["hostname"]=>
      string(12) "master1_host"
      ["user"]=>
      string(12) "master1_user"
      ["port"]=>
      int(3306)
      ["socket"]=>
      string(14) "master1_socket"
    }
  }
  ["slaves"]=>
  array(2) {
```

```
[0]=>
array(5) {
    ["name_from_config"]=>
    string(7) "slave_0"
    ["hostname"]=>
    string(11) "slave0_host"
    ["user"]=>
    string(11) "slave0_user"
    ["port"]=>
    int(3306)
    ["socket"]=>
    string(13) "slave0_socket"
}
[1]=>
array(5) {
    ["name_from_config"]=>
    string(7) "slave_1"
    ["hostname"]=>
    string(11) "slave1_host"
    ["user"]=>
    string(12) "global_user"
    ["port"]=>
    int(1234)
    ["socket"]=>
    string(13) "global_socket"
}
}
```

6.7.8.2 mysqlnd_ms_fabric_select_global

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_ms_fabric_select_global`

Switch to global sharding server for a given table

Description

```
array mysqlnd_ms_fabric_select_global(
    mixed connection,
    mixed table_name);
```

Warning

This function is currently not documented; only its argument list is available.

MySQL Fabric related.

Switch the connection to the nodes handling global sharding queries for the given table name.

Parameters

connection A MySQL connection handle obtained from any of the connect functions of the [mysqli](#), [mysql](#) or [PDO_MYSQL](#) extensions.

table_name The table name to ask Fabric about.

Return Values

[FALSE](#) on error. Otherwise, [TRUE](#)

Notes

Note

`mysqlnd_ms_fabric_select_global` requires PECL mysqlnd_ms >> 1.6.0.

6.7.8.3 `mysqlnd_ms_fabric_select_shard`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_ms_fabric_select_shard`

Switch to shard

Description

```
array mysqlnd_ms_fabric_select_shard(  
    mixed connection,  
    mixed table_name,  
    mixed shard_key);
```

Warning

This function is currently not documented; only its argument list is available.

MySQL Fabric related.

Switch the connection to the shards responsible for the given table name and shard key.

Parameters

<i>connection</i>	A MySQL connection handle obtained from any of the connect functions of the mysqli , mysql or PDO_MYSQL extensions.
<i>table_name</i>	The table name to ask Fabric about.
<i>shard_key</i>	The shard key to ask Fabric about.

Return Values

[FALSE](#) on error. Otherwise, [TRUE](#)

Notes

Note

`mysqlnd_ms_fabric_select_shard` requires PECL mysqlnd_ms >> 1.6.0.

6.7.8.4 `mysqlnd_ms_get_last_gtid`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_ms_get_last_gtid`

Returns the latest global transaction ID

Description

```
string mysqlnd_ms_get_last_gtid(  
    mixed connection);
```

Returns a global transaction identifier which belongs to a write operation no older than the last write performed by the client. It is not guaranteed that the global transaction identifier is identical to that one created for the last write transaction performed by the client.

Parameters

<i>connection</i>	A PECL/mysqlnd_ms connection handle to a MySQL server of the type PDO_MYSQL , mysqli or ext/mysql . The connection handle is obtained when opening a connection with a host name that matches a mysqlnd_ms configuration file entry using any of the above three MySQL driver extensions.

Return Values

Returns a global transaction ID (GTID) on success. Otherwise, returns [FALSE](#).

The function [mysqlnd_ms_get_last_gtid](#) returns the GTID obtained when executing the SQL statement from the [fetch_last_gtid](#) entry of the [global_transaction_id_injection](#) section from the plugins configuration file.

The function may be called after the GTID has been incremented.

Notes

Note

[mysqlnd_ms_get_last_gtid](#) requires PHP $\geq$ 5.4.0 and PECL [mysqlnd_ms](#) $\geq$ 1.2.0. Internally, it is using a [mysqlnd](#) library C functionality not available with PHP 5.3.

Please note, all MySQL 5.6 production versions do not provide clients with enough information to use GTIDs for enforcing session consistency. In the worst case, the plugin will choose the master only.

Examples

Example 6.304 [mysqlnd_ms_get_last_gtid](#) example

```
<?php
/* Open mysqlnd_ms connection using mysqli, PDO_MySQL or mysql extension */
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("DROP TABLE IF EXISTS test"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
printf("GTID after transaction %s\n", mysqlnd_ms_get_last_gtid($mysqli));
/* auto commit mode, transaction on master, GTID must be incremented */
if (!$mysqli->query("CREATE TABLE test(id INT)"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
printf("GTID after transaction %s\n", mysqlnd_ms_get_last_gtid($mysqli));
?>
```

See Also

[Global Transaction IDs](#)

6.7.8.5 [mysqlnd_ms_get_last_used_connection](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_ms_get_last_used_connection](#)

Returns an array which describes the last used connection

Description

```
array mysqlnd_ms_get_last_used_connection(
    mixed connection);
```

Returns an array which describes the last used connection from the plugins connection pool currently pointed to by the user connection handle. If using the plugin, a user connection handle represents a pool of database connections. It is not possible to tell from the user connection handles properties to which database server from the pool the user connection handle points.

The function can be used to debug or monitor PECL mysqlnd_ms.

Parameters

connection A MySQL connection handle obtained from any of the connect functions of the [mysqli](#), [mysql](#) or [PDO_MYSQL](#) extensions.

Return Values

FALSE on error. Otherwise, an array which describes the connection used to execute the last statement on.

Array which describes the connection.

Property	Description	Version
scheme	Connection scheme. Either tcp://host:port or unix://host:socket . If you want to distinguish connections from each other use a combination of scheme and thread_id as a unique key. Neither scheme nor thread_id alone are sufficient to distinguish two connections from each other. Two servers may assign the same thread_id to two different connections. Thus, connections in the pool may have the same thread_id . Also, do not rely on uniqueness of scheme in a pool. Your QA engineers may use the same MySQL server instance for two distinct logical roles and add it multiple times to the pool. This hack is used, for example, in the test suite.	Since 1.1.0.
host	Database server host used with the connection. The host is only set with TCP/IP connections. It is empty with Unix domain or Windows named pipe connections,	Since 1.1.0.
host_info	A character string representing the server hostname and the connection type.	Since 1.1.2.
port	Database server port used with the connection.	Since 1.1.0.
socket_domain	Unix domain socket or Windows named pipe used with the connection. The value is empty for TCP/IP connections.	Since 1.1.2.
thread_id	Connection thread id.	Since 1.1.0.
last_message	Info message obtained from the MySQL C API function mysql_info() . Please, see mysqli_info for a description.	Since 1.1.0.
errno	Error code.	Since 1.1.0.
error	Error message.	Since 1.1.0.
sqlstate	Error SQLstate code.	Since 1.1.0.

Notes

Note

[mysqlnd_ms_get_last_used_connection](#) requires PHP >= 5.4.0 and PECL mysqlnd_ms >> 1.1.0. Internally, it is using a [mysqlnd](#) library C call not available with PHP 5.3.

Examples

The example assumes that [myapp](#) refers to a plugin configuration file section and represents a connection pool.

Example 6.305 [mysqlnd_ms_get_last_used_connection](#) example

```
<?php
```

```
$link = new mysqli("myapp", "user", "password", "database");
$res = $link->query("SELECT 1 FROM DUAL");
var_dump(mysqlnd_ms_get_last_used_connection($link));
?>
```

The above example will output:

```
array(10) {
  ["scheme"]=>
  string(22) "unix:///tmp/mysql.sock"
  ["host_info"]=>
  string(25) "Localhost via UNIX socket"
  ["host"]=>
  string(0) ""
  ["port"]=>
  int(3306)
  ["socket_or_pipe"]=>
  string(15) "/tmp/mysql.sock"
  ["thread_id"]=>
  int(46253)
  ["last_message"]=>
  string(0) ""
  ["errno"]=>
  int(0)
  ["error"]=>
  string(0) ""
  ["sqlstate"]=>
  string(5) "00000"
}
```

6.7.8.6 mysqlnd_ms_get_stats

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_ms_get_stats](#)

Returns query distribution and connection statistics

Description

```
array mysqlnd_ms_get_stats();
```

Returns an array of statistics collected by the replication and load balancing plugin.

The PHP configuration setting [mysqlnd_ms.collect_statistics](#) controls the collection of statistics. The collection of statistics is disabled by default for performance reasons.

The scope of the statistics is the [PHP](#) process. Depending on your deployment model a [PHP](#) process may handle one or multiple requests.

Statistics are aggregated for all connections and all storage handler. It is not possible to tell how much queries originating from [mysqli](#), [PDO_MySQL](#) or [mysql](#) API calls have contributed to the aggregated data values.

Parameters

This function has no parameters.

Return Values

Returns [NULL](#) if the PHP configuration directive [mysqlnd_ms.enable](#) has disabled the plugin. Otherwise, returns array of statistics.

Array of statistics

Statistic	Description	Version
use_slave	The semantics of this statistic has changed between 1.0.1 - 1.1.0. The meaning for version 1.0.1 is as follows. Number of statements considered as read-only by the built-in query analyzer. Neither statements which begin with a SQL hint to force use of slave nor statements directed to a slave by an user-defined callback are included. The total number of statements sent to the slaves is use_slave + use_slave_sql_hint + use_slave_callback. PECL/mysqLnd_ms 1.1.0 introduces a new concept of chained filters. The statistics is now set by the internal load balancing filter. With version 1.1.0 the load balancing filter is always the last in the filter chain, if used. In future versions a load balancing filter may be followed by other filters causing another change in the meaning of the statistic. If, in the future, a load balancing filter is followed by another filter it is no longer guaranteed that the statement, which increments use_slave, will be executed on the slaves. The meaning for version 1.1.0 is as follows. Number of statements sent to the slaves. Statements directed to a slave by the user filter (an user-defined callback) are not included. The latter are counted by use_slave_callback.	Since 1.0.0.
use_master	The semantics of this statistic has changed between 1.0.1 - 1.1.0. The meaning for version 1.0.1 is as follows. Number of statements not considered as read-only by the built-in query analyzer. Neither statements which begin with a SQL hint to force use of master nor statements directed to a master by an user-defined callback are included. The total number of statements sent to the master is use_master + use_master_sql_hint + use_master_callback. PECL/mysqLnd_ms 1.1.0 introduces a new concept of chained filters. The statistics is now set by the internal load balancing filter. With version 1.1.0 the load balancing filter is always the last in the filter chain, if used. In future versions a load balancing filter may be followed by other filters causing another change in the meaning of the statistic. If, in the future, a load balancing filter is followed by another filter it is no longer guaranteed that the statement, which increments use_master, will be executed on the slaves. The meaning for version 1.1.0 is as follows. Number of statements sent to the masters. Statements directed to a master by the user filter (an user-defined callback) are not included. The latter are counted by use_master_callback.	Since 1.0.0.
use_slave_recommend	Number of statements the built-in query analyzer recommends sending to a slave because they contain no SQL hint to force use of a certain server. The recommendation may be overruled in the following. It is not guaranteed whether the statement will be executed on a slave or not. This is how often the internal is_select function has guessed that a slave shall be used. Please, see also the user space function mysqLnd_ms_query_is_select.	Since 1.1.0.
use_master_recommend	Number of statements the built-in query analyzer recommends sending to a master because they contain no SQL hint to force use of a certain server. The recommendation may be overruled in the following. It is not guaranteed whether the statement will be executed on a slave or	Since 1.1.0.

Statistic	Description	Version
	not. This is how often the internal <code>is_select</code> function has guessed that a master shall be used. Please, see also the user space function <code>mysqld_ms_query_is_select</code> .	
<code>use_slave</code>	Number of statements sent to a slave because statement begins with the SQL hint to force use of slave.	Since 1.0.0.
<code>use_master</code>	Number of statements sent to a master because statement begins with the SQL hint to force use of master.	Since 1.0.0.
<code>use_last</code>	Number of statements sent to server which has run the previous statement, because statement begins with the SQL hint to force use of previously used server.	Since 1.0.0.
<code>use_slave_callback</code>	Number of statements sent to a slave because an user-defined callback has chosen a slave server for statement execution.	Since 1.0.0.
<code>use_master_callback</code>	Number of statements sent to a master because an user-defined callback has chosen a master server for statement execution.	Since 1.0.0.
<code>non_lazy_slave</code>	Number of successfully opened slave connections from configurations not using <code>lazy connections</code> . The total number of successfully opened slave connections is <code>non_lazy_connections_slave_success</code> + <code>lazy_connections_slave_success</code>	Since 1.0.0.
<code>non_lazy_slave_failure</code>	Number of failed slave connection attempts from configurations not using <code>lazy connections</code> . The total number of failed slave connection attempts is <code>non_lazy_connections_slave_failure</code> + <code>lazy_connections_slave_failure</code>	Since 1.0.0.
<code>non_lazy_master</code>	Number of successfully opened master connections from configurations not using <code>lazy connections</code> . The total number of successfully opened master connections is <code>non_lazy_connections_master_success</code> + <code>lazy_connections_master_success</code>	Since 1.0.0.
<code>non_lazy_master_failure</code>	Number of failed master connection attempts from configurations not using <code>lazy connections</code> . The total number of failed master connection attempts is <code>non_lazy_connections_master_failure</code> + <code>lazy_connections_master_failure</code>	Since 1.0.0.
<code>lazy_slave</code>	Number of successfully opened slave connections from configurations using <code>lazy connections</code> .	Since 1.0.0.
<code>lazy_slave_failure</code>	Number of failed slave connection attempts from configurations using <code>lazy connections</code> .	Since 1.0.0.
<code>lazy_master</code>	Number of successfully opened master connections from configurations using <code>lazy connections</code> .	Since 1.0.0.
<code>lazy_master_failure</code>	Number of failed master connection attempts from configurations using <code>lazy connections</code> .	Since 1.0.0.
<code>trx_autocommit</code>	Number of <code>autocommit</code> mode activations via API calls. This figure may be used to monitor activity related to the plugin configuration setting <code>trx_stickiness</code> . If, for example, you want to know if a certain API call invokes the <code>mysqld</code> library function <code>trx_autocommit()</code> , which is a requirement for <code>trx_stickiness</code> , you may call the user API function in question and check if the statistic has changed. The statistic is modified only by the plugins internal subclassed <code>trx_autocommit()</code> method.	Since 1.0.0.
<code>trx_autocommit_deactivation</code>	Number of <code>autocommit</code> mode deactivations via API calls.	Since 1.0.0.

Statistic	Description	Version
<code>trx_master</code>	Number of statements redirected to the master while <code>trx_stickiness=master</code> and <code>autocommit</code> mode is disabled.	Since 1.0.0.
<code>gtid_auto</code>	Number of successful SQL injections in autocommit mode as part of the plugins client-side <code>global transaction id emulation</code> .	Since 1.2.0.
<code>gtid_auto</code>	Number of failed SQL injections in autocommit mode as part of the plugins client-side <code>global transaction id emulation</code> .	Since 1.2.0.
<code>gtid_commit</code>	Number of successful SQL injections in commit mode as part of the plugins client-side <code>global transaction id emulation</code> .	Since 1.2.0.
<code>gtid_commit</code>	Number of failed SQL injections in commit mode as part of the plugins client-side <code>global transaction id emulation</code> .	Since 1.2.0.
<code>gtid_implicit</code>	Number of successful SQL injections when implicit commit is detected as part of the plugins client-side <code>global transaction id emulation</code> . Implicit commit happens, for example, when autocommit has been turned off, a query is executed and autocommit is enabled again. In that case, the statement will be committed by the server and SQL to maintain is injected before the autocommit is re-enabled. Another sequence causing an implicit commit is <code>begin()</code> , <code>query()</code> , <code>begin()</code> . The second call to <code>begin()</code> will implicitly commit the transaction started by the first call to <code>begin()</code> . <code>begin()</code> refers to internal library calls not actual PHP user API calls.	Since 1.2.0.
<code>gtid_implicit</code>	Number of failed SQL injections when implicit commit is detected as part of the plugins client-side <code>global transaction id emulation</code> . Implicit commit happens, for example, when autocommit has been turned off, a query is executed and autocommit is enabled again. In that case, the statement will be committed by the server and SQL to maintain is injected before the autocommit is re-enabled.	Since 1.2.0.
<code>transient_error</code>	How often an operation has been retried when a transient error was detected. See also, <code>transient_error</code> plugin configuration file setting.	Since 1.6.0.
<code>fabric_successful</code>	Number of successful sharding <code>lookup_servers</code> remote procedure calls to MySQL Fabric. A call is considered successful if the plugin could reach MySQL Fabric and got any reply. The reply itself may or may not be understood by the plugin. Success refers to the network transport only. If the reply was not understood or indicates a valid error condition, <code>fabric_sharding_lookup_servers_xml_failure</code> gets incremented.	Since 1.6.0.
<code>fabric_failed</code>	Number of failed sharding <code>lookup_servers</code> remote procedure calls to MySQL Fabric. A remote procedure call is considered failed if there was a network error in connecting to, writing to or reading from MySQL Fabric.	Since 1.6.0.
<code>fabric_time</code>	Time spent connecting to, writing to and reading from MySQL Fabric during the <code>sharding.lookup_servers</code> remote procedure call. The value is aggregated for all calls. Time is measured in microseconds.	Since 1.6.0.
<code>fabric_bytes</code>	Total number of bytes received from MySQL Fabric in reply to <code>sharding.lookup_servers</code> calls.	Since 1.6.0.
<code>fabric_failure</code>	How often a reply from MySQL Fabric to <code>sharding.lookup_servers</code> calls was not understood. Please note, the current experimental implementation does not distinguish between valid errors returned and malformed replies.	Since 1.6.0.

Statistic	Description	Version
xa_begin	How many XA/distributed transactions have been started using mysqLnd_ms_xa_begin .	Since 1.6.0.
xa_commit	How many XA/distributed transactions have been successfully committed using mysqLnd_ms_xa_commit .	Since 1.6.0.
xa_commit_failure	How many XA/distributed transactions failed to commit during mysqLnd_ms_xa_commit .	Since 1.6.0.
xa_rollback	How many XA/distributed transactions have been successfully rolled back using mysqLnd_ms_xa_rollback . The figure does not include implicit rollbacks performed as a result of mysqLnd_ms_xa_commit failure.	Since 1.6.0.
xa_rollback_failure	How many XA/distributed transactions could not be rolled back. This includes failures of mysqLnd_ms_xa_rollback but also failed during rollback when closing a connection, if rollback_on_close is set. Please, see also xa_rollback_on_close below.	Since 1.6.0.
xa_participants	Total number of participants in any XA transaction started with mysqLnd_ms_xa_begin .	Since 1.6.0.
xa_rollback_implicit	How many XA transactions have been rolled back implicitly when a connection was close and rollback_on_close is set. Depending on your coding policies, this may hint a flaw in your code as you may prefer to explicitly clean up resources.	Since 1.6.0.
pool_master	Number of master servers (connections) in the internal connection pool.	Since 1.6.0.
pool_slave	Number of slave servers (connections) in the internal connection pool.	Since 1.6.0.
pool_master_inuse	Number of master servers (connections) from the internal connection pool which are currently used for picking a connection.	Since 1.6.0.
pool_slave_inuse	Number of slave servers (connections) from the internal connection pool which are currently used for picking a connection.	Since 1.6.0.
pool_update	How often the active connection list has been replaced and a new set of master and slave servers had been installed.	Since 1.6.0.
pool_master_flush	How often a master connection has been reused after being flushed from the active list.	Since 1.6.0.
pool_slave_flush	How often a slave connection has been reused after being flushed from the active list.	Since 1.6.0.

Examples

Example 6.306 [mysqLnd_ms_get_stats](#) example

```
<?php
printf("mysqLnd_ms.enable = %d\n", ini_get("mysqLnd_ms.enable"));
printf("mysqLnd_ms.collect_statistics = %d\n", ini_get("mysqLnd_ms.collect_statistics"));
var_dump(mysqLnd_ms_get_stats());
?>
```

The above example will output:

```
mysqLnd_ms.enable = 1
mysqLnd_ms.collect_statistics = 1
array(26) {
```



```
["use_slave"]=>
string(1) "0"
["use_master"]=>
string(1) "0"
["use_slave_guess"]=>
string(1) "0"
["use_master_guess"]=>
string(1) "0"
["use_slave_sql_hint"]=>
string(1) "0"
["use_master_sql_hint"]=>
string(1) "0"
["use_last_used_sql_hint"]=>
string(1) "0"
["use_slave_callback"]=>
string(1) "0"
["use_master_callback"]=>
string(1) "0"
["non_lazy_connections_slave_success"]=>
string(1) "0"
["non_lazy_connections_slave_failure"]=>
string(1) "0"
["non_lazy_connections_master_success"]=>
string(1) "0"
["non_lazy_connections_master_failure"]=>
string(1) "0"
["lazy_connections_slave_success"]=>
string(1) "0"
["lazy_connections_slave_failure"]=>
string(1) "0"
["lazy_connections_master_success"]=>
string(1) "0"
["lazy_connections_master_failure"]=>
string(1) "0"
["trx_autocommit_on"]=>
string(1) "0"
["trx_autocommit_off"]=>
string(1) "0"
["trx_master_forced"]=>
string(1) "0"
["gtid_autocommit_injections_success"]=>
string(1) "0"
["gtid_autocommit_injections_failure"]=>
string(1) "0"
["gtid_commit_injections_success"]=>
string(1) "0"
["gtid_commit_injections_failure"]=>
string(1) "0"
["gtid_implicit_commit_injections_success"]=>
string(1) "0"
["gtid_implicit_commit_injections_failure"]=>
string(1) "0"
["transient_error_retries"]=>
string(1) "0"
}
```

See Also

[Runtime configuration](#)
[mysqlnd_ms.collect_statistics](#)
[mysqlnd_ms.enable](#)
[Monitoring](#)

6.7.8.7 mysqlnd_ms_match_wild

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_ms_match_wild](#)

Finds whether a table name matches a wildcard pattern or not

Description

```
bool mysqlnd_ms_match_wild(
    string table_name,
    string wildcard);
```

Finds whether a table name matches a wildcard pattern or not.

This function is not of much practical relevance with PECL mysqlnd_ms 1.1.0 because the plugin does not support MySQL replication table filtering yet.

Parameters

table_name

The table name to check if it is matched by the wildcard.

wildcard

The wildcard pattern to check against the table name. The wildcard pattern supports the same placeholders as MySQL replication filters do.

MySQL replication filters can be configured by using the MySQL Server configuration options `--replicate-wild-do-table` and `--replicate-wild-do-db`. Please, consult the MySQL Reference Manual to learn more about this MySQL Server feature.

The supported placeholders are:

- `%` - zero or more literals
- `_` - one literal

Placeholders can be escaped using `\`.

Return Values

Returns `TRUE` if *table_name* is matched by *wildcard*. Otherwise, returns `FALSE`

Examples

Example 6.307 mysqlnd_ms_match_wild example

```
<?php
var_dump(mysqlnd_ms_match_wild("schema_name.table_name", "schema%"));
var_dump(mysqlnd_ms_match_wild("abc", "_"));
var_dump(mysqlnd_ms_match_wild("table1", "table_"));
var_dump(mysqlnd_ms_match_wild("asia_customers", "%customers"));
var_dump(mysqlnd_ms_match_wild("funny%table", "funny\\%table"));
var_dump(mysqlnd_ms_match_wild("funnytable", "funny%table"));
?>
```

The above example will output:

```
bool(true)
bool(false)
bool(true)
bool(true)
bool(true)
bool(true)
```

6.7.8.8 mysqlnd_ms_query_is_select

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_ms_query_is_select`

Find whether to send the query to the master, the slave or the last used MySQL server

Description

```
int mysqlnd_ms_query_is_select(
    string query);
```

Finds whether to send the query to the master, the slave or the last used MySQL server.

The plugins built-in read/write split mechanism will be used to analyze the query string to make a recommendation where to send the query. The built-in read/write split mechanism is very basic and simple. The plugin will recommend sending all queries to the MySQL replication master server but those which begin with `SELECT`, or begin with a SQL hint which enforces sending the query to a slave server. Due to the basic but fast algorithm the plugin may propose to run some read-only statements such as `SHOW TABLES` on the replication master.

Parameters

query Query string to test.

Return Values

A return value of `MYSQLND_MS_QUERY_USE_MASTER` indicates that the query should be send to the MySQL replication master server. The function returns a value of `MYSQLND_MS_QUERY_USE_SLAVE` if the query can be run on a slave because it is considered read-only. A value of `MYSQLND_MS_QUERY_USE_LAST_USED` is returned to recommend running the query on the last used server. This can either be a MySQL replication master server or a MySQL replication slave server.

If read write splitting has been disabled by setting `mysqlnd_ms.disable_rw_split`, the function will always return `MYSQLND_MS_QUERY_USE_MASTER` or `MYSQLND_MS_QUERY_USE_LAST_USED`.

Examples

Example 6.308 mysqlnd_ms_query_is_select example

```
<?php
function is_select($query)
{
    switch (mysqlnd_ms_query_is_select($query))
    {
        case MYSQLND_MS_QUERY_USE_MASTER:
            printf("'%s' should be run on the master.\n", $query);
            break;
        case MYSQLND_MS_QUERY_USE_SLAVE:
            printf("'%s' should be run on a slave.\n", $query);
            break;
        case MYSQLND_MS_QUERY_USE_LAST_USED:
            printf("'%s' should be run on the server that has run the previous query\n", $query);
            break;
        default:
            printf("No suggestion where to run the '%s', fallback to master recommended\n", $query);
            break;
    }
}
is_select("INSERT INTO test(id) VALUES (1)");
is_select("SELECT 1 FROM DUAL");
is_select("/*" . MYSQLND_MS_LAST_USED_SWITCH . "*/SELECT 2 FROM DUAL");
?>
```

The above example will output:

```
INSERT INTO test(id) VALUES (1) should be run on the master.  
SELECT 1 FROM DUAL should be run on a slave.  
/*ms=last_used*/SELECT 2 FROM DUAL should be run on the server that has run the previous query
```

See Also

Predefined Constants

[user](#) filter

Runtime configuration

[mysqlnd_ms.disable_rw_split](#)

[mysqlnd_ms.enable](#)

6.7.8.9 [mysqlnd_ms_set_qos](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_ms_set_qos](#)

Sets the quality of service needed from the cluster

Description

```
bool mysqlnd_ms_set_qos(  
    mixed connection,  
    int service_level,  
    int service_level_option,  
    mixed option_value);
```

Sets the quality of service needed from the cluster. A database cluster delivers a certain quality of service to the user depending on its architecture. A major aspect of the quality of service is the consistency level the cluster can offer. An asynchronous MySQL replication cluster defaults to eventual consistency for slave reads: a slave may serve stale data, current data, or it may have not the requested data at all, because it is not synchronous to the master. In a MySQL replication cluster, only master accesses can give strong consistency, which promises that all clients see each others changes.

PECL/mysqlnd_ms hides the complexity of choosing appropriate nodes to achieve a certain level of service from the cluster. The "Quality of Service" filter implements the necessary logic. The filter can either be configured in the plugins configuration file, or at runtime using [mysqlnd_ms_set_qos](#).

Similar results can be achieved with PECL mysqlnd_ms < 1.2.0, if using SQL hints to force the use of a certain type of node or using the [master_on_write](#) plugin configuration option. The first requires more code and causes more work on the application side. The latter is less refined than using the quality of service filter. Settings made through the function call can be reversed, as shown in the example below. The example temporarily switches to a higher service level (session consistency, read your writes) and returns back to the clusters default after it has performed all operations that require the better service. This way, read load on the master can be minimized compared to using [master_on_write](#), which would continue using the master after the first write.

Since 1.5.0 calls will fail when done in the middle of a transaction if [transaction stickiness](#) is enabled and transaction boundaries have been detected. properly.

Parameters

connection

A PECL/mysqlnd_ms connection handle to a MySQL server of the type [PDO_MYSQL](#), [mysqli](#) or [ext/mysql](#) for which a service level is to be set. The connection handle is obtained when opening

a connection with a host name that matches a `mysqLnd_ms` configuration file entry using any of the above three MySQL driver extensions.

service_level

The requested service level:

`MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL`,
`MYSQLND_MS_QOS_CONSISTENCY_SESSION` or
`MYSQLND_MS_QOS_CONSISTENCY_STRONG`.

service_level_option

An option to parameterize the requested service level. The option can either be `MYSQLND_MS_QOS_OPTION_GTID` or `MYSQLND_MS_QOS_OPTION_AGE`.

The option `MYSQLND_MS_QOS_OPTION_GTID` can be used to refine the service level `MYSQLND_MS_QOS_CONSISTENCY_SESSION`. It must be combined with a fourth function parameter, the *option_value*. The *option_value* shall be a global transaction ID obtained from `mysqLnd_ms_get_last_gtid`. If set, the plugin considers both master servers and asynchronous slaves for session consistency (read your writes). Otherwise, only masters are used to achieve session consistency. A slave is considered up-to-date and checked if it has already replicated the global transaction ID from *option_value*. Please note, searching appropriate slaves is an expensive and slow operation. Use the feature sparsely, if the master cannot handle the read load alone.

The `MYSQLND_MS_QOS_OPTION_AGE` option can be combined with the `MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL` service level, to filter out asynchronous slaves that lag more seconds behind the master than *option_value*. If set, the plugin will only consider slaves for reading if `SHOW SLAVE STATUS` reports `Slave_IO_Running=Yes`, `Slave_SQL_Running=Yes` and `Seconds_Behind_Master <= option_value`. Please note, searching appropriate slaves is an expensive and slow operation. Use the feature sparsely in version 1.2.0. Future versions may improve the algorithm used to identify candidates. Please, see the MySQL reference manual about the precision, accuracy and limitations of the MySQL administrative command `SHOW SLAVE STATUS`.

option_value

Parameter value for the service level option. See also the *service_level_option* parameter.

Return Values

Returns `TRUE` if the connections service level has been switched to the requested. Otherwise, returns `FALSE`

Notes

Note

`mysqLnd_ms_set_qos` requires PHP $\geq$ 5.4.0 and PECL `mysqLnd_ms` $\geq$ 1.2.0. Internally, it is using a `mysqLnd` library C functionality not available with PHP 5.3.

Please note, all MySQL 5.6 production versions do not provide clients with enough information to use GTIDs for enforcing session consistency. In the worst case, the plugin will choose the master only.

Examples

Example 6.309 `mysqlnd_ms_set_qos` example

```
<?php
/* Open mysqlnd_ms connection using mysqli, PDO_MySQL or mysql extension */
$mysqli = new mysqli("myapp", "username", "password", "database");
if (!$mysqli)
    /* Of course, your error handling is nicer... */
    die(sprintf("[%d] %s\n", mysqli_connect_errno(), mysqli_connect_error()));
/* Session consistency: read your writes */
$ret = mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_SESSION);
if (!$ret)
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
/* Will use master and return fresh data, client can see his last write */
if (!$res = $mysqli->query("SELECT item, price FROM orders WHERE order_id = 1"))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
/* Back to default: use of all slaves and masters permitted, stale data can happen */
if (!mysqlnd_ms_set_qos($mysqli, MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL))
    die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
?>
```

See Also

[mysqlnd_ms_get_last_gtid](#)
[Service level and consistency concept](#)
[Filter concept](#)

6.7.8.10 `mysqlnd_ms_set_user_pick_server`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_ms_set_user_pick_server`

Sets a callback for user-defined read/write splitting

Description

```
bool mysqlnd_ms_set_user_pick_server(
    string function);
```

Sets a callback for user-defined read/write splitting. The plugin will call the callback only if `pick[]=user` is the default rule for server picking in the relevant section of the plugins configuration file.

The plugins built-in read/write query split mechanism decisions can be overwritten in two ways. The easiest way is to prepend the query string with the SQL hints `MYSQLND_MS_MASTER_SWITCH`, `MYSQLND_MS_SLAVE_SWITCH` or `MYSQLND_MS_LAST_USED_SWITCH`. Using SQL hints one can control, for example, whether a query shall be send to the MySQL replication master server or one of the slave servers. By help of SQL hints it is not possible to pick a certain slave server for query execution.

Full control on server selection can be gained using a callback function. Use of a callback is recommended to expert users only because the callback has to cover all cases otherwise handled by the plugin.

The plugin will invoke the callback function for selecting a server from the lists of configured master and slave servers. The callback function inspects the query to run and picks a server for query execution by returning the hosts URI, as found in the master and slave list.

If the lazy connections are enabled and the callback chooses a slave server for which no connection has been established so far and establishing the connection to the slave fails, the plugin will return an error upon the next action on the failed connection, for example, when running a query. It is the

responsibility of the application developer to handle the error. For example, the application can re-run the query to trigger a new server selection and callback invocation. If so, the callback must make sure to select a different slave, or check slave availability, before returning to the plugin to prevent an endless loop.

Parameters

function

The function to be called. Class methods may also be invoked statically using this function by passing `array($classname, $methodname)` to this parameter. Additionally class methods of an object instance may be called by passing `array($objectinstance, $methodname)` to this parameter.

Return Values

Host to run the query on. The host URI is to be taken from the master and slave connection lists passed to the callback function. If callback returns a value neither found in the master nor in the slave connection lists the plugin will fallback to the second pick method configured via the `pick[]` setting in the plugin configuration file. If not second pick method is given, the plugin falls back to the build-in default pick method for server selection.

Notes

Note

`mysqlnd_ms_set_user_pick_server` is available with PECL mysqlnd_ms < 1.1.0. It has been replaced by the `user` filter. Please, check the [Change History](#) for upgrade notes.

Examples

Example 6.310 `mysqlnd_ms_set_user_pick_server` example

```
[myapp]
master[] = localhost
slave[] = 192.168.2.27:3306
slave[] = 192.168.78.136:3306
pick[] = user
```

```
<?php
function pick_server($connected, $query, $master, $slaves, $last_used)
{
    static $slave_idx = 0;
    static $num_slaves = NULL;
    if (is_null($num_slaves))
        $num_slaves = count($slaves);
    /* default: fallback to the plugins build-in logic */
    $ret = NULL;
    printf("User has connected to '%s'...\n", $connected);
    printf("... deciding where to run '%s'\n", $query);
    $where = mysqlnd_ms_query_is_select($query);
    switch ($where)
    {
        case MYSQLND_MS_QUERY_USE_MASTER:
            printf("... using master\n");
            $ret = $master[0];
            break;
        case MYSQLND_MS_QUERY_USE_SLAVE:
            /* SELECT or SQL hint for using slave */
            if (strpos($query, "FROM table_on_slave_a_only"))
            {

```

```
/* a table which is only on the first configured slave */
printf(... access to table available only on slave A detected\n");
$ret = $slaves[0];
}
else
{
/* round robin */
printf(... some read-only query for a slave\n");
$ret = $slaves[$slave_idx++ % $num_slaves];
}
break;
case MYSQLND_MS_QUERY_LAST_USED:
printf(... using last used server\n");
$ret = $last_used;
break;
}
printf(... ret = '%s'\n", $ret);
return $ret;
}
mysqlnd_ms_set_user_pick_server("pick_server");
$mysqli = new mysqli("myapp", "root", "root", "test");
if (!($res = $mysqli->query("SELECT 1 FROM DUAL")))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();
if (!($res = $mysqli->query("SELECT 2 FROM DUAL")))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();
if (!($res = $mysqli->query("SELECT * FROM table_on_slave_a_only")))
    printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
else
    $res->close();
$mysqli->close();
?>
```

The above example will output:

```
User has connected to 'myapp'...
... deciding where to run 'SELECT 1 FROM DUAL'
... some read-only query for a slave
... ret = 'tcp://192.168.2.27:3306'
User has connected to 'myapp'...
... deciding where to run 'SELECT 2 FROM DUAL'
... some read-only query for a slave
... ret = 'tcp://192.168.78.136:3306'
User has connected to 'myapp'...
... deciding where to run 'SELECT * FROM table_on_slave_a_only'
... access to table available only on slave A detected
... ret = 'tcp://192.168.2.27:3306'
```

See Also

[mysqlnd_ms_query_is_select](#)

[Filter concept](#)

[user filter](#)

6.7.8.11 [mysqlnd_ms_xa_begin](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_ms_xa_begin](#)

Starts a distributed/XA transaction among MySQL servers

Description

```
int mysqlnd_ms_xa_begin(
    mixed connection,
    string gtrid,
    int timeout);
```

Starts a XA transaction among MySQL servers. PECL/mysqlnd_ms acts as a transaction coordinator the distributed transaction.

Once a global transaction has been started, the plugin injects appropriate **XA BEGIN** SQL statements on all MySQL servers used in the following. The global transaction is either ended by calling `mysqlnd_ms_xa_commit`, `mysqlnd_ms_xa_rollback` or by an implicit rollback in case of an error.

During a global transaction, the plugin tracks all server switches, for example, when switching from one MySQL shard to another MySQL shard. Immediately before a query is run on a server that has not been participating in the global transaction yet, **XA BEGIN** is executed on the server. From a users perspective the injection happens during a call to a query execution function such as `mysqli_query`. Should the injection fail an error is reported to the caller of the query execution function. The failing server does not become a participant in the global transaction. The user may retry executing a query on the server and hereby retry injecting **XA BEGIN**, abort the global transaction because not all required servers can participate, or ignore and continue the global without the failed server.

Reasons to fail executing **XA BEGIN** include but are not limited to a server being unreachable or the server having an open, concurrent XA transaction using the same xid.

Please note, global and local transactions are mutually exclusive. You cannot start a XA transaction when you have a local transaction open. The local transaction must be ended first. The plugin tries to detect this conflict as early as possible. It monitors API calls for controlling local transactions to learn about the current state. However, if using SQL statements for local transactions such as **BEGIN**, the plugin may not know the current state and the conflict is not detected before **XA BEGIN** is injected and executed.

The use of other XA resources but MySQL servers is not supported by the function. To carry out a global transaction among, for example, a MySQL server and another vendors database system, you should issue the systems SQL commands yourself.

Experimental

The feature is currently under development. There may be issues and/or feature limitations. Do not use in production environments.

Parameters

connection

A MySQL connection handle obtained from any of the connect functions of the `mysqli`, `mysql` or `PDO_MYSQL` extensions.

gtrid

Global transaction identifier (gtrid). The gtrid is a binary string up to 64 bytes long. Please note, depending on your character set settings, 64 characters may require more than 64 bytes to store.

In accordance with the MySQL SQL syntax, XA transactions use identifiers made of three parts. An xid consists of a global transaction identifier (gtrid), a branch qualifier (bqual) and a format identifier (formatID). Only the global transaction identifier can and needs to be set.

The branch qualifier and format identifier are set automatically. The details should be considered implementation dependent, which may change without prior notice. In version 1.6 the branch qualifier is consecutive number which is incremented whenever a participant joins the global transaction.

timeout

Timeout in seconds. The default value is 60 seconds.

The timeout is a hint to the garbage collection. If a transaction is recorded to take longer than expected, the garbage collection begins checking the transactions status.

Setting a low value may make the garbage collection check the progress too often. Please note, checking the status of a global transaction may involve connecting to all recorded participants and possibly issuing queries on the servers.

Return Values

Returns [TRUE](#) if there is no open local or global transaction and a new global transaction can be started. Otherwise, returns [FALSE](#)

See Also

[Quickstart XA/Distributed transactions](#)
[Runtime configuration](#)
[mysqlnd_ms_get_stats](#)

6.7.8.12 mysqlnd_ms_xa_commit

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_ms_xa_commit](#)

Commits a distributed/XA transaction among MySQL servers

Description

```
int mysqlnd_ms_xa_commit(
    mixed connection,
    string gtrid);
```

Commits a global transaction among MySQL servers started by [mysqlnd_ms_xa_begin](#).

If any of the global transaction participants fails to commit an implicit rollback is performed. It may happen that not all cases can be handled during the rollback. For example, no attempts will be made to reconnect to a participant after the connection to the participant has been lost. Solving cases that cannot easily be rolled back is left to the garbage collection.

Experimental

The feature is currently under development. There may be issues and/or feature limitations. Do not use in production environments.

Parameters

connection

A MySQL connection handle obtained from any of the connect functions of the [mysqli](#), [mysql](#) or [PDO_MYSQL](#) extensions.

gtrid

Global transaction identifier (gtrid).

Return Values

Returns [TRUE](#) if the global transaction has been committed. Otherwise, returns [FALSE](#)

See Also

[Quickstart XA/Distributed transactions](#)
[Runtime configuration](#)
[mysqlnd_ms_get_stats](#)

6.7.8.13 `mysqlnd_ms_xa_gc`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_ms_xa_gc`

Garbage collects unfinished XA transactions after severe errors

Description

```
int mysqlnd_ms_xa_gc(
    mixed connection,
    string gtrid,
    boolean ignore_max_retries);
```

Garbage collects unfinished XA transactions.

The XA protocol is a blocking protocol. There exist cases when servers participating in a global transaction cannot make progress when the transaction coordinator crashes or disconnects. In such a case, the MySQL servers keep waiting for instructions to finish the XA transaction in question. Because transactions occupy resources, transactions should always be terminated properly.

Garbage collection requires configuring a state store to track global transactions. Should a PHP client crash in the middle of a transaction and a new PHP client be started, then the built-in garbage collection can learn about the aborted global transaction and terminate it. If you do not configure a state store, the garbage collection cannot perform any cleanup tasks.

The state store should be crash-safe and be highly available to survive its own crash. Currently, only MySQL is supported as a state store.

Garbage collection can also be performed automatically in the background. See the plugin configuration directive [garbage_collection](#) for details.

Experimental

The feature is currently under development. There may be issues and/or feature limitations. Do not use in production environments.

Parameters

<i>connection</i>	A MySQL connection handle obtained from any of the connect functions of the mysqli , mysql or PDO_MYSQL extensions.
<i>gtrid</i>	Global transaction identifier (gtrid). If given, the garbage collection considers the transaction only. Otherwise, the state store is scanned for any unfinished transaction.
<i>ignore_max_retries</i>	Whether to ignore the plugin configuration max_retries setting. If garbage collection continuously fails and the max_retries limit is reached prior to finishing the failed global transaction, you can attempt further runs prior to investigating the cause and solving the issue manually by issuing appropriate SQL statements on the participants. Setting the parameter has the same effect as temporarily setting max_retries = 0.

Return Values

Returns [TRUE](#) if garbage collection was successful. Otherwise, returns [FALSE](#)

See Also

[Quickstart XA/Distributed transactions](#)
[Runtime configuration](#)
[State store configuration](#)
[mysqlnd_ms_get_stats](#)

6.7.8.14 `mysqlnd_ms_xa_rollback`

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_ms_xa_rollback`

Rolls back a distributed/XA transaction among MySQL servers

Description

```
int mysqlnd_ms_xa_rollback(  
    mixed connection,  
    string gtrid);
```

Rolls back a global transaction among MySQL servers started by `mysqlnd_ms_xa_begin`.

If any of the global transaction participants fails to rollback the situation is left to be solved by the garbage collection.

Experimental

The feature is currently under development. There may be issues and/or feature limitations. Do not use in production environments.

Parameters

connection A MySQL connection handle obtained from any of the connect functions of the `mysqli`, `mysql` or `PDO_MYSQL` extensions.

gtrid Global transaction identifier (gtrid).

Return Values

Returns `TRUE` if the global transaction has been rolled back. Otherwise, returns `FALSE`

See Also

[Quickstart XA/Distributed transactions](#)
[Runtime configuration](#)
[mysqlnd_ms_get_stats](#)

6.7.9 Change History

Copyright 1997-2014 the PHP Documentation Group.

This change history is a high level summary of selected changes that may impact applications and/or break backwards compatibility.

See also the [CHANGES](#) file in the source distribution for a complete list of changes.

6.7.9.1 PECL/mysqlnd_ms 1.6 series

Copyright 1997-2014 the PHP Documentation Group.

1.6.0-alpha

- Release date: TBD
- Motto/theme: Maintenance and initial MySQL Fabric support

Note

This is the current development series. All features are at an early stage. Changes may happen at any time without prior notice. Please, do not use this version in production environments.

■ The documentation may not reflect all changes yet.

Bug fixes

- Won't fix: #66616 R/W split fails: QOS with `mysqlnd_get_last_gtid` with built-in MySQL GTID

This is not a bug in the plugins implementation but a server side feature limitation not considered and documented before. MySQL 5.6 built-in GTIDs cannot be used to ensure session consistency when reading from slaves in all cases. In the worst case the plugin will not consider using the slaves and fallback to using the master. There will be no wrong results but no benefit from doing GTID checks either.

- Fixed #66064 - Random once load balancer ignoring weights

Due to a config parsing bug random load balancing has ignored node weights if, and only if, the sticky flag was set (random once).

- Fixed #65496 - Wrong check for slave delay

The quality of service filter has erroneously ignored slaves that lag for zero (0) seconds if a any maximum lag had been set. Although a slave was not lagging behind, it was excluded from the load balancing list if a maximum age was set by the QoS filter. This was due to using the wrong comparison operator in the source of the filter.

- Fixed #65408 - Compile failure with `-Werror=format-security`

Feature changes

- Introduced an internal connection pool. When using Fabric and switching from shard group A to shard group B, we are replacing the entire list of masters and slaves. This troubles the connections state alignment logic and some filters. Some filters cache information on the master and slave lists. The new internal connection pool abstraction allows us to inform the filters of changes, hence they can update their caches.

Later on, the pool can also be used to reduce connection overhead. Assume you are switching from a shard group to another and back again. Whenever the switch is done, the pool's active server (and connection) lists are replaced. However, no longer used connections are not necessarily closed immediately but can be kept in the pool for later reuse.

Please note, the connection pool is internal at this point. There are some new statistics to monitor it. However, you cannot yet configure pool size or behaviour.

- Added a basic distributed transaction abstraction. XA transactions can be supported ever since using standard SQL calls. This is inconvenient as XA participants must be managed manually. PECL/mysqlnd_ms introduces API calls to control XA transaction among MySQL servers. When using the new functions, PECL/mysqlnd_ms acts as a transaction coordinator. After starting a distributed transaction, the plugin tracks all servers involved until the transaction is ended and issues appropriate SQL statements on the XA participants.

This is useful, for example, when using Fabric and sharding. When using Fabric the actual shard servers involved in a business transaction may not be known in advance. Thus, manually controlling a transaction that spawns multiple shards becomes difficult. Please, be warned about [current limitations](#).

- Introduced automatic retry loop for [transient errors](#) and [corresponding statistic](#) to count the number of implicit retries. Some distributed database clusters use transient errors to hint a client to retry its operation in a bit. Most often, the client is then supposed to halt execution (sleep) for a short moment before retrying the desired operation. Immediately failing over to another node is not necessary in response to the error. Instead, a retry loop can be performed. Common situation when using MySQL Cluster.

- Introduced automatic retry loop for [transient errors](#) and [corresponding statistic](#) to count the number of implicit retries. Some distributed database clusters use transient errors to hint a client to retry its operation in a bit. Most often, the client is then supposed to halt execution (sleep) for a short moment before retrying the desired operation. Immediately failing over to another node is not necessary in response to the error. Instead, a retry loop can be performed. Common situation when using MySQL Cluster.
- Introduced [most basic support](#) for the MySQL Fabric High Availability and sharding framework.

Please, consider this pre-alpha quality. Both the server side framework and the client side code is supposed to work flawless considering the MySQL Fabric quickstart examples only. However, testing has not been performed to the level of prior plugin alpha releases. Either sides are moving targets, API changes may happen at any time without prior warning.

As this is work in progress, the manual may not yet reflect allow feature limitations and known bugs.

- New [statistics](#) to monitor the Fabric XML RPC call [sharding.lookup_servers](#):
[fabric_sharding_lookup_servers_success](#),
[fabric_sharding_lookup_servers_failure](#),
[fabric_sharding_lookup_servers_time_total](#),
[fabric_sharding_lookup_servers_bytes_total](#),
[fabric_sharding_lookup_servers_xml_failure](#).
- New functions related to MySQL Fabric: [mysqlnd_ms_fabric_select_shard](#),
[mysqlnd_ms_fabric_select_global](#), [mysqlnd_ms_dump_servers](#).

6.7.9.2 PECL/mysqlnd_ms 1.5 series

[Copyright 1997-2014 the PHP Documentation Group.](#)

1.5.1-stable

- Release date: 06/2013
- Motto/theme: Sharding support, improved transaction support

Note

This is the current stable series. Use this version in production environments.

The documentation is complete.

1.5.0-alpha

- Release date: 03/2013
- Motto/theme: Sharding support, improved transaction support

Bug fixes

- Fixed #60605 PHP segmentation fault when [mysqlnd_ms](#) is enabled.
- Setting transaction stickiness disables all load balancing, including automatic failover, for the duration of a transaction. So far connection switches could have happened in the middle of a transaction in multi-master configurations and during automatic failover although transaction monitoring had detected transaction boundaries properly.
- BC break and bug fix. SQL hints enforcing the use of a specific kind of server ([MYSQLND_MS_MASTER_SWITCH](#), [MYSQLND_MS_SLAVE_SWITCH](#), [MYSQLND_MS_LAST_USED_SWITCH](#)) are ignored for the duration of a transaction of transaction stickiness is enabled and transaction boundaries have been detected properly.

This is a change in behaviour. However, it is also a bug fix and a step to align behaviour. If, in previous versions, transaction stickiness, one of the above listed SQL hints and the quality of service filtering was combined it could happen that the SQL hints got ignored. In some cases the SQL hints did work, in other cases they did not. The new behaviour is more consistent. SQL hints will always be ignored for the duration of a transaction, if [transaction stickiness](#) is enabled.

Please note, transaction boundary detection continues to be based on API call monitoring. SQL commands controlling transactions are not monitored.

- BC break and bug fix. Calls to [mysqlnd_ms_set_qos](#) will fail when done in the middle of a transaction if [transaction stickiness](#) is enabled. Connection switches are not allowed for the duration of a transaction. Changing the quality of service likely results in a different set of servers qualifying for query execution, possibly making it necessary to switch connections. Thus, the call is not allowed in during an active transaction. The quality of server can, however, be changed in between transactions.

Feature changes

- Introduced the [node_group](#) filter. The filter lets you organize servers (master and slaves) into groups. Queries can be directed to a certain group of servers by prefixing the query statement with a SQL hint/comment that contains the groups configured name. Grouping can be used for partitioning and sharding, and also to optimize for local caching. In the case of sharding, a group name can be thought of like a shard key. All queries for a given shard key will be executed on the configured shard. Note: both the client and server must support sharding for sharding to function with [mysqlnd_ms](#).

- Extended configuration file validation during PHP startup (RINIT). An [E_WARNING](#) level error will be thrown if the configuration file can not be read (permissions), is empty, or the file (JSON) could not be parsed. Warnings may appear in log files, which depending on how PHP is configured.

Distributions that aim to provide a pre-configured setup, including a configuration file stub, are asked to put `{}` into the configuration file to prevent this warning about an invalid configuration file.

Further configuration file validation is done when parsing sections upon opening a connection. Please, note that there may still be situations when an invalid plugin configuration file does not lead to proper error messages but a failure to connect.

- As of PHP 5.5.0, improved support for transaction boundaries detection was added for [mysqli](#). The [mysqli](#) extension has been modified to use the new C API calls of the [mysqlnd](#) library to begin, commit, and rollback a transaction or savepoint. If [trx_stickiness](#) is used to enable transaction aware load balancing, the [mysqli_begin](#), [mysqli_commit](#) and [mysqli_rollback](#) functions will now be monitored by the plugin, to go along with the [mysqli_autocommit](#) function that was already supported. All SQL features to control transactions are also available through the improved [mysqli](#) transaction control related functions. This means that it is not required to issue SQL statements instead of using API calls. Applications using the appropriate API calls can be load balanced by PECL/mysqlnd_ms in a completely transaction-aware way.

Please note, [PDO_MySQL](#) has not been updated yet to utilize the new [mysqlnd](#) API calls. Thus, transaction boundary detection with [PDO_MySQL](#) continues to be limited to the monitoring by passing in [PDO::ATTR_AUTOCOMMIT](#) to [PDO::setAttribute](#).

- Introduced [trx_stickiness=on](#). This [trx_stickiness](#) option differs from [trx_stickiness=master](#) as it tries to execute a read-only transaction on a slave, if quality of service (consistency level) allows the use of a slave. Read-only transactions were introduced in MySQL 5.6, and they offer performance gains.
- Query cache support is considered beta if used with the [mysqli](#) API. It should work fine with primary copy based clusters. For all other APIs, this feature continues to be called experimental.
- The code examples in the [mysqlnd_ms](#) source were updated.

6.7.9.3 PECL/mysqlnd_ms 1.4 series

Copyright 1997-2014 the PHP Documentation Group.

1.4.2-stable

- Release date: 08/2012
- Motto/theme: Tweaking based on user feedback

1.4.1-beta

- Release date: 08/2012
- Motto/theme: Tweaking based on user feedback

Bug fixes

- Fixed build with PHP 5.5

1.4.0-alpha

- Release date: 07/2012
- Motto/theme: Tweaking based on user feedback

Feature changes

- BC break: Renamed plugin configuration setting `ini_file` to `config_file`. In early versions the plugin configuration file used ini style. Back then the configuration setting was named accordingly. It has now been renamed to reflect the newer file format and to distinguish it from PHP's own ini file (configuration directives file).
- Introduced new default charset setting `server_charset` to allow proper escaping before a connection is opened. This is most useful when using lazy connections, which are a default.
- Introduced `wait_for_gtid_timeout` setting to throttle slave reads that need session consistency. If global transaction identifier are used and the service level is set to session consistency, the plugin tries to find up-to-date slaves. The slave status check is done by a SQL statement. If nothing else is set, the slave status is checked only one can the search for more up-to-date slaves continues immediately thereafter. Setting `wait_for_gtid_timeout` instructs the plugin to poll a slaves status for `wait_for_gtid_timeout` seconds if the first execution of the SQL statement has shown that the slave is not up-to-date yet. The poll will be done once per second. This way, the plugin will wait for slaves to catch up and throttle the client.
- New failover strategy `loop_before_master`. By default the plugin does no failover. It is possible to enable automatic failover if a connection attempt fails. Upto version 1.3 only `master` strategy existed to failover to a master if a slave connection fails. `loop_before_master` is similar but tries all other slaves before attempting to connect to the master if a slave connection fails.

The number of attempts can be limited using the `max_retries` option. Failed hosts can be remembered and skipped in load balancing for the rest of the web request. `max_retries` and `remember_failed` are considered experimental although decent stability is given. Syntax and semantics may change in the future without prior notice.

6.7.9.4 PECL/mysqlnd_ms 1.3 series

Copyright 1997-2014 the PHP Documentation Group.

1.3.2-stable

- Release date: 04/2012
- Motto/theme: see 1.3.0-alpha

Bug fixes

- Fixed problem with multi-master where although in a transaction the queries to the master weren't sticky and were spread all over the masters (RR). Still not sticky for Random. Random_once is not affected.

1.3.1-beta

- Release date: 04/2012
- Motto/theme: see 1.3.0-alpha

Bug fixes

- Fixed problem with building together with QC.

1.3.0-alpha

- Release date: 04/2012
- Motto/theme: Query caching through quality-of-service concept

The 1.3 series aims to improve the performance of applications and the overall load of an asynchronous MySQL cluster, for example, a MySQL cluster using MySQL Replication. This is done by transparently replacing a slave access with a local cache access, if the application allows it by setting an appropriate quality of service flag. When using MySQL replication a slave can serve stale data. An application using MySQL replication must continue to work correctly with stale data. Given that the application is know to work correctly with stale data, the slave access can transparently be replace with a local cache access.

[PECL/mysqlnd_qc](#) serves as a cache backend. PECL/mysqlnd_qc supports use of various storage locations, among others main memory, [APC](#) and [MEMCACHE](#).

Feature changes

- Added cache option to quality-of-service (QoS) filter.
 - New configure option `enable-mysqlnd-ms-cache-support`
 - New constant `MYSQLND_MS_HAVE_CACHE_SUPPORT`.
 - New constant `MYSQLND_MS_QOS_OPTION_CACHE` to be used with `mysqlnd_ms_set_qos`.
- Support for built-in global transaction identifier feature of MySQL 5.6.5-m8 or newer.

6.7.9.5 PECL/mysqlnd_ms 1.2 series

[Copyright 1997-2014 the PHP Documentation Group.](#)

1.2.1-beta

- Release date: 01/2012
- Motto/theme: see 1.2.0-alpha

Minor test changes.**1.2.0-alpha**

- Release date: 11/2011
- Motto/theme: Global Transaction ID injection and quality-of-service concept

In version 1.2 the focus continues to be on supporting MySQL database clusters with asynchronous replication. The plugin tries to make using the cluster introducing a quality-of-service filter which applications can use to define what service quality they need from the cluster. Service levels provided

are eventual consistency with optional maximum age/slave lag, session consistency and strong consistency.

Additionally the plugin can do client-side global transaction id injection to make manual master failover easier.

Feature changes

- Introduced quality-of-service (QoS) filter. Service levels provided by QoS filter:
 - eventual consistency, optional option slave lag
 - session consistency, optional option GTID
 - strong consistency
- Added the `mysqlnd_ms_set_qos` function to set the required connection quality at runtime. The new constants related to `mysqlnd_ms_set_qos` are:
 - `MYSQLND_MS_QOS_CONSISTENCY_STRONG`
 - `MYSQLND_MS_QOS_CONSISTENCY_SESSION`
 - `MYSQLND_MS_QOS_CONSISTENCY_EVENTUAL`
 - `MYSQLND_MS_QOS_OPTION_GTID`
 - `MYSQLND_MS_QOS_OPTION_AGE`
- Added client-side global transaction id injection (GTID).
- New statistics related to GTID:
 - `gtid_autocommit_injections_success`
 - `gtid_autocommit_injections_failure`
 - `gtid_commit_injections_success`
 - `gtid_commit_injections_failure`
 - `gtid_implicit_commit_injections_success`
 - `gtid_implicit_commit_injections_failure`
- Added `mysqlnd_ms_get_last_gtid` to fetch the last global transaction id.
- Enabled support for multi master without slaves.

6.7.9.6 PECL/mysqlnd_ms 1.1 series

Copyright 1997-2014 the PHP Documentation Group.

1.1.0

- Release date: 09/2011
- Motto/theme: Cover replication basics with production quality

The 1.1 and 1.0 series expose a similar feature set. Internally, the 1.1 series has been refactored to plan for future feature additions. A new configuration file format has been introduced, and limitations have been lifted. And the code quality and quality assurance has been improved.

Feature changes

- Added the (chainable) [filter concept](#):
 - BC break: [mysqlnd_ms_set_user_pick_server](#) has been removed. The http://svn.php.net/viewvc/pecl/mysqlnd_ms/trunk/user filter has been introduced to replace it. The filter offers similar functionality, but see below for an explanation of the differences.
- New powerful JSON based configuration syntax.
- [Lazy connections improved](#): security relevant, and state changing commands are covered.
- Support for (native) prepared statements.
- New statistics: [use_master_guess](#), [use_slave_guess](#).
 - BC break: Semantics of statistics changed for [use_slave](#), [use_master](#). Future changes are likely. Please see, [mysqlnd_ms_get_stats](#).
- List of broadcasted messages extended by [ssl_set](#).
- Library calls now monitored to remember settings for lazy connections: [change_user](#), [select_db](#), [set_charset](#), [set_autocommit](#).
- Introduced [mysqlnd_ms.disable_rw_split](#). The configuration setting allows using the load balancing and lazy connection functionality independently of read write splitting.

Bug fixes

- Fixed PECL #22724 - Server switching ([mysqlnd_ms_query_is_select\(\)](#) case sensitive)
- Fixed PECL #22784 - Using [mysql_connect](#) and [mysql_select_db](#) did not work
- Fixed PECL #59982 - Unusable extension with `--enable-mysqlnd-ms-table-filter`. Use of the option is NOT supported. You must not use it. Added note to m4.
- Fixed Bug #60119 - `host="localhost"` lost in [mysqlnd_ms_get_last_used_connection\(\)](#)

The [mysqlnd_ms_set_user_pick_server](#) function was removed, and replaced in favor of a new [user](#) filter. You can no longer set a callback function using [mysqlnd_ms_set_user_pick_server](#) at runtime, but instead have to configure it in the plugins configuration file. The [user](#) filter will pass the same arguments to the callback as before. Therefore, you can continue to use the same procedural function as a callback. It is no longer possible to use static class methods, or class methods of an object instance, as a callback. Doing so will cause the function executing a statement handled by the plugin to emit an [E_RECOVERABLE_ERROR](#) level error, which might look like: `"(mysqlnd_ms) Specified callback (picker) is not a valid callback."` Note: this may halt your application.

6.7.9.7 PECL/mysqlnd_ms 1.0 series

[Copyright 1997-2014 the PHP Documentation Group.](#)

1.0.1-alpha

- Release date: 04/2011
- Motto/theme: bug fix release

1.0.0-alpha

- Release date: 04/2011
- Motto/theme: Cover replication basics to test user feedback

The first release of practical use. It features basic automatic read-write splitting, SQL hints to overrule automatic redirection, load balancing of slave requests, lazy connections, and optional, automatic use of the master after the first write.

The public feature set is close to that of the 1.1 release.

1.0.0-pre-alpha

- Release date: 09/2010
- Motto/theme: Proof of concept

Initial check-in. Essentially a demo of the [mysqlnd](#) plugin API.

6.8 Mysqlnd query result cache plugin

[Copyright 1997-2014 the PHP Documentation Group.](#)

The mysqlnd query result cache plugin adds easy to use client-side query caching to all PHP MySQL extensions using [mysqlnd](#).

As of version PHP 5.3.3 the MySQL native driver for PHP ([mysqlnd](#)) features an internal plugin C API. C plugins, such as the query cache plugin, can extend the functionality of [mysqlnd](#).

Mysqlnd plugins such as the query cache plugin operate transparent from a user perspective. The cache plugin supports all PHP applications and all PHP MySQL extensions ([mysqli](#), [mysql](#), [PDO_MYSQL](#)). It does not change existing APIs.

No significant application changes are required to cache a query. The cache has two operation modes. It will either cache all queries (not recommended) or only those queries marked with a certain SQL hint (recommended).

6.8.1 Key Features

[Copyright 1997-2014 the PHP Documentation Group.](#)

- Transparent and therefore easy to use
 - supports all PHP MySQL extensions
 - no API changes
 - very little application changes required
- Flexible invalidation strategy
 - Time-to-Live (TTL)
 - user-defined
- Storage with different scope and life-span
 - Default (Hash, process memory)
 - [APC](#)
 - MEMCACHE
 - sqlite
 - user-defined
- Built-in slam defense to prevent cache stampeding.

6.8.2 Limitations

[Copyright 1997-2014 the PHP Documentation Group.](#)

The current 1.0.1 release of PECL `mysqlnd_qc` does not support PHP 5.4. Version 1.1.0-alpha lifts this limitation.

Prepared statements and unbuffered queries are fully supported. Thus, the plugin is capable of caching all statements issued with `mysqli` or `PDO_MySQL`, which are the only two PHP MySQL APIs to offer prepared statement support.

6.8.3 On the name

Copyright 1997-2014 the PHP Documentation Group.

The shortcut `mysqlnd_qc` stands for `mysqlnd query cache plugin`. The name was chosen for a quick-and-dirty proof-of-concept. In the beginning the developers did not expect to continue using the code base. Sometimes PECL/`mysqlnd_qc` has also been called `client-side query result set cache`.

6.8.4 Quickstart and Examples

Copyright 1997-2014 the PHP Documentation Group.

The `mysqlnd` query cache plugin is easy to use. This quickstart will demo typical use-cases, and provide practical advice on getting started.

It is strongly recommended to read the reference sections in addition to the quickstart. It is safe to begin with the quickstart. However, before using the plugin in mission critical environments we urge you to read additionally the background information from the reference sections.

Most of the examples use the `mysqli` extension because it is the most feature complete PHP MySQL extension. However, the plugin can be used with any PHP MySQL extension that is using the `mysqlnd` library.

6.8.4.1 Architecture and Concepts

Copyright 1997-2014 the PHP Documentation Group.

The query cache plugin is implemented as a PHP extension. It is written in C and operates under the hood of PHP. During the startup of the PHP interpreter, it gets registered as a `mysqlnd` plugin to replace selected `mysqlnd` C methods. Hereby, it can change the behaviour of any PHP MySQL extension (`mysqli`, `PDO_MYSQL`, `mysql`) compiled to use the `mysqlnd` library without changing the extensions API. This makes the plugin compatible with each and every PHP MySQL application. Because existing APIs are not changed, it is almost transparent to use. Please, see the [mysqlnd plugin API description](#) for a discussion of the advantages of the plugin architecture and a comparison with proxy based solutions.

Transparent to use

At PHP run time PECL/`mysqlnd_qc` can proxy queries send from PHP (`mysqlnd`) to the MySQL server. It then inspects the statement string to find whether it shall cache its results. If so, result set is cached using a storage handler and further executions of the statement are served from the cache for a user-defined period. The Time to Live (TTL) of the cache entry can either be set globally or on a per statement basis.

A statement is either cached if the plugin is instructed to cache all statements globally using a or, if the query string starts with the SQL hint (`/*qc=on*/`). The plugin is capable of caching any query issued by calling appropriate API calls of any of the existing PHP MySQL extensions.

Flexible storage: various storage handler

Various storage handler are supported to offer different scopes for cache entries. Different scopes allow for different degrees in sharing cache entries among clients.

- `default` (built-in): process memory, scope: process, one or more web requests depending on PHP deployment model used

- [APC](#): shared memory, scope: single server, multiple web requests
- [SQLite](#): memory or file, scope: single server, multiple web requests
- [MEMCACHE](#): main memory, scope: single or multiple server, multiple web requests
- [user](#) (built-in): user-defined - any, scope: user-defined - any

Support for the [APC](#), [SQLite](#) and [MEMCACHE](#) storage handler has to be enabled at compile time. The [default](#) and [user](#) handler are built-in. It is possible to switch between compiled-in storage handlers on a per query basis at run time. However, it is recommended to pick one storage handler and use it for all cache entries.

Built-in slam defense to avoid overloading

To avoid overload situations the cache plugin has a built-in slam defense mechanism. If a popular cache entry expires many clients using the cache entry will try to refresh the cache entry. For the duration of the refresh many clients may access the database server concurrently. In the worst case, the database server becomes overloaded and it takes more and more time to refresh the cache entry, which in turn lets more and more clients try to refresh the cache entry. To prevent this from happening the plugin has a slam defense mechanism. If slam defense is enabled and the plugin detects an expired cache entry it extends the life time of the cache entry before it refreshes the cache entry. This way other concurrent accesses to the expired cache entry are still served from the cache for a certain time. The other concurrent accesses do not trigger a concurrent refresh. Ideally, the cache entry gets refreshed by the client which extended the cache entry's lifespan before other clients try to refresh the cache and potentially cause an overload situation.

Unique approach to caching

PECL/mysqlnd_qc has a unique approach to caching result sets that is superior to application based cache solutions. Application based solutions first fetch a result set into PHP variables. Then, the PHP variables are serialized for storage in a persistent cache, and then unserialized when fetching. The mysqlnd query cache stores the raw wire protocol data sent from MySQL to PHP in its cache and replays it, if still valid, on a cache hit. This way, it saves an extra serialization step for a cache put that all application based solutions have to do. It can store the raw wire protocol data in the cache without having to serialize into a PHP variable first and deserializing the PHP variable for storing in the cache again.

6.8.4.2 Setup

Copyright 1997-2014 the PHP Documentation Group.

The plugin is implemented as a PHP extension. See also the [installation instructions](#) to install the [PECL/mysqlnd_qc](#) extension.

Compile or configure the PHP MySQL extension ([mysqli](#), [PDO_MYSQL](#), [mysql](#)) that you plan to use with support for the [mysqlnd](#) library. PECL/mysqlnd_qc is a plugin for the mysqlnd library. To use the plugin with any of the existing PHP MySQL extensions (APIs), the extension has to use the mysqlnd library.

Then, load the extension into PHP and activate the plugin in the PHP configuration file using the PHP configuration directive named [mysqlnd_qc.enable_qc](#).

Example 6.311 Enabling the plugin (php.ini)

```
mysqlnd_qc.enable_qc=1
```

6.8.4.3 Caching queries

Copyright 1997-2014 the PHP Documentation Group.

There are four ways to trigger caching of a query.

- Use of SQL hints on a per query basis
- User supplied callbacks to decide on a per query basis, for example, using `mysqlnd_qc_is_select`
- `mysqlnd_set_cache_condition` for rule based automatic per query decisions
- `mysqlnd_qc.cache_by_default = 1` to cache all queries blindly

Use of SQL hints and `mysqlnd_qc.cache_by_default = 1` are explained below. Please, refer to the function reference on `mysqlnd_qc_is_select` for a description of using a callback and, `mysqlnd_qc_set_cache_condition` on how to set rules for automatic caching.

A SQL hint is a SQL standards compliant comment. As a SQL comment it is ignored by the database. A statement is considered eligible for caching if it either begins with the SQL hint enabling caching or it is a `SELECT` statement.

An individual query which shall be cached must begin with the SQL hint `/*qc=on*/`. It is recommended to use the PHP constant `MYSQLND_QC_ENABLE_SWITCH` instead of using the string value.

- not eligible for caching and not cached: `INSERT INTO test(id) VALUES (1)`
- not eligible for caching and not cached: `SHOW ENGINES`
- eligible for caching but uncached: `SELECT id FROM test`
- eligible for caching and cached: `/*qc=on*/SELECT id FROM test`

The examples `SELECT` statement string is prefixed with the `MYSQLND_QC_ENABLE_SWITCH` SQL hint to enable caching of the statement. The SQL hint must be given at the very beginning of the statement string to enable caching.

Example 6.312 Using the `MYSQLND_QC_ENABLE_SWITCH` SQL hint

```
mysqlnd_qc.enable_qc=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");
/* Will be cached because of the SQL hint */
$start = microtime(true);
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
printf("Total time uncached query: %.6fs\n", microtime(true) - $start);
/* Cache hit */
$start = microtime(true);
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
printf("Total time cached query: %.6fs\n", microtime(true) - $start);
?>
```

The above examples will output something similar to:

```

array(1) {
    ["id"]=>
    string(1) "1"
}
Total time uncached query: 0.000740s
array(1) {
    ["id"]=>
    string(1) "1"
}
Total time cached query: 0.000098s

```

If nothing else is configured, as it is the case in the quickstart example, the plugin will use the built-in `default` storage handler. The `default` storage handler uses process memory to hold a cache entry. Depending on the PHP deployment model, a PHP process may serve one or more web requests. Please, consult the web server manual for details. Details make no difference for the examples given in the quickstart.

The query cache plugin will cache all queries regardless if the query string begins with the SQL hint which enables caching or not, if the PHP configuration directive `mysqlnd_qc.cache_by_default` is set to `1`. The setting `mysqlnd_qc.cache_by_default` is evaluated by the core of the query cache plugins. Neither the built-in nor user-defined storage handler can overrule the setting.

The SQL hint `/*qc=off*/` can be used to disable caching of individual queries if `mysqlnd_qc.cache_by_default = 1`. It is recommended to use the PHP constant `MYSQLND_QC_DISABLE_SWITCH` instead of using the string value.

Example 6.313 Using the `MYSQLND_QC_DISABLE_SWITCH` SQL hint

```

mysqlnd_qc.enable_qc=1
mysqlnd_qc.cache_by_default=1

```

```

<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");
/* Will be cached although no SQL hint is present because of mysqlnd_qc.cache_by_default = 1 */
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
$mysqli->query("DELETE FROM test WHERE id = 1");
/* Cache hit - no automatic invalidation and still valid! */
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
/* Cache miss - query must not be cached because of the SQL hint */
$res = $mysqli->query("/*" . MYSQLND_QC_DISABLE_SWITCH . "*/SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
?>

```

The above examples will output:

```

array(1) {

```



```

["id"]=>
string(1) "1"
}
array(1) {
    ["id"]=>
    string(1) "1"
}
NULL

```

PECL/mysqlnd_qc forbids caching of statements for which at least one column from the statements result set shows no table name in its meta data by default. This is usually the case for columns originating from SQL functions such as `NOW()` or `LAST_INSERT_ID()`. The policy aims to prevent pitfalls if caching by default is used.

Example 6.314 Example showing which type of statements are not cached

```

mysqlnd_qc.enable_qc=1
mysqlnd_qc.cache_by_default=1

```

```

<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1)");
for ($i = 0; $i < 3; $i++) {
    $start = microtime(true);
    /* Note: statement will not be cached because of NOW() use */
    $res = $mysqli->query("SELECT id, NOW() AS _time FROM test");
    $row = $res->fetch_assoc();
    /* dump results */
    var_dump($row);
    printf("Total time: %.6fs\n", microtime(true) - $start);
    /* pause one second */
    sleep(1);
}
?>

```

The above examples will output something similar to:

```

array(2) {
    ["id"]=>
    string(1) "1"
    ["_time"]=>
    string(19) "2012-01-11 15:43:10"
}
Total time: 0.000540s
array(2) {
    ["id"]=>
    string(1) "1"
    ["_time"]=>
    string(19) "2012-01-11 15:43:11"
}
Total time: 0.000555s
array(2) {
    ["id"]=>
    string(1) "1"
    ["_time"]=>
    string(19) "2012-01-11 15:43:12"
}

```

```
}
Total time: 0.000549s
```

It is possible to enable caching for all statements including those which has columns in their result set for which MySQL reports no table, such as the statement from the example. Set `mysqlnd_qc.cache_no_table = 1` to enable caching of such statements. Please, note the difference in the measured times for the above and below examples.

Example 6.315 Enabling caching for all statements using the `mysqlnd_qc.cache_no_table` ini setting

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.cache_by_default=1
mysqlnd_qc.cache_no_table=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1)");
for ($i = 0; $i < 3; $i++) {
    $start = microtime(true);
    /* Note: statement will not be cached because of NOW() use */
    $res = $mysqli->query("SELECT id, NOW() AS _time FROM test");
    $row = $res->fetch_assoc();
    /* dump results */
    var_dump($row);
    printf("Total time: %.6fs\n", microtime(true) - $start);
    /* pause one second */
    sleep(1);
}
?>
```

The above examples will output something similar to:

```
array(2) {
    ["id"]=>
    string(1) "1"
    ["_time"]=>
    string(19) "2012-01-11 15:47:45"
}
Total time: 0.000546s
array(2) {
    ["id"]=>
    string(1) "1"
    ["_time"]=>
    string(19) "2012-01-11 15:47:45"
}
Total time: 0.000187s
array(2) {
    ["id"]=>
    string(1) "1"
    ["_time"]=>
    string(19) "2012-01-11 15:47:45"
}
Total time: 0.000167s
```

Note

Although `mysqlnd_qc.cache_no_table = 1` has been created for use with `mysqlnd_qc.cache_by_default = 1` it is bound it. The plugin will evaluate the `mysqlnd_qc.cache_no_table` whenever a query is to be cached, no matter whether caching has been enabled using a SQL hint or any other measure.

6.8.4.4 Setting the TTL

Copyright 1997-2014 the PHP Documentation Group.

The default invalidation strategy of the query cache plugin is Time to Live (TTL). The built-in storage handlers will use the default TTL defined by the PHP configuration value `mysqlnd_qc.ttl` unless the query string contains a hint for setting a different TTL. The TTL is specified in seconds. By default cache entries expire after 30 seconds

The example sets `mysqlnd_qc.ttl=3` to cache statements for three seconds by default. Every second it updates a database table record to hold the current time and executes a `SELECT` statement to fetch the record from the database. The `SELECT` statement is cached for three seconds because it is prefixed with the SQL hint enabling caching. The output verifies that the query results are taken from the cache for the duration of three seconds before they are refreshed.

Example 6.316 Setting the TTL with the `mysqlnd_qc.ttl` ini setting

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.ttl=3
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id VARCHAR(255))");
for ($i = 0; $i < 7; $i++) {
    /* update DB row */
    if (!$mysqli->query("DELETE FROM test") ||
        !$mysqli->query("INSERT INTO test(id) VALUES (NOW())"))
        /* Of course, a real-life script should do better error handling */
        die(sprintf("[%d] %s\n", $mysqli->errno, $mysqli->error));
    /* select latest row but cache results */
    $query = "/*" . MYSQLND_QC_ENABLE_SWITCH . "*/";
    $query .= "SELECT id AS _time FROM test";
    if (!($res = $mysqli->query($query)) ||
        !($row = $res->fetch_assoc()))
    {
        printf("[%d] %s\n", $mysqli->errno, $mysqli->error);
    }
    $res->free();
    printf("Wall time %s - DB row time %s\n", date("H:i:s"), $row['_time']);
    /* pause one second */
    sleep(1);
}
?>
```

The above examples will output something similar to:

```
Wall time 14:55:59 - DB row time 2012-01-11 14:55:59
```

```

Wall time 14:56:00 - DB row time 2012-01-11 14:55:59
Wall time 14:56:01 - DB row time 2012-01-11 14:55:59
Wall time 14:56:02 - DB row time 2012-01-11 14:56:02
Wall time 14:56:03 - DB row time 2012-01-11 14:56:02
Wall time 14:56:04 - DB row time 2012-01-11 14:56:02
Wall time 14:56:05 - DB row time 2012-01-11 14:56:05

```

As can be seen from the example, any **TTL** based cache can serve stale data. Cache entries are not automatically invalidated, if underlying data changes. Applications using the default **TTL** invalidation strategy must be able to work correctly with stale data.

A user-defined cache storage handler can implement any invalidation strategy to work around this limitation.

The default **TTL** can be overruled using the SQL hint `/*qc_tt=seconds*/`. The SQL hint must be appear immediately after the SQL hint which enables caching. It is recommended to use the PHP constant `MYSQLND_QC_TTL_SWITCH` instead of using the string value.

Example 6.317 Setting TTL with SQL hints

```

<?php
$start = microtime(true);
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");
printf("Default TTL\t: %d seconds\n", ini_get("mysqlnd_qc.ttl"));
/* Will be cached for 2 seconds */
$sql = sprintf("/*%s*/%s*/SELECT id FROM test WHERE id = 1", MYSQLND_QC_ENABLE_SWITCH, MYSQLND_QC_TTL_SWITCH);
$res = $mysqli->query($sql);
var_dump($res->fetch_assoc());
$res->free();
$mysqli->query("DELETE FROM test WHERE id = 1");
sleep(1);
/* Cache hit - no automatic invalidation and still valid! */
$res = $mysqli->query($sql);
var_dump($res->fetch_assoc());
$res->free();
sleep(2);
/* Cache miss - cache entry has expired */
$res = $mysqli->query($sql);
var_dump($res->fetch_assoc());
$res->free();
printf("Script runtime\t: %d seconds\n", microtime(true) - $start);
?>

```

The above examples will output something similar to:

```

Default TTL      : 30 seconds
array(1) {
  ["id"]=>
  string(1) "1"
}
array(1) {
  ["id"]=>
  string(1) "1"
}
NULL
Script runtime   : 3 seconds

```

6.8.4.5 Pattern based caching

Copyright 1997-2014 the PHP Documentation Group.

An application has three options for telling PECL/mysqlnd_qc whether a particular statement shall be used. The most basic approach is to cache all statements by setting `mysqlnd_qc.cache_by_default = 1`. This approach is often of little practical value. But it enables users to make a quick estimation about the maximum performance gains from caching. An application designed to use a cache may be able to prefix selected statements with the appropriate SQL hints. However, altering an applications source code may not always be possible or desired, for example, to avoid problems with software updates. Therefore, PECL/mysqlnd_qc allows setting a callback which decides if a query is to be cached.

The callback is installed with the `mysqlnd_qc_set_is_select` function. The callback is given the statement string of every statement inspected by the plugin. Then, the callback can decide whether to cache the function. The callback is supposed to return `FALSE` if the statement shall not be cached. A return value of `TRUE` makes the plugin try to add the statement into the cache. The cache entry will be given the default TTL (`mysqlnd_qc.ttl`). If the callback returns a numerical value it is used as the TTL instead of the global default.

Example 6.318 Setting a callback with `mysqlnd_qc_set_is_select`

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.collect_statistics=1
```

```
<?php
/* callback which decides if query is cached */
function is_select($query) {
    static $patterns = array(
        /* true - use default from mysqlnd_qc.ttl */
        "@SELECT\s+.*\s+FROM\s+test@ismU" => true,
        /* 3 - use TTL = 3 seconds */
        "@SELECT\s+.*\s+FROM\s+news@ismU" => 3
    );
    /* check if query does match pattern */
    foreach ($patterns as $pattern => $ttl) {
        if (preg_match($pattern, $query)) {
            printf("is_select(%45s): cache\n", $query);
            return $ttl;
        }
    }
    printf("is_select(%45s): do not cache\n", $query);
    return false;
}

/* install callback */
mysqlnd_qc_set_is_select("is_select");
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");
/* cache put */
$mysqli->query("SELECT id FROM test WHERE id = 1");
/* cache hit */
$mysqli->query("SELECT id FROM test WHERE id = 1");
/* cache put */
$mysqli->query("SELECT * FROM test");
$stats = mysqlnd_qc_get_core_stats();
printf("Cache put: %d\n", $stats['cache_put']);
printf("Cache hit: %d\n", $stats['cache_hit']);
?>
```

The above examples will output something similar to:

```
is_select(          DROP TABLE IF EXISTS test): do not cache
is_select(          CREATE TABLE test(id INT)): do not cache
is_select(  INSERT INTO test(id) VALUES (1), (2), (3)): do not cache
is_select(          SELECT id FROM test WHERE id = 1): cache
is_select(          SELECT id FROM test WHERE id = 1): cache
is_select(          SELECT * FROM test): cache
Cache put: 2
Cache hit: 1
```

The examples callback tests if a statement string matches a pattern. If this is the case, it either returns **TRUE** to cache the statement using the global default TTL or an alternative TTL.

To minimize application changes the callback can put into and registered in an auto prepend file.

6.8.4.6 Slam defense

Copyright 1997-2014 the PHP Documentation Group.

A badly designed cache can do more harm than good. In the worst case a cache can increase database server load instead of minimizing it. An overload situation can occur if a highly shared cache entry expires (cache stampeding).

Cache entries are shared and reused to a different degree depending on the storage used. The default storage handler stores cache entries in process memory. Thus, a cache entry can be reused for the life-span of a process. Other PHP processes cannot access it. If Memcache is used, a cache entry can be shared among multiple PHP processes and even among multiple machines, depending on the set up being used.

If a highly shared cache entry stored, for example, in Memcache expires, many clients gets a cache miss. Many client requests can no longer be served from the cache but try to run the underlying query on the database server. Until the cache entry is refreshed, more and more clients contact the database server. In the worst case, a total lost of service is the result.

The overload can be avoided using a storage handler which limits the reuse of cache entries to few clients. Then, at the average, its likely that only a limited number of clients will try to refresh a cache entry concurrently.

Additionally, the built-in slam defense mechanism can and should be used. If slam defense is activated an expired cache entry is given an extended life time. The first client getting a cache miss for the expired cache entry tries to refresh the cache entry within the extended life time. All other clients requesting the cache entry are temporarily served from the cache although the original **TTL** of the cache entry has expired. The other clients will not experience a cache miss before the extended life time is over.

Example 6.319 Enabling the slam defense mechanism

```
mysqlnd_qc.slam_defense=1
mysqlnd_qc.slam_defense_ttl=1
```

The slam defense mechanism is enabled with the PHP configuration directive `mysqlnd_qc.slam_defense`. The extended life time of a cache entry is set with `mysqlnd_qc.slam_defense_ttl`.

The function `mysqlnd_qc_get_core_stats` returns an array of statistics. The statistics `slam_stale_refresh` and `slam_stale_hit` are incremented if slam defense takes place.

It is not possible to give a one-fits-all recommendation on the slam defense configuration. Users are advised to monitor and test their setup and derive settings accordingly.

6.8.4.7 Finding cache candidates

Copyright 1997-2014 the PHP Documentation Group.

A statement should be considered for caching if it is executed often and has a long run time. Cache candidates are found by creating a list of statements sorted by the product of the number of executions multiplied by the statements run time. The function `mysqlnd_qc_get_query_trace_log` returns a query log which help with the task.

Collecting a query trace is a slow operation. Thus, it is disabled by default. The PHP configuration directive `mysqlnd_qc.collect_query_trace` is used to enable it. The functions trace contains one entry for every query issued before the function is called.

Example 6.320 Collecting a query trace

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.collect_query_trace=1
```

```
<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
/* dummy queries to fill the query trace */
for ($i = 0; $i < 2; $i++) {
    $res = $mysqli->query("SELECT 1 AS _one FROM DUAL");
    $res->free();
}
/* dump trace */
var_dump(mysqlnd_qc_get_query_trace_log());
?>
```

The above examples will output:

```
array(2) {
  [0]=>
    array(8) {
      ["query"]=>
        string(26) "SELECT 1 AS _one FROM DUAL"
      ["origin"]=>
        string(102) "#0 qc.php(7): mysqli->query('SELECT 1 AS _on...')
#1 {main}"
      ["run_time"]=>
        int(0)
      ["store_time"]=>
        int(25)
      ["eligible_for_caching"]=>
        bool(false)
      ["no_table"]=>
        bool(false)
      ["was_added"]=>
        bool(false)
      ["was_already_in_cache"]=>
        bool(false)
    }
  [1]=>
    array(8) {
      ["query"]=>
```

```

string(26) "SELECT 1 AS _one FROM DUAL"
["origin"]=>
string(102) "#0 qc.php(7): mysqli->query('SELECT 1 AS _on...')
#1 {main}
  ["run_time"]=>
  int(0)
  ["store_time"]=>
  int(8)
  ["eligible_for_caching"]=>
  bool(false)
  ["no_table"]=>
  bool(false)
  ["was_added"]=>
  bool(false)
  ["was_already_in_cache"]=>
  bool(false)
}
}

```

Assorted information is given in the trace. Among them timings and the origin of the query call. The origin property holds a code backtrace to identify the source of the query. The depth of the backtrace can be limited with the PHP configuration directive `mysqlnd_qc.query_trace_bt_depth`. The default depth is 3.

Example 6.321 Setting the backtrace depth with the `mysqlnd_qc.query_trace_bt_depth` ini setting

```

mysqlnd_qc.enable_qc=1
mysqlnd_qc.collect_query_trace=1

```

```

<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");
/* dummy queries to fill the query trace */
for ($i = 0; $i < 3; $i++) {
    $res = $mysqli->query("SELECT id FROM test WHERE id = " . $mysqli->real_escape_string($i));
    $res->free();
}
$trace = mysqlnd_qc_get_query_trace_log();
$summary = array();
foreach ($trace as $entry) {
    if (!isset($summary[$entry['query']])) {
        $summary[$entry['query']] = array(
            "executions" => 1,
            "time"       => $entry['run_time'] + $entry['store_time'],
        );
    } else {
        $summary[$entry['query']]['executions']++;
        $summary[$entry['query']]['time'] += $entry['run_time'] + $entry['store_time'];
    }
}
foreach ($summary as $query => $details) {
    printf("%45s: %5dms (%dx)\n",
        $query, $details['time'], $details['executions']);
}
?>

```

The above examples will output something similar to:


```

DROP TABLE IF EXISTS test:      0ms (1x)
CREATE TABLE test(id INT):      0ms (1x)
INSERT INTO test(id) VALUES (1), (2), (3): 0ms (1x)
SELECT id FROM test WHERE id = 0: 25ms (1x)
SELECT id FROM test WHERE id = 1: 10ms (1x)
SELECT id FROM test WHERE id = 2:  9ms (1x)

```

6.8.4.8 Measuring cache efficiency

Copyright 1997-2014 the PHP Documentation Group.

PECL/mysqlnd_qc offers three ways to measure the cache efficiency. The function `mysqlnd_qc_get_normalized_query_trace_log` returns statistics aggregated by the normalized query string, `mysqlnd_qc_get_cache_info` gives storage handler specific information which includes a list of all cached items, depending on the storage handler. Additionally, the core of PECL/mysqlnd_qc collects high-level summary statistics aggregated per PHP process. The high-level statistics are returned by `mysqlnd_qc_get_core_stats`.

The functions `mysqlnd_qc_get_normalized_query_trace_log` and `mysqlnd_qc_get_core_stats` will not collect data unless data collection has been enabled through their corresponding PHP configuration directives. Data collection is disabled by default for performance considerations. It is configurable with the `mysqlnd_qc.time_statistics` option, which determines if timing information should be collected. Collection of time statistics is enabled by default but only performed if data collection as such has been enabled. Recording time statistics causes extra system calls. In most cases, the benefit of the monitoring outweighs any potential performance penalty of the additional system calls.

Example 6.322 Collecting statistics data with the `mysqlnd_qc.time_statistics` ini setting

```

mysqlnd_qc.enable_qc=1
mysqlnd_qc.collect_statistics=1

```

```

<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");
/* dummy queries */
for ($i = 1; $i <= 4; $i++) {
    $query = sprintf("/*%s*/SELECT id FROM test WHERE id = %d", MYSQLND_QC_ENABLE_SWITCH, $i % 2);
    $res = $mysqli->query($query);

    $res->free();
}
var_dump(mysqlnd_qc_get_core_stats());
?>

```

The above examples will output something similar to:

```

array(26) {
    ["cache_hit"]=>
    string(1) "2"
    ["cache_miss"]=>
    string(1) "2"

```

```

["cache_put"]=>
string(1) "2"
["query_should_cache"]=>
string(1) "4"
["query_should_not_cache"]=>
string(1) "3"
["query_not_cached"]=>
string(1) "3"
["query_could_cache"]=>
string(1) "4"
["query_found_in_cache"]=>
string(1) "2"
["query_uncached_other"]=>
string(1) "0"
["query_uncached_no_table"]=>
string(1) "0"
["query_uncached_no_result"]=>
string(1) "0"
["query_uncached_use_result"]=>
string(1) "0"
["query_aggr_run_time_cache_hit"]=>
string(2) "28"
["query_aggr_run_time_cache_put"]=>
string(3) "900"
["query_aggr_run_time_total"]=>
string(3) "928"
["query_aggr_store_time_cache_hit"]=>
string(2) "14"
["query_aggr_store_time_cache_put"]=>
string(2) "40"
["query_aggr_store_time_total"]=>
string(2) "54"
["receive_bytes_recorded"]=>
string(3) "136"
["receive_bytes_replayed"]=>
string(3) "136"
["send_bytes_recorded"]=>
string(2) "84"
["send_bytes_replayed"]=>
string(2) "84"
["slam_stale_refresh"]=>
string(1) "0"
["slam_stale_hit"]=>
string(1) "0"
["request_counter"]=>
int(1)
["process_hash"]=>
int(1929695233)
}

```

For a quick overview, call [mysqlnd_qc_get_core_stats](#). It delivers cache usage, cache timing and traffic related statistics. Values are aggregated on a per process basis for all queries issued by any PHP MySQL API call.

Some storage handler, such as the default handler, can report cache entries, statistics related to the entries and meta data for the underlying query through the [mysqlnd_qc_get_cache_info](#) function. Please note, that the information returned depends on the storage handler. Values are aggregated on a per process basis.

Example 6.323 Example [mysqlnd_qc_get_cache_info](#) usage

```
mysqlnd_qc.enable_qc=1
```

```
<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");
/* dummy queries to fill the query trace */
for ($i = 1; $i <= 4; $i++) {
    $query = sprintf("/*%s*/SELECT id FROM test WHERE id = %d", MYSQLND_QC_ENABLE_SWITCH, $i % 2);
    $res = $mysqli->query($query);

    $res->free();
}
var_dump(mysqlind_qc_get_cache_info());
?>
```

The above examples will output something similar to:

```
array(4) {
  ["num_entries"]=>
  int(2)
  ["handler"]=>
  string(7) "default"
  ["handler_version"]=>
  string(5) "1.0.0"
  ["data"]=>
  array(2) {
    ["Localhost via UNIX socket
3306
root
test|/*qc=on*/SELECT id FROM test WHERE id = 1"]=>
    array(2) {
      ["statistics"]=>
      array(11) {
        ["rows"]=>
        int(1)
        ["stored_size"]=>
        int(71)
        ["cache_hits"]=>
        int(1)
        ["run_time"]=>
        int(391)
        ["store_time"]=>
        int(27)
        ["min_run_time"]=>
        int(16)
        ["max_run_time"]=>
        int(16)
        ["min_store_time"]=>
        int(8)
        ["max_store_time"]=>
        int(8)
        ["avg_run_time"]=>
        int(8)
        ["avg_store_time"]=>
        int(4)
      }
      ["metadata"]=>
      array(1) {
        [0] =>
        array(8) {
          ["name"] =>
          string(2) "id"
          ["orig_name"] =>
          string(2) "id"
          ["table"] =>
          string(4) "test"
          ["orig_table"] =>
```

```

        string(4) "test"
        ["db"]=>
        string(4) "test"
        ["max_length"]=>
        int(1)
        ["length"]=>
        int(11)
        ["type"]=>
        int(3)
    }
}
}
["Localhost via UNIX socket
3306
root
test|/*qc=on*/SELECT id FROM test WHERE id = 0"]=>
array(2) {
    ["statistics"]=>
    array(11) {
        ["rows"]=>
        int(0)
        ["stored_size"]=>
        int(65)
        ["cache_hits"]=>
        int(1)
        ["run_time"]=>
        int(299)
        ["store_time"]=>
        int(13)
        ["min_run_time"]=>
        int(11)
        ["max_run_time"]=>
        int(11)
        ["min_store_time"]=>
        int(6)
        ["max_store_time"]=>
        int(6)
        ["avg_run_time"]=>
        int(5)
        ["avg_store_time"]=>
        int(3)
    }
    ["metadata"]=>
    array(1) {
        [0]=>
        array(8) {
            ["name"]=>
            string(2) "id"
            ["orig_name"]=>
            string(2) "id"
            ["table"]=>
            string(4) "test"
            ["orig_table"]=>
            string(4) "test"
            ["db"]=>
            string(4) "test"
            ["max_length"]=>
            int(0)
            ["length"]=>
            int(11)
            ["type"]=>
            int(3)
        }
    }
}
}
}
}

```

It is possible to further break down the granularity of statistics to the level of the normalized statement string. The normalized statement string is the statements string with all parameters replaced with

question marks. For example, the two statements `SELECT id FROM test WHERE id = 0` and `SELECT id FROM test WHERE id = 1` are normalized into `SELECT id FROM test WHERE id = ?`. Their both statistics are aggregated into one entry for `SELECT id FROM test WHERE id = ?`.

Example 6.324 Example `mysqlnd_qc_get_normalized_query_trace_log` usage

```
mysqlnd_qc.enable_qc=1
mysqlnd_qc.collect_normalized_query_trace=1
```

```
<?php
/* connect to MySQL */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");
/* dummy queries to fill the query trace */
for ($i = 1; $i <= 4; $i++) {
    $query = sprintf("/*%s*/SELECT id FROM test WHERE id = %d", MYSQLND_QC_ENABLE_SWITCH, $i % 2);
    $res = $mysqli->query($query);

    $res->free();
}
var_dump(mysqlnd_qc_get_normalized_query_trace_log());
?>
```

The above examples will output something similar to:

```
array(4) {
  [0]=>
  array(9) {
    ["query"]=>
    string(25) "DROP TABLE IF EXISTS test"
    ["occurrences"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["avg_run_time"]=>
    int(0)
    ["min_run_time"]=>
    int(0)
    ["max_run_time"]=>
    int(0)
    ["avg_store_time"]=>
    int(0)
    ["min_store_time"]=>
    int(0)
    ["max_store_time"]=>
    int(0)
  }
  [1]=>
  array(9) {
    ["query"]=>
    string(27) "CREATE TABLE test (id INT )"
    ["occurrences"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["avg_run_time"]=>
    int(0)
    ["min_run_time"]=>
    int(0)
    ["max_run_time"]=>
```

```

    int(0)
    ["avg_store_time"]=>
    int(0)
    ["min_store_time"]=>
    int(0)
    ["max_store_time"]=>
    int(0)
}
[2]=>
array(9) {
    ["query"]=>
    string(46) "INSERT INTO test (id ) VALUES ( ? ), ( ? ), ( ? )"
    ["occurrences"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["avg_run_time"]=>
    int(0)
    ["min_run_time"]=>
    int(0)
    ["max_run_time"]=>
    int(0)
    ["avg_store_time"]=>
    int(0)
    ["min_store_time"]=>
    int(0)
    ["max_store_time"]=>
    int(0)
}
[3]=>
array(9) {
    ["query"]=>
    string(31) "SELECT id FROM test WHERE id =?"
    ["occurrences"]=>
    int(4)
    ["eligible_for_caching"]=>
    bool(true)
    ["avg_run_time"]=>
    int(179)
    ["min_run_time"]=>
    int(11)
    ["max_run_time"]=>
    int(393)
    ["avg_store_time"]=>
    int(12)
    ["min_store_time"]=>
    int(7)
    ["max_store_time"]=>
    int(25)
}
}

```

The source distribution of PECL/mysqlnd_qc contains a directory [web/](#) in which web based monitoring scripts can be found which give an example how to write a cache monitor. Please, follow the instructions given in the source.

Since PECL/mysqlnd_qc 1.1.0 it is possible to write statistics into a log file. Please, see [mysqlnd_qc.collect_statistics_log_file](#).

6.8.4.9 Beyond TTL: user-defined storage

Copyright 1997-2014 the PHP Documentation Group.

The query cache plugin supports the use of user-defined storage handler. User-defined storage handler can use arbitrarily complex invalidation algorithms and support arbitrary storage media.

All user-defined storage handlers have to provide a certain interface. The functions of the user-defined storage handler will be called by the core of the cache plugin. The necessary interface consists

of seven public functions. Both procedural and object oriented user-defined storage handler must implement the same set of functions.

Example 6.325 Using a user-defined storage handler

```
<?php
/* Enable default caching of all statements */
ini_set("mysqlnd_qc.cache_by_default", 1);
/* Procedural user defined storage handler functions */
$__cache = array();
function get_hash($host_info, $port, $user, $db, $query) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());
    return md5(sprintf("%s%s%s%s", $host_info, $port, $user, $db, $query));
}
function find_query_in_cache($key) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());
    if (isset($__cache[$key])) {
        $tmp = $__cache[$key];
        if ($tmp["valid_until"] < time()) {
            unset($__cache[$key]);
            $ret = NULL;
        } else {
            $ret = $__cache[$key]["data"];
        }
    } else {
        $ret = NULL;
    }
    return $ret;
}
function return_to_cache($key) {
    /*
     * Called on cache hit after cached data has been processed,
     * may be used for reference counting
     */
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());
}
function add_query_to_cache_if_not_exists($key, $data, $ttl, $run_time, $store_time, $row_count) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());
    $__cache[$key] = array(
        "data" => $data,
        "row_count" => $row_count,
        "valid_until" => time() + $ttl,
        "hits" => 0,
        "run_time" => $run_time,
        "store_time" => $store_time,
        "cached_run_times" => array(),
        "cached_store_times" => array(),
    );
    return TRUE;
}
function query_is_select($query) {
    printf("\t%s('%s'):", __FUNCTION__, $query);
    $ret = FALSE;
    if (strpos($query, "SELECT") !== FALSE) {
        /* cache for 5 seconds */
        $ret = 5;
    }
    printf("%s\n", (FALSE === $ret) ? "FALSE" : $ret);
    return $ret;
}
function update_query_run_time_stats($key, $run_time, $store_time) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());
    if (isset($__cache[$key])) {
        $__cache[$key]['hits']++;
        $__cache[$key]["cached_run_times"][] = $run_time;
        $__cache[$key]["cached_store_times"][] = $store_time;
    }
}
```

```

    }
}
function get_stats($key = NULL) {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());
    if ($key && isset($__cache[$key])) {
        $stats = $__cache[$key];
    } else {
        $stats = array();
        foreach ($__cache as $key => $details) {
            $stats[$key] = array(
                'hits' => $details['hits'],
                'bytes' => strlen($details['data']),
                'uncached_run_time' => $details['run_time'],
                'cached_run_time' => (count($details['cached_run_times'])
                    ? array_sum($details['cached_run_times']) / count($details['cached_run_times']) : 0,
            );
        }
    }
    return $stats;
}
function clear_cache() {
    global $__cache;
    printf("\t%s(%d)\n", __FUNCTION__, func_num_args());
    $__cache = array();
    return TRUE;
}
/* Install procedural user-defined storage handler */
if (!mysqlnd_qc_set_user_handlers("get_hash", "find_query_in_cache",
    "return_to_cache", "add_query_to_cache_if_not_exists",
    "query_is_select", "update_query_run_time_stats", "get_stats", "clear_cache")) {
    printf("Failed to install user-defined storage handler\n");
}
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");
printf("\nCache put/cache miss\n");
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
/* Delete record to verify we get our data from the cache */
$mysqli->query("DELETE FROM test WHERE id = 1");
printf("\nCache hit\n");
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
printf("\nDisplay cache statistics\n");
var_dump(mysqlnd_qc_get_cache_info());
printf("\nFlushing cache, cache put/cache miss");
var_dump(mysqlnd_qc_clear_cache());
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
?>

```

The above examples will output something similar to:

```

query_is_select('DROP TABLE IF EXISTS test'): FALSE
query_is_select('CREATE TABLE test(id INT)': FALSE
query_is_select('INSERT INTO test(id) VALUES (1), (2)': FALSE
Cache put/cache miss
query_is_select('SELECT id FROM test WHERE id = 1'): 5
get_hash(5)
find_query_in_cache(1)

```



```

        add_query_to_cache_if_not_exists(6)
array(1) {
    ["id"]=>
    string(1) "1"
}
        query_is_select('DELETE FROM test WHERE id = 1'): FALSE
Cache hit
        query_is_select('SELECT id FROM test WHERE id = 1'): 5
        get_hash(5)
        find_query_in_cache(1)
        return_to_cache(1)
        update_query_run_time_stats(3)
array(1) {
    ["id"]=>
    string(1) "1"
}
Display cache statistics
        get_stats(0)
array(4) {
    ["num_entries"]=>
    int(1)
    ["handler"]=>
    string(4) "user"
    ["handler_version"]=>
    string(5) "1.0.0"
    ["data"]=>
    array(1) {
        ["18683c177dc89bb352b29965d112fdaa"]=>
        array(4) {
            ["hits"]=>
            int(1)
            ["bytes"]=>
            int(71)
            ["uncached_run_time"]=>
            int(398)
            ["cached_run_time"]=>
            int(4)
        }
    }
}
}
Flushing cache, cache put/cache miss    clear_cache(0)
bool(true)
        query_is_select('SELECT id FROM test WHERE id = 1'): 5
        get_hash(5)
        find_query_in_cache(1)
        add_query_to_cache_if_not_exists(6)
NULL

```

6.8.5 Installing/Configuring

Copyright 1997-2014 the PHP Documentation Group.

6.8.5.1 Requirements

Copyright 1997-2014 the PHP Documentation Group.

PHP 5.3.3 or a newer version of **PHP**.

PECL/mysqlnd_qc is a mysqlnd plugin. It plugs into the mysqlnd library. To use you this plugin with a PHP MySQL extension, the extension ([mysqli](#), [mysql](#), or [PDO_MYSQL](#)) must enable the mysqlnd library.

For using the [APC](#) storage handler with PECL/mysqlnd_qc 1.0 [APC 3.1.3p1-beta](#) or newer. PECL/mysqlnd_qc 1.2 has been tested with [APC 3.1.13-beta](#). The APC storage handler cannot be used with a shared build. You cannot use the PHP configuration directive [extension](#) to load the APC and PECL/mysqlnd_qc extensions if PECL/mysqlnd_qc will use APC as a storage handler. For using the

APC storage handler, you have to statically compile PHP with APC and PECL/mysqlnd_qc support into PHP.

For using [MEMCACHE](#) storage handler: Use [libmemcache 0.38](#) or newer. PECL/mysqlnd_qc 1.2 has been tested with [libmemcache 1.4.0](#).

For using [sqlite](#) storage handler: Use the [sqlite3](#) extension that bundled with PHP.

6.8.5.2 Installation

Copyright 1997-2014 the PHP Documentation Group.

This [PECL](#) extension is not bundled with PHP.

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: http://pecl.php.net/package/mysqlnd_qc

A DLL for this PECL extension is currently unavailable. See also the [building on Windows](#) section.

6.8.5.3 Runtime Configuration

Copyright 1997-2014 the PHP Documentation Group.

The behaviour of these functions is affected by settings in [php.ini](#).

Table 6.43 mysqlnd_qc Configure Options

Name	Default	Changeable	Changelog
mysqlnd_qc.enable_qc	1	PHP_INI_SYSTEM	
mysqlnd_qc.ttl	30	PHP_INI_ALL	
mysqlnd_qc.cache_by_default	0	PHP_INI_ALL	
mysqlnd_qc.cache_no_table	0	PHP_INI_ALL	
mysqlnd_qc.use_request_time	0	PHP_INI_ALL	
mysqlnd_qc.time_statistics	1	PHP_INI_ALL	
mysqlnd_qc.collect_statistics	0	PHP_INI_ALL	
mysqlnd_qc.collect_statistics_filepath	tmp/mysqlnd_qc.stats	PHP_INI_SYSTEM	
mysqlnd_qc.collect_query_trace	0	PHP_INI_SYSTEM	
mysqlnd_qc.query_trace_buffer_depth	8	PHP_INI_SYSTEM	
mysqlnd_qc.collect_normalized_query_trace	0	PHP_INI_SYSTEM	
mysqlnd_qc.ignore_sql_comments	1	PHP_INI_ALL	
mysqlnd_qc.slam_defence	0	PHP_INI_SYSTEM	
mysqlnd_qc.slam_defence_ttl	30	PHP_INI_SYSTEM	
mysqlnd_qc.std_data_copy	0	PHP_INI_SYSTEM	
mysqlnd_qc.apc_prefix	qc_	PHP_INI_ALL	
mysqlnd_qc.memc_server	127.0.0.1	PHP_INI_ALL	
mysqlnd_qc.memc_port	11211	PHP_INI_ALL	
mysqlnd_qc.sqlite_data_file	memory:	PHP_INI_ALL	

Here's a short explanation of the configuration directives.

[mysqlnd_qc.enable_qc](#) integer Enables or disables the plugin. If disabled the extension will not plug into [mysqlnd](#) to proxy internal [mysqlnd](#) C API calls.

<code>mysqlnd_qc.ttl</code> integer	Default Time-to-Live (TTL) for cache entries in seconds.
<code>mysqlnd_qc.cache_by_default</code> integer	Cache all queries regardless if they begin with the SQL hint that enables caching of a query or not. Storage handler cannot overrule the setting. It is evaluated by the core of the plugin.
<code>mysqlnd_qc.cache_no_table</code> integer	Whether to cache queries with no table name in any of columns meta data of their result set, for example, <code>SELECT SLEEP(1)</code> , <code>SELECT NOW()</code> , <code>SELECT SUBSTRING()</code> .
<code>mysqlnd_qc.use_request_time</code> integer	Use PHP global request time to avoid <code>gettimeofday()</code> system calls? If using <code>APC</code> storage handler it should be set to the value of <code>apc.use_request_time</code> , if not warnings will be generated.
<code>mysqlnd_qc.time_statistics</code> integer	Collect run time and store time statistics using <code>gettimeofday()</code> system call? Data will be collected only if you also set <code>mysqlnd_qc.collect_statistics = 1</code> ,
<code>mysqlnd_qc.collect_statistics</code> integer	Collect statistics for <code>mysqlnd_qc_get_core_stats</code> ? Does not influence storage handler statistics! Handler statistics can be an integral part of the handler internal storage format. Therefore, collection of some handler statistics cannot be disabled.
<code>mysqlnd_qc.collect_statistics_log-file</code> integer	If <code>mysqlnd_qc.collect_statistics</code> and <code>mysqlnd_qc.collect_statistics_log_file</code> are set, the plugin will dump statistics into the specified log file at every 10th web request during PHP request shutdown. The log file needs to be writable by the web server user. Since 1.1.0.
<code>mysqlnd_qc.collect_query_traces</code> integer	Collect query back traces?
<code>mysqlnd_qc.query_trace_bt_max</code> integer	Maximum depth/level of a query code traceback.
<code>mysqlnd_qc.ignore_sql_comments</code> integer	Whether to remove SQL comments from a query string before hashing it to generate a cache key. Disable if you do not want two statements such as <code>SELECT /*my_source_ip=123*/ id FROM test</code> and <code>SELECT /*my_source_ip=456*/ id FROM test</code> to refer to the same cache entry. Since 1.1.0.
<code>mysqlnd_qc.slam_defense</code> integer	Activates handler based slam defense (cache stampeding protection) if available. Supported by <code>Default</code> and <code>APC</code> storage handler
<code>mysqlnd_qc.slam_defense_ttl</code> integer	TTL for stale cache entries which are served while another client updates the entries. Supported by <code>APC</code> storage handler.
<code>mysqlnd_qc.collect_normalized_queries</code> integer	Collect aggregated normalized query traces? The setting has no effect by default. You compile the extension using the define <code>NORM_QUERY_TRACE_LOG</code> to make use of the setting.
<code>mysqlnd_qc.std_data_copy</code> integer	Default storage handler: copy cached wire data? EXPERIMENTAL – use default setting!
<code>mysqlnd_qc.apc_prefix</code> string	The <code>APC</code> storage handler stores data in the <code>APC</code> user cache. The setting sets a prefix to be used for cache entries.

<code>mysqlnd_qc.memc_server</code> string	MEMCACHE storage handler: memcache server host.
<code>mysqlnd_qc.memc_port</code> integer	MEMCACHE storage handler: memcached server port.
<code>mysqlnd_qc.sqlite_data_file</code> string	sqlite storage handler: data file. Any setting but :memory: may be of little practical value.

6.8.6 Predefined Constants

Copyright 1997-2014 the PHP Documentation Group.

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

SQL hint related

Example 6.326 Using SQL hint constants

The query cache is controlled by SQL hints. SQL hints are used to enable and disable caching. SQL hints can be used to set the **TTL** of a query.

The SQL hints recognized by the query cache can be manually changed at compile time. This makes it possible to use `mysqlnd_qc` in environments in which the default SQL hints are already taken and interpreted by other systems. Therefore it is recommended to use the SQL hint string constants instead of manually adding the default SQL hints to the query string.

```
<?php
/* Use constants for maximum portability */
$query = "/*" . MYSQLND_QC_ENABLE_SWITCH . "*/SELECT id FROM test";
/* Valid but less portable: default TTL */
$query = "/*qc=on*/SELECT id FROM test";
/* Valid but less portable: per statement TTL */
$query = "/*qc=on*//*qc_ttl=5*/SELECT id FROM test";
printf("MYSQLND_QC_ENABLE_SWITCH: %s\n", MYSQLND_QC_ENABLE_SWITCH);
printf("MYSQLND_QC_DISABLE_SWITCH: %s\n", MYSQLND_QC_DISABLE_SWITCH);
printf("MYSQLND_QC_TTL_SWITCH: %s\n", MYSQLND_QC_TTL_SWITCH);
?>
```

The above examples will output:

```
MYSQLND_QC_ENABLE_SWITCH: qc=on
MYSQLND_QC_DISABLE_SWITCH: qc=off
MYSQLND_QC_TTL_SWITCH: qc_ttl=
```

MYSQLND_QC_ENABLE_SWITCH SQL hint used to enable caching of a query.
(string)

MYSQLND_QC_DISABLE_SWITCH SQL hint used to disable caching of a query if
(string) `mysqlnd_qc.cache_by_default = 1`.

MYSQLND_QC_TTL_SWITCH SQL hint used to set the TTL of a result set.
(string)

MYSQLND_QC_SERVER_ID_SWITCH This SQL hint should not be used in general.
(string)

It is needed by [PECL/mysqlnd_ms](#) to group cache entries for one statement but originating from different physical connections. If

the hint is used connection settings such as user, hostname and charset are not considered for generating a cache key of a query. Instead the given value and the query string are used as input to the hashing function that generates the key.

PECL/mysqlnd_ms may, if instructed, cache results from MySQL Replication slaves. Because it can hold many connections to the slave the cache key shall not be formed from the user, hostname or other settings that may vary for the various slave connections. Instead, PECL/mysqlnd_ms provides an identifier which refers to the group of slave connections that shall be enabled to share cache entries no matter which physical slave connection was to generate the cache entry.

Use of this feature outside of PECL/mysqlnd_ms is not recommended.

mysqlnd_qc_set_cache_condition related

Example 6.327 Example *mysqlnd_qc_set_cache_condition* usage

The function *mysqlnd_qc_set_cache_condition* allows setting conditions for automatic caching of statements which don't begin with the SQL hints necessary to manually enable caching.

```
<?php
/* Cache all accesses to tables with the name "new%" in schema/database "db_example" for 1 second */
if (!mysqlnd_qc_set_cache_condition(MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN, "db_example.new%", 1)) {
    die("Failed to set cache condition!");
}
$mysqli = new mysqli("host", "user", "password", "db_example", "port");
/* cached although no SQL hint given */
$mysqli->query("SELECT id, title FROM news");
$pdo_mysql = new PDO("mysql:host=host;dbname=db_example;port=port", "user", "password");
/* not cached: no SQL hint, no pattern match */
$pdo_mysql->query("SELECT id, title FROM latest_news");
/* cached: TTL 1 second, pattern match */
$pdo_mysql->query("SELECT id, title FROM news");
?>
```

MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN Used as a parameter of *mysqlnd_qc_set_cache_condition* to set conditions for schema based automatic caching.

Other

The plugin version number can be obtained using either *MYSQLND_QC_VERSION*, which is the string representation of the numerical version number, or *MYSQLND_QC_VERSION_ID*, which is an integer such as 10000. Developers can calculate the version number as follows.

Version (part)	Example
Major*10000	1*10000 = 10000
Minor*100	0*100 = 0
Patch	0 = 0
MYSQLND_QC_VERSION_ID	10000

MYSQLND_QC_VERSION (string) Plugin version string, for example, "1.0.0-prototype".

MYSQLND_QC_VERSION_ID (int) Plugin version number, for example, 10000.

6.8.7 mysqlnd_qc Functions

Copyright 1997-2014 the PHP Documentation Group.

6.8.7.1 mysqlnd_qc_clear_cache

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_qc_clear_cache`

Flush all cache contents

Description

```
bool mysqlnd_qc_clear_cache();
```

Flush all cache contents.

Flushing the cache is a storage handler responsibility. All built-in storage handler but the `memcache` storage handler support flushing the cache. The `memcache` storage handler cannot flush its cache contents.

User-defined storage handler may or may not support the operation.

Parameters

This function has no parameters.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

A return value of `FALSE` indicates that flushing all cache contents has failed or the operation is not supported by the active storage handler. Applications must not expect that calling the function will always flush the cache.

6.8.7.2 mysqlnd_qc_get_available_handlers

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_qc_get_available_handlers`

Returns a list of available storage handler

Description

```
array mysqlnd_qc_get_available_handlers();
```

Which storage are available depends on the compile time configuration of the query cache plugin. The `default` storage handler is always available. All other storage handler must be enabled explicitly when building the extension.

Parameters

This function has no parameters.

Return Values

Returns an array of available built-in storage handler. For each storage handler the version number and version string is given.

Examples

Example 6.328 `mysqlnd_qc_get_available_handlers` example

```
<?php
var_dump(mysqlnd_qc_get_available_handlers());
?>
```

The above examples will output:

```
array(5) {
  ["default"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
  ["user"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
  ["APC"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
  ["MEMCACHE"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
  ["sqlite"]=>
  array(2) {
    ["version"]=>
    string(5) "1.0.0"
    ["version_number"]=>
    int(100000)
  }
}
```

See Also

[Installation](#)

[mysqlnd_qc_set_storage_handler](#)

6.8.7.3 mysqlnd_qc_get_cache_info

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_qc_get_cache_info](#)

Returns information on the current handler, the number of cache entries and cache entries, if available

Description

```
array mysqlnd_qc_get_cache_info();
```

Parameters

This function has no parameters.

Return Values

Returns information on the current handler, the number of cache entries and cache entries, if available. If and what data will be returned for the cache entries is subject to the active storage handler. Storage handler are free to return any data. Storage handler are recommended to return at least the data provided by the default handler, if technically possible.

The scope of the information is the PHP process. Depending on the PHP deployment model a process may serve one or more web requests.

Values are aggregated for all cache activities on a per storage handler basis. It is not possible to tell how much queries originating from `mysqli`, `PDO_MySQL` or `mysql.API` calls have contributed to the aggregated data values. Use `mysqlnd_qc_get_core_stats` to get timing data aggregated for all storage handlers.

Array of cache information

<code>handler</code> string	The active storage handler. All storage handler. Since 1.0.0.
<code>handler_version</code> string	The version of the active storage handler. All storage handler. Since 1.0.0.
<code>num_entries</code> int	The number of cache entries. The value depends on the storage handler in use. The default, APC and SQLite storage handler provide the actual number of cache entries. The MEMCACHE storage handler always returns 0. MEMCACHE does not support counting the number of cache entries. If a user defined handler is used, the number of entries of the <code>data</code> property is reported. Since 1.0.0.
<code>data</code> array	The version of the active storage handler. Additional storage handler dependent data on the cache entries. Storage handler are requested to provide similar and comparable information. A user defined storage handler is free to return any data. Since 1.0.0. The following information is provided by the default storage handler for the <code>data</code> property. The <code>data</code> property holds a hash. The hash is indexed by the internal cache entry identifier of the storage handler. The cache entry identifier is human-readable and contains the query string leading to the cache entry. Please, see also the example below. The following data is given for every cache entry.
<code>statistics</code> array	Statistics of the cache entry. Since 1.0.0.

Property	Description	Version
<code>rows</code>	Number of rows of the cached result set.	Since 1.0.0.
<code>store_size</code>	The size of the cached result set in bytes. This is the size of the payload. The value is not suited for calculating the total memory consumption of all cache entries including the administrative overhead of the cache entries.	Since 1.0.0.
<code>cache_hits</code>	How often the cached entry has been returned.	Since 1.0.0.
<code>run_time</code>	Run time of the statement to which the cache entry belongs. This is the run time of the uncached statement. It is the time between sending the statement to MySQL receiving a reply from MySQL. Run time saved by using the query cache plugin can be calculated like this: <code>cache_hits * ((run_time - avg_run_time) + (store_time - avg_store_time))</code> .	Since 1.0.0.
<code>store_time</code>	Store time of the statements result set to which the cache entry belongs. This is the time it took to fetch and store the results of the uncached statement.	Since 1.0.0.
<code>min_run_time</code>	Minimum run time of the cached statement. How long it took to find the statement in the cache.	Since 1.0.0.
<code>min_store_time</code>	Minimum store time of the cached statement. The time taken for fetching the cached	Since 1.0.0.

Property	Description	Version
	result set from the storage medium and decoding	
avg_time	Average time of the cached statement.	Since 1.0.0.
avg_store_time	Average store time of the cached statement.	Since 1.0.0.
max_time	Average time of the cached statement.	Since 1.0.0.
max_store_time	Average store time of the cached statement.	Since 1.0.0.
validity	Timestamp when the cache entry expires.	Since 1.1.0.

metadata array

Metadata of the cache entry. This is the metadata provided by MySQL together with the result set of the statement in question. Different versions of the MySQL server may return different metadata. Unlike with some of the PHP MySQL extensions no attempt is made to hide MySQL server version dependencies and version details from the caller. Please, refer to the MySQL C API documentation that belongs to the MySQL server in use for further details.

The metadata list contains one entry for every column.

Since 1.0.0.

Property	Description	Version
name	The field name. Depending on the MySQL version this may be the fields alias name.	Since 1.0.0.
org_name	The field name.	Since 1.0.0.
table	The table name. If an alias name was used for the table, this usually holds the alias name.	Since 1.0.0.
org_table	The table name.	Since 1.0.0.
db	The database/schema name.	Since 1.0.0.

Property	Description	Version
<code>maxlen</code>	The maximum width of the field. Details may vary by MySQL server version.	Since 1.0.0.
<code>length</code>	The width of the field. Details may vary by MySQL server version.	Since 1.0.0.
<code>type</code>	The data type of the field. Details may vary by the MySQL server in use. This is the MySQL C API type constants value. It is recommended to use type constants provided by the <code>mysqli</code> extension to test for its meaning. You should not test for certain type values by comparing with certain numbers.	Since 1.0.0.

The APC storage handler returns the same information for the `data` property but no `metadata`. The `metadata` of a cache entry is set to `NULL`.

The MEMCACHE storage handler does not fill the `data` property. Statistics are not available on a per cache entry basis with the MEMCACHE storage handler.

A user defined storage handler is free to provide any data.

Examples

Example 6.329 `mysqlnd_qc_get_cache_info` example

The example shows the output from the built-in default storage handler. Other storage handler may report different data.

```
<?php
/* Populate the cache, e.g. using mysqli */
$mysqli = new mysqli("host", "user", "password", "schema");
$mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/SELECT id FROM test");
/* Display cache information */
var_dump(mysqlnd_qc_get_cache_info());
?>
```

The above examples will output:

```
array(4) {
  ["num_entries"]=>
  int(1)
  ["handler"]=>
```

```
string(7) "default"
["handler_version"]=>
string(5) "1.0.0"
["data"]=>
array(1) {
  ["Localhost via UNIX socket 3306 user schema/*qc=on*/SELECT id FROM test"]=>
  array(2) {
    ["statistics"]=>
    array(11) {
      ["rows"]=>
      int(6)
      ["stored_size"]=>
      int(101)
      ["cache_hits"]=>
      int(0)
      ["run_time"]=>
      int(471)
      ["store_time"]=>
      int(27)
      ["min_run_time"]=>
      int(0)
      ["max_run_time"]=>
      int(0)
      ["min_store_time"]=>
      int(0)
      ["max_store_time"]=>
      int(0)
      ["avg_run_time"]=>
      int(0)
      ["avg_store_time"]=>
      int(0)
    }
    ["metadata"]=>
    array(1) {
      [0]=>
      array(8) {
        ["name"]=>
        string(2) "id"
        ["orig_name"]=>
        string(2) "id"
        ["table"]=>
        string(4) "test"
        ["orig_table"]=>
        string(4) "test"
        ["db"]=>
        string(4) "schema"
        ["max_length"]=>
        int(1)
        ["length"]=>
        int(11)
        ["type"]=>
        int(3)
      }
    }
  }
}
```

See Also

[mysqlnd_qc_get_core_stats](#)

6.8.7.4 [mysqlnd_qc_get_core_stats](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_qc_get_core_stats](#)

Statistics collected by the core of the query cache

Description

```
array mysqlnd_qc_get_core_stats();
```

Returns an array of statistics collected by the core of the cache plugin. The same data fields will be reported for any storage handler because the data is collected by the core.

The [PHP](#) configuration setting `mysqlnd_qc.collect_statistics` controls the collection of statistics. The collection of statistics is disabled by default for performance reasons. Disabling the collection of statistics will also disable the collection of time related statistics.

The [PHP](#) configuration setting `mysqlnd_qc.collect_time_statistics` controls the collection of time related statistics.

The scope of the core statistics is the [PHP](#) process. Depending on your deployment model a [PHP](#) process may handle one or multiple requests.

Statistics are aggregated for all cache entries and all storage handler. It is not possible to tell how much queries originating from [mysqli](#), [PDO_MySQL](#) or [mysql](#) API calls have contributed to the aggregated data values.

Parameters

This function has no parameters.

Return Values

Array of core statistics

Statistic	Description	Version
cache_hit	Statement is considered cacheable and cached data has been reused. Statement is considered cacheable and a cache miss happened but the statement got cached by someone else while we process it and thus we can fetch the result from the refreshed cache.	Since 1.0.0.
cache_miss	Statement is considered cacheable... • ... and has been added to the cache • ... but the PHP configuration directive setting of <code>mysqlnd_qc.cache_no_table = 1</code> has prevented caching. • ... but an unbuffered result set is requested. • ... but a buffered result set was empty.	Since 1.0.0.
cache_put	Statement is considered cacheable and has been added to the cache. Take care when calculating derived statistics. Storage handler with a storage life time beyond process scope	Since 1.0.0.

Statistic	Description	Version
	may report <code>cache_put = 0</code> together with <code>cache_hit > 0</code> , if another process has filled the cache. You may want to use <code>num_entries</code> from <code>mysqlnd_qc_get_cache_info</code> if the handler supports it (<code>default</code> , <code>APC</code>).	
<code>query_should_cache</code>	Statement is considered cacheable based on query string analysis. The statement may or may not be added to the cache. See also <code>cache_put</code> .	Since 1.0.0.
<code>query_should_not_cache</code>	Statement is considered not cacheable based on query string analysis.	Since 1.0.0.
<code>query_not_cached</code>	Statement is considered not cacheable or it is considered cachable but the storage handler has not returned a hash key for it.	Since 1.0.0.
<code>query_could_cache</code>	Statement is considered cacheable... • ... and statement has been run without errors • ... and meta data shows at least one column in the result set The statement may or may not be in the cache already. It may or may not be added to the cache later on.	Since 1.0.0.
<code>query_found_in_cache</code>	Statement is considered cacheable and we have found it in the cache but we have not replayed the cached data yet and we have not send the result set to the client yet. This is not considered a cache hit because the client might not fetch the result or the cached data may be faulty.	Since 1.0.0.
<code>query_uncached_other</code>	Statement is considered cacheable and it may or may not be in the cache already but either replaying cached data has failed, no result set is available or some other error has happened.	
<code>query_uncached_no_table</code>	Statement has not been cached because the result set has at least one column which has no	Since 1.0.0.

Statistic	Description	Version
	table name in its meta data. An example of such a query is <code>SELECT SLEEP(1)</code> . To cache those statements you have to change default value of the PHP configuration directive <code>mysqlnd_qc.cache_no_table</code> and set <code>mysqlnd_qc.cache_no_table = 1</code> . Often, it is not desired to cache such statements.	
<code>query_uncached_use_result</code>	Statement would have been cached if a buffered result set had been used. The situation is also considered as a cache miss and <code>cache_miss</code> will be incremented as well.	Since 1.0.0.
<code>query_aggr_run_time_cache</code>	Aggregated run time (ms) of all cached queries. Cached queries are those which have incremented <code>cache_hit</code> .	Since 1.0.0.
<code>query_aggr_run_time_cache_miss</code>	Aggregated run time (ms) of all uncached queries that have been put into the cache. See also <code>cache_put</code> .	Since 1.0.0.
<code>query_aggr_run_time_total</code>	Aggregated run time (ms) of all uncached and cached queries that have been inspected and executed by the query cache.	Since 1.0.0.
<code>query_aggr_store_time_cache</code>	Aggregated store time (ms) of all cached queries. Cached queries are those which have incremented <code>cache_hit</code> .	Since 1.0.0.
<code>query_aggr_store_time_cache_miss</code>	Aggregated store time (ms) of all uncached queries that have been put into the cache. See also <code>cache_put</code> .	Since 1.0.0.
<code>query_aggr_store_time_total</code>	Aggregated store time (ms) of all uncached and cached queries that have been inspected and executed by the query cache.	Since 1.0.0.
<code>receive_bytes_recorded</code>	Recorded incoming network traffic (bytes) send from MySQL to PHP. The traffic may or may not have been added to the cache. The traffic is the total for all queries regardless if cached or not.	Since 1.0.0.
<code>receive_bytes_replayed</code>	Network traffic replayed during cache. This is the total amount of incoming traffic saved because of the usage of the query cache plugin.	Since 1.0.0.

Statistic	Description	Version
<code>send_bytes_recorded</code>	Recorded outgoing network traffic (bytes) send from MySQL to PHP. The traffic may or may not have been added to the cache. The traffic is the total for all queries regardless if cached or not.	Since 1.0.0.
<code>send_bytes_replayed</code>	Network traffic replayed during cache. This is the total amount of outgoing traffic saved because of the usage of the query cache plugin.	Since 1.0.0.
<code>slam_stale_refresh</code>	Number of cache misses which triggered serving stale data until the client causing the cache miss has refreshed the cache entry.	Since 1.0.0.
<code>slam_stale_hit</code>	Number of cache hits while a stale cache entry gets refreshed.	Since 1.0.0.

Examples

Example 6.330 `mysqlnd_qc_get_core_stats` example

```
<?php
/* Enable collection of statistics - default: disabled */
ini_set("mysqlnd_qc.collect_statistics", 1);
/* Enable collection of all timing related statistics -
default: enabled but overruled by mysqlnd_qc.collect_statistics = 0 */
ini_set("mysqlnd_qc.collect_time_statistics", 1);
/* Populate the cache, e.g. using mysqli */
$mysqli = new mysqli('host', 'user', 'password', 'schema');
/* Cache miss and cache put */
$mysqli->query("/*qc=on*/SELECT id FROM test");
/* Cache hit */
$mysqli->query("/*qc=on*/SELECT id FROM test");
/* Display core statistics */
var_dump(mysqlnd_qc_get_core_stats());
?>
```

The above examples will output:

```
array(26) {
  ["cache_hit"]=>
    string(1) "1"
  ["cache_miss"]=>
    string(1) "1"
  ["cache_put"]=>
    string(1) "1"
  ["query_should_cache"]=>
    string(1) "2"
  ["query_should_not_cache"]=>
    string(1) "0"
  ["query_not_cached"]=>
    string(1) "0"
  ["query_could_cache"]=>
    string(1) "2"
  ["query_found_in_cache"]=>
```



```
string(1) "1"
["query_uncached_other"]=>
string(1) "0"
["query_uncached_no_table"]=>
string(1) "0"
["query_uncached_no_result"]=>
string(1) "0"
["query_uncached_use_result"]=>
string(1) "0"
["query_aggr_run_time_cache_hit"]=>
string(1) "4"
["query_aggr_run_time_cache_put"]=>
string(3) "395"
["query_aggr_run_time_total"]=>
string(3) "399"
["query_aggr_store_time_cache_hit"]=>
string(1) "2"
["query_aggr_store_time_cache_put"]=>
string(1) "8"
["query_aggr_store_time_total"]=>
string(2) "10"
["receive_bytes_recorded"]=>
string(2) "65"
["receive_bytes_replayed"]=>
string(2) "65"
["send_bytes_recorded"]=>
string(2) "29"
["send_bytes_replayed"]=>
string(2) "29"
["slam_stale_refresh"]=>
string(1) "0"
["slam_stale_hit"]=>
string(1) "0"
["request_counter"]=>
int(1)
["process_hash"]=>
int(3547549858)
}
```

See Also

[Runtime configuration](#)
[mysqlnd_qc.collect_statistics](#)
[mysqlnd_qc.time_statistics](#)
[mysqlnd_qc.get_cache_info](#)

6.8.7.5 mysqlnd_qc_get_normalized_query_trace_log

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_qc_get_normalized_query_trace_log](#)

Returns a normalized query trace log for each query inspected by the query cache

Description

```
array mysqlnd_qc_get_normalized_query_trace_log();
```

Returns a normalized query trace log for each query inspected by the query cache. The collection of the trace log is disabled by default. To collect the trace log you have to set the PHP configuration directive `mysqlnd_qc.collect_normalized_query_trace` to `1`

Entries in the trace log are grouped by the normalized query statement. The normalized query statement is the query statement with all statement parameter values being replaced with a question mark. For example, the two statements `SELECT id FROM test WHERE id = 1` and `SELECT`

`id FROM test WHERE id = 2` are normalized as `SELECT id FROM test WHERE id = ?`. Whenever a statement is inspected by the query cache which matches the normalized statement pattern, its statistics are grouped by the normalized statement string.

Parameters

This function has no parameters.

Return Values

An array of query log. Every list entry contains the normalized query string and further detail information.

Key	Description
<code>query</code>	Normalized statement string.
<code>occurrences</code>	How many statements have matched the normalized statement string in addition to the one which has created the log entry. The value is zero if a statement has been normalized, its normalized representation has been added to the log but no further queries inspected by PECL/mysqlnd_qc have the same normalized statement string.
<code>eligible_for_cache</code>	Whether the statement could be cached. An statement eligible for caching has not necessarily been cached. It not possible to tell for sure if or how many cached statement have contributed to the aggregated normalized statement log entry. However, comparing the minimum and average run time one can make an educated guess.
<code>avg_run_time</code>	The average run time of all queries contributing to the query log entry. The run time is the time between sending the query statement to MySQL and receiving an answer from MySQL.
<code>avg_store_time</code>	The average store time of all queries contributing to the query log entry. The store time is the time needed to fetch a statements result set from the server to the client and, storing it on the client.
<code>min_run_time</code>	The minimum run time of all queries contributing to the query log entry.
<code>min_store_time</code>	The minimum store time of all queries contributing to the query log entry.
<code>max_run_time</code>	The maximum run time of all queries contributing to the query log entry.
<code>max_store_time</code>	The maximum store time of all queries contributing to the query log entry.

Examples

Example 6.331 `mysqlnd_qc_get_normalized_query_trace_log` example

```
mysqlnd_qc.collect_normalized_query_trace=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");
/* not cached */
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
/* cache put */
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 2");
var_dump($res->fetch_assoc());
```

```
$res->free();
/* cache hit */
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 2");
var_dump($res->fetch_assoc());
$res->free();
var_dump(mysqlnd_qc_get_normalized_query_trace_log());
?>
```

The above examples will output:

```
array(1) {
    ["id"]=>
    string(1) "1"
}
array(1) {
    ["id"]=>
    string(1) "2"
}
array(1) {
    ["id"]=>
    string(1) "2"
}
array(4) {
    [0]=>
    array(9) {
        ["query"]=>
        string(25) "DROP TABLE IF EXISTS test"
        ["occurrences"]=>
        int(0)
        ["eligible_for_caching"]=>
        bool(false)
        ["avg_run_time"]=>
        int(0)
        ["min_run_time"]=>
        int(0)
        ["max_run_time"]=>
        int(0)
        ["avg_store_time"]=>
        int(0)
        ["min_store_time"]=>
        int(0)
        ["max_store_time"]=>
        int(0)
    }
    [1]=>
    array(9) {
        ["query"]=>
        string(27) "CREATE TABLE test (id INT )"
        ["occurrences"]=>
        int(0)
        ["eligible_for_caching"]=>
        bool(false)
        ["avg_run_time"]=>
        int(0)
        ["min_run_time"]=>
        int(0)
        ["max_run_time"]=>
        int(0)
        ["avg_store_time"]=>
        int(0)
        ["min_store_time"]=>
        int(0)
        ["max_store_time"]=>
        int(0)
    }
    [2]=>
    array(9) {
        ["query"]=>
```

```
string(40) "INSERT INTO test (id ) VALUES ( ? ), ( ? )"
["occurrences"]=>
int(0)
["eligible_for_caching"]=>
bool(false)
["avg_run_time"]=>
int(0)
["min_run_time"]=>
int(0)
["max_run_time"]=>
int(0)
["avg_store_time"]=>
int(0)
["min_store_time"]=>
int(0)
["max_store_time"]=>
int(0)
}
[3]=>
array(9) {
    ["query"]=>
    string(31) "SELECT id FROM test WHERE id =?"
    ["occurrences"]=>
    int(2)
    ["eligible_for_caching"]=>
    bool(true)
    ["avg_run_time"]=>
    int(159)
    ["min_run_time"]=>
    int(12)
    ["max_run_time"]=>
    int(307)
    ["avg_store_time"]=>
    int(10)
    ["min_store_time"]=>
    int(8)
    ["max_store_time"]=>
    int(13)
}
```

See Also

[Runtime configuration](#)
[mysqlnd_qc.collect_normalized_query_trace](#)
[mysqlnd_qc.time_statistics](#)
[mysqlnd_qc_get_query_trace_log](#)

6.8.7.6 [mysqlnd_qc_get_query_trace_log](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_qc_get_query_trace_log](#)

Returns a backtrace for each query inspected by the query cache

Description

```
array mysqlnd_qc_get_query_trace_log();
```

Returns a backtrace for each query inspected by the query cache. The collection of the backtrace is disabled by default. To collect the backtrace you have to set the PHP configuration directive [mysqlnd_qc.collect_query_trace](#) to 1

The maximum depth of the backtrace is limited to the depth set with the PHP configuration directive [mysqlnd_qc.query_trace_bt_depth](#).

Parameters

This function has no parameters.

Return Values

An array of query backtrace. Every list entry contains the query string, a backtrace and further detail information.

Key	Description
<code>query</code>	Query string.
<code>origin</code>	Code backtrace.
<code>run_time</code>	Query run time in milliseconds. The collection of all times and the necessary <code>gettimeofday</code> system calls can be disabled by setting the PHP configuration directive <code>mysqlnd_qc.time_statistics</code> to <code>0</code>
<code>store_time</code>	Query result set store time in milliseconds. The collection of all times and the necessary <code>gettimeofday</code> system calls can be disabled by setting the PHP configuration directive <code>mysqlnd_qc.time_statistics</code> to <code>0</code>
<code>eligible_for_caching</code>	<code>TRUE</code> if query is cacheable otherwise <code>FALSE</code> .
<code>no_table</code>	<code>TRUE</code> if the query has generated a result set and at least one column from the result set has no table name set in its metadata. This is usually the case with queries which one probably do not want to cache such as <code>SELECT SLEEP(1)</code> . By default any such query will not be added to the cache. See also PHP configuration directive <code>mysqlnd_qc.cache_no_table</code> .
<code>was_added</code>	<code>TRUE</code> if the query result has been put into the cache, otherwise <code>FALSE</code> .
<code>was_already_in_cache</code>	<code>TRUE</code> if the query result would have been added to the cache if it was not already in the cache (cache hit). Otherwise <code>FALSE</code> .

Examples

Example 6.332 `mysqlnd_qc_get_query_trace_log` example

```
mysqlnd_qc.collect_query_trace=1
```

```
<?php
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema", "port", "socket");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2)");
/* not cached */
$res = $mysqli->query("SELECT id FROM test WHERE id = 1");
var_dump($res->fetch_assoc());
$res->free();
/* cache put */
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 2");
var_dump($res->fetch_assoc());
$res->free();
/* cache hit */
$res = $mysqli->query("/*" . MYSQLND_QC_ENABLE_SWITCH . "*/" . "SELECT id FROM test WHERE id = 2");
var_dump($res->fetch_assoc());
$res->free();
var_dump(mysqlnd_qc_get_query_trace_log());
?>
```

The above examples will output:

```
array(1) {
  ["id"]=>
  string(1) "1"
}
array(1) {
  ["id"]=>
  string(1) "2"
}
array(1) {
  ["id"]=>
  string(1) "2"
}
array(6) {
  [0]=>
  array(8) {
    ["query"]=>
    string(25) "DROP TABLE IF EXISTS test"
    ["origin"]=>
    string(102) "#0 qc.php(4): mysqli->query('DROP TABLE IF E...')
#1 {main}"
    ["run_time"]=>
    int(0)
    ["store_time"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
    bool(false)
  }
  [1]=>
  array(8) {
    ["query"]=>
    string(25) "CREATE TABLE test(id INT)"
    ["origin"]=>
    string(102) "#0 qc.php(5): mysqli->query('CREATE TABLE te...')
#1 {main}"
    ["run_time"]=>
    int(0)
    ["store_time"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
    bool(false)
  }
  [2]=>
  array(8) {
    ["query"]=>
    string(36) "INSERT INTO test(id) VALUES (1), (2)"
    ["origin"]=>
    string(102) "#0 qc.php(6): mysqli->query('INSERT INTO tes...')
#1 {main}"
    ["run_time"]=>
    int(0)
    ["store_time"]=>
    int(0)
    ["eligible_for_caching"]=>
    bool(false)
```

```

    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
    bool(false)
  }
  [3]=>
  array(8) {
    ["query"]=>
    string(32) "SELECT id FROM test WHERE id = 1"
    ["origin"]=>
    string(102) "#0 qc.php(9): mysqli->query('SELECT id FROM ...')
#1 {main}"
    ["run_time"]=>
    int(0)
    ["store_time"]=>
    int(25)
    ["eligible_for_caching"]=>
    bool(false)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
    bool(false)
  }
  [4]=>
  array(8) {
    ["query"]=>
    string(41) "/*qc=on*/SELECT id FROM test WHERE id = 2"
    ["origin"]=>
    string(103) "#0 qc.php(14): mysqli->query('/*qc=on*/SELECT...')
#1 {main}"
    ["run_time"]=>
    int(311)
    ["store_time"]=>
    int(13)
    ["eligible_for_caching"]=>
    bool(true)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(true)
    ["was_already_in_cache"]=>
    bool(false)
  }
  [5]=>
  array(8) {
    ["query"]=>
    string(41) "/*qc=on*/SELECT id FROM test WHERE id = 2"
    ["origin"]=>
    string(103) "#0 qc.php(19): mysqli->query('/*qc=on*/SELECT...')
#1 {main}"
    ["run_time"]=>
    int(13)
    ["store_time"]=>
    int(8)
    ["eligible_for_caching"]=>
    bool(true)
    ["no_table"]=>
    bool(false)
    ["was_added"]=>
    bool(false)
    ["was_already_in_cache"]=>
    bool(true)
  }
}

```

See Also

Runtime configuration
mysqlnd_qc.collect_query_trace
mysqlnd_qc.query_trace_bt_depth
mysqlnd_qc.time_statistics
mysqlnd_qc.cache_no_table
mysqlnd_qc.get_normalized_query_trace_log

6.8.7.7 mysqlnd_qc_set_cache_condition

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_qc_set_cache_condition`

Set conditions for automatic caching

Description

```
bool mysqlnd_qc_set_cache_condition(  
    int condition_type,  
    mixed condition,  
    mixed condition_option);
```

Sets a condition for automatic caching of statements which do not contain the necessary SQL hints to enable caching of them.

Parameters

condition_type

Type of the condition. The only allowed value is `MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN`.

condition

Parameter for the condition set with `condition_type`. Parameter type and structure depend on `condition_type`

If `condition_type` equals `MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN` `condition` must be a string. The string sets a pattern. Statements are cached if table and database meta data entry of their result sets match the pattern. The pattern is checked for a match with the `db` and `org_table` meta data entries provided by the underlying MySQL client server library. Please, check the MySQL Reference manual for details about the two entries. The `db` and `org_table` values are concatenated with a dot (.) before matched against `condition`. Pattern matching supports the wildcards `%` and `_`. The wildcard `%` will match one or many arbitrary characters. `_` will match one arbitrary character. The escape symbol is backslash.

condition_option

Option for `condition`. Type and structure depend on `condition_type`.

If `condition_type` equals `MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN` `condition_options` is the TTL to be used.

Examples

Example 6.333 mysqlnd_qc_set_cache_condition example

```
<?php  
/* Cache all accesses to tables with the name "new%" in schema/database "db_example" for 1 second */  
if (!mysqlnd_qc_set_cache_condition(MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN, "db_example.new%", 1)) {
```



```
    die("Failed to set cache condition!");
}
$mysqli = new mysqli("host", "user", "password", "db_example", "port");
/* cached although no SQL hint given */
$mysqli->query("SELECT id, title FROM news");
$pdo_mysql = new PDO("mysql:host=host;dbname=db_example;port=port", "user", "password");
/* not cached: no SQL hint, no pattern match */
$pdo_mysql->query("SELECT id, title FROM latest_news");
/* cached: TTL 1 second, pattern match */
$pdo_mysql->query("SELECT id, title FROM news");
?>
```

Return Values

Returns TRUE on success or FALSE on FAILURE.

See Also

[Quickstart: pattern based caching](#)

6.8.7.8 mysqlnd_qc_set_is_select

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_qc_set_is_select](#)

Installs a callback which decides whether a statement is cached

Description

```
mixed mysqlnd_qc_set_is_select(
    string callback);
```

Installs a callback which decides whether a statement is cached.

There are several ways of hinting PECL/mysqlnd_qc to cache a query. By default, PECL/mysqlnd_qc attempts to cache a if caching of all statements is enabled or the query string begins with a certain SQL hint. The plugin internally calls a function named [is_select\(\)](#) to find out. This internal function can be replaced with a user-defined callback. Then, the user-defined callback is responsible to decide whether the plugin attempts to cache a statement. Because the internal function is replaced with the callback, the callback gains full control. The callback is free to ignore the configuration setting [mysqlnd_qc.cache_by_default](#) and SQL hints.

The callback is invoked for every statement inspected by the plugin. It is given the statements string as a parameter. The callback returns [FALSE](#) if the statement shall not be cached. It returns [TRUE](#) to make the plugin attempt to cache the statements result set, if any. A so-created cache entry is given the default TTL set with the PHP configuration directive [mysqlnd_qc.ttl](#). If a different TTL shall be used, the callback returns a numeric value to be used as the TTL.

The internal [is_select](#) function is part of the internal cache storage handler interface. Thus, a user-defined storage handler offers the same capabilities.

Parameters

This function has no parameters.

Return Values

Returns [TRUE](#) on success or [FALSE](#) on failure.

Examples

Example 6.334 `mysqlnd_qc_set_is_select` example

```

<?php
/* callback which decides if query is cached */
function is_select($query) {
    static $patterns = array(
        /* true - use default from mysqlnd_qc.ttl */
        "@SELECT\s+.*\s+FROM\s+test@ismU" => true,
        /* 3 - use TTL = 3 seconds */
        "@SELECT\s+.*\s+FROM\s+news@ismU" => 3
    );
    /* check if query does match pattern */
    foreach ($patterns as $pattern => $ttl) {
        if (preg_match($pattern, $query)) {
            printf("is_select(%45s): cache\n", $query);
            return $ttl;
        }
    }
    printf("is_select(%45s): do not cache\n", $query);
    return false;
}
mysqlnd_qc_set_is_select("is_select");
/* Connect, create and populate test table */
$mysqli = new mysqli("host", "user", "password", "schema");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)");
/* cache put */
$mysqli->query("SELECT id FROM test WHERE id = 1");
/* cache hit */
$mysqli->query("SELECT id FROM test WHERE id = 1");
/* cache put */
$mysqli->query("SELECT * FROM test");
?>

```

The above examples will output:

```

is_select(                DROP TABLE IF EXISTS test): do not cache
is_select(                CREATE TABLE test(id INT)): do not cache
is_select(  INSERT INTO test(id) VALUES (1), (2), (3)): do not cache
is_select(                SELECT id FROM test WHERE id = 1): cache
is_select(                SELECT id FROM test WHERE id = 1): cache
is_select(                SELECT * FROM test): cache

```

See Also

[Runtime configuration](#)
[mysqlnd_qc.ttl](#)
[mysqlnd_qc.cache_by_default](#)
[mysqlnd_qc_set_user_handlers](#)

6.8.7.9 `mysqlnd_qc_set_storage_handler`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_qc_set_storage_handler](#)

Change current storage handler

Description

```
bool mysqlnd_qc_set_storage_handler(
```

```
string handler);
```

Sets the storage handler used by the query cache. A list of available storage handler can be obtained from [mysqlnd_qc_get_available_handlers](#). Which storage are available depends on the compile time configuration of the query cache plugin. The `default` storage handler is always available. All other storage handler must be enabled explicitly when building the extension.

Parameters

handler Handler can be of type string representing the name of a built-in storage handler or an object of type [mysqlnd_qc_handler_default](#). The names of the built-in storage handler are `default`, `APC`, `MEMCACHE`, `sqlite`.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

If changing the storage handler fails a catchable fatal error will be thrown. The query cache cannot operate if the previous storage handler has been shutdown but no new storage handler has been installed.

Examples

Example 6.335 [mysqlnd_qc_set_storage_handler](#) example

The example shows the output from the built-in default storage handler. Other storage handler may report different data.

```
<?php
var_dump(mysqlnd_qc_set_storage_handler("memcache"));
if (true === mysqlnd_qc_set_storage_handler("default"))
    printf("Default storage handler activated");
/* Catchable fatal error */
var_dump(mysqlnd_qc_set_storage_handler("unknown"));
?>
```

The above examples will output:

```
bool(true)
Default storage handler activated
Catchable fatal error: mysqlnd_qc_set_storage_handler(): Unknown handler 'unknown' in (file) on line (1
```

See Also

[Installation](#)
[mysqlnd_qc_get_available_handlers](#)

6.8.7.10 [mysqlnd_qc_set_user_handlers](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_qc_set_user_handlers](#)

Sets the callback functions for a user-defined procedural storage handler

Description

```
bool mysqlnd_qc_set_user_handlers(
    string get_hash,
    string find_query_in_cache,
    string return_to_cache,
    string add_query_to_cache_if_not_exists,
    string query_is_select,
    string update_query_run_time_stats,
    string get_stats,
    string clear_cache);
```

Sets the callback functions for a user-defined procedural storage handler.

Parameters

<i>get_hash</i>	Name of the user function implementing the storage handler <i>get_hash</i> functionality.
<i>find_query_in_cache</i>	Name of the user function implementing the storage handler <i>find_in_cache</i> functionality.
<i>return_to_cache</i>	Name of the user function implementing the storage handler <i>return_to_cache</i> functionality.
<i>add_query_to_cache_if_not_exists</i>	Name of the user function implementing the storage handler <i>add_query_to_cache_if_not_exists</i> functionality.
<i>query_is_select</i>	Name of the user function implementing the storage handler <i>query_is_select</i> functionality.
<i>update_query_run_time_stats</i>	Name of the user function implementing the storage handler <i>update_query_run_time_stats</i> functionality.
<i>get_stats</i>	Name of the user function implementing the storage handler <i>get_stats</i> functionality.
<i>clear_cache</i>	Name of the user function implementing the storage handler <i>clear_cache</i> functionality.

Return Values

Returns TRUE on success or FALSE on FAILURE.

See Also

[Procedural user-defined storage handler example](#)

6.8.8 Change History

[Copyright 1997-2014 the PHP Documentation Group.](#)

This change history is a high level summary of selected changes that may impact applications and/or break backwards compatibility.

See also the [CHANGES](#) file in the source distribution for a complete list of changes.

6.8.8.1 PECL/mysqlnd_qc 1.2 series

[Copyright 1997-2014 the PHP Documentation Group.](#)

1.2.0 - alpha

- Release date: 03/2013

- Motto/theme: PHP 5.5 compatibility

Feature changes

- Update build for PHP 5.5 (Credits: Remi Collet)
- APC storage handler update
 - Fix build for APC 3.1.13-beta and trunk
- Introduced [MYSQLND_QC_VERSION](#) and [MYSQLND_QC_VERSION_ID](#).

6.8.8.2 PECL/mysqlnd_qc 1.1 series

Copyright 1997-2014 the PHP Documentation Group.

1.1.0 - stable

- Release date: 04/2012
- Motto/theme: PHP 5.4 compatibility, schema pattern based caching and mysqlnd_ms support

1.1.0 - beta

- Release date: 04/2012
- Motto/theme: PHP 5.4 compatibility, schema pattern based caching and mysqlnd_ms support

1.1.0 - alpha

- Release date: 04/2012
- Motto/theme: PHP 5.4 compatibility, schema pattern based caching and mysqlnd_ms support

Feature changes

- APC storage handler update
 - Fix build for APC 3.1.9+
 - Note: Use of the APC storage handler is currently not recommended due to stability issues of APC itself.
- New PHP configuration directives
 - [mysqlnd_qc.collect_statistics_log_file](#). Aggregated cache statistics log file written after every 10th request served by the PHP process.
 - [mysqlnd_qc.ignore_sql_comments](#). Control whether SQL comments are ignored for cache key hash generation.
- New constants and SQL hints
 - [MYSQLND_QC_SERVER_ID_SWITCH](#) allows grouping of cache entries from different physical connections. This is needed by PECL/mysqlnd_ms.
 - [MYSQLND_QC_CONDITION_META_SCHEMA_PATTERN](#) to be used with [mysqlnd_qc_set_cache_condition](#).
- New function [mysqlnd_qc_set_cache_condition](#) for built-in schema pattern based caching. Likely to support a wider range of conditions in the future.
- Report [valid_until](#) timestamp for cache entries of the default handler through [mysqlnd_qc_get_cache_info](#).

- Include charset number for cache entry hashing. This should prevent serving result sets which have the wrong charset.

API change: `get_hash_key` expects new "charsetnr" (int) parameter after "port".

- API change: changing `is_select()` signature from `bool is_select()` to `mixed is_select()`. Mixed can be either boolean or array(long ttl, string server_id). This is needed by PECL/mysqlnd_ms.

Other

- Support acting as a cache backend for [PECL/mysqlnd_ms](#) 1.3.0-beta or later to transparently replace MySQL Replication slave reads with cache accesses, if the user explicitly allows.

Bug fixes

- Fixed Bug #59959 (config.m4, wrong library - 64bit memcached handler builds) (Credits: Remi Collet)

6.8.8.3 PECL/mysqlnd_qc 1.0 series

[Copyright 1997-2014 the PHP Documentation Group.](#)

1.0.1-stable

- Release date: 12/2010
- Motto/theme: Prepared statement support

Added support for Prepared statements and unbuffered queries.

1.0.0-beta

- Release date: 07/2010
- Motto/theme: TTL-based cache with various storage options (Memcache, APC, SQLite, user-defined)

Initial public release of the transparent TTL-based query result cache. Flexible storage of cached results. Various storage media supported.

6.9 Mysqlnd user handler plugin

[Copyright 1997-2014 the PHP Documentation Group.](#)

The mysqlnd user handler plugin ([mysqlnd_uh](#)) allows users to set hooks for most internal calls of the MySQL native driver for PHP ([mysqlnd](#)). Mysqlnd and its plugins, including PECL/mysqlnd_uh, operate on a layer beneath the PHP MySQL extensions. A mysqlnd plugin can be considered as a proxy between the PHP MySQL extensions and the MySQL server as part of the PHP executable on the client-side. Because the plugins operates on their own layer below the PHP MySQL extensions, they can monitor and change application actions without requiring application changes. If the PHP MySQL extensions ([mysqli](#), [mysql](#), [PDO_MYSQL](#)) are compiled to use [mysqlnd](#) this can be used for:

- Monitoring
 - Queries executed by any of the PHP MySQL extensions
 - Prepared statements executing by any of the PHP MySQL extensions
- Auditing
 - Detection of database usage
 - SQL injection protection using black and white lists

- Assorted
- Load Balancing connections

The MySQL native driver for PHP ([mysqlnd](#)) features an internal plugin C API. C plugins, such as the [mysqlnd](#) user handler plugin, can extend the functionality of [mysqlnd](#). PECL/mysqlnd_uh makes parts of the internal plugin C API available to the PHP user for plugin development with PHP.

Status

The [mysqlnd](#) user handler plugin is in alpha status. Take appropriate care before using it in production environments.

6.9.1 Security considerations

Copyright 1997-2014 the PHP Documentation Group.

PECL/mysqlnd_uh gives users access to MySQL user names, MySQL password used by any of the PHP MySQL extensions to connect to MySQL. It allows monitoring of all queries and prepared statements exposing the statement string to the user. Therefore, the extension should be installed with care. The [PHP_INI_SYSTEM](#) configuration setting [mysqlnd_uh.enable](#) can be used to prevent users from hooking [mysqlnd](#) calls.

Code obfuscators and similar technologies are not suitable to prevent monitoring of [mysqlnd](#) library activities if PECL/mysqlnd_uh is made available and the user can install a proxy, for example, using [auto_prepend_file](#).

6.9.2 Documentation note

Copyright 1997-2014 the PHP Documentation Group.

Many of the [mysqlnd_uh](#) functions are briefly described because the [mysqli](#) extension is a thin abstraction layer on top of the MySQL C API that the [mysqlnd](#) library provides. Therefore, the corresponding [mysqli](#) documentation (along with the MySQL reference manual) can be consulted to receive more information about a particular function.

6.9.3 On the name

Copyright 1997-2014 the PHP Documentation Group.

The shortcut [mysqlnd_uh](#) stands for [mysqlnd user handler](#), and has been the name since early development.

6.9.4 Quickstart and Examples

Copyright 1997-2014 the PHP Documentation Group.

The [mysqlnd](#) user handler plugin can be understood as a client-side proxy for all PHP MySQL extensions ([mysqli](#), [mysql](#), [PDO_MYSQL](#)), if they are compiled to use the [mysqlnd](#) library. The extensions use the [mysqlnd](#) library internally, at the C level, to communicate with the MySQL server. PECL/mysqlnd_uh allows it to hook many [mysqlnd](#) calls. Therefore, most activities of the PHP MySQL extensions can be monitored.

Because monitoring happens at the level of the library, at a layer below the application, it is possible to monitor applications without changing them.

On the C level, the [mysqlnd](#) library is structured in modules or classes. The extension hooks almost all methods of the [mysqlnd](#) internal [connection](#) class and exposes them through the user space class [MysqlndUhConnection](#). Some few methods of the [mysqlnd](#) internal [statement](#) class are made available to the PHP user with the class [MysqlndUhPreparedStatement](#). By subclassing

the classes `MysqlndUhConnection` and `MysqlndUhPreparedStatement` users get access to `mysqlnd` internal function calls.

Note

The internal `mysqlnd` function calls are not designed to be exposed to the PHP user. Manipulating their activities may cause PHP to crash or leak memory. Often, this is not considered a bug. Please, keep in mind that you are accessing C library functions through PHP which are expected to take certain actions, which you may not be able to emulate in user space. Therefore, it is strongly recommended to always call the parent method implementation when subclassing `MysqlndUhConnection` or `MysqlndUhPreparedStatement`. To prevent the worst case, the extension performs some sanity checks. Please, see also the [Mysqlnd_uh Configure Options](#).

6.9.4.1 Setup

Copyright 1997-2014 the PHP Documentation Group.

The plugin is implemented as a PHP extension. See the [installation instructions](#) to install the [PECL/mysqlnd_uh](#) extension. Then, load the extension into PHP and activate the plugin in the PHP configuration file using the PHP configuration directive named `mysqlnd_uh.enable`. The below example shows the default settings of the extension.

Example 6.336 Enabling the plugin (php.ini)

```
mysqlnd_uh.enable=1
mysqlnd_uh.report_wrong_types=1
```

6.9.4.2 How it works

Copyright 1997-2014 the PHP Documentation Group.

This describes the background and inner workings of the `mysqlnd_uh` extension.

Two classes are provided by the extension: `MysqlndUhConnection` and `MysqlndUhPreparedStatement`. `MysqlndUhConnection` lets you access almost all methods of the `mysqlnd` internal `connection` class. The latter exposes some selected methods of the `mysqlnd` internal `statement` class. For example, `MysqlndUhConnection::connect` maps to the `mysqlnd` library C function `mysqlnd_conn__connect`.

As a `mysqlnd` plugin, the `PECL/mysqlnd_uh` extension replaces `mysqlnd` library C functions with its own functions. Whenever a PHP MySQL extension compiled to use `mysqlnd` calls a `mysqlnd` function, the functions installed by the plugin are executed instead of the original `mysqlnd` ones. For example, `mysqli_connect` invokes `mysqlnd_conn__connect`, so the connect function installed by `PECL/mysqlnd_uh` will be called. The functions installed by `PECL/mysqlnd_uh` are the methods of the built-in classes.

The built-in PHP classes and their methods do nothing but call their `mysqlnd` C library counterparts, to behave exactly like the original `mysqlnd` function they replace. The code below illustrates in pseudo-code what the extension does.

Example 6.337 Pseudo-code: what a built-in class does

```
class MysqlndUhConnection {
    public function connect(($conn, $host, $user, $passwd, $db, $port, $socket, $mysql_flags) {
        MYSQLND* c_mysqlnd_connection = convert_from_php_to_c($conn);
```



```

    ...
    return call_c_function(mysqlnd_conn__connect(c_mysqlnd_connection, ...));
}
}

```

The build-in classes behave like a transparent proxy. It is possible for you to replace the proxy with your own. This is done by subclassing [MysqlndUhConnection](#) or [MysqlndUhPreparedStatement](#) to extend the functionality of the proxy, followed by registering a new proxy object. Proxy objects are installed by [mysqlnd_uh_set_connection_proxy](#) and [mysqlnd_uh_set_statement_proxy](#).

Example 6.338 Installing a proxy

```

<?php
class proxy extends MysqlndUhConnection {
    public function connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
?>

```

The above example will output:

```

proxy::connect(array (
    0 => NULL,
    1 => 'localhost',
    2 => 'root',
    3 => '',
    4 => 'test',
    5 => 3306,
    6 => NULL,
    7 => 131072,
))
proxy::connect returns true

```

6.9.4.3 Installing a proxy

Copyright 1997-2014 the PHP Documentation Group.

The extension provides two built-in classes: [MysqlndUhConnection](#) and [MysqlndUhPreparedStatement](#). The classes are used for hooking [mysqlnd](#) library calls. Their methods correspond to [mysqlnd](#) internal functions. By default they act like a transparent proxy and do nothing but call their [mysqlnd](#) counterparts. By subclassing the classes you can install your own proxy to monitor [mysqlnd](#).

See also the [How it works](#) guide to learn about the inner workings of this extension.

Connection proxies are objects of the type [MysqlndUhConnection](#). Connection proxy objects are installed by [mysqlnd_uh_set_connection_proxy](#). If you install the built-in class [MysqlndUhConnection](#) as a proxy, nothing happens. It behaves like a transparent proxy.

Example 6.339 Proxy registration, `mysqlnd_uh.enable=1`

```
<?php
mysqlnd_uh_set_connection_proxy(new MysqlndUhConnection());
$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The `PHP_INI_SYSTEM` configuration setting `mysqlnd_uh.enable` controls whether a proxy may be set. If disabled, the extension will throw errors of type `E_WARNING`

Example 6.340 Proxy installation disabled

```
mysqlnd_uh.enable=0
```

```
<?php
mysqlnd_uh_set_connection_proxy(new MysqlndUhConnection());
$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The above example will output:

```
PHP Warning: MysqlndUhConnection::__construct(): (Mysqlnd User Handler) The plugin has been disabled by set
PHP Warning: mysqlnd_uh_set_connection_proxy(): (Mysqlnd User Handler) The plugin has been disabled by set
```

To monitor `mysqlnd`, you have to write your own proxy object subclassing `MysqlndUhConnection`. Please, see the function reference for a the list of methods that can be subclassed. Alternatively, you can use reflection to inspect the built-in `MysqlndUhConnection`.

Create a new class `proxy`. Derive it from the built-in class `MysqlndUhConnection`. Replace the `MysqlndUhConnection::connect` method. Print out the host parameter value passed to the method. Make sure that you call the parent implementation of the `connect` method. Failing to do so may give unexpected and undesired results, including memory leaks and crashes.

Register your proxy and open three connections using the PHP MySQL extensions `mysqli`, `mysql`, `PDO_MYSQL`. If the extensions have been compiled to use the `mysqlnd` library, the `proxy::connect` method will be called three times, once for each connection opened.

Example 6.341 Connection proxy

```
<?php
class proxy extends MysqlndUhConnection {
    public function connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags) {
        printf("Connection opened to '%s'\n", $host);
        /* Always call the parent implementation! */
        return parent::connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags);
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysql = mysql_connect("localhost", "root", "");
$pdo = new PDO("mysql:host=localhost;dbname=test", "root", "");
?>
```

The above example will output:

```
Connection opened to 'localhost'
Connection opened to 'localhost'
Connection opened to 'localhost'
```

The use of prepared statement proxies follows the same pattern: create a proxy object of the type [MysqlndUhPreparedStatement](#) and install the proxy using [mysqlnd_uh_set_statement_proxy](#).

Example 6.342 Prepared statement proxy

```
<?php
class stmt_proxy extends MysqlndUhPreparedStatement {
    public function prepare($res, $query) {
        printf("%s(%s)\n", __METHOD__, $query);
        return parent::prepare($res, $query);
    }
}
mysqlnd_uh_set_statement_proxy(new stmt_proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$stmt = $mysqli->prepare("SELECT 'mysqlnd hacking made easy' AS _msg FROM DUAL");
?>
```

The above example will output:

```
stmt_proxy::prepare(SELECT 'mysqlnd hacking made easy' AS _msg FROM DUAL)
```

6.9.4.4 Basic query monitoring

[Copyright 1997-2014 the PHP Documentation Group.](#)

Basic monitoring of a query statement is easy with PECL/mysqlnd_uh. Combined with [debug_print_backtrace](#) it can become a powerful tool, for example, to find the origin of certain statement. This may be desired when searching for slow queries but also after database refactoring to find code still accessing deprecated databases or tables. The latter may be a complicated matter to do otherwise, especially if the application uses auto-generated queries.

Example 6.343 Basic Monitoring

```
<?php
class conn_proxy extends MysqlndUhConnection {
    public function query($res, $query) {
        debug_print_backtrace();
        return parent::query($res, $query);
    }
}
class stmt_proxy extends MysqlndUhPreparedStatement {
    public function prepare($res, $query) {
        debug_print_backtrace();
        return parent::prepare($res, $query);
    }
}
mysqlnd_uh_set_connection_proxy(new conn_proxy());
mysqlnd_uh_set_statement_proxy(new stmt_proxy());
```

```
printf("Proxies installed...\n");
$pdo = new PDO("mysql:host=localhost;dbname=test", "root", "");
var_dump($pdo->query("SELECT 1 AS _one FROM DUAL")->fetchAll(PDO::FETCH_ASSOC));
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->prepare("SELECT 1 AS _two FROM DUAL");
?>
```

The above example will output:

```
#0 conn_proxy->query(Resource id #19, SELECT 1 AS _one FROM DUAL)
#1 PDO->query(SELECT 1 AS _one FROM DUAL) called at [example.php:19]
array(1) {
  [0]=>
  array(1) {
    ["_one"]=>
    string(1) "1"
  }
}
#0 stmt_proxy->prepare(Resource id #753, SELECT 1 AS _two FROM DUAL)
#1 mysqli->prepare(SELECT 1 AS _two FROM DUAL) called at [example.php:22]
```

For basic query monitoring you should install a connection and a prepared statement proxy. The connection proxy should subclass `mysqli_connection::query`. All database queries not using native prepared statements will call this method. In the example the `query` function is invoked by a PDO call. By default, `PDO_MYSQL` is using prepared statement emulation.

All native prepared statements are prepared with the `prepare` method of `mysqli` exported through `mysqli_prepared_statement::prepare`. Subclass `mysqli_prepared_statement` and overwrite `prepare` for native prepared statement monitoring.

6.9.5 Installing/Configuring

Copyright 1997-2014 the PHP Documentation Group.

6.9.5.1 Requirements

Copyright 1997-2014 the PHP Documentation Group.

PHP 5.3.3 or later. It is recommended to use **PHP 5.4.0** or later to get access to the latest `mysqli` features.

The `mysqli_uh` user handler plugin supports all PHP applications and all available PHP MySQL extensions (`mysqli`, `mysql`, `PDO_MYSQL`). The PHP MySQL extension must be configured to use `mysqli` in order to be able to use the `mysqli_uh` plugin for `mysqli`.

The alpha versions makes use of some `mysqli` features. You must enable `mysqli` to compile the plugin. This requirement may be removed in the future. Note, that this requirement does not restrict you to use the plugin only with `mysqli`. You can use the plugin to monitor `mysql`, `mysqli` and `PDO_MYSQL`.

6.9.5.2 Installation

Copyright 1997-2014 the PHP Documentation Group.

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: <http://pecl.php.net/package/mysqli-uh>

PECL/`mysqli_uh` is currently not available on Windows. The source code of the extension makes use of C99 constructs not allowed with PHP Windows builds.

6.9.5.3 Runtime Configuration

Copyright 1997-2014 the PHP Documentation Group.

The behaviour of these functions is affected by settings in `php.ini`.

Table 6.44 Mysqlnd_uh Configure Options

Name	Default	Changeable	Changelog
<code>mysqlnd_uh.enable</code>	1	PHP_INI_SYSTEM	
<code>mysqlnd_uh.report_wrong_types</code>	1	PHP_INI_ALL	

Here's a short explanation of the configuration directives.

`mysqlnd_uh.enable` integer Enables or disables the plugin. If set to disabled, the extension will not allow users to plug into `mysqlnd` to hook `mysqlnd` calls.

`mysqlnd_uh.report_wrong_types` integer Whether to report wrong return value types of user hooks as `E_WARNING` level errors. This is recommended for detecting errors.

6.9.5.4 Resource Types

Copyright 1997-2014 the PHP Documentation Group.

This extension has no resource types defined.

6.9.6 Predefined Constants

Copyright 1997-2014 the PHP Documentation Group.

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Most of the constants refer to details of the MySQL Client Server Protocol. Please, refer to the MySQL reference manual to learn about their meaning. To avoid content duplication, only short descriptions are given.

`mysqlndUhConnection::simpleCommand` related

The following constants can be used to detect what command is to be send through `mysqlndUhConnection::simpleCommand`.

`MYSQLND_UH_MYSQLND_COM_SLEEP` MySQL Client Server protocol command: `COM_SLEEP`.
(integer)

`MYSQLND_UH_MYSQLND_COM_QUIT` MySQL Client Server protocol command: `COM_QUIT`.
(integer)

`MYSQLND_UH_MYSQLND_COM_INIT_DB` MySQL Client Server protocol command: `COM_INIT_DB`.
(integer)

`MYSQLND_UH_MYSQLND_COM_QUERY` MySQL Client Server protocol command: `COM_QUERY`.
(integer)

`MYSQLND_UH_MYSQLND_COM_FIELD_LIST` MySQL Client Server protocol command: `COM_FIELD_LIST`.
(integer)

`MYSQLND_UH_MYSQLND_COM_CREATE_DB` MySQL Client Server protocol command: `COM_CREATE_DB`.
(integer)

`MYSQLND_UH_MYSQLND_COM_DROP_DB` MySQL Client Server protocol command: `COM_DROP_DB`.
(integer)

<code>MYSQLND_UH_MYSQLND_COM_REFRESH</code> (integer)	MySQL Client Server protocol command: COM_REFRESH.
<code>MYSQLND_UH_MYSQLND_COM_SHUTDOWN</code> (integer)	MySQL Client Server protocol command: COM_SHUTDOWN.
<code>MYSQLND_UH_MYSQLND_COM_STATISTICS</code> (integer)	MySQL Client Server protocol command: COM_STATISTICS.
<code>MYSQLND_UH_MYSQLND_COM_PROCESS_INFO</code> (integer)	MySQL Client Server protocol command: COM_PROCESS_INFO.
<code>MYSQLND_UH_MYSQLND_COM_CONNECT</code> (integer)	MySQL Client Server protocol command: COM_CONNECT.
<code>MYSQLND_UH_MYSQLND_COM_PROCESS_KILL</code> (integer)	MySQL Client Server protocol command: COM_PROCESS_KILL.
<code>MYSQLND_UH_MYSQLND_COM_DEBUG</code> (integer)	MySQL Client Server protocol command: COM_DEBUG.
<code>MYSQLND_UH_MYSQLND_COM_PING</code> (integer)	MySQL Client Server protocol command: COM_PING.
<code>MYSQLND_UH_MYSQLND_COM_TIME</code> (integer)	MySQL Client Server protocol command: COM_TIME.
<code>MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT</code> (integer)	MySQL Client Server protocol command: COM_DELAYED_INSERT.
<code>MYSQLND_UH_MYSQLND_COM_CHANGE_USER</code> (integer)	MySQL Client Server protocol command: COM_CHANGE_USER.
<code>MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP</code> (integer)	MySQL Client Server protocol command: COM_BINLOG_DUMP.
<code>MYSQLND_UH_MYSQLND_COM_TABLE_DUMP</code> (integer)	MySQL Client Server protocol command: COM_TABLE_DUMP.
<code>MYSQLND_UH_MYSQLND_COM_CONNECT_OUT</code> (integer)	MySQL Client Server protocol command: COM_CONNECT_OUT.
<code>MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVE</code> (integer)	MySQL Client Server protocol command: COM_REGISTER_SLAVE.
<code>MYSQLND_UH_MYSQLND_COM_STMT_PREPARE</code> (integer)	MySQL Client Server protocol command: COM_STMT_PREPARE.
<code>MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE</code> (integer)	MySQL Client Server protocol command: COM_STMT_EXECUTE.
<code>MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA</code> (integer)	MySQL Client Server protocol command: COM_STMT_SEND_LONG_DATA.
<code>MYSQLND_UH_MYSQLND_COM_STMT_CLOSE</code> (integer)	MySQL Client Server protocol command: COM_STMT_CLOSE.
<code>MYSQLND_UH_MYSQLND_COM_STMT_RESET</code> (integer)	MySQL Client Server protocol command: COM_STMT_RESET.
<code>MYSQLND_UH_MYSQLND_COM_SET_OPTION</code> (integer)	MySQL Client Server protocol command: COM_SET_OPTION.

`MYSQLND_UH_MYSQLND_COM_STMT_FETCH` MySQL Client Server protocol command: COM_STMT_FETCH.
(integer)

`MYSQLND_UH_MYSQLND_COM_DAEMON` MySQL Client Server protocol command: COM_DAEMON.
(integer)

`MYSQLND_UH_MYSQLND_COM_END` MySQL Client Server protocol command: COM_END.
(integer)

The following constants can be used to analyze the `ok_packet` argument of `MysqlndUhConnection::simpleCommand`.

`MYSQLND_UH_MYSQLND_PROT_GREETING` MySQL Client Server protocol packet: greeting.
(integer)

`MYSQLND_UH_MYSQLND_PROT_AUTH` MySQL Client Server protocol packet: authentication.
(integer)

`MYSQLND_UH_MYSQLND_PROT_OK` MySQL Client Server protocol packet: OK.
(integer)

`MYSQLND_UH_MYSQLND_PROT_EOF` MySQL Client Server protocol packet: EOF.
(integer)

`MYSQLND_UH_MYSQLND_PROT_CMD` MySQL Client Server protocol packet: command.
(integer)

`MYSQLND_UH_MYSQLND_PROT_RS_HEADER` MySQL Client Server protocol packet: result set header.
(integer)

`MYSQLND_UH_MYSQLND_PROT_RS_FIELD` MySQL Client Server protocol packet: resultset field.
(integer)

`MYSQLND_UH_MYSQLND_PROT_ROW` MySQL Client Server protocol packet: row.
(integer)

`MYSQLND_UH_MYSQLND_PROT_STMT_STATS` MySQL Client Server protocol packet: stats.
(integer)

`MYSQLND_UH_MYSQLND_PREPARE_OK` MySQL Client Server protocol packet: prepare response.
(integer)

`MYSQLND_UH_MYSQLND_CHG_USER` MySQL Client Server protocol packet: change user response.
(integer)

`MYSQLND_UH_MYSQLND_PROT_LAST` No practical meaning. Last entry marker of internal C data structure list.

`MysqlndUhConnection::close` related

The following constants can be used to detect why a connection has been closed through `MysqlndUhConnection::close()`.

`MYSQLND_UH_MYSQLND_CLOSE_USER` User has called `mysqlnd` to close the connection.
(integer)

`MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT` Implicitly closed, for example, during garbage connection.
(integer)

`MYSQLND_UH_MYSQLND_CLOSE_ERROR` Connection error.
(integer)

`MYSQLND_UH_MYSQLND_CLOSE_LAST` No practical meaning. Last entry marker of internal C data structure list.

MysqlndUhConnection::setServerOption() related

The following constants can be used to detect which option is set through `MysqlndUhConnection::setServerOption()`.

`MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENT_SUPPORT` Option: enables multi statement support.
(integer)

`MYSQLND_UH_SERVER_OPTION_DISABLE_MULTI_STATEMENT_SUPPORT` Option: disables multi statement support.
(integer)

MysqlndUhConnection::setClientOption() related

The following constants can be used to detect which option is set through `MysqlndUhConnection::setClientOption()`.

`MYSQLND_UH_MYSQLND_OPTION_CONNECTION_TIMEOUT` Option: connection timeout.
(integer)

`MYSQLND_UH_MYSQLND_OPTION_COMPRESSED_PROTOCOL` Option: whether the MySQL compressed protocol is to be used.
(integer)

`MYSQLND_UH_MYSQLND_OPTION_NAMED_PIPE` Option: named pipe to use for connection (Windows).
(integer)

`MYSQLND_UH_MYSQLND_OPTION_INIT_COMMAND` Option: init command to execute upon connect.
(integer)

`MYSQLND_UH_MYSQLND_READ_DEFAULTS` Option: MySQL server default file to read upon connect.
(integer)

`MYSQLND_UH_MYSQLND_READ_DEFAULT_GROUP` Option: MySQL server default file group to read upon connect.
(integer)

`MYSQLND_UH_MYSQLND_SET_CHARSET_DIR` Option: charset description files directory.
(integer)

`MYSQLND_UH_MYSQLND_SET_CHARSET_NAME` Option: charset name.
(integer)

`MYSQLND_UH_MYSQLND_OPT_LOCAL_INFILE` Option: Whether to allow `LOAD DATA LOCAL INFILE` use.
(integer)

`MYSQLND_UH_MYSQLND_OPT_PROTOCOL` Option: supported protocol version.
(integer)

`MYSQLND_UH_MYSQLND_SHARED_MEMORY_BASE_NAME` Option: shared memory base name for shared memory connections.
(integer)

`MYSQLND_UH_MYSQLND_OPT_READ_TIMEOUT` Option: connection read timeout.
(integer)

`MYSQLND_UH_MYSQLND_OPT_WRITE_TIMEOUT` Option: connection write timeout.
(integer)

`MYSQLND_UH_MYSQLND_OPT_UNBUFFERED_RESULT_SETS` Option: unbuffered result sets.
(integer)

`MYSQLND_UH_MYSQLND_OPT_USE_EMBEDDED_SERVER` Embedded server related.
(integer)

`MYSQLND_UH_MYSQLND_OPT_USE_MYSQLND` Embedded server related.
(integer)

`MYSQLND_UH_MYSQLND_OPT_GUESS_CONNECTION`
(integer)

`MYSQLND_UH_MYSQLND_SET_CLIENT_IP`
(integer)

`MYSQLND_UH_MYSQLND_SECURE`
(integer)

`MYSQLND_UH_MYSQLND_REPORT_OPTION` Option: Whether to report data truncation.
(integer)

`MYSQLND_UH_MYSQLND_OPT_RECONNECT` Option: Whether to reconnect automatically.
(integer)

`MYSQLND_UH_MYSQLND_OPT_SSL_OPTION` Option: `SERVER_CERT`
(integer)

`MYSQLND_UH_MYSQLND_OPT_NETWORK_BUFFER_SIZE` Option: MySQLnd network buffer size for commands.
(integer)

`MYSQLND_UH_MYSQLND_OPT_NETWORK_BUFFER_SIZE` Option: MySQLnd network buffer size for reading from the server.
(integer)

`MYSQLND_UH_MYSQLND_OPT_SSL_KEY` Option: SSL key.
(integer)

`MYSQLND_UH_MYSQLND_OPT_SSL_CERT` Option: SSL certificate.
(integer)

`MYSQLND_UH_MYSQLND_OPT_SSL_CA` Option: SSL CA.
(integer)

`MYSQLND_UH_MYSQLND_OPT_SSL_CA_PATH` Option: Path to SSL CA.
(integer)

`MYSQLND_UH_MYSQLND_OPT_SSL_CIPHER` Option: SSL cipher.
(integer)

`MYSQLND_UH_MYSQLND_OPT_SSL_PASSPHRASE` Option: SSL passphrase.
(integer)

`MYSQLND_UH_SERVER_OPTION_PLUGIN_DIRECTORY` Option: server plugin directory.
(integer)

`MYSQLND_UH_SERVER_OPTION_DEFAULT_AUTHENTICATION_METHOD` Option: Default authentication method.
(integer)

`MYSQLND_UH_SERVER_OPTION_SET_CLIENT_IP`
(integer)

`MYSQLND_UH_MYSQLND_OPT_MAX_ALLOWED_PACKET_SIZE` Option: Maximum allowed packet size. Available as of [PHP 5.4.0](#).
(integer)

`MYSQLND_UH_MYSQLND_OPT_AUTOMATIC_RETRY` Option: `TODO`. Available as of [PHP 5.4.0](#).
(integer)

`MYSQLND_UH_MYSQLND_OPT_INTEGER_FLOAT_RETURN` Option: Make MySQLnd return integer and float columns as long even when using the MySQL Client Server text protocol. Only available with a custom build of MySQLnd.
(integer)

Other

The plugins version number can be obtained using [MYSQLND_UH_VERSION](#) or [MYSQLND_UH_VERSION_ID](#). [MYSQLND_UH_VERSION](#) is the string representation of the numerical version number [MYSQLND_UH_VERSION_ID](#), which is an integer such as 10000. Developers can calculate the version number as follows.

Version (part)	Example
Major*10000	1*10000 = 10000
Minor*100	0*100 = 0
Patch	0 = 0
MYSQLND_UH_VERSION_ID	10000

[MYSQLND_UH_VERSION](#) (string) Plugin version string, for example, "1.0.0-alpha".

[MYSQLND_UH_VERSION_ID](#) (integer) Plugin version number, for example, 10000.

6.9.7 The MysqlndUhConnection class

Copyright 1997-2014 the PHP Documentation Group.

```
MysqlndUhConnection {
MysqlndUhConnection
    Methods

    public bool MysqlndUhConnection::changeUser(
        mysqlnd_connection connection,
        string user,
        string password,
        string database,
        bool silent,
        int passwd_len);

    public string MysqlndUhConnection::charsetName(
        mysqlnd_connection connection);

    public bool MysqlndUhConnection::close(
        mysqlnd_connection connection,
        int close_type);

    public bool MysqlndUhConnection::connect(
        mysqlnd_connection connection,
        string host,
        string use",
        string password,
        string database,
        int port,
        string socket,
        int mysql_flags);

    public MysqlndUhConnection::__construct();

    public bool MysqlndUhConnection::endPSession(
        mysqlnd_connection connection);

    public string MysqlndUhConnection::escapeString(
        mysqlnd_connection connection,
        string escape_string);

    public int MysqlndUhConnection::getAffectedRows(
        mysqlnd_connection connection);

    public int MysqlndUhConnection::getErrorMessage(
        mysqlnd_connection connection);
```

```
public string MysqlndUhConnection::getErrorString(
    mysqlnd_connection connection);

public int MysqlndUhConnection::getFieldCount(
    mysqlnd_connection connection);

public string MysqlndUhConnection::getHostInformation(
    mysqlnd_connection connection);

public int MysqlndUhConnection::getLastInsertId(
    mysqlnd_connection connection);

public void MysqlndUhConnection::getLastMessage(
    mysqlnd_connection connection);

public string MysqlndUhConnection::getProtocolInformation(
    mysqlnd_connection connection);

public string MysqlndUhConnection::getServerInformation(
    mysqlnd_connection connection);

public string MysqlndUhConnection::getServerStatistics(
    mysqlnd_connection connection);

public int MysqlndUhConnection::getServerVersion(
    mysqlnd_connection connection);

public string MysqlndUhConnection::getSqlstate(
    mysqlnd_connection connection);

public array MysqlndUhConnection::getStatistics(
    mysqlnd_connection connection);

public int MysqlndUhConnection::getThreadId(
    mysqlnd_connection connection);

public int MysqlndUhConnection::getWarningCount(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::init(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::killConnection(
    mysqlnd_connection connection,
    int pid);

public array MysqlndUhConnection::listFields(
    mysqlnd_connection connection,
    string table,
    string achtung_wild);

public void MysqlndUhConnection::listMethod(
    mysqlnd_connection connection,
    string query,
    string achtung_wild,
    string par1);

public bool MysqlndUhConnection::moreResults(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::nextResult(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::ping(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::query(
    mysqlnd_connection connection,
    string query);

public bool MysqlndUhConnection::queryReadResultsetHeader(
    mysqlnd_connection connection,
    mysqlnd_statement mysqlnd_stmt);
```

```
public bool MysqlndUhConnection::reapQuery(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::refreshServer(
    mysqlnd_connection connection,
    int options);

public bool MysqlndUhConnection::restartPSession(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::selectDb(
    mysqlnd_connection connection,
    string database);

public bool MysqlndUhConnection::sendClose(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::sendQuery(
    mysqlnd_connection connection,
    string query);

public bool MysqlndUhConnection::serverDumpDebugInformation(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::setAutocommit(
    mysqlnd_connection connection,
    int mode);

public bool MysqlndUhConnection::setCharset(
    mysqlnd_connection connection,
    string charset);

public bool MysqlndUhConnection::setClientOption(
    mysqlnd_connection connection,
    int option,
    int value);

public void MysqlndUhConnection::setServerOption(
    mysqlnd_connection connection,
    int option);

public void MysqlndUhConnection::shutdownServer(
    string MYSQLND_UH_RES_MYSQLND_NAME,
    string "level");

public bool MysqlndUhConnection::simpleCommand(
    mysqlnd_connection connection,
    int command,
    string arg,
    int ok_packet,
    bool silent,
    bool ignore_upsert_status);

public bool MysqlndUhConnection::simpleCommandHandleResponse(
    mysqlnd_connection connection,
    int ok_packet,
    bool silent,
    int command,
    bool ignore_upsert_status);

public bool MysqlndUhConnection::sslSet(
    mysqlnd_connection connection,
    string key,
    string cert,
    string ca,
    string capath,
    string cipher);

public resource MysqlndUhConnection::stmtInit(
    mysqlnd_connection connection);

public resource MysqlndUhConnection::storeResult(
    mysqlnd_connection connection);
```

```
public bool MysqlndUhConnection::txCommit(
    mysqlnd_connection connection);

public bool MysqlndUhConnection::txRollback(
    mysqlnd_connection connection);

public resource MysqlndUhConnection::useResult(
    mysqlnd_connection connection);
}
```

6.9.7.1 MysqlndUhConnection::changeUser

Copyright 1997-2014 the PHP Documentation Group.

- **MysqlndUhConnection::changeUser**

Changes the user of the specified mysqlnd database connection

Description

```
public bool MysqlndUhConnection::changeUser(
    mysqlnd_connection connection,
    string user,
    string password,
    string database,
    bool silent,
    int passwd_len);
```

Changes the user of the specified mysqlnd database connection

Parameters

<i>connection</i>	Mysqlnd connection handle. Do not modify!
<i>user</i>	The MySQL user name.
<i>password</i>	The MySQL password.
<i>database</i>	The MySQL database to change to.
<i>silent</i>	Controls if mysqlnd is allowed to emit errors or not.
<i>passwd_len</i>	Length of the MySQL password.

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples

Example 6.344 MysqlndUhConnection::changeUser example

```
<?php
class proxy extends MysqlndUhConnection {
    /* Hook mysqlnd's connection::change_user call */
    public function changeUser($res, $user, $passwd, $db, $silent, $passwd_len) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::changeUser($res, $user, $passwd, $db, $silent, $passwd_len);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
/* Install proxy/hooks to be used with all future mysqlnd connection */
mysqlnd_uh_set_connection_proxy(new proxy());
/* Create mysqli connection which is using the mysqlnd library */
$mysqli = new mysqli("localhost", "root", "", "test");
```

```
/* Example of a user API call which triggers the hooked mysqlnd call */
var_dump($mysqli->change_user("root", "bar", "test"));
?>
```

The above example will output:

```
proxy::changeUser(array (
  0 => NULL,
  1 => 'root',
  2 => 'bar',
  3 => 'test',
  4 => false,
  5 => 3,
))
proxy::changeUser returns false
bool(false)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_change_user](#)

6.9.7.2 MysqlndUhConnection::charsetName

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::charsetName](#)

Returns the default character set for the database connection

Description

```
public string MysqlndUhConnection::charsetName(
    mysqlnd_connection connection);
```

Returns the default character set for the database connection.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

The default character set.

Examples

Example 6.345 [MysqlndUhConnection::charsetName](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function charsetName($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::charsetName($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump(mysqli_character_set_name($mysqli));
```

```
?>
```

The above example will output:

```
proxy::charsetName(array (
    0 => NULL,
))
proxy::charsetName returns 'latin1'
string(6) "latin1"
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_character_set_name](#)

6.9.7.3 MysqlndUhConnection::close

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::close](#)

Closes a previously opened database connection

Description

```
public bool MysqlndUhConnection::close(
    mysqlnd_connection connection,
    int close_type);
```

Closes a previously opened database connection.

Note

Failing to call the parent implementation may cause memory leaks or crash PHP. This is not considered a bug. Please, keep in mind that the [mysqlnd](#) library functions have never been designed to be exposed to the user space.

Parameters

connection	The connection to be closed. Do not modify!
close_type	Why the connection is to be closed. The value of close_type is one of MYSQLND_UH_MYSQLND_CLOSE_EXPLICIT , MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT , MYSQLND_UH_MYSQLND_CLOSE_DISCONNECTED or MYSQLND_UH_MYSQLND_CLOSE_LAST . The latter should never be seen, unless the default behaviour of the mysqlnd library has been changed by a plugin.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.346 MysqlndUhConnection::close example

```
<?php
function close_type_to_string($close_type) {
```

```
$mapping = array(
    MYSQLND_UH_MYSQLND_CLOSE_DISCONNECTED => "MYSQLND_UH_MYSQLND_CLOSE_DISCONNECTED",
    MYSQLND_UH_MYSQLND_CLOSE_EXPLICIT => "MYSQLND_UH_MYSQLND_CLOSE_EXPLICIT",
    MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT => "MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT",
    MYSQLND_UH_MYSQLND_CLOSE_LAST => "MYSQLND_UH_MYSQLND_CLOSE_IMPLICIT"
);
return (isset($mapping[$close_type])) ? $mapping[$close_type] : 'unknown';
}
class proxy extends MysqlndUhConnection {
    public function close($res, $close_type) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("close_type = %s\n", close_type_to_string($close_type));
        /* WARNING: you must call the parent */
        $ret = parent::close($res, $close_type);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->close();
?>
```

The above example will output:

```
proxy::close(array (
    0 => NULL,
    1 => 0,
))
close_type = MYSQLND_UH_MYSQLND_CLOSE_EXPLICIT
proxy::close returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_close](#)
[mysql_close](#)

6.9.7.4 MysqlndUhConnection::connect

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::connect](#)

Open a new connection to the MySQL server

Description

```
public bool MysqlndUhConnection::connect(
    mysqlnd_connection connection,
    string host,
    string use",
    string password,
    string database,
    int port,
    string socket,
    int mysql_flags);
```

Open a new connection to the MySQL server.

Parameters

[connection](#)

Mysqlnd connection handle. Do not modify!

<i>host</i>	Can be either a host name or an IP address. Passing the <code>NULL</code> value or the string "localhost" to this parameter, the local host is assumed. When possible, pipes will be used instead of the TCP/IP protocol.
<i>user</i>	The MySQL user name.
<i>password</i>	If not provided or <code>NULL</code> , the MySQL server will attempt to authenticate the user against those user records which have no password only. This allows one username to be used with different permissions (depending on if a password as provided or not).
<i>database</i>	If provided will specify the default database to be used when performing queries.
<i>port</i>	Specifies the port number to attempt to connect to the MySQL server.
<i>socket</i>	Specifies the socket or named pipe that should be used. If <code>NULL</code> , mysqlnd will default to <code>/tmp/mysql.sock</code> .
<i>mysql_flags</i>	Connection options.

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 6.347 `MysqlndUhConnection::connect` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::connect($res, $host, $user, $passwd, $db, $port, $socket, $mysql_flags);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The above example will output:

```
proxy::connect(array (
    0 => NULL,
    1 => 'localhost',
    2 => 'root',
    3 => '',
    4 => 'test',
    5 => 3306,
    6 => NULL,
    7 => 131072,
))
proxy::connect returns true
```

See Also

```
mysqlnd_uh_set_connection_proxy  
mysqli_connect  
mysql_connect
```

6.9.7.5 MysqlndUhConnection::__construct

Copyright 1997-2014 the PHP Documentation Group.

- `MysqlndUhConnection::__construct`

The `__construct` purpose

Description

```
public MysqlndUhConnection::__construct();
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

6.9.7.6 MysqlndUhConnection::endPSession

Copyright 1997-2014 the PHP Documentation Group.

- `MysqlndUhConnection::endPSession`

End a persistent connection

Description

```
public bool MysqlndUhConnection::endPSession(  
    mysqlnd_connection connection);
```

End a persistent connection

Warning

This function is currently not documented; only its argument list is available.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns `TRUE` on success. Otherwise, returns `FALSE`

Examples

Example 6.348 MysqlndUhConnection::endPSession example

```
<?php  
class proxy extends MysqlndUhConnection {  
    public function endPSession($conn) {  
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));  
        $ret = parent::endPSession($conn);  
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));  
    }  
}
```

```
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("p:localhost", "root", "", "test");
$mysqli->close();
?>
```

The above example will output:

```
proxy::endPSession(array (
    0 => NULL,
))
proxy::endPSession returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.7 MysqlndUhConnection::escapeString

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::escapeString](#)

Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection

Description

```
public string MysqlndUhConnection::escapeString(
    mysqlnd_connection connection,
    string escape_string);
```

Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection.

Parameters

[MYSQLND_UH_RES_MYSQLND_NAME](#) Mysqlnd connection handle. Do not modify!

[escape_string](#) The string to be escaped.

Return Values

The escaped string.

Examples

Example 6.349 MysqlndUhConnection::escapeString example

```
<?php
class proxy extends MysqlndUhConnection {
    public function escapeString($res, $string) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::escapeString($res, $string);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
```

```
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->set_charset("latin1");
$mysqli->real_escape_string("test0'test");
?>
```

The above example will output:

```
proxy::escapeString(array (
    0 => NULL,
    1 => 'test0\'test',
))
proxy::escapeString returns 'test0\\\'test'
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_real_escape_string](#)
[mysql_real_escape_string](#)

6.9.7.8 MysqlndUhConnection::getAffectedRows

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getAffectedRows](#)

Gets the number of affected rows in a previous MySQL operation

Description

```
public int MysqlndUhConnection::getAffectedRows(
    mysqlnd_connection connection);
```

Gets the number of affected rows in a previous MySQL operation.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Number of affected rows.

Examples

Example 6.350 MysqlndUhConnection::getAffectedRows example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getAffectedRows($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getAffectedRows($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT)");
$mysqli->query("INSERT INTO test(id) VALUES (1)");
var_dump($mysqli->affected_rows);
```

```
?>
```

The above example will output:

```
proxy::getAffectedRows(array (
    0 => NULL,
))
proxy::getAffectedRows returns 1
int(1)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_affected_rows](#)
[mysql_affected_rows](#)

6.9.7.9 MysqlndUhConnection::getErrorNumber

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getErrorNumber](#)

Returns the error code for the most recent function call

Description

```
public int MysqlndUhConnection::getErrorNumber(
    mysqlnd_connection connection);
```

Returns the error code for the most recent function call.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

Error code for the most recent function call.

Examples

[MysqlndUhConnection::getErrorNumber](#) is not only executed after the invocation of a user space API call which maps directly to it but also called internally.

Example 6.351 MysqlndUhConnection::getErrorNumber example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getErrorNumber($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getErrorNumber($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
printf("connect...\n");
$mysqli = new mysqli("localhost", "root", "", "test");
printf("query...\n");
$mysqli->query("PLEASE_LET_THIS_BE_INVALID_SQL");
printf("errno...\n");
```

```
var_dump($mysqli->errno);
printf("close...\n");
$mysqli->close();
?>
```

The above example will output:

```
connect...
proxy::getErrorNumber(array (
    0 => NULL,
))
proxy::getErrorNumber returns 0
query...
errno...
proxy::getErrorNumber(array (
    0 => NULL,
))
proxy::getErrorNumber returns 1064
int(1064)
close...
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[MysqlndUhConnection::getErrorString](#)
[mysqli_errno](#)
[mysql_errno](#)

6.9.7.10 MysqlndUhConnection::getErrorString

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getErrorString](#)

Returns a string description of the last error

Description

```
public string MysqlndUhConnection::getErrorString(
    mysqlnd_connection connection);
```

Returns a string description of the last error.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Error string for the most recent function call.

Examples

[MysqlndUhConnection::getErrorString](#) is not only executed after the invocation of a user space API call which maps directly to it but also called internally.

Example 6.352 MysqlndUhConnection::getErrorString example

```
<?php
class proxy extends MysqlndUhConnection {
```

```
public function getErrorString($res) {
    printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
    $ret = parent::getErrorString($res);
    printf("%s returns %s\n", __METHOD__, var_export($ret, true));
    return $ret;
}
}
mysqlnd_uh_set_connection_proxy(new proxy());
printf("connect...\n");
$mysqli = new mysqli("localhost", "root", "", "test");
printf("query...\n");
$mysqli->query("WILL_I_EVER_LEARN_SQL?");
printf("errno...\n");
var_dump($mysqli->error);
printf("close...\n");
$mysqli->close();
?>
```

The above example will output:

```
connect...
proxy::getErrorString(array (
    0 => NULL,
))
proxy::getErrorString returns ''
query...
errno...
proxy::getErrorString(array (
    0 => NULL,
))
proxy::getErrorString returns 'You have an error in your SQL syntax; check the manual that corresponds
string(168) "You have an error in your SQL syntax; check the manual that corresponds to your MySQL serv
close...
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[MysqlndUhConnection::getErrorMessage](#)
[mysqli_error](#)
[mysql_error](#)

6.9.7.11 MysqlndUhConnection::getFieldCount

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getFieldCount](#)

Returns the number of columns for the most recent query

Description

```
public int MysqlndUhConnection::getFieldCount(
    mysqlnd_connection connection);
```

Returns the number of columns for the most recent query.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Number of columns.

Examples

[MysqlndUhConnection::getFieldCount](#) is not only executed after the invocation of a user space API call which maps directly to it but also called internally.

Example 6.353 [MysqlndUhConnection::getFieldCount](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getFieldCount($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getFieldCount($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("WILL_I_EVER_LEARN_SQL?");
var_dump($mysqli->field_count);
$mysqli->query("SELECT 1, 2, 3 FROM DUAL");
var_dump($mysqli->field_count);
?>
```

The above example will output:

```
proxy::getFieldCount(array (
    0 => NULL,
))
proxy::getFieldCount returns 0
int(0)
proxy::getFieldCount(array (
    0 => NULL,
))
proxy::getFieldCount returns 3
proxy::getFieldCount(array (
    0 => NULL,
))
proxy::getFieldCount returns 3
int(3)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_field_count](#)

6.9.7.12 [MysqlndUhConnection::getHostInformation](#)

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getHostInformation](#)

Returns a string representing the type of connection used

Description

```
public string MysqlndUhConnection::getHostInformation(
    mysqlnd_connection connection);
```

Returns a string representing the type of connection used.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Connection description.

Examples**Example 6.354 MysqlndUhConnection::getHostInformation example**

```
<?php
class proxy extends MysqlndUhConnection {
    public function getHostInformation($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getHostInformation($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->host_info);
?>
```

The above example will output:

```
proxy::getHostInformation(array (
    0 => NULL,
))
proxy::getHostInformation returns 'Localhost via UNIX socket'
string(25) "Localhost via UNIX socket"
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_host_info](#)
[mysql_get_host_info](#)

6.9.7.13 MysqlndUhConnection::getLastInsertId

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getLastInsertId](#)

Returns the auto generated id used in the last query.

Description

```
public int MysqlndUhConnection::getLastInsertId(
    mysqlnd_connection connection);
```

Returns the auto generated id used in the last query.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Last insert id.

Examples

Example 6.355 `MysqlndUhConnection::getLastInsertId` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getLastInsertId($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getLastInsertId($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("DROP TABLE IF EXISTS test");
$mysqli->query("CREATE TABLE test(id INT AUTO_INCREMENT PRIMARY KEY, col VARCHAR(255))");
$mysqli->query("INSERT INTO test(col) VALUES ('a')");
var_dump($mysqli->insert_id);
?>
```

The above example will output:

```
proxy::getLastInsertId(array (
    0 => NULL,
))
proxy::getLastInsertId returns 1
int(1)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_insert_id](#)
[mysql_insert_id](#)

6.9.7.14 `MysqlndUhConnection::getLastMessage`

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getLastMessage](#)

Retrieves information about the most recently executed query

Description

```
public void MysqlndUhConnection::getLastMessage(
    mysqlnd_connection connection);
```

Retrieves information about the most recently executed query.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Last message. Trying to return a string longer than 511 bytes will cause an error of the type [E_WARNING](#) and result in the string being truncated.

Examples

Example 6.356 `MysqlndUhConnection::getLastMessage` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getLastMessage($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getLastMessage($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->info);
$mysqli->query("DROP TABLE IF EXISTS test");
var_dump($mysqli->info);
?>
```

The above example will output:

```
proxy::getLastMessage(array (
    0 => NULL,
))
proxy::getLastMessage returns ''
string(0) ""
proxy::getLastMessage(array (
    0 => NULL,
))
proxy::getLastMessage returns ''
string(0) ""
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_info](#)
[mysql_info](#)

6.9.7.15 `MysqlndUhConnection::getProtocolInformation`

Copyright 1997-2014 the PHP Documentation Group.

- `MysqlndUhConnection::getProtocolInformation`

Returns the version of the MySQL protocol used

Description

```
public string MysqlndUhConnection::getProtocolInformation(
    mysqlnd_connection connection);
```

Returns the version of the MySQL protocol used.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

The protocol version.

Examples

Example 6.357 `MysqlndUhConnection::getProtocolInformation` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getProtocolInformation($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getProtocolInformation($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->protocol_version);
?>
```

The above example will output:

```
proxy::getProtocolInformation(array (
  0 => NULL,
))
proxy::getProtocolInformation returns 10
int(10)
```

See Also

`mysqlnd_uh_set_connection_proxy`
`mysqli_get_proto_info`
`mysql_get_proto_info`

6.9.7.16 `MysqlndUhConnection::getServerInformation`

Copyright 1997-2014 the PHP Documentation Group.

- `MysqlndUhConnection::getServerInformation`

Returns the version of the MySQL server

Description

```
public string MysqlndUhConnection::getServerInformation(
    mysqlnd_connection connection);
```

Returns the version of the MySQL server.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

The server version.

Examples

Example 6.358 `MysqlndUhConnection::getServerInformation` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getServerInformation($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getServerInformation($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->server_info);
?>
```

The above example will output:

```
proxy::getServerInformation(array (
    0 => NULL,
))
proxy::getServerInformation returns '5.1.45-debug-log'
string(16) "5.1.45-debug-log"
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_server_info](#)
[mysql_get_server_info](#)

6.9.7.17 MysqlndUhConnection::getServerStatistics

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getServerStatistics](#)

Gets the current system status

Description

```
public string MysqlndUhConnection::getServerStatistics(
    mysqlnd_connection connection);
```

Gets the current system status.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

The system status message.

Examples

Example 6.359 MysqlndUhConnection::getServerStatistics example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getServerStatistics($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getServerStatistics($res);
```

```
    printf("%s returns %s\n", __METHOD__, var_export($ret, true));
    return $ret;
}
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump(mysqli_stat($mysqli));
?>
```

The above example will output:

```
proxy::getServerStatistics(array (
    0 => NULL,
))
proxy::getServerStatistics returns 'Uptime: 2059995  Threads: 1  Questions: 126157  Slow queries: 0  Opens:
string(140) "Uptime: 2059995  Threads: 1  Questions: 126157  Slow queries: 0  Opens: 6377  Flush tables: 1
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_stat](#)
[mysql_stat](#)

6.9.7.18 MysqlndUhConnection::getServerVersion

[Copyright 1997-2014 the PHP Documentation Group.](#)

- [MysqlndUhConnection::getServerVersion](#)

Returns the version of the MySQL server as an integer

Description

```
public int MysqlndUhConnection::getServerVersion(
    mysqlnd_connection connection);
```

Returns the version of the MySQL server as an integer.

Parameters

connection MySQLnd connection handle. Do not modify!

Return Values

The MySQL version.

Examples

Example 6.360 MysqlndUhConnection::getServerVersion example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getServerVersion($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getServerVersion($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
```

```
var_dump($mysqli->server_version);
?>
```

The above example will output:

```
proxy::getServerVersion(array (
  0 => NULL,
))
proxy::getServerVersion returns 50145
int(50145)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_server_version](#)
[mysql_get_server_version](#)

6.9.7.19 MysqlndUhConnection::getSqlstate

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getSqlstate](#)

Returns the SQLSTATE error from previous MySQL operation

Description

```
public string MysqlndUhConnection::getSqlstate(
    mysqlnd_connection connection);
```

Returns the SQLSTATE error from previous MySQL operation.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

The SQLSTATE code.

Examples

Example 6.361 [MysqlndUhConnection::getSqlstate](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getSqlstate($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getSqlstate($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->sqlstate);
$mysqli->query("AN_INVALID_REQUEST_TO_PROVOKE_AN_ERROR");
var_dump($mysqli->sqlstate);
?>
```

The above example will output:

```
proxy::getSqlstate(array (
  0 => NULL,
))
proxy::getSqlstate returns '00000'
string(5) "00000"
proxy::getSqlstate(array (
  0 => NULL,
))
proxy::getSqlstate returns '42000'
string(5) "42000"
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_sql_state](#)

6.9.7.20 MysqlndUhConnection::getStatistics

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getStatistics](#)

Returns statistics about the client connection.

Description

```
public array MysqlndUhConnection::getStatistics(
    mysqlnd_connection connection);
```

Returns statistics about the client connection.

Warning

This function is currently not documented; only its argument list is available.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Connection statistics collected by mysqlnd.

Examples

Example 6.362 MysqlndUhConnection::getStatistics example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getStatistics($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getStatistics($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->get_connection_stats());
?>
```


The above example will output:

```

proxy::getStatistics(array (
    0 => NULL,
))
proxy::getStatistics returns array (
    'bytes_sent' => '73',
    'bytes_received' => '77',
    'packets_sent' => '2',
    'packets_received' => '2',
    'protocol_overhead_in' => '8',
    'protocol_overhead_out' => '8',
    'bytes_received_ok_packet' => '0',
    'bytes_received_eof_packet' => '0',
    'bytes_received_rset_header_packet' => '0',
    'bytes_received_rset_field_meta_packet' => '0',
    'bytes_received_rset_row_packet' => '0',
    'bytes_received_prepare_response_packet' => '0',
    'bytes_received_change_user_packet' => '0',
    'packets_sent_command' => '0',
    'packets_received_ok' => '0',
    'packets_received_eof' => '0',
    'packets_received_rset_header' => '0',
    'packets_received_rset_field_meta' => '0',
    'packets_received_rset_row' => '0',
    'packets_received_prepare_response' => '0',
    'packets_received_change_user' => '0',
    'result_set_queries' => '0',
    'non_result_set_queries' => '0',
    'no_index_used' => '0',
    'bad_index_used' => '0',
    'slow_queries' => '0',
    'buffered_sets' => '0',
    'unbuffered_sets' => '0',
    'ps_buffered_sets' => '0',
    'ps_unbuffered_sets' => '0',
    'flushed_normal_sets' => '0',
    'flushed_ps_sets' => '0',
    'ps_prepared_never_executed' => '0',
    'ps_prepared_once_executed' => '0',
    'rows_fetched_from_server_normal' => '0',
    'rows_fetched_from_server_ps' => '0',
    'rows_buffered_from_client_normal' => '0',
    'rows_buffered_from_client_ps' => '0',
    'rows_fetched_from_client_normal_buffered' => '0',
    'rows_fetched_from_client_normal_unbuffered' => '0',
    'rows_fetched_from_client_ps_buffered' => '0',
    'rows_fetched_from_client_ps_unbuffered' => '0',
    'rows_fetched_from_client_ps_cursor' => '0',
    'rows_affected_normal' => '0',
    'rows_affected_ps' => '0',
    'rows_skipped_normal' => '0',
    'rows_skipped_ps' => '0',
    'copy_on_write_saved' => '0',
    'copy_on_write_performed' => '0',
    'command_buffer_too_small' => '0',
    'connect_success' => '1',
    'connect_failure' => '0',
    'connection_reused' => '0',
    'reconnect' => '0',
    'pconnect_success' => '0',
    'active_connections' => '1',
    'active_persistent_connections' => '0',
    'explicit_close' => '0',
    'implicit_close' => '0',
    'disconnect_close' => '0',
    'in_middle_of_command_close' => '0',
    'explicit_free_result' => '0',

```

```

'implicit_free_result' => '0',
'explicit_stmt_close' => '0',
'implicit_stmt_close' => '0',
'mem_emalloc_count' => '0',
'mem_emalloc_amount' => '0',
'mem_ecalloc_count' => '0',
'mem_ecalloc_amount' => '0',
'mem_erealloc_count' => '0',
'mem_erealloc_amount' => '0',
'mem_efree_count' => '0',
'mem_efree_amount' => '0',
'mem_malloc_count' => '0',
'mem_malloc_amount' => '0',
'mem_calloc_count' => '0',
'mem_calloc_amount' => '0',
'mem_realloc_count' => '0',
'mem_realloc_amount' => '0',
'mem_free_count' => '0',
'mem_free_amount' => '0',
'mem_strndup_count' => '0',
'mem_strndup_count' => '0',
'mem_strndup_count' => '0',
'mem_strdup_count' => '0',
'proto_text_fetched_null' => '0',
'proto_text_fetched_bit' => '0',
'proto_text_fetched_tinyint' => '0',
'proto_text_fetched_short' => '0',
'proto_text_fetched_int24' => '0',
'proto_text_fetched_int' => '0',
'proto_text_fetched_bigint' => '0',
'proto_text_fetched_decimal' => '0',
'proto_text_fetched_float' => '0',
'proto_text_fetched_double' => '0',
'proto_text_fetched_date' => '0',
'proto_text_fetched_year' => '0',
'proto_text_fetched_time' => '0',
'proto_text_fetched_datetime' => '0',
'proto_text_fetched_timestamp' => '0',
'proto_text_fetched_string' => '0',
'proto_text_fetched_blob' => '0',
'proto_text_fetched_enum' => '0',
'proto_text_fetched_set' => '0',
'proto_text_fetched_geometry' => '0',
'proto_text_fetched_other' => '0',
'proto_binary_fetched_null' => '0',
'proto_binary_fetched_bit' => '0',
'proto_binary_fetched_tinyint' => '0',
'proto_binary_fetched_short' => '0',
'proto_binary_fetched_int24' => '0',
'proto_binary_fetched_int' => '0',
'proto_binary_fetched_bigint' => '0',
'proto_binary_fetched_decimal' => '0',
'proto_binary_fetched_float' => '0',
'proto_binary_fetched_double' => '0',
'proto_binary_fetched_date' => '0',
'proto_binary_fetched_year' => '0',
'proto_binary_fetched_time' => '0',
'proto_binary_fetched_datetime' => '0',
'proto_binary_fetched_timestamp' => '0',
'proto_binary_fetched_string' => '0',
'proto_binary_fetched_blob' => '0',
'proto_binary_fetched_enum' => '0',
'proto_binary_fetched_set' => '0',
'proto_binary_fetched_geometry' => '0',
'proto_binary_fetched_other' => '0',
'init_command_executed_count' => '0',
'init_command_failed_count' => '0',
'com_quit' => '0',
'com_init_db' => '0',
'com_query' => '0',
'com_field_list' => '0',
'com_create_db' => '0',

```

```

'com_drop_db' => '0',
'com_refresh' => '0',
'com_shutdown' => '0',
'com_statistics' => '0',
'com_process_info' => '0',
'com_connect' => '0',
'com_process_kill' => '0',
'com_debug' => '0',
'com_ping' => '0',
'com_time' => '0',
'com_delayed_insert' => '0',
'com_change_user' => '0',
'com_binlog_dump' => '0',
'com_table_dump' => '0',
'com_connect_out' => '0',
'com_register_slave' => '0',
'com_stmt_prepare' => '0',
'com_stmt_execute' => '0',
'com_stmt_send_long_data' => '0',
'com_stmt_close' => '0',
'com_stmt_reset' => '0',
'com_stmt_set_option' => '0',
'com_stmt_fetch' => '0',
'com_deamon' => '0',
'bytes_received_real_data_normal' => '0',
'bytes_received_real_data_ps' => '0',
)
array(160) {
  ["bytes_sent"]=>
  string(2) "73"
  ["bytes_received"]=>
  string(2) "77"
  ["packets_sent"]=>
  string(1) "2"
  ["packets_received"]=>
  string(1) "2"
  ["protocol_overhead_in"]=>
  string(1) "8"
  ["protocol_overhead_out"]=>
  string(1) "8"
  ["bytes_received_ok_packet"]=>
  string(1) "0"
  ["bytes_received_eof_packet"]=>
  string(1) "0"
  ["bytes_received_rset_header_packet"]=>
  string(1) "0"
  ["bytes_received_rset_field_meta_packet"]=>
  string(1) "0"
  ["bytes_received_rset_row_packet"]=>
  string(1) "0"
  ["bytes_received_prepare_response_packet"]=>
  string(1) "0"
  ["bytes_received_change_user_packet"]=>
  string(1) "0"
  ["packets_sent_command"]=>
  string(1) "0"
  ["packets_received_ok"]=>
  string(1) "0"
  ["packets_received_eof"]=>
  string(1) "0"
  ["packets_received_rset_header"]=>
  string(1) "0"
  ["packets_received_rset_field_meta"]=>
  string(1) "0"
  ["packets_received_rset_row"]=>
  string(1) "0"
  ["packets_received_prepare_response"]=>
  string(1) "0"
  ["packets_received_change_user"]=>
  string(1) "0"
  ["result_set_queries"]=>
  string(1) "0"

```

```
["non_result_set_queries"]=>
string(1) "0"
["no_index_used"]=>
string(1) "0"
["bad_index_used"]=>
string(1) "0"
["slow_queries"]=>
string(1) "0"
["buffered_sets"]=>
string(1) "0"
["unbuffered_sets"]=>
string(1) "0"
["ps_buffered_sets"]=>
string(1) "0"
["ps_unbuffered_sets"]=>
string(1) "0"
["flushed_normal_sets"]=>
string(1) "0"
["flushed_ps_sets"]=>
string(1) "0"
["ps_prepared_never_executed"]=>
string(1) "0"
["ps_prepared_once_executed"]=>
string(1) "0"
["rows_fetched_from_server_normal"]=>
string(1) "0"
["rows_fetched_from_server_ps"]=>
string(1) "0"
["rows_buffered_from_client_normal"]=>
string(1) "0"
["rows_buffered_from_client_ps"]=>
string(1) "0"
["rows_fetched_from_client_normal_buffered"]=>
string(1) "0"
["rows_fetched_from_client_normal_unbuffered"]=>
string(1) "0"
["rows_fetched_from_client_ps_buffered"]=>
string(1) "0"
["rows_fetched_from_client_ps_unbuffered"]=>
string(1) "0"
["rows_fetched_from_client_ps_cursor"]=>
string(1) "0"
["rows_affected_normal"]=>
string(1) "0"
["rows_affected_ps"]=>
string(1) "0"
["rows_skipped_normal"]=>
string(1) "0"
["rows_skipped_ps"]=>
string(1) "0"
["copy_on_write_saved"]=>
string(1) "0"
["copy_on_write_performed"]=>
string(1) "0"
["command_buffer_too_small"]=>
string(1) "0"
["connect_success"]=>
string(1) "1"
["connect_failure"]=>
string(1) "0"
["connection_reused"]=>
string(1) "0"
["reconnect"]=>
string(1) "0"
["pconnect_success"]=>
string(1) "0"
["active_connections"]=>
string(1) "1"
["active_persistent_connections"]=>
string(1) "0"
["explicit_close"]=>
string(1) "0"
```

```
["implicit_close"]=>
string(1) "0"
["disconnect_close"]=>
string(1) "0"
["in_middle_of_command_close"]=>
string(1) "0"
["explicit_free_result"]=>
string(1) "0"
["implicit_free_result"]=>
string(1) "0"
["explicit_stmt_close"]=>
string(1) "0"
["implicit_stmt_close"]=>
string(1) "0"
["mem_emalloc_count"]=>
string(1) "0"
["mem_emalloc_amount"]=>
string(1) "0"
["mem_ecalloc_count"]=>
string(1) "0"
["mem_ecalloc_amount"]=>
string(1) "0"
["mem_erealloc_count"]=>
string(1) "0"
["mem_erealloc_amount"]=>
string(1) "0"
["mem_efree_count"]=>
string(1) "0"
["mem_efree_amount"]=>
string(1) "0"
["mem_malloc_count"]=>
string(1) "0"
["mem_malloc_amount"]=>
string(1) "0"
["mem_calloc_count"]=>
string(1) "0"
["mem_calloc_amount"]=>
string(1) "0"
["mem_realloc_count"]=>
string(1) "0"
["mem_realloc_amount"]=>
string(1) "0"
["mem_free_count"]=>
string(1) "0"
["mem_free_amount"]=>
string(1) "0"
["mem_strdup_count"]=>
string(1) "0"
["mem_strndup_count"]=>
string(1) "0"
["mem_estndup_count"]=>
string(1) "0"
["mem_strdup_count"]=>
string(1) "0"
["proto_text_fetched_null"]=>
string(1) "0"
["proto_text_fetched_bit"]=>
string(1) "0"
["proto_text_fetched_tinyint"]=>
string(1) "0"
["proto_text_fetched_short"]=>
string(1) "0"
["proto_text_fetched_int24"]=>
string(1) "0"
["proto_text_fetched_int"]=>
string(1) "0"
["proto_text_fetched_bigint"]=>
string(1) "0"
["proto_text_fetched_decimal"]=>
string(1) "0"
["proto_text_fetched_float"]=>
string(1) "0"
```

```
["proto_text_fetched_double"]=>
string(1) "0"
["proto_text_fetched_date"]=>
string(1) "0"
["proto_text_fetched_year"]=>
string(1) "0"
["proto_text_fetched_time"]=>
string(1) "0"
["proto_text_fetched_datetime"]=>
string(1) "0"
["proto_text_fetched_timestamp"]=>
string(1) "0"
["proto_text_fetched_string"]=>
string(1) "0"
["proto_text_fetched_blob"]=>
string(1) "0"
["proto_text_fetched_enum"]=>
string(1) "0"
["proto_text_fetched_set"]=>
string(1) "0"
["proto_text_fetched_geometry"]=>
string(1) "0"
["proto_text_fetched_other"]=>
string(1) "0"
["proto_binary_fetched_null"]=>
string(1) "0"
["proto_binary_fetched_bit"]=>
string(1) "0"
["proto_binary_fetched_tinyint"]=>
string(1) "0"
["proto_binary_fetched_short"]=>
string(1) "0"
["proto_binary_fetched_int24"]=>
string(1) "0"
["proto_binary_fetched_int"]=>
string(1) "0"
["proto_binary_fetched_bigint"]=>
string(1) "0"
["proto_binary_fetched_decimal"]=>
string(1) "0"
["proto_binary_fetched_float"]=>
string(1) "0"
["proto_binary_fetched_double"]=>
string(1) "0"
["proto_binary_fetched_date"]=>
string(1) "0"
["proto_binary_fetched_year"]=>
string(1) "0"
["proto_binary_fetched_time"]=>
string(1) "0"
["proto_binary_fetched_datetime"]=>
string(1) "0"
["proto_binary_fetched_timestamp"]=>
string(1) "0"
["proto_binary_fetched_string"]=>
string(1) "0"
["proto_binary_fetched_blob"]=>
string(1) "0"
["proto_binary_fetched_enum"]=>
string(1) "0"
["proto_binary_fetched_set"]=>
string(1) "0"
["proto_binary_fetched_geometry"]=>
string(1) "0"
["proto_binary_fetched_other"]=>
string(1) "0"
["init_command_executed_count"]=>
string(1) "0"
["init_command_failed_count"]=>
string(1) "0"
["com_quit"]=>
string(1) "0"
```

```
["com_init_db"]=>
string(1) "0"
["com_query"]=>
string(1) "0"
["com_field_list"]=>
string(1) "0"
["com_create_db"]=>
string(1) "0"
["com_drop_db"]=>
string(1) "0"
["com_refresh"]=>
string(1) "0"
["com_shutdown"]=>
string(1) "0"
["com_statistics"]=>
string(1) "0"
["com_process_info"]=>
string(1) "0"
["com_connect"]=>
string(1) "0"
["com_process_kill"]=>
string(1) "0"
["com_debug"]=>
string(1) "0"
["com_ping"]=>
string(1) "0"
["com_time"]=>
string(1) "0"
["com_delayed_insert"]=>
string(1) "0"
["com_change_user"]=>
string(1) "0"
["com_binlog_dump"]=>
string(1) "0"
["com_table_dump"]=>
string(1) "0"
["com_connect_out"]=>
string(1) "0"
["com_register_slave"]=>
string(1) "0"
["com_stmt_prepare"]=>
string(1) "0"
["com_stmt_execute"]=>
string(1) "0"
["com_stmt_send_long_data"]=>
string(1) "0"
["com_stmt_close"]=>
string(1) "0"
["com_stmt_reset"]=>
string(1) "0"
["com_stmt_set_option"]=>
string(1) "0"
["com_stmt_fetch"]=>
string(1) "0"
["com_deamon"]=>
string(1) "0"
["bytes_received_real_data_normal"]=>
string(1) "0"
["bytes_received_real_data_ps"]=>
string(1) "0"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_get_connection_stats](#)

6.9.7.21 MysqlndUhConnection::getThreadId

Copyright 1997-2014 the PHP Documentation Group.

- `MysqlndUhConnection::getThreadId`

Returns the thread ID for the current connection

Description

```
public int MysqlndUhConnection::getThreadId(
    mysqlnd_connection connection);
```

Returns the thread ID for the current connection.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Connection thread id.

Examples

Example 6.363 MysqlndUhConnection::getThreadId example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getThreadId($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getThreadId($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->thread_id);
?>
```

The above example will output:

```
proxy::getThreadId(array (
    0 => NULL,
))
proxy::getThreadId returns 27646
int(27646)
```

See Also

`mysqlnd_uh_set_connection_proxy`
`mysqli_thread_id`
`mysql_thread_id`

6.9.7.22 MysqlndUhConnection::getWarningCount

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::getWarningCount](#)

Returns the number of warnings from the last query for the given link

Description

```
public int MysqlndUhConnection::getWarningCount(
    mysqlnd_connection connection);
```

Returns the number of warnings from the last query for the given link.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

Number of warnings.

Examples

Example 6.364 [MysqlndUhConnection::getWarningCount](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function getWarningCount($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::getWarningCount($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
var_dump($mysqli->warning_count);
?>
```

The above example will output:

```
proxy::getWarningCount(array (
    0 => NULL,
))
proxy::getWarningCount returns 0
int(0)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_warning_count](#)

6.9.7.23 [MysqlndUhConnection::init](#)

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::init](#)

Initialize mysqlnd connection

Description

```
public bool MysqlndUhConnection::init(
    mysqlnd_connection connection);
```

Initialize mysqlnd connection. This is an mysqlnd internal call to initialize the connection object.

Note

Failing to call the parent implementation may cause memory leaks or crash PHP. This is not considered a bug. Please, keep in mind that the [mysqlnd](#) library functions have never been designed to be exposed to the user space.

Parameters

[connection](#)

Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples**Example 6.365 [MysqlndUhConnection::init](#) example**

```
<?php
class proxy extends MysqlndUhConnection {
    public function init($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::init($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
?>
```

The above example will output:

```
proxy::init(array (
    0 => NULL,
))
proxy::init returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.24 [MysqlndUhConnection::killConnection](#)

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::killConnection](#)

Asks the server to kill a MySQL thread

Description

```
public bool MysqlndUhConnection::killConnection(
    mysqlnd_connection connection,
```

```
int pid);
```

Asks the server to kill a MySQL thread.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

[pid](#) Thread Id of the connection to be killed.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.366 [MysqlndUhConnection::kill](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function killConnection($res, $pid) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::killConnection($res, $pid);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->kill($mysqli->thread_id);
?>
```

The above example will output:

```
proxy::killConnection(array (
    0 => NULL,
    1 => 27650,
))
proxy::killConnection returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_kill](#)

6.9.7.25 [MysqlndUhConnection::listFields](#)

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::listFields](#)

List MySQL table fields

Description

```
public array MysqlndUhConnection::listFields(
    mysqlnd_connection connection,
    string table,
    string achtung_wild);
```

List MySQL table fields.

Warning

This function is currently not documented; only its argument list is available.

Parameters

<i>connection</i>	Mysqlnd connection handle. Do not modify!
<i>table</i>	The name of the table that's being queried.
<i>pattern</i>	Name pattern.

Return Values**Examples****Example 6.367 MysqlndUhConnection::listFields example**

```
<?php
class proxy extends MysqlndUhConnection {
    public function listFields($res, $table, $pattern) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::listFields($res, $table, $pattern);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysql = mysql_connect("localhost", "root", "");
mysql_select_db("test", $mysql);
mysql_query("DROP TABLE IF EXISTS test_a", $mysql);
mysql_query("CREATE TABLE test_a(id INT, col1 VARCHAR(255))", $mysql);
$res = mysql_list_fields("test", "test_a", $mysql);
printf("num_rows = %d\n", mysql_num_rows($res));
while ($row = mysql_fetch_assoc($res))
    var_dump($row);
?>
```

The above example will output:

```
proxy::listFields(array (
  0 => NULL,
  1 => 'test_a',
  2 => '',
))
proxy::listFields returns NULL
num_rows = 0
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysql_list_fields](#)

6.9.7.26 MysqlndUhConnection::listMethod

Copyright 1997-2014 the PHP Documentation Group.

- `MysqlndUhConnection::listMethod`

Wrapper for assorted list commands

Description

```
public void MysqlndUhConnection::listMethod(
    mysqlnd_connection connection,
    string query,
    string achtung_wild,
    string par1);
```

Wrapper for assorted list commands.

Warning

This function is currently not documented; only its argument list is available.

Parameters

connection Mysqlnd connection handle. Do not modify!

query SHOW command to be executed.

achtung_wild

par1

Return Values

Return Values

TODO

Examples

Example 6.368 `MysqlndUhConnection::listMethod` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function listMethod($res, $query, $pattern, $par1) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::listMethod($res, $query, $pattern, $par1);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysql = mysql_connect("localhost", "root", "");
$res = mysql_list_dbs($mysql);
printf("num_rows = %d\n", mysql_num_rows($res));
while ($row = mysql_fetch_assoc($res))
    var_dump($row);
?>
```

The above example will output:

```
proxy::listMethod(array (
  0 => NULL,
  1 => 'SHOW DATABASES',
```

```
    2 => '',
    3 => '',
))
proxy::listMethod returns NULL
num_rows = 6
array(1) {
    ["Database"]=>
        string(18) "information_schema"
}
array(1) {
    ["Database"]=>
        string(5) "mysql"
}
array(1) {
    ["Database"]=>
        string(8) "oxid_new"
}
array(1) {
    ["Database"]=>
        string(7) "phpptest"
}
array(1) {
    ["Database"]=>
        string(7) "pushphp"
}
array(1) {
    ["Database"]=>
        string(4) "test"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysql_list_dbs](#)

6.9.7.27 MysqlndUhConnection::moreResults

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::moreResults](#)

Check if there are any more query results from a multi query

Description

```
public bool MysqlndUhConnection::moreResults(
    mysqlnd_connection connection);
```

Check if there are any more query results from a multi query.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.369 [MysqlndUhConnection::moreResults](#) example

```
<?php
class proxy extends MysqlndUhConnection {
```

```
public function moreResults($res) {
    printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
    $ret = parent::moreResults($res);
    printf("%s returns %s\n", __METHOD__, var_export($ret, true));
    return $ret;
}
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->multi_query("SELECT 1 AS _one; SELECT 2 AS _two");
do {
    $res = $mysqli->store_result();
    var_dump($res->fetch_assoc());
    printf("%s\n", str_repeat("-", 40));
} while ($mysqli->more_results() && $mysqli->next_result());
?>
```

The above example will output:

```
array(1) {
    ["_one"]=>
    string(1) "1"
}
-----
proxy::moreResults(array (
    0 => NULL,
))
proxy::moreResults returns true
proxy::moreResults(array (
    0 => NULL,
))
proxy::moreResults returns true
array(1) {
    ["_two"]=>
    string(1) "2"
}
-----
proxy::moreResults(array (
    0 => NULL,
))
proxy::moreResults returns false
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_more_results](#)

6.9.7.28 MysqlndUhConnection::nextResult

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::nextResult](#)

Prepare next result from multi_query

Description

```
public bool MysqlndUhConnection::nextResult(
    mysqlnd_connection connection);
```

Prepare next result from multi_query.

Parameters

connection

Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.370 `MysqlndUhConnection::nextResult` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function nextResult($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::nextResult($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->multi_query("SELECT 1 AS _one; SELECT 2 AS _two");
do {
    $res = $mysqli->store_result();
    var_dump($res->fetch_assoc());
    printf("%s\n", str_repeat("-", 40));
} while ($mysqli->more_results() && $mysqli->next_result());
?>
```

The above example will output:

```
array(1) {
    ["_one"]=>
    string(1) "1"
}
-----
proxy::nextResult(array (
    0 => NULL,
))
proxy::nextResult returns true
array(1) {
    ["_two"]=>
    string(1) "2"
}
-----
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_next_result](#)

6.9.7.29 `MysqlndUhConnection::ping`

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::ping](#)

Pings a server connection, or tries to reconnect if the connection has gone down

Description


```
public bool MysqlndUhConnection::ping(
    mysqlnd_connection connection);
```

Pings a server connection, or tries to reconnect if the connection has gone down.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.371 [MysqlndUhConnection::ping](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function ping($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::ping($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->ping();
?>
```

The above example will output:

```
proxy::ping(array (
    0 => NULL,
))
proxy::ping returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_ping](#)
[mysql_ping](#)

6.9.7.30 [MysqlndUhConnection::query](#)

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::query](#)

Performs a query on the database

Description

```
public bool MysqlndUhConnection::query(
    mysqlnd_connection connection,
    string query);
```

Performs a query on the database (COM_QUERY).

Parameters

<i>connection</i>	Mysqlnd connection handle. Do not modify!
<i>query</i>	The query string.

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples**Example 6.372 MysqlndUhConnection::query example**

```
<?php
class proxy extends MysqlndUhConnection {
    public function query($res, $query) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $query = "SELECT 'How about query rewriting?'";
        $ret = parent::query($res, $query);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$res = $mysqli->query("SELECT 'welcome mysqlnd_uh!' FROM DUAL");
var_dump($res->fetch_assoc());
?>
```

The above example will output:

```
proxy::query(array (
  0 => NULL,
  1 => 'SELECT \'welcome mysqlnd_uh!\' FROM DUAL',
))
proxy::query returns true
array(1) {
  ["How about query rewriting?"]=>
  string(26) "How about query rewriting?"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_query](#)
[mysql_query](#)

6.9.7.31 MysqlndUhConnection::queryReadResultsetHeader

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::queryReadResultsetHeader](#)

Read a result set header

Description

```
public bool MysqlndUhConnection::queryReadResultsetHeader(
    mysqlnd_connection connection,
```

```
mysqlnd_statement mysqlnd_stmt);
```

Read a result set header.

Parameters

<i>connection</i>	Mysqlnd connection handle. Do not modify!
<i>mysqlnd_stmt</i>	Mysqlnd statement handle. Do not modify! Set to NULL , if function is not used in the context of a prepared statement.

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples

Example 6.373 **MysqlndUhConnection::queryReadResultsetHeader** example

```
<?php
class proxy extends MysqlndUhConnection {
    public function queryReadResultsetHeader($res, $stmt) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::queryReadResultsetHeader($res, $stmt);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$res = $mysqli->query("SELECT 'Welcome mysqlnd_uh!' FROM DUAL");
var_dump($res->fetch_assoc());
?>
```

The above example will output:

```
proxy::queryReadResultsetHeader(array (
  0 => NULL,
  1 => NULL,
))
proxy::queryReadResultsetHeader returns true
array(1) {
  ["Welcome mysqlnd_uh!"]=>
    string(19) "Welcome mysqlnd_uh!"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.32 **MysqlndUhConnection::reapQuery**

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::reapQuery](#)

Get result from async query

Description

```
public bool MysqlndUhConnection::reapQuery(
    mysqlnd_connection connection);
```

Get result from async query.

Parameters

connection

Mysqlnd connection handle. Do not modify!

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples

Example 6.374 `MysqlndUhConnection::reapQuery` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function reapQuery($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::reapQuery($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}

mysqlnd_uh_set_connection_proxy(new proxy());
$conn1 = new mysqli("localhost", "root", "", "test");
$conn2 = new mysqli("localhost", "root", "", "test");
$conn1->query("SELECT 1 as 'one', SLEEP(1) AS _sleep FROM DUAL", MYSQLI_ASYNC | MYSQLI_USE_RESULT);
$conn2->query("SELECT 1.1 as 'one dot one' FROM DUAL", MYSQLI_ASYNC | MYSQLI_USE_RESULT);
$links = array(
    $conn1->thread_id => array('link' => $conn1, 'processed' => false),
    $conn2->thread_id => array('link' => $conn2, 'processed' => false)
);
$saved_errors = array();
do {
    $poll_links = $poll_errors = $poll_reject = array();
    foreach ($links as $thread_id => $link) {
        if (!$link['processed']) {
            $poll_links[] = $link['link'];
            $poll_errors[] = $link['link'];
            $poll_reject[] = $link['link'];
        }
    }
    if (0 == count($poll_links))
        break;
    if (0 == ($num_ready = mysqli_poll($poll_links, $poll_errors, $poll_reject, 0, 200000)))
        continue;
    if (!empty($poll_errors)) {
        die(var_dump($poll_errors));
    }
    foreach ($poll_links as $link) {
        $thread_id = mysqli_thread_id($link);
        $links[$thread_id]['processed'] = true;
        if (is_object($res = mysqli_reap_async_query($link))) {
            // result set object
            while ($row = mysqli_fetch_assoc($res)) {
                // eat up all results
                var_dump($row);
            }
            mysqli_free_result($res);
        } else {
            // either there is no result (no SELECT) or there is an error
            if (mysqli_errno($link) > 0) {
                $saved_errors[$thread_id] = mysqli_errno($link);
                printf("%s' caused %d\n", $links[$thread_id]['query'], mysqli_errno($link));
            }
        }
    }
}
```

```
}  
} while (true);  
?>
```

The above example will output:

```
proxy::reapQuery(array (  
    0 => NULL,  
))  
proxy::reapQuery returns true  
array(1) {  
    ["one dot one"]=>  
        string(3) "1.1"  
}  
proxy::reapQuery(array (  
    0 => NULL,  
))  
proxy::reapQuery returns true  
array(2) {  
    ["one"]=>  
        string(1) "1"  
    ["_sleep"]=>  
        string(1) "0"  
}  
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_real_async_query](#)

6.9.7.33 MysqlndUhConnection::refreshServer

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::refreshServer](#)

Flush or reset tables and caches

Description

```
public bool MysqlndUhConnection::refreshServer(  
    mysqlnd_connection connection,  
    int options);
```

Flush or reset tables and caches.

Warning

This function is currently not documented; only its argument list is available.

Parameters

<i>connection</i>	Mysqlnd connection handle. Do not modify!
<i>options</i>	What to refresh.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.375 MysqlndUhConnection::refreshServer example

```
<?php
class proxy extends MysqlndUhConnection {
    public function refreshServer($res, $option) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::refreshServer($res, $option);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
mysqli_refresh($mysqli, 1);
?>
```

The above example will output:

```
proxy::refreshServer(array (
    0 => NULL,
    1 => 1,
))
proxy::refreshServer returns false
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.34 MysqlndUhConnection::restartPSession

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::restartPSession](#)

Restart a persistent mysqlnd connection

Description

```
public bool MysqlndUhConnection::restartPSession(
    mysqlnd_connection connection);
```

Restart a persistent mysqlnd connection.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples**Example 6.376 MysqlndUhConnection::restartPSession example**

```
<?php
class proxy extends MysqlndUhConnection {
    public function ping($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
    }
}
```

```
$ret = parent::ping($res);
printf("%s returns %s\n", __METHOD__, var_export($ret, true));
return $ret;
}
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->ping();
?>
```

The above example will output:

```
proxy::restartPSession(array (
    0 => NULL,
))
proxy::restartPSession returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.35 MysqlndUhConnection::selectDb

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::selectDb](#)

Selects the default database for database queries

Description

```
public bool MysqlndUhConnection::selectDb(
    mysqlnd_connection connection,
    string database);
```

Selects the default database for database queries.

Parameters

connection MySQLnd connection handle. Do not modify!

database The database name.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.377 MysqlndUhConnection::selectDb example

```
<?php
class proxy extends MysqlndUhConnection {
    public function selectDb($res, $database) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::selectDb($res, $database);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
```

```
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->select_db("mysql");
?>
```

The above example will output:

```
proxy::selectDb(array (
    0 => NULL,
    1 => 'mysql',
))
proxy::selectDb returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_select_db](#)
[mysql_select_db](#)

6.9.7.36 MysqlndUhConnection::sendClose

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::sendClose](#)

Sends a close command to MySQL

Description

```
public bool MysqlndUhConnection::sendClose(
    mysqlnd_connection connection);
```

Sends a close command to MySQL.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.378 MysqlndUhConnection::sendClose example

```
<?php
class proxy extends MysqlndUhConnection {
    public function sendClose($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::sendClose($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->close();
?>
```


The above example will output:

```
proxy::sendClose(array (
    0 => NULL,
))
proxy::sendClose returns true
proxy::sendClose(array (
    0 => NULL,
))
proxy::sendClose returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.37 MysqlndUhConnection::sendQuery

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::sendQuery](#)

Sends a query to MySQL

Description

```
public bool MysqlndUhConnection::sendQuery(
    mysqlnd_connection connection,
    string query);
```

Sends a query to MySQL.

Parameters

connection MySQLnd connection handle. Do not modify!

query The query string.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.379 MysqlndUhConnection::sendQuery example

```
<?php
class proxy extends MysqlndUhConnection {
    public function sendQuery($res, $query) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::sendQuery($res, $query);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("SELECT 1");
?>
```

The above example will output:

```
proxy::sendQuery(array (
  0 => NULL,
  1 => 'SELECT 1',
))
proxy::sendQuery returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.38 MysqlndUhConnection::serverDumpDebugInformation

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::serverDumpDebugInformation](#)

Dump debugging information into the log for the MySQL server

Description

```
public bool MysqlndUhConnection::serverDumpDebugInformation(
    mysqlnd_connection connection);
```

Dump debugging information into the log for the MySQL server.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.380 MysqlndUhConnection::serverDumpDebugInformation example

```
<?php
class proxy extends MysqlndUhConnection {
    public function serverDumpDebugInformation($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::serverDumpDebugInformation($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->dump_debug_info();
?>
```

The above example will output:

```
proxy::serverDumpDebugInformation(array (
  0 => NULL,
))
```

```
proxy::serverDumpDebugInformation returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_dump_debug_info](#)

6.9.7.39 MysqlndUhConnection::setAutocommit

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::setAutocommit](#)

Turns on or off auto-committing database modifications

Description

```
public bool MysqlndUhConnection::setAutocommit(
    mysqlnd_connection connection,
    int mode);
```

Turns on or off auto-committing database modifications

Parameters

<i>connection</i>	Mysqlnd connection handle. Do not modify!
<i>mode</i>	Whether to turn on auto-commit or not.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.381 MysqlndUhConnection::setAutocommit example

```
<?php
class proxy extends MysqlndUhConnection {
    public function setAutocommit($res, $mode) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::setAutocommit($res, $mode);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->autocommit(false);
$mysqli->autocommit(true);
?>
```

The above example will output:

```
proxy::setAutocommit(array (
    0 => NULL,
    1 => 0,
))
proxy::setAutocommit returns true
proxy::setAutocommit(array (
```

```
    0 => NULL,  
    1 => 1,  
))  
proxy::setAutocommit returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_autocommit](#)

6.9.7.40 MysqlndUhConnection::setCharset

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::setCharset](#)

Sets the default client character set

Description

```
public bool MysqlndUhConnection::setCharset(  
    mysqlnd_connection connection,  
    string charset);
```

Sets the default client character set.

Parameters

connection Mysqlnd connection handle. Do not modify!

charset The charset to be set as default.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.382 MysqlndUhConnection::setCharset example

```
<?php  
class proxy extends MysqlndUhConnection {  
    public function setCharset($res, $charset) {  
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));  
        $ret = parent::setCharset($res, $charset);  
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));  
        return $ret;  
    }  
}  
mysqlnd_uh_set_connection_proxy(new proxy());  
$mysqli = new mysqli("localhost", "root", "", "test");  
$mysqli->set_charset("latin1");  
?>
```

The above example will output:

```
proxy::setCharset(array (  
    0 => NULL,  
    1 => 'latin1',  
))
```

```
proxy::setCharset returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_set_charset](#)

6.9.7.41 MysqlndUhConnection::setClientOption

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::setClientOption](#)

Sets a client option

Description

```
public bool MysqlndUhConnection::setClientOption(  
    mysqlnd_connection connection,  
    int option,  
    int value);
```

Sets a client option.

Parameters

<i>connection</i>	Mysqlnd connection handle. Do not modify!
<i>option</i>	The option to be set.
<i>value</i>	Optional option value, if required.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.383 MysqlndUhConnection::setClientOption example

```
<?php  
function client_option_to_string($option) {  
    static $mapping = array(  
        MYSQLND_UH_MYSQLND_OPTION_OPT_CONNECT_TIMEOUT => "MYSQLND_UH_MYSQLND_OPTION_OPT_CONNECT_TIMEOUT",  
        MYSQLND_UH_MYSQLND_OPTION_OPT_COMPRESS => "MYSQLND_UH_MYSQLND_OPTION_OPT_COMPRESS",  
        MYSQLND_UH_MYSQLND_OPTION_OPT_NAMED_PIPE => "MYSQLND_UH_MYSQLND_OPTION_OPT_NAMED_PIPE",  
        MYSQLND_UH_MYSQLND_OPTION_INIT_COMMAND => "MYSQLND_UH_MYSQLND_OPTION_INIT_COMMAND",  
        MYSQLND_UH_MYSQLND_READ_DEFAULT_FILE => "MYSQLND_UH_MYSQLND_READ_DEFAULT_FILE",  
        MYSQLND_UH_MYSQLND_READ_DEFAULT_GROUP => "MYSQLND_UH_MYSQLND_READ_DEFAULT_GROUP",  
        MYSQLND_UH_MYSQLND_SET_CHARSET_DIR => "MYSQLND_UH_MYSQLND_SET_CHARSET_DIR",  
        MYSQLND_UH_MYSQLND_SET_CHARSET_NAME => "MYSQLND_UH_MYSQLND_SET_CHARSET_NAME",  
        MYSQLND_UH_MYSQLND_OPT_LOCAL_INFILE => "MYSQLND_UH_MYSQLND_OPT_LOCAL_INFILE",  
        MYSQLND_UH_MYSQLND_OPT_PROTOCOL => "MYSQLND_UH_MYSQLND_OPT_PROTOCOL",  
        MYSQLND_UH_MYSQLND_SHARED_MEMORY_BASE_NAME => "MYSQLND_UH_MYSQLND_SHARED_MEMORY_BASE_NAME",  
        MYSQLND_UH_MYSQLND_OPT_READ_TIMEOUT => "MYSQLND_UH_MYSQLND_OPT_READ_TIMEOUT",  
        MYSQLND_UH_MYSQLND_OPT_WRITE_TIMEOUT => "MYSQLND_UH_MYSQLND_OPT_WRITE_TIMEOUT",  
        MYSQLND_UH_MYSQLND_OPT_USE_RESULT => "MYSQLND_UH_MYSQLND_OPT_USE_RESULT",  
        MYSQLND_UH_MYSQLND_OPT_USE_REMOTE_CONNECTION => "MYSQLND_UH_MYSQLND_OPT_USE_REMOTE_CONNECTION",  
        MYSQLND_UH_MYSQLND_OPT_USE_EMBEDDED_CONNECTION => "MYSQLND_UH_MYSQLND_OPT_USE_EMBEDDED_CONNECTION",  
        MYSQLND_UH_MYSQLND_OPT_GUESS_CONNECTION => "MYSQLND_UH_MYSQLND_OPT_GUESS_CONNECTION",  
        MYSQLND_UH_MYSQLND_SET_CLIENT_IP => "MYSQLND_UH_MYSQLND_SET_CLIENT_IP",  
        MYSQLND_UH_MYSQLND_SECURE_AUTH => "MYSQLND_UH_MYSQLND_SECURE_AUTH",  
        MYSQLND_UH_MYSQLND_REPORT_DATA_TRUNCATION => "MYSQLND_UH_MYSQLND_REPORT_DATA_TRUNCATION",  
        MYSQLND_UH_MYSQLND_OPT_RECONNECT => "MYSQLND_UH_MYSQLND_OPT_RECONNECT",
```

```

MYSQLND_UH_MYSQLND_OPT_SSL_VERIFY_SERVER_CERT => "MYSQLND_UH_MYSQLND_OPT_SSL_VERIFY_SERVER_CERT",
MYSQLND_UH_MYSQLND_OPT_NET_CMD_BUFFER_SIZE => "MYSQLND_UH_MYSQLND_OPT_NET_CMD_BUFFER_SIZE",
MYSQLND_UH_MYSQLND_OPT_NET_READ_BUFFER_SIZE => "MYSQLND_UH_MYSQLND_OPT_NET_READ_BUFFER_SIZE",
MYSQLND_UH_MYSQLND_OPT_SSL_KEY => "MYSQLND_UH_MYSQLND_OPT_SSL_KEY",
MYSQLND_UH_MYSQLND_OPT_SSL_CERT => "MYSQLND_UH_MYSQLND_OPT_SSL_CERT",
MYSQLND_UH_MYSQLND_OPT_SSL_CA => "MYSQLND_UH_MYSQLND_OPT_SSL_CA",
MYSQLND_UH_MYSQLND_OPT_SSL_CAPATH => "MYSQLND_UH_MYSQLND_OPT_SSL_CAPATH",
MYSQLND_UH_MYSQLND_OPT_SSL_CIPHER => "MYSQLND_UH_MYSQLND_OPT_SSL_CIPHER",
MYSQLND_UH_MYSQLND_OPT_SSL_PASSPHRASE => "MYSQLND_UH_MYSQLND_OPT_SSL_PASSPHRASE",
MYSQLND_UH_SERVER_OPTION_PLUGIN_DIR => "MYSQLND_UH_SERVER_OPTION_PLUGIN_DIR",
MYSQLND_UH_SERVER_OPTION_DEFAULT_AUTH => "MYSQLND_UH_SERVER_OPTION_DEFAULT_AUTH",
MYSQLND_UH_SERVER_OPTION_SET_CLIENT_IP => "MYSQLND_UH_SERVER_OPTION_SET_CLIENT_IP"
);
if (version_compare(PHP_VERSION, '5.3.99-dev', '>')) {
    $mapping[MYSQLND_UH_MYSQLND_OPT_MAX_ALLOWED_PACKET] = "MYSQLND_UH_MYSQLND_OPT_MAX_ALLOWED_PACKET";
    $mapping[MYSQLND_UH_MYSQLND_OPT_AUTH_PROTOCOL] = "MYSQLND_UH_MYSQLND_OPT_AUTH_PROTOCOL";
}
if (defined("MYSQLND_UH_MYSQLND_OPT_INT_AND_FLOAT_NATIVE")) {
    /* special mysqlnd build */
    $mapping["MYSQLND_UH_MYSQLND_OPT_INT_AND_FLOAT_NATIVE"] = "MYSQLND_UH_MYSQLND_OPT_INT_AND_FLOAT_NATIVE";
}
return (isset($mapping[$option])) ? $mapping[$option] : 'unknown';
}
class proxy extends MysqlndUhConnection {
    public function setClientOption($res, $option, $value) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("Option '%s' set to %s\n", client_option_to_string($option), var_export($value, true));
        $ret = parent::setClientOption($res, $option, $value);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
?>

```

The above example will output:

```

proxy::setClientOption(array (
    0 => NULL,
    1 => 210,
    2 => 3221225472,
))
Option 'MYSQLND_UH_MYSQLND_OPT_MAX_ALLOWED_PACKET' set to 3221225472
proxy::setClientOption returns true
proxy::setClientOption(array (
    0 => NULL,
    1 => 211,
    2 => 'mysql_native_password',
))
Option 'MYSQLND_UH_MYSQLND_OPT_AUTH_PROTOCOL' set to 'mysql_native_password'
proxy::setClientOption returns true
proxy::setClientOption(array (
    0 => NULL,
    1 => 8,
    2 => 1,
))
Option 'MYSQLND_UH_MYSQLND_OPT_LOCAL_INFILE' set to 1
proxy::setClientOption returns true

```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_real_connect](#)
[mysqli_options](#)

6.9.7.42 MysqlndUhConnection::setServerOption

Copyright 1997-2014 the PHP Documentation Group.

- **MysqlndUhConnection::setServerOption**

Sets a server option

Description

```
public void MysqlndUhConnection::setServerOption(
    mysqlnd_connection connection,
    int option);
```

Sets a server option.

Parameters

connection Mysqlnd connection handle. Do not modify!

option The option to be set.

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples

Example 6.384 MysqlndUhConnection::setServerOption example

```
<?php
function server_option_to_string($option) {
    $ret = 'unknown';
    switch ($option) {
        case MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_ON:
            $ret = 'MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_ON';
            break;
        case MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_OFF:
            $ret = 'MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_ON';
            break;
    }
    return $ret;
}
class proxy extends MysqlndUhConnection {
    public function setServerOption($res, $option) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("Option '%s' set\n", server_option_to_string($option));
        $ret = parent::setServerOption($res, $option);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->multi_query("SELECT 1; SELECT 2");
?>
```

The above example will output:

```
proxy::setServerOption(array (
    0 => NULL,
    1 => 0,
```

```
))  
Option 'MYSQLND_UH_SERVER_OPTION_MULTI_STATEMENTS_ON' set  
proxy::setServerOption returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_real_connect](#)
[mysqli_options](#)
[mysqli_multi_query](#)

6.9.7.43 MysqlndUhConnection::shutdownServer

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::shutdownServer](#)

The shutdownServer purpose

Description

```
public void MysqlndUhConnection::shutdownServer(  
    string MYSQLND_UH_RES_MYSQLND_NAME,  
    string "level");
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

MYSQLND_UH_RES_MYSQLND_NAME

"level"

Return Values

6.9.7.44 MysqlndUhConnection::simpleCommand

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::simpleCommand](#)

Sends a basic COM_* command

Description

```
public bool MysqlndUhConnection::simpleCommand(  
    mysqlnd_connection connection,  
    int command,  
    string arg,  
    int ok_packet,  
    bool silent,  
    bool ignore_upsert_status);
```

Sends a basic COM_* command to MySQL.

Parameters

connection

Mysqlnd connection handle. Do not modify!

<i>command</i>	The COM command to be send.
<i>arg</i>	Optional COM command arguments.
<i>ok_packet</i>	The OK packet type.
<i>silent</i>	Whether mysqlnd may emit errors.
<i>ignore_upsert_status</i>	Whether to ignore UPDATE/INSERT status.

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples

Example 6.385 MysqlndUhConnection::simpleCommand example

```
<?php
function server_cmd_2_string($command) {
    $mapping = array(
        MYSQLND_UH_MYSQLND_COM_SLEEP => "MYSQLND_UH_MYSQLND_COM_SLEEP",
        MYSQLND_UH_MYSQLND_COM_QUIT => "MYSQLND_UH_MYSQLND_COM_QUIT",
        MYSQLND_UH_MYSQLND_COM_INIT_DB => "MYSQLND_UH_MYSQLND_COM_INIT_DB",
        MYSQLND_UH_MYSQLND_COM_QUERY => "MYSQLND_UH_MYSQLND_COM_QUERY",
        MYSQLND_UH_MYSQLND_COM_FIELD_LIST => "MYSQLND_UH_MYSQLND_COM_FIELD_LIST",
        MYSQLND_UH_MYSQLND_COM_CREATE_DB => "MYSQLND_UH_MYSQLND_COM_CREATE_DB",
        MYSQLND_UH_MYSQLND_COM_DROP_DB => "MYSQLND_UH_MYSQLND_COM_DROP_DB",
        MYSQLND_UH_MYSQLND_COM_REFRESH => "MYSQLND_UH_MYSQLND_COM_REFRESH",
        MYSQLND_UH_MYSQLND_COM_SHUTDOWN => "MYSQLND_UH_MYSQLND_COM_SHUTDOWN",
        MYSQLND_UH_MYSQLND_COM_STATISTICS => "MYSQLND_UH_MYSQLND_COM_STATISTICS",
        MYSQLND_UH_MYSQLND_COM_PROCESS_INFO => "MYSQLND_UH_MYSQLND_COM_PROCESS_INFO",
        MYSQLND_UH_MYSQLND_COM_CONNECT => "MYSQLND_UH_MYSQLND_COM_CONNECT",
        MYSQLND_UH_MYSQLND_COM_PROCESS_KILL => "MYSQLND_UH_MYSQLND_COM_PROCESS_KILL",
        MYSQLND_UH_MYSQLND_COM_DEBUG => "MYSQLND_UH_MYSQLND_COM_DEBUG",
        MYSQLND_UH_MYSQLND_COM_PING => "MYSQLND_UH_MYSQLND_COM_PING",
        MYSQLND_UH_MYSQLND_COM_TIME => "MYSQLND_UH_MYSQLND_COM_TIME",
        MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT => "MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT",
        MYSQLND_UH_MYSQLND_COM_CHANGE_USER => "MYSQLND_UH_MYSQLND_COM_CHANGE_USER",
        MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP => "MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP",
        MYSQLND_UH_MYSQLND_COM_TABLE_DUMP => "MYSQLND_UH_MYSQLND_COM_TABLE_DUMP",
        MYSQLND_UH_MYSQLND_COM_CONNECT_OUT => "MYSQLND_UH_MYSQLND_COM_CONNECT_OUT",
        MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVE => "MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVE",
        MYSQLND_UH_MYSQLND_COM_STMT_PREPARE => "MYSQLND_UH_MYSQLND_COM_STMT_PREPARE",
        MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE => "MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE",
        MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA => "MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA",
        MYSQLND_UH_MYSQLND_COM_STMT_CLOSE => "MYSQLND_UH_MYSQLND_COM_STMT_CLOSE",
        MYSQLND_UH_MYSQLND_COM_STMT_RESET => "MYSQLND_UH_MYSQLND_COM_STMT_RESET",
        MYSQLND_UH_MYSQLND_COM_SET_OPTION => "MYSQLND_UH_MYSQLND_COM_SET_OPTION",
        MYSQLND_UH_MYSQLND_COM_STMT_FETCH => "MYSQLND_UH_MYSQLND_COM_STMT_FETCH",
        MYSQLND_UH_MYSQLND_COM_DAEMON => "MYSQLND_UH_MYSQLND_COM_DAEMON",
        MYSQLND_UH_MYSQLND_COM_END => "MYSQLND_UH_MYSQLND_COM_END",
    );
    return (isset($mapping[$command])) ? $mapping[$command] : 'unknown';
}

function ok_packet_2_string($ok_packet) {
    $mapping = array(
        MYSQLND_UH_MYSQLND_PROT_GREET_PACKET => "MYSQLND_UH_MYSQLND_PROT_GREET_PACKET",
        MYSQLND_UH_MYSQLND_PROT_AUTH_PACKET => "MYSQLND_UH_MYSQLND_PROT_AUTH_PACKET",
        MYSQLND_UH_MYSQLND_PROT_OK_PACKET => "MYSQLND_UH_MYSQLND_PROT_OK_PACKET",
        MYSQLND_UH_MYSQLND_PROT_EOF_PACKET => "MYSQLND_UH_MYSQLND_PROT_EOF_PACKET",
        MYSQLND_UH_MYSQLND_PROT_CMD_PACKET => "MYSQLND_UH_MYSQLND_PROT_CMD_PACKET",
        MYSQLND_UH_MYSQLND_PROT_RSET_HEADER_PACKET => "MYSQLND_UH_MYSQLND_PROT_RSET_HEADER_PACKET",
        MYSQLND_UH_MYSQLND_PROT_RSET_FLD_PACKET => "MYSQLND_UH_MYSQLND_PROT_RSET_FLD_PACKET",
        MYSQLND_UH_MYSQLND_PROT_ROW_PACKET => "MYSQLND_UH_MYSQLND_PROT_ROW_PACKET",
        MYSQLND_UH_MYSQLND_PROT_STATS_PACKET => "MYSQLND_UH_MYSQLND_PROT_STATS_PACKET",
        MYSQLND_UH_MYSQLND_PREPARE_RESP_PACKET => "MYSQLND_UH_MYSQLND_PREPARE_RESP_PACKET",
        MYSQLND_UH_MYSQLND_CHG_USER_RESP_PACKET => "MYSQLND_UH_MYSQLND_CHG_USER_RESP_PACKET",
    );
}
```

```

    MYSQLND_UH_MYSQLND_PROT_LAST => "MYSQLND_UH_MYSQLND_PROT_LAST",
);
return (isset($mapping[$ok_packet])) ? $mapping[$ok_packet] : 'unknown';
}
class proxy extends MysqlndUhConnection {
    public function simpleCommand($conn, $command, $arg, $ok_packet, $silent, $ignore_upsert_status) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("Command '%s'\n", server_cmd_2_string($command));
        printf("OK packet '%s'\n", ok_packet_2_string($ok_packet));
        $ret = parent::simpleCommand($conn, $command, $arg, $ok_packet, $silent, $ignore_upsert_status);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->query("SELECT 1");
?>

```

The above example will output:

```

proxy::simpleCommand(array (
    0 => NULL,
    1 => 3,
    2 => 'SELECT 1',
    3 => 13,
    4 => false,
    5 => false,
))
Command 'MYSQLND_UH_MYSQLND_COM_QUERY'
OK packet 'MYSQLND_UH_MYSQLND_PROT_LAST'
proxy::simpleCommand returns true
:)proxy::simpleCommand(array (
    0 => NULL,
    1 => 1,
    2 => '',
    3 => 13,
    4 => true,
    5 => true,
))
Command 'MYSQLND_UH_MYSQLND_COM_QUIT'
OK packet 'MYSQLND_UH_MYSQLND_PROT_LAST'
proxy::simpleCommand returns true

```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.45 MysqlndUhConnection::simpleCommandHandleResponse

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::simpleCommandHandleResponse](#)

Process a response for a basic COM_* command send to the client

Description

```

public bool MysqlndUhConnection::simpleCommandHandleResponse(
    mysqlnd_connection connection,
    int ok_packet,
    bool silent,
    int command,
    bool ignore_upsert_status);

```

Process a response for a basic COM_* command send to the client.

Parameters

<i>connection</i>	Mysqlnd connection handle. Do not modify!
<i>ok_packet</i>	The OK packet type.
<i>silent</i>	Whether mysqlnd may emit errors.
<i>command</i>	The COM command to process results from.
<i>ignore_upsert_status</i>	Whether to ignore UPDATE/INSERT status.

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.386 `MysqlndUhConnection::simpleCommandHandleResponse` example

```
<?php
function server_cmd_2_string($command) {
    $mapping = array(
        MYSQLND_UH_MYSQLND_COM_SLEEP => "MYSQLND_UH_MYSQLND_COM_SLEEP",
        MYSQLND_UH_MYSQLND_COM_QUIT => "MYSQLND_UH_MYSQLND_COM_QUIT",
        MYSQLND_UH_MYSQLND_COM_INIT_DB => "MYSQLND_UH_MYSQLND_COM_INIT_DB",
        MYSQLND_UH_MYSQLND_COM_QUERY => "MYSQLND_UH_MYSQLND_COM_QUERY",
        MYSQLND_UH_MYSQLND_COM_FIELD_LIST => "MYSQLND_UH_MYSQLND_COM_FIELD_LIST",
        MYSQLND_UH_MYSQLND_COM_CREATE_DB => "MYSQLND_UH_MYSQLND_COM_CREATE_DB",
        MYSQLND_UH_MYSQLND_COM_DROP_DB => "MYSQLND_UH_MYSQLND_COM_DROP_DB",
        MYSQLND_UH_MYSQLND_COM_REFRESH => "MYSQLND_UH_MYSQLND_COM_REFRESH",
        MYSQLND_UH_MYSQLND_COM_SHUTDOWN => "MYSQLND_UH_MYSQLND_COM_SHUTDOWN",
        MYSQLND_UH_MYSQLND_COM_STATISTICS => "MYSQLND_UH_MYSQLND_COM_STATISTICS",
        MYSQLND_UH_MYSQLND_COM_PROCESS_INFO => "MYSQLND_UH_MYSQLND_COM_PROCESS_INFO",
        MYSQLND_UH_MYSQLND_COM_CONNECT => "MYSQLND_UH_MYSQLND_COM_CONNECT",
        MYSQLND_UH_MYSQLND_COM_PROCESS_KILL => "MYSQLND_UH_MYSQLND_COM_PROCESS_KILL",
        MYSQLND_UH_MYSQLND_COM_DEBUG => "MYSQLND_UH_MYSQLND_COM_DEBUG",
        MYSQLND_UH_MYSQLND_COM_PING => "MYSQLND_UH_MYSQLND_COM_PING",
        MYSQLND_UH_MYSQLND_COM_TIME => "MYSQLND_UH_MYSQLND_COM_TIME",
        MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT => "MYSQLND_UH_MYSQLND_COM_DELAYED_INSERT",
        MYSQLND_UH_MYSQLND_COM_CHANGE_USER => "MYSQLND_UH_MYSQLND_COM_CHANGE_USER",
        MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP => "MYSQLND_UH_MYSQLND_COM_BINLOG_DUMP",
        MYSQLND_UH_MYSQLND_COM_TABLE_DUMP => "MYSQLND_UH_MYSQLND_COM_TABLE_DUMP",
        MYSQLND_UH_MYSQLND_COM_CONNECT_OUT => "MYSQLND_UH_MYSQLND_COM_CONNECT_OUT",
        MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVE => "MYSQLND_UH_MYSQLND_COM_REGISTER_SLAVE",
        MYSQLND_UH_MYSQLND_COM_STMT_PREPARE => "MYSQLND_UH_MYSQLND_COM_STMT_PREPARE",
        MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE => "MYSQLND_UH_MYSQLND_COM_STMT_EXECUTE",
        MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA => "MYSQLND_UH_MYSQLND_COM_STMT_SEND_LONG_DATA",
        MYSQLND_UH_MYSQLND_COM_STMT_CLOSE => "MYSQLND_UH_MYSQLND_COM_STMT_CLOSE",
        MYSQLND_UH_MYSQLND_COM_STMT_RESET => "MYSQLND_UH_MYSQLND_COM_STMT_RESET",
        MYSQLND_UH_MYSQLND_COM_SET_OPTION => "MYSQLND_UH_MYSQLND_COM_SET_OPTION",
        MYSQLND_UH_MYSQLND_COM_STMT_FETCH => "MYSQLND_UH_MYSQLND_COM_STMT_FETCH",
        MYSQLND_UH_MYSQLND_COM_DAEMON => "MYSQLND_UH_MYSQLND_COM_DAEMON",
        MYSQLND_UH_MYSQLND_COM_END => "MYSQLND_UH_MYSQLND_COM_END",
    );
    return (isset($mapping[$command])) ? $mapping[$command] : 'unknown';
}

function ok_packet_2_string($ok_packet) {
    $mapping = array(
        MYSQLND_UH_MYSQLND_PROT_GREET_PACKET => "MYSQLND_UH_MYSQLND_PROT_GREET_PACKET",
        MYSQLND_UH_MYSQLND_PROT_AUTH_PACKET => "MYSQLND_UH_MYSQLND_PROT_AUTH_PACKET",
        MYSQLND_UH_MYSQLND_PROT_OK_PACKET => "MYSQLND_UH_MYSQLND_PROT_OK_PACKET",
        MYSQLND_UH_MYSQLND_PROT_EOF_PACKET => "MYSQLND_UH_MYSQLND_PROT_EOF_PACKET",
        MYSQLND_UH_MYSQLND_PROT_CMD_PACKET => "MYSQLND_UH_MYSQLND_PROT_CMD_PACKET",
        MYSQLND_UH_MYSQLND_PROT_RSET_HEADER_PACKET => "MYSQLND_UH_MYSQLND_PROT_RSET_HEADER_PACKET",
        MYSQLND_UH_MYSQLND_PROT_RSET_FLD_PACKET => "MYSQLND_UH_MYSQLND_PROT_RSET_FLD_PACKET",
    );
}
```

```

MYSQLND_UH_MYSQLND_PROT_ROW_PACKET => "MYSQLND_UH_MYSQLND_PROT_ROW_PACKET",
MYSQLND_UH_MYSQLND_PROT_STATS_PACKET => "MYSQLND_UH_MYSQLND_PROT_STATS_PACKET",
MYSQLND_UH_MYSQLND_PREPARE_RESP_PACKET => "MYSQLND_UH_MYSQLND_PREPARE_RESP_PACKET",
MYSQLND_UH_MYSQLND_CHG_USER_RESP_PACKET => "MYSQLND_UH_MYSQLND_CHG_USER_RESP_PACKET",
MYSQLND_UH_MYSQLND_PROT_LAST => "MYSQLND_UH_MYSQLND_PROT_LAST",
);
return (isset($mapping[$ok_packet])) ? $mapping[$ok_packet] : 'unknown';
}
class proxy extends MysqlndUhConnection {
    public function simpleCommandHandleResponse($conn, $ok_packet, $silent, $command, $ignore_upsert_status) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        printf("Command '%s'\n", server_cmd_2_string($command));
        printf("OK packet '%s'\n", ok_packet_2_string($ok_packet));
        $ret = parent::simpleCommandHandleResponse($conn, $ok_packet, $silent, $command, $ignore_upsert_status);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysql = mysql_connect("localhost", "root", "");
mysql_query("SELECT 1 FROM DUAL", $mysql);
?>

```

The above example will output:

```

proxy::simpleCommandHandleResponse(array (
    0 => NULL,
    1 => 5,
    2 => false,
    3 => 27,
    4 => true,
))
Command 'MYSQLND_UH_MYSQLND_COM_SET_OPTION'
OK packet 'MYSQLND_UH_MYSQLND_PROT_EOF_PACKET'
proxy::simpleCommandHandleResponse returns true

```

See Also

[mysqlnd_uh_set_connection_proxy](#)

6.9.7.46 MysqlndUhConnection::sslSet

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::sslSet](#)

Used for establishing secure connections using SSL

Description

```

public bool MysqlndUhConnection::sslSet(
    mysqlnd_connection connection,
    string key,
    string cert,
    string ca,
    string capath,
    string cipher);

```

Used for establishing secure connections using SSL.

Parameters

connection

Mysqlnd connection handle. Do not modify!

<i>key</i>	The path name to the key file.
<i>cert</i>	The path name to the certificate file.
<i>ca</i>	The path name to the certificate authority file.
<i>capath</i>	The pathname to a directory that contains trusted SSL CA certificates in PEM format.
<i>cipher</i>	A list of allowable ciphers to use for SSL encryption.

Return Values

Returns **TRUE** on success. Otherwise, returns **FALSE**

Examples

Example 6.387 **MysqlndUhConnection::sslSet** example

```
<?php
class proxy extends MysqlndUhConnection {
    public function sslSet($conn, $key, $cert, $ca, $capath, $cipher) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::sslSet($conn, $key, $cert, $ca, $capath, $cipher);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->ssl_set("key", "cert", "ca", "capath", "cipher");
?>
```

The above example will output:

```
proxy::sslSet(array (
    0 => NULL,
    1 => 'key',
    2 => 'cert',
    3 => 'ca',
    4 => 'capath',
    5 => 'cipher',
))
proxy::sslSet returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_ssl_set](#)

6.9.7.47 **MysqlndUhConnection::stmtInit**

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::stmtInit](#)

Initializes a statement and returns a resource for use with `mysqli_statement::prepare`

Description

```
public resource MysqlndUhConnection::stmtInit(
    mysqlnd_connection connection);
```

Initializes a statement and returns a resource for use with `mysqli_statement::prepare`.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Resource of type [Mysqlnd Prepared Statement \(internal only - you must not modify it!\)](#). The documentation may also refer to such resources using the alias name `mysqlnd_prepared_statement`.

Examples

Example 6.388 `MysqlndUhConnection::stmtInit` example

```
<?php
class proxy extends MysqlndUhConnection {
    public function stmtInit($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        var_dump($res);
        $ret = parent::stmtInit($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        var_dump($ret);
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$stmt = $mysqli->prepare("SELECT 1 AS _one FROM DUAL");
$stmt->execute();
$one = NULL;
$stmt->bind_result($one);
$stmt->fetch();
var_dump($one);
?>
```

The above example will output:

```
proxy::stmtInit(array (
    0 => NULL,
))
resource(19) of type (Mysqlnd Connection)
proxy::stmtInit returns NULL
resource(246) of type (Mysqlnd Prepared Statement (internal only - you must not modify it!))
int(1)
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_stmt_init](#)

6.9.7.48 `MysqlndUhConnection::storeResult`

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::storeResult](#)

Transfers a result set from the last query

Description

```
public resource MysqlndUhConnection::storeResult(
    mysqlnd_connection connection);
```

Transfers a result set from the last query.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Resource of type [Mysqlnd Resultset \(internal only - you must not modify it!\)](#). The documentation may also refer to such resources using the alias name [mysqlnd_resultset](#).

Examples

Example 6.389 [MysqlndUhConnection::storeResult](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function storeResult($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::storeResult($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        var_dump($ret);
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$res = $mysqli->query("SELECT 'Also called buffered result' AS _msg FROM DUAL");
var_dump($res->fetch_assoc());
$mysqli->real_query("SELECT 'Good morning!' AS _msg FROM DUAL");
$res = $mysqli->store_result();
var_dump($res->fetch_assoc());
?>
```

The above example will output:

```
proxy::storeResult(array (
    0 => NULL,
))
proxy::storeResult returns NULL
resource(475) of type (Mysqlnd Resultset (internal only - you must not modify it!))
array(1) {
    ["_msg"]=>
    string(27) "Also called buffered result"
}
proxy::storeResult(array (
    0 => NULL,
))
proxy::storeResult returns NULL
resource(730) of type (Mysqlnd Resultset (internal only - you must not modify it!))
array(1) {
    ["_msg"]=>
    string(13) "Good morning!"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_store_result](#)
[mysqli_real_query](#)

6.9.7.49 MysqlndUhConnection::txCommit

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::txCommit](#)

Commits the current transaction

Description

```
public bool MysqlndUhConnection::txCommit(
    mysqlnd_connection connection);
```

Commits the current transaction.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples**Example 6.390 MysqlndUhConnection::txCommit example**

```
<?php
class proxy extends MysqlndUhConnection {
    public function txCommit($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::txCommit($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->commit();
?>
```

The above example will output:

```
proxy::txCommit(array (
    0 => NULL,
))
proxy::txCommit returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_commit](#)

6.9.7.50 MysqlndUhConnection::txRollback

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::txRollback](#)

Rolls back current transaction

Description

```
public bool MysqlndUhConnection::txRollback(
    mysqlnd_connection connection);
```

Rolls back current transaction.

Parameters

[connection](#) Mysqlnd connection handle. Do not modify!

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.391 MysqlndUhConnection::txRollback example

```
<?php
class proxy extends MysqlndUhConnection {
    public function txRollback($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::txRollback($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->rollback();
?>
```

The above example will output:

```
proxy::txRollback(array (
    0 => NULL,
))
proxy::txRollback returns true
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_commit](#)

6.9.7.51 MysqlndUhConnection::useResult

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhConnection::useResult](#)

Initiate a result set retrieval

Description

```
public resource MysqlndUhConnection::useResult(
    mysqlnd_connection connection);
```

Initiate a result set retrieval.

Parameters

connection Mysqlnd connection handle. Do not modify!

Return Values

Resource of type [Mysqlnd Resultset \(internal only - you must not modify it!\)](#). The documentation may also refer to such resources using the alias name [mysqlnd_resultset](#).

Examples

Example 6.392 [MysqlndUhConnection::useResult](#) example

```
<?php
class proxy extends MysqlndUhConnection {
    public function useResult($res) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $ret = parent::useResult($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        var_dump($ret);
        return $ret;
    }
}
mysqlnd_uh_set_connection_proxy(new proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$mysqli->real_query("SELECT 'Good morning!' AS _msg FROM DUAL");
$res = $mysqli->use_result();
var_dump($res->fetch_assoc());
?>
```

The above example will output:

```
proxy::useResult(array (
    0 => NULL,
))
proxy::useResult returns NULL
resource(425) of type (Mysqlnd Resultset (internal only - you must not modify it!))
array(1) {
    ["_msg"]=>
    string(13) "Good morning!"
}
```

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqli_use_result](#)
[mysqli_real_query](#)

6.9.8 The MysqlndUhPreparedStatement class

Copyright 1997-2014 the PHP Documentation Group.

```
MysqlndUhPreparedStatement {
MysqlndUhPreparedStatement
    Methods

    public MysqlndUhPreparedStatement::__construct();

    public bool MysqlndUhPreparedStatement::execute(
        mysqlnd_prepared_statement statement);

    public bool MysqlndUhPreparedStatement::prepare(
        mysqlnd_prepared_statement statement,
        string query);
}
```

6.9.8.1 MysqlndUhPreparedStatement::__construct

[Copyright 1997-2014 the PHP Documentation Group.](#)

- [MysqlndUhPreparedStatement::__construct](#)

The __construct purpose

Description

```
public MysqlndUhPreparedStatement::__construct();
```

Warning

This function is currently not documented; only its argument list is available.

Parameters

This function has no parameters.

Return Values

6.9.8.2 MysqlndUhPreparedStatement::execute

[Copyright 1997-2014 the PHP Documentation Group.](#)

- [MysqlndUhPreparedStatement::execute](#)

Executes a prepared Query

Description

```
public bool MysqlndUhPreparedStatement::execute(
    mysqlnd_prepared_statement statement);
```

Executes a prepared Query.

Parameters

<i>statement</i>	Mysqlnd prepared statement handle. Do not modify! Resource of type Mysqlnd Prepared Statement (internal only - you must not modify it!) .

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

Examples

Example 6.393 MysqlndUhPreparedStatement::execute example

```
<?php
class stmt_proxy extends MysqlndUhPreparedStatement {
    public function execute($res) {
        printf("%s(", __METHOD__);
        var_dump($res);
        printf(")\n");
        $ret = parent::execute($res);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        var_dump($ret);
        return $ret;
    }
}
mysqlnd_uh_set_statement_proxy(new stmt_proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$stmt = $mysqli->prepare("SELECT 'Labskaus' AS _msg FROM DUAL");
$stmt->execute();
$msg = NULL;
$stmt->bind_result($msg);
$stmt->fetch();
var_dump($msg);
?>
```

The above example will output:

```
stmt_proxy::execute(resource(256) of type (Mysqlnd Prepared Statement (internal only - you must not modify
))
stmt_proxy::execute returns true
bool(true)
string(8) "Labskaus"
```

See Also

[mysqlnd_uh_set_statement_proxy](#)
[mysqli_stmt_execute](#)

6.9.8.3 MysqlndUhPreparedStatement::prepare

Copyright 1997-2014 the PHP Documentation Group.

- [MysqlndUhPreparedStatement::prepare](#)

Prepare an SQL statement for execution

Description

```
public bool MysqlndUhPreparedStatement::prepare(
    mysqlnd_prepared_statement statement,
    string query);
```

Prepare an SQL statement for execution.

Parameters

statement

Mysqlnd prepared statement handle. Do not modify! Resource of type [Mysqlnd Prepared Statement \(internal only - you must not modify it!\)](#).

query

The query to be prepared.

Return ValuesReturns **TRUE** on success. Otherwise, returns **FALSE****Examples****Example 6.394 MysqlndUhPreparedStatement::prepare example**

```
<?php
class stmt_proxy extends MysqlndUhPreparedStatement {
    public function prepare($res, $query) {
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));
        $query = "SELECT 'No more you-know-what-I-mean for lunch, please' AS _msg FROM DUAL";
        $ret = parent::prepare($res, $query);
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));
        var_dump($ret);
        return $ret;
    }
}
mysqlnd_uh_set_statement_proxy(new stmt_proxy());
$mysqli = new mysqli("localhost", "root", "", "test");
$stmt = $mysqli->prepare("SELECT 'Labskaus' AS _msg FROM DUAL");
$stmt->execute();
$msg = NULL;
$stmt->bind_result($msg);
$stmt->fetch();
var_dump($msg);
?>
```

The above example will output:

```
stmt_proxy::prepare(array (
  0 => NULL,
  1 => 'SELECT \'Labskaus\' AS _msg FROM DUAL',
))
stmt_proxy::prepare returns true
bool(true)
string(46) "No more you-know-what-I-mean for lunch, please"
```

See Also

[mysqlnd_uh_set_statement_proxy](#)
[mysqli_stmt_prepare](#)
[mysqli_prepare](#)

6.9.9 Mysqlnd_uh Functions

Copyright 1997-2014 the PHP Documentation Group.

6.9.9.1 mysqlnd_uh_convert_to_mysqlnd

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_uh_convert_to_mysqlnd](#)

Converts a MySQL connection handle into a mysqlnd connection handle

Description

```
resource mysqlnd_uh_convert_to_mysqlnd(
    mysqli mysql_connection);
```

Converts a MySQL connection handle into a mysqlnd connection handle. After conversion you can execute mysqlnd library calls on the connection handle. This can be used to access mysqlnd functionality not made available through user space API calls.

The function can be disabled with `mysqlnd_uh.enable`. If `mysqlnd_uh.enable` is set to `FALSE` the function will not install the proxy and always return `TRUE`. Additionally, an error of the type `E_WARNING` may be emitted. The error message may read like `PHP Warning: mysqlnd_uh_convert_to_mysqlnd(): (Mysqlnd User Handler) The plugin has been disabled by setting the configuration parameter mysqlnd_uh.enable = false. You are not allowed to call this function [...]`.

Parameters

MySQL connection handle A MySQL connection handle of type `mysql`, `mysqli` or `PDO_MySQL`.

Return Values

A mysqlnd connection handle.

Changelog

Version	Description
5.4.0	The <i>mysql_connection</i> parameter can now be of type <code>mysql</code> , <code>PDO_MySQL</code> , or <code>mysqli</code> . Before, only the <code>mysqli</code> type was allowed.

Examples

Example 6.395 `mysqlnd_uh_convert_to_mysqlnd` example

```
<?php
/* PDO user API gives no access to connection thread id */
$mysql_connection = new PDO("mysql:host=localhost;dbname=test", "root", "");
/* Convert PDO MySQL handle to mysqlnd handle */
$mysqlnd = mysqlnd_uh_convert_to_mysqlnd($mysql_connection);
/* Create Proxy to call mysqlnd connection class methods */
$obj = new MySQLndUHConnection();
/* Call mysqlnd_conn::get_thread_id */
var_dump($obj->getThreadId($mysqlnd));
/* Use SQL to fetch connection thread id */
var_dump($mysql_connection->query("SELECT CONNECTION_ID()")->fetchAll());
?>
```

The above example will output:

```
int(27054)
array(1) {
  [0]=>
    array(2) {
      ["CONNECTION_ID()"]=>
        string(5) "27054"
      [0]=>
        string(5) "27054"
    }
}
```

See Also[mysqlnd_uh.enable](#)**6.9.9.2 mysqlnd_uh_set_connection_proxy**

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_uh_set_connection_proxy](#)

Installs a proxy for mysqlnd connections

Description

```
bool mysqlnd_uh_set_connection_proxy(  
    MysqlndUhConnection connection_proxy,  
    mysqli mysqli_connection);
```

Installs a proxy object to hook mysqlnd's connection objects methods. Once installed, the proxy will be used for all MySQL connections opened with [mysqli](#), [mysql](#) or [PDO_MYSQL](#), assuming that the listed extensions are compiled to use the [mysqlnd](#) library.

The function can be disabled with [mysqlnd_uh.enable](#). If [mysqlnd_uh.enable](#) is set to [FALSE](#) the function will not install the proxy and always return [TRUE](#). Additionally, an error of the type [E_WARNING](#) may be emitted. The error message may read like [PHP Warning: mysqlnd_uh_set_connection_proxy\(\): \(Mysqlnd User Handler\) The plugin has been disabled by setting the configuration parameter mysqlnd_uh.enable = false. The proxy has not been installed \[...\]](#).

Parameters

connection_proxy	A proxy object of type MysqlndUhConnection .
mysqli_connection	Object of type mysqli . If given, the proxy will be set for this particular connection only.

Return ValuesReturns [TRUE](#) on success. Otherwise, returns [FALSE](#)**Examples****Example 6.396 mysqlnd_uh_set_connection_proxy example**

```
<?php  
$mysqli = new mysqli("localhost", "root", "", "test");  
$mysqli->query("SELECT 'No proxy installed, yet'");  
class proxy extends MysqlndUhConnection {  
    public function query($res, $query) {  
        printf("%s(%s)\n", __METHOD__, var_export(func_get_args(), true));  
        $ret = parent::query($res, $query);  
        printf("%s returns %s\n", __METHOD__, var_export($ret, true));  
        return $ret;  
    }  
}  
mysqlnd_uh_set_connection_proxy(new proxy());  
$mysqli->query("SELECT 'mysqlnd rocks!'");  
$mysql = mysql_connect("localhost", "root", "", "test");  
mysql_query("SELECT 'Ahoj Andrej!'", $mysql);  
$pdo = new PDO("mysql:host=localhost;dbname=test", "root", "");  
$pdo->query("SELECT 'Moin Johannes!'");  
?>
```

The above example will output:

```
proxy::query(array (
  0 => NULL,
  1 => 'SELECT \'mysqlnd rocks!\'',
))
proxy::query returns true
proxy::query(array (
  0 => NULL,
  1 => 'SELECT \'Ahoy Andrew!\'',
))
proxy::query returns true
proxy::query(array (
  0 => NULL,
  1 => 'SELECT \'Moin Johannes!\'',
))
proxy::query returns true
```

See Also

[mysqlnd_uh_set_statement_proxy](#)
[mysqlnd_uh.enable](#)

6.9.9.3 [mysqlnd_uh_set_statement_proxy](#)

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_uh_set_statement_proxy](#)

Installs a proxy for mysqlnd statements

Description

```
bool mysqlnd_uh_set_statement_proxy(
    MysqlndUhStatement statement_proxy);
```

Installs a proxy for mysqlnd statements. The proxy object will be used for all mysqlnd prepared statement objects, regardless which PHP MySQL extension ([mysqli](#), [mysql](#), [PDO_MYSQL](#)) has created them as long as the extension is compiled to use the [mysqlnd](#) library.

The function can be disabled with [mysqlnd_uh.enable](#). If [mysqlnd_uh.enable](#) is set to [FALSE](#) the function will not install the proxy and always return [TRUE](#). Additionally, an error of the type [E_WARNING](#) may be emitted. The error message may read like [PHP Warning: mysqlnd_uh_set_statement_proxy\(\): \(Mysqlnd User Handler\) The plugin has been disabled by setting the configuration parameter mysqlnd_uh.enable = false. The proxy has not been installed \[...\]](#).

Parameters

<i>statement_proxy</i>	The mysqlnd statement proxy object of type MysqlndUhStatement

Return Values

Returns [TRUE](#) on success. Otherwise, returns [FALSE](#)

See Also

[mysqlnd_uh_set_connection_proxy](#)
[mysqlnd_uh.enable](#)

6.9.10 Change History

Copyright 1997-2014 the PHP Documentation Group.

The Change History lists major changes users need to be aware of if upgrading from one version to another. It is a high level summary of selected changes that may impact applications or might even break backwards compatibility. See also the [CHANGES](#) file contained in the source for additional changelog information. The commit history is also available.

6.9.10.1 PECL/mysqlnd_uh 1.0 series

Copyright 1997-2014 the PHP Documentation Group.

1.0.1-alpha

- Release date: TBD
- Motto/theme: bug fix release

Feature changes

- Support of PHP 5.4.0 or later.
- BC break: `mysqlndUhConnection::changeUser` requires additional `passwd_len` parameter.
- BC break: `MYSQLND_UH_VERSION_STR` renamed to `MYSQLND_UH_VERSION`.
`MYSQLND_UH_VERSION` renamed to `MYSQLND_UH_VERSION_ID`.
- BC break: `mysqlnd_uh.enabled` configuration setting renamed to `mysqlnd_uh.enable`.

1.0.0-alpha

- Release date: 08/2010
- Motto/theme: Initial release

6.10 Mysqlnd connection multiplexing plugin

Copyright 1997-2014 the PHP Documentation Group.

The mysqlnd multiplexing plugin (`mysqlnd_mux`) multiplexes MySQL connections established by all PHP MySQL extensions that use the MySQL native driver (`mysqlnd`) for PHP.

The MySQL native driver for PHP features an internal C API for plugins, such as the connection multiplexing plugin, which can extend the functionality of mysqlnd. See the [mysqlnd](#) for additional details about its benefits over the MySQL Client Library libmysqlclient.

Mysqlnd plugins like `mysqlnd_mux` operate, for the most part, transparently from a user perspective. The connection multiplexing plugin supports all PHP applications, and all MySQL PHP extensions. It does not change existing APIs. Therefore, it can easily be used with existing PHP applications.

Note

This is a proof-of-concept. All features are at an early stage. Not all kinds of queries are handled by the plugin yet. Thus, it cannot be used in a drop-in fashion at the moment.

Please, do not use this version in production environments.

6.10.1 Key Features

Copyright 1997-2014 the PHP Documentation Group.

The key features of `mysqlnd_mux` are as follows:

- Transparent and therefore easy to use:
 - Supports all of the PHP MySQL extensions.
 - Little to no application changes are required, dependent on the required usage scenario.
- Reduces server load and connection establishment latency:
 - Opens less connections to the MySQL server.
 - Less connections to MySQL mean less work for the MySQL server. In a client-server environment scaling the server is often more difficult than scaling the client. Multiplexing helps with horizontal scale-out (scale-by-client).
 - Pooling saves connection time.
- Multiplexed connection: multiple user handles share the same network connection. Once opened, a network connection is cached and shared among multiple user handles. There is a 1:n relationship between internal network connection and user connection handles.
- Persistent connection: a network connection is kept open at the end of the web request, if the PHP deployment model allows. Thus, subsequently web requests can reuse a previously opened connection. Like other resources, network connections are bound to the scope of a process. Thus, they can be reused for all web requests served by a process.

6.10.2 Limitations

[Copyright 1997-2014 the PHP Documentation Group.](#)

The proof-of-concept does not support unbuffered queries, prepared statements, and asynchronous queries.

The connection pool is using a combination of the transport method and hostname as keys. As a consequence, two connections to the same host using the same transport method (TCP/IP, Unix socket, Windows named pipe) will be linked to the same pooled connection even if username and password differ. Be aware of the possible security implications.

The proof-of-concept is transaction agnostic. It does not know about SQL transactions.

Note

Applications must be aware of the consequences of connection sharing connections.

6.10.3 About the name `mysqlnd_mux`

[Copyright 1997-2014 the PHP Documentation Group.](#)

The shortcut `mysqlnd_mux` stands for `mysqlnd connection multiplexing plugin`.

6.10.4 Concepts

[Copyright 1997-2014 the PHP Documentation Group.](#)

This explains the architecture and related concepts for this plugin. Reading and understanding these concepts is required to successfully use this plugin.

6.10.4.1 Architecture

[Copyright 1997-2014 the PHP Documentation Group.](#)

The `mysqlnd` connection multiplexing plugin is implemented as a PHP extension. It is written in C and operates under the hood of PHP. During the startup of the PHP interpreter, in the module initialization phase of the PHP engine, it gets registered as a [mysqlnd](#) plugin to replace specific `mysqlnd` C methods.

The `mysqlnd` library uses PHP streams to communicate with the MySQL server. PHP streams are accessed by the `mysqlnd` library through its `net` module. The `mysqlnd` connection multiplexing plugin proxies methods of the `mysqlnd` library `net` module to control opening and closing of network streams.

Upon opening a user connection to MySQL using the appropriate connection functions of either [mysqli](#), [PDO_MYSQL](#) or [ext/mysql](#), the plugin will search its connection pool for an open network connection. If the pool contains a network connection to the host specified by the connect function using the transport method requested (TCP/IP, Unix domain socket, Windows named pipe), the pooled connection is linked to the user handle. Otherwise, a new network connection is opened, put into the pool and associated with the user connection handle. This way, multiple user handles can be linked to the same network connection.

6.10.4.2 Connection pool

[Copyright 1997-2014 the PHP Documentation Group.](#)

The plugin's connection pool is created when PHP initializes its modules ([MINIT](#)) and free'd when PHP shuts down the modules ([MSHUTDOWN](#)). This is the same as for persistent MySQL connections.

Depending on the deployment model, the pool is used for the duration of one or multiple web requests. Network connections are bound to the lifespan of an operating system level process. If the PHP process serves multiple web requests as it is the case for Fast-CGI or threaded web server deployments, then the pooled connections can be reused over multiple connections. Because multiplexing means sharing connections, it can even happen with a threaded deployment that two threads or two distinct web requests are linked to one pooled network connections.

A pooled connection is explicitly closed once the last reference to it is released. An implicit close happens when PHP shuts down its modules.

6.10.4.3 Sharing connections

[Copyright 1997-2014 the PHP Documentation Group.](#)

The PHP `mysqlnd` connection multiplexing plugin changes the relationship between a user's connection handle and the underlying MySQL connection. Without the plugin, every MySQL connection belongs to exactly one user connection at a time. The multiplexing plugin changes. A MySQL connection is shared among multiple user handles. There is no one-to-one relation if using the plugin.

Sharing pooled connections has an impact on the connection state. State changing operations from multiple user handles pointing to one MySQL connection are not isolated from each other. If, for example, a session variable is set through one user connection handle, the session variable becomes visible to all other user handles that reference the same underlying MySQL connection.

This is similar in concept to connection state related phenomena described for the PHP `mysqlnd` replication and load balancing plugin. Please, check the [PECL/mysqlnd_ms documentation](#) for more details on the state of a connection.

The proof-of-concept takes no measures to isolate multiplexed connections from each other.

6.10.5 Installing/Configuring

[Copyright 1997-2014 the PHP Documentation Group.](#)

6.10.5.1 Requirements

[Copyright 1997-2014 the PHP Documentation Group.](#)

PHP 5.5.0 or newer. Some advanced functionality requires PHP 5.5.0 or newer.

The `mysqlnd_mux` replication and load balancing plugin supports all PHP applications and all available PHP MySQL extensions (`mysqli`, `mysql`, `PDO_MYSQL`). The PHP MySQL extension must be configured to use `mysqlnd` in order to be able to use the `mysqlnd_mux` plugin for `mysqlnd`.

6.10.5.2 Installation

Copyright 1997-2014 the PHP Documentation Group.

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: http://pecl.php.net/package/mysqlnd_mux

6.10.5.3 Runtime Configuration

Copyright 1997-2014 the PHP Documentation Group.

The behaviour of these functions is affected by settings in `php.ini`.

Table 6.45 Mysqlnd_mux Configure Options

Name	Default	Changeable	Changelog
<code>mysqlnd_mux.enable</code>	0	PHP_INI_SYSTEM	

Here's a short explanation of the configuration directives.

`mysqlnd_mux.enable` integer Enables or disables the plugin. If disabled, the extension will not plug into `mysqlnd` to proxy internal `mysqlnd` C API calls.

6.10.6 Predefined Constants

Copyright 1997-2014 the PHP Documentation Group.

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

Other

The plugins version number can be obtained using `MYSQLND_MUX_VERSION` or `MYSQLND_MUX_VERSION_ID`. `MYSQLND_MUX_VERSION` is the string representation of the numerical version number `MYSQLND_MUX_VERSION_ID`, which is an integer such as 10000. Developers can calculate the version number as follows.

Version (part)	Example
Major*10000	1*10000 = 10000
Minor*100	0*100 = 0
Patch	0 = 0
MYSQLND_MUX_VERSION_ID	10000

`MYSQLND_MUX_VERSION` (string) Plugin version string, for example, "1.0.0-prototype".

`MYSQLND_MUX_VERSION_ID` (integer) Plugin version number, for example, 10000.

6.10.7 Change History

Copyright 1997-2014 the PHP Documentation Group.

This change history is a high level summary of selected changes that may impact applications and/or break backwards compatibility.

See also the [CHANGES](#) file in the source distribution for a complete list of changes.

6.10.7.1 PECL/mysqlnd_mux 1.0 series

Copyright 1997-2014 the PHP Documentation Group.

1.0.0-pre-alpha

- Release date: no package released, initial check-in 09/2012
- Motto/theme: Proof of concept

Initial check-in. Essentially a demo of the [mysqlnd](#) plugin API.

Note

This is the current development series. All features are at an early stage. Changes may happen at any time without prior notice. Please, do not use this version in production environments.

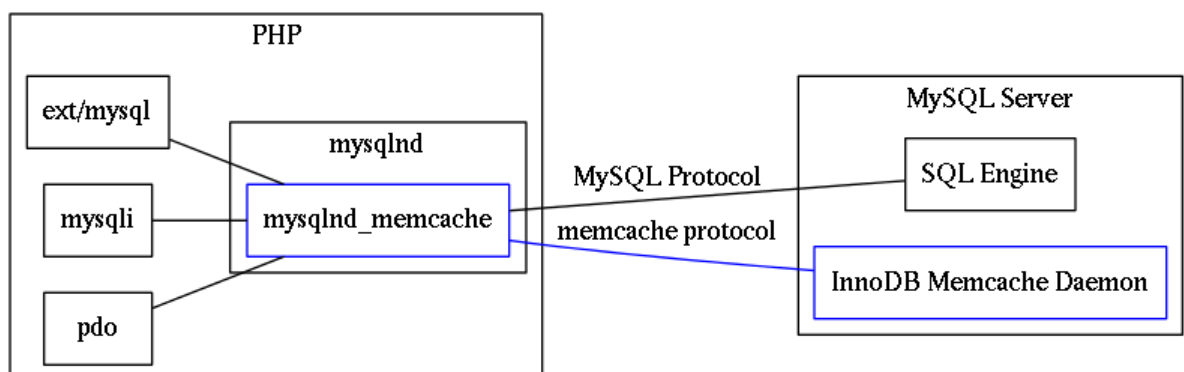
The documentation may not reflect all changes yet.

6.11 Mysqlnd Memcache plugin

Copyright 1997-2014 the PHP Documentation Group.

The [mysqlnd memcache plugin](#) ([mysqlnd_memcache](#)) is a PHP extension for transparently translating SQL into requests for the MySQL InnoDB Memcached Daemon Plugin (server plugin). It includes experimental support for the MySQL Cluster Memcached Daemon. The server plugin provides access to data stored inside MySQL InnoDB (respectively MySQL Cluster NDB) tables using the Memcache protocol. This PHP extension, which supports all PHP MySQL extensions that use [mysqlnd](#), will identify tables exported in this way and will translate specific SELECT queries into Memcache requests.

Figure 6.1 [mysqlnd_memcache](#) data flow



Note

This plugin depends on the MySQL InnoDB Memcached Daemon Plugin. It is not provided to be used with a stand-alone Memcached. For a generic query cache using Memcached look at the [mysqlnd query cache plugin](#). For direct Memcache access look at the [memcache](#) and [memcached](#) extensions.

The MySQL native driver for PHP is a C library that ships together with PHP as of PHP 5.3.0. It serves as a drop-in replacement for the MySQL Client Library (libmysqlclient). Using mysqlnd has several advantages: no extra downloads are required because it's bundled with PHP, it's under the PHP license, there is lower memory consumption in certain cases, and it contains new functionality such as asynchronous queries.

The `mysqlnd_mmemcache` operates, for the most part, transparently from a user perspective. The mysqlnd memcache plugin supports all PHP applications, and all MySQL PHP extensions. It does not change existing APIs. Therefore, it can easily be used with existing PHP applications.

The MySQL Memcache plugins add key-value style access method for data stored in InnoDB resp. NDB (MySQL Cluster) SQL tables through the Memcache protocol. This type of key-value access is often faster than using SQL.

6.11.1 Key Features

[Copyright 1997-2014 the PHP Documentation Group.](#)

The key features of PECL/mysqlnd_memcache are as follows.

- Possible performance benefits
 - Client-side: light-weight protocol.
 - Server-side: no SQL parsing, direct access to the storage.
- Please, run your own benchmarks! Actual performance results are highly dependent on setup and hardware used.

6.11.2 Limitations

[Copyright 1997-2014 the PHP Documentation Group.](#)

The initial version is not binary safe. Due to the way the MySQL Memcache plugins works there are restrictions related to separators.

Prepared statements and asynchronous queries are not supported. Result set meta data support is limited.

The mapping information for tables accessible via Memcache is not cached in the plugin between requests but fetched from the MySQL server each time a MySQL connection is associated with a Memcache connection. See `mysqlnd_memcache_set` for details.

6.11.3 On the name

[Copyright 1997-2014 the PHP Documentation Group.](#)

The shortcut `mysqlnd_memcache` stands for `mysqlnd memcache plugin`. Memcache refers to support of the MySQL Memcache plugins for InnoDB and NDB (MySQL Cluster). The plugin is not related to the Memcached cache server.

6.11.4 Quickstart and Examples

[Copyright 1997-2014 the PHP Documentation Group.](#)

The mysqlnd memcache plugin is easy to use. This quickstart will demo typical use-cases, and provide practical advice on getting started.

It is strongly recommended to read the reference sections in addition to the quickstart. The quickstart tries to avoid discussing theoretical concepts and limitations. Instead, it will link to the reference

sections. It is safe to begin with the quickstart. However, before using the plugin in mission critical environments we urge you to read additionally the background information from the reference sections.

6.11.4.1 Setup

Copyright 1997-2014 the PHP Documentation Group.

The plugin is implemented as a PHP extension. See also the [installation instructions](#) to install this extension.

Compile or configure the PHP MySQL extension (API) ([mysqli](#), [PDO_MYSQL](#), [mysql](#)). That extension must use the [mysqlnd](#) library as because `mysqlnd_memcache` is a plugin for the `mysqlnd` library. For additional information, refer to the [mysqlnd_memcache installation instructions](#).

Then, load this extension into PHP and activate the plugin in the PHP configuration file using the PHP configuration directive named [mysqlnd_memcache.enable](#).

Example 6.397 Enabling the plugin (php.ini)

```
; On Windows the filename is php_mysqlnd_memcache.dll
; Load the extension
extension=mysqlnd_memcache.so
; Enable it
mysqlnd_memcache.enable=1
```

Follow the instructions given in the [MySQL Reference Manual on installing the Memcache plugins](#) for the MySQL server. Activate the plugins and configure Memcache access for SQL tables.

The examples in this quickguide assume that the following table exists, and that Memcache is configured with access to it.

Example 6.398 SQL table used for the Quickstart

```
CREATE TABLE test(
  id CHAR(16),
  f1 VARCHAR(255),
  f2 VARCHAR(255),
  f3 VARCHAR(255),
  flags INT NOT NULL,
  cas_column INT,
  expire_time_column INT,
  PRIMARY KEY(id)
) ENGINE=InnoDB;
INSERT INTO test (id, f1, f2, f3) VALUES (1, 'Hello', 'World', '!');
INSERT INTO test (id, f1, f2, f3) VALUES (2, 'Lady', 'and', 'the tramp');
INSERT INTO innodb_memcache.containers(
  name, db_schema, db_table, key_columns, value_columns,
  flags, cas_column, expire_time_column, unique_idx_name_on_key)
VALUES (
  'plugin_test', 'test', 'test', 'id', 'f1,f2,f3',
  'flags', 'cas_column', 'expire_time_column', 'PRIMARY KEY');
```

6.11.4.2 Usage

Copyright 1997-2014 the PHP Documentation Group.

After associating a MySQL connection with a Memcache connection using [mysqlnd_memcache_set](#) the plugin attempts to transparently replace SQL [SELECT](#) statements by a memcache access. For that purpose the plugin monitors all SQL statements executed and tries to match the statement string

against `MYSQLND_MEMCACHE_DEFAULT_REGEX`. In case of a match, the `mysqlnd` memcache plugin checks whether the `SELECT` is accessing only columns of a mapped table and the `WHERE` clause is limited to a single key lookup.

In case of the example SQL table, the plugin will use the Memcache interface of the MySQL server to fetch results for a SQL query like `SELECT f1, f2, f3 WHERE id = n`.

Example 6.399 Basic example.

```
<?php
$mysqli = new mysqli("host", "user", "passwd", "database");
$memc = new Memcached();
$memc->addServer("host", 11211);
mysqlnd_memcache_set($mysqli, $memc);
/*
   This is a query which queries table test using id as key in the WHERE part
   and is accessing fields f1, f2 and f3. Therefore, mysqlnd_memcache
   will intercept it and route it via memcache.
*/
$result = $mysqli->query("SELECT f1, f2, f3 FROM test WHERE id = 1");
while ($row = $result->fetch_row()) {
    print_r($row);
}
/*
   This is a query which queries table test but using f1 in the WHERE clause.
   Therefore, mysqlnd_memcache can't intercept it. This will be executed
   using the MySQL protocol
*/
$mysqli->query("SELECT id FROM test WHERE f1 = 'Lady'");
while ($row = $result->fetch_row()) {
    print_r($row);
}
?>
```

The above example will output:

```
array(
    [f1] => Hello
    [f2] => World
    [f3] => !
)
array(
    [id] => 2
)
```

6.11.5 Installing/Configuring

Copyright 1997-2014 the PHP Documentation Group.

6.11.5.1 Requirements

Copyright 1997-2014 the PHP Documentation Group.

PHP: this extension requires PHP 5.4+, version PHP 5.4.4 or never. The required PHP extensions are `PCRE` (enabled by default), and the `memcached` extension version 2.0.x.

The `mysqlnd_memcache` Memcache plugin supports all PHP applications and all available PHP MySQL extensions (`mysqli`, `mysql`, `PDO_MYSQL`). The PHP MySQL extension must be configured with `mysqlnd` support.

For accessing [InnoDB](#) tables, this PHP extension requires [MySQL Server 5.6.6](#) or newer with the InnoDB Memcache Daemon Plugin enabled.

For accessing [MySQL Cluster NDB](#) tables, this PHP extension requires [MySQL Cluster 7.2](#) or newer with the NDB Memcache API nodes enabled.

6.11.5.2 Installation

Copyright 1997-2014 the PHP Documentation Group.

This [PECL](#) extension is not bundled with PHP.

Information for installing this PECL extension may be found in the manual chapter titled [Installation of PECL extensions](#). Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: http://pecl.php.net/package/mysqld_memcache

A DLL for this PECL extension is currently unavailable. See also the [building on Windows](#) section.

6.11.5.3 Runtime Configuration

Copyright 1997-2014 the PHP Documentation Group.

The behaviour of these functions is affected by settings in [php.ini](#).

Table 6.46 `Mysqld_memcache` Configure Options

Name	Default	Changeable	Changelog
<code>mysqld_memcache.enable</code>	<code>1</code>	<code>PHP_INI_SYSTEM</code>	Available since 1.0.0

Here's a short explanation of the configuration directives.

`mysqld_memcache.enable` integer Enables or disables the plugin. If disabled, the extension will not plug into `mysqld` to proxy internal `mysqld` C API calls.

Note

This option is mainly used by developers to build this extension statically into PHP. General users are encouraged to build this extension as a shared object, and to unload it completely when it is not needed.

6.11.6 Predefined Constants

Copyright 1997-2014 the PHP Documentation Group.

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

MySQL Memcache Plugin related

`MYSQLND_MEMCACHE_DEFAULT_REGEXP` Default regular expression (PCRE style) used for matching `SELECT` statements that will be mapped into a MySQL Memcache Plugin access point, if possible.
(string)

It is also possible to use `mysqld_memcache_set`, but the default approach is using this regular expression for pattern matching.

Assorted

The version number of this plugin can be obtained by using `MYSQLND_MEMCACHE_VERSION` or `MYSQLND_MEMCACHE_VERSION_ID`. `MYSQLND_MEMCACHE_VERSION` is the string representation of the numerical version number `MYSQLND_MEMCACHE_VERSION_ID`, which is an integer such as 10000. Developers can calculate the version number as follows.

Version (part)	Example
Major*10000	1*10000 = 10000
Minor*100	0*100 = 0
Patch	0 = 0
MYSQLND_MEMCACHE_VERSION_ID	10000

`MYSQLND_MEMCACHE_VERSION` Plugin version string, for example, "1.0.0-alpha".
(string)

`MYSQLND_MEMCACHE_VERSION_ID` Plugin version number, for example, 10000.
(integer)

6.11.7 Mysqlnd_memcache Functions

Copyright 1997-2014 the PHP Documentation Group.

6.11.7.1 mysqlnd_memcache_get_config

Copyright 1997-2014 the PHP Documentation Group.

- `mysqlnd_memcache_get_config`

Returns information about the plugin configuration

Description

```
array mysqlnd_memcache_get_config(
    mixed connection);
```

This function returns an array of all `mysqlnd_memcache` related configuration information that is attached to the MySQL connection. This includes MySQL, the Memcache object provided via `mysqlnd_memcache_set`, and the table mapping configuration that was automatically collected from the MySQL Server.

Parameters

connection A handle to a MySQL Server using one of the MySQL API extensions for PHP, which are `PDO_MYSQL`, `mysqli` or `ext/mysql`.

Return Values

An array of `mysqlnd_memcache` configuration information on success, otherwise `FALSE`.

The returned array has these elements:

Table 6.47 `mysqlnd_memcache_get_config` array structure

Array Key	Description
memcached	Instance of Memcached associated to this MySQL connection by <code>mysqlnd_memcache_set</code> . You can use this to change settings of the memcache connection, or directly by querying the server on this connection.
pattern	The PCRE regular expression used to match the SQL query sent to the server. Queries matching

Array Key	Description
	this pattern will be further analyzed to decide whether the query can be intercepted and sent via the memcache interface or whether the query is sent using the general MySQL protocol to the server. The pattern is either the default pattern (MYSQLND_MEMCACHE_DEFAULT_REGEXP) or it is set via mysqlnd_memcache_set .
mappings	An associative array with a list of all configured containers as they were discovered by this plugin. The key for these elements is the name of the container in the MySQL configuration. The value is described below. The contents of this field is created by querying the MySQL Server during association to MySQL and a memcache connection using mysqlnd_memcache_set .
mapping_query	An SQL query used during mysqlnd_memcache_set to identify the available containers and mappings. The result of that query is provided in the mappings element.

Table 6.48 Mapping entry structure

Array Key	Description
prefix	A prefix used while accessing data via memcache. With the MySQL InnoDB Memcache Deamon plugin, this usually begins with @@ and ends with a configurable separator. This prefix is placed in front of the key value while using the memcache protocol.
schema_name	Name of the schema (database) which contains the table being accessed.
table_name	Name of the table which contains the data accessible via memcache protocol.
id_field_name	Name of the database field (column) with the id used as key when accessing the table via memcache. Often this is the database field having a primary key.
separator	The separator used to split the different field values. This is needed as memcache only provides access to a single value while MySQL can map multiple columns to this value. Note The separator, which can be set in the MySQL Server configuration, should not be part of any value retrieved via memcache because proper mapping can't be guaranteed.
fields	An array with the name of all fields available for this mapping.

Examples

Example 6.400 `mysqlnd_memcache_get_config` example

```
<?php
$mysqli = new mysqli("host", "user", "passwd", "database");
$memc = new Memcached();
$memc->addServer("host", 11211);
mysqlnd_memcache_set($mysqli, $memc);
var_dump(mysqlnd_memcache_get_config($mysqli));
?>
```

The above example will output:

```
array(4) {
  ["memcached"]=>
  object(Memcached)#2 (0) {
  }
  ["pattern"]=>
  string(125) "/^\\s*SELECT\\s*(.+)\\s*FROM\\s*`?([a-z0-9_]+)`\\s*WHERE\\s*`?([a-z0-9_]+)`\\s*=\\s*(?(['"])([^\r\n`\"'"])*\\1)?\\s*$/"
  ["mappings"]=>
  array(1) {
    ["mymem_test"]=>
    array(6) {
      ["prefix"]=>
      string(13) "@@mymem_test."
      ["schema_name"]=>
      string(4) "test"
      ["table_name"]=>
      string(10) "mymem_test"
      ["id_field_name"]=>
      string(2) "id"
      ["separator"]=>
      string(1) "|"
      ["fields"]=>
      array(3) {
        [0]=>
        string(2) "f1"
        [1]=>
        string(2) "f2"
        [2]=>
        string(2) "f3"
      }
    }
  }
  ["mapping_query"]=>
  string(209) "SELECT c.name,
              CONCAT('@@', c.name, (SELECT value FROM innodb_memcache.config_options WHERE name = 'prefix')) AS prefix,
              c.db_schema,
              c.db_table,
              c.key_columns,
              c.value_columns,
              (SELECT value FROM innodb_memcache.config_options WHERE name = 'separator') AS separator
              FROM innodb_memcache.containers c"
```

See Also

[mysqlnd_memcache_set](#)

6.11.7.2 `mysqlnd_memcache_set`

Copyright 1997-2014 the PHP Documentation Group.

- [mysqlnd_memcache_set](#)

Associate a MySQL connection with a Memcache connection

Description

```
bool mysqlnd_memcache_set(
    mixed mysql_connection,
    Memcached memcache_connection,
    string pattern,
    callback callback);
```

Associate *mysql_connection* with *memcache_connection* using *pattern* as a PCRE regular expression, and *callback* as a notification callback or to unset the association of *mysql_connection*.

While associating a MySQL connection with a Memcache connection, this function will query the MySQL Server for its configuration. It will automatically detect whether the server is configured to use the InnoDB Memcache Daemon Plugin or MySQL Cluster NDB Memcache support. It will also query the server to automatically identify exported tables and other configuration options. The results of this automatic configuration can be retrieved using [mysqlnd_memcache_get_config](#).

Parameters

<i>mysql_connection</i>	A handle to a MySQL Server using one of the MySQL API extensions for PHP, which are PDO_MYSQL , mysqli or ext/mysql .
<i>memcache_connection</i>	A Memcached instance with a connection to the MySQL Memcache Daemon plugin. If this parameter is omitted, then <i>mysql_connection</i> will be unassociated from any memcache connection. And if a previous association exists, then it will be replaced.
<i>pattern</i>	A regular expression in Perl Compatible Regular Expression syntax used to identify potential Memcache-queries. The query should have three sub patterns. The first subpattern contains the requested field list, the second the name of the ID column from the query and the third the requested value. If this parameter is omitted or is set to NULL , then a default pattern will be used.
<i>callback</i>	A callback which will be used whenever a query is being sent to MySQL. The callback will receive a single boolean parameter telling if a query was sent via Memcache.

Return Values

[TRUE](#) if the association or disassociation is successful, otherwise [FALSE](#) if there is an error.

Examples

Example 6.401 [mysqlnd_memcache_set](#) example with [var_dump](#) as a simple debugging callback.

```
<?php
$mysqli = new mysqli("host", "user", "passwd", "database");
$memc = new Memcached();
$memc->addServer("host", 11211);
mysqlnd_memcache_set($mysqli, $memc, NULL, 'var_dump');
/* This query will be intercepted and executed via Memcache protocol */
echo "Sending query for id via Memcache: ";
$mysqli->query("SELECT f1, f2, f3 FROM test WHERE id = 1");
/* f1 is not configured as valid key field, this won't be sent via Memcache */
```

```
echo "Sending query for f1 via Memcache: ";
$mysqli->query("SELECT id FROM test WHERE f1 = 1");
mysqlind_memcache_set($mysqli);
/* Now the regular MySQL protocol will be used */
echo "var_dump won't be invoked: ";
$mysqli->query("SELECT f1, f2, f3 WHERE id = 1");
?>
```

The above example will output:

```
Sending query for id via Memcache: bool(true)
Sending query for f1 via Memcache: bool(false)
var_dump won't be invoked:
```

See Also

[mysqlind_memcache_get_config](#)

6.11.8 Change History

Copyright 1997-2014 the PHP Documentation Group.

This change history is a high level summary of selected changes that may impact applications and/or break backwards compatibility.

See also the [CHANGES](#) file in the source distribution for a complete list of changes.

6.11.8.1 PECL/mysqlind_memcache 1.0 series

Copyright 1997-2014 the PHP Documentation Group.

1.0.0-alpha

- Release date: TBD
- Motto/theme: Basic mapping of SQL SELECT to a MySQL Memcache plugin access.

The initial release does map basic SQL SELECT statements to a MySQL Memcache plugin access. This bares potential performance benefits as the direct key-value access to MySQL storage using the Memcache protocol is usually faster than using SQL access.

6.12 Common Problems with MySQL and PHP

- **Error: Maximum Execution Time Exceeded:** This is a PHP limit; go into the [php.ini](#) file and set the maximum execution time up from 30 seconds to something higher, as needed. It is also not a bad idea to double the RAM allowed per script to 16MB instead of 8MB.
- **Fatal error: Call to unsupported or undefined function mysql_connect() in ...:** This means that your PHP version isn't compiled with MySQL support. You can either compile a dynamic MySQL module and load it into PHP or recompile PHP with built-in MySQL support. This process is described in detail in the PHP manual.
- **Error: Undefined reference to 'uncompress':** This means that the client library is compiled with support for a compressed client/server protocol. The fix is to add **-lz** last when linking with **-lmysqlclient**.
- **Error: Client does not support authentication protocol:** This is most often encountered when trying to use the older [mysql](#) extension with MySQL 4.1.1 and later. Possible

solutions are: downgrade to MySQL 4.0; switch to PHP 5 and the newer [mysqli](#) extension; or configure the MySQL server with the [old_passwords](#) system variable set to 1. (See [Client does not support authentication protocol](#), for more information.)

Chapter 7 MySQL Connector/C Developer Guide

Table of Contents

7.1 Introduction to Connector/C	1135
7.2 Connector/C Versions	1135
7.3 Connector/C Distribution Contents	1136
7.4 Installing Connector/C	1137
7.4.1 Installing Connector/C from a Binary Distribution	1137
7.4.2 Installing Connector/C from Source	1138
7.4.3 Postinstallation Steps	1140
7.4.4 Testing Connector/C	1141
7.5 Building Connector/C Applications	1141

7.1 Introduction to Connector/C

Connector/C is a client library that implements the C API for client/server communication. It is a standalone replacement for the MySQL client library shipped with MySQL Server distributions. See [MySQL C API Implementations](#).

To see which platforms are supported, visit [MySQL Connector/C downloads](#).

Reasons to use Connector/C:

- If you need only the client library, Connector/C provides everything required. There is no need to compile or install the MySQL Server package, which is much larger.
- Connector/C does not rely on the MySQL Server release cycle. Bug fixes and new features can be distributed independently of MySQL Server releases.

For documentation of the C API implemented by Connector/C, see [MySQL C API](#).

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

The following discussion covers these topics:

- Connector/C versions and supported platforms
- Connector/C distribution contents
- Obtaining and installing Connector/C
- Building client programs that use Connector/C

7.2 Connector/C Versions

These versions of Connector/C are available:

- Connector/C 6.1: Based on the C API parts of current MySQL sources and kept up to date with those sources.
- Connector/C 6.0: Created originally from a branch of the MySQL source tree, but now out of date with respect to C API changes in that tree.

Consequently, Connector/C 6.1 is preferred over 6.0. Connector/C 6.1 provides these features not present in 6.0:

- Support for the pluggable authentication framework that enables implementation of authentication methods as plugins. This framework can be used for MySQL native authentication as well as external authentication methods. See [Pluggable Authentication](#).
- Client-side support for the SHA-256, PAM, and Windows native authentication plugins. See [The SHA-256 Authentication Plugin](#), [The PAM Authentication Plugin](#), and [The Windows Native Authentication Plugin](#).

The older Connector/C 6.0 can connect only to accounts that use native MySQL passwords. If a client program attempts to connect to an account that requires a different authentication method, an “Access denied for user” error occurs.

- Support for connecting to accounts that have expired passwords. See [Password Expiration and Sandbox Mode](#).
- Support for prepared [CALL](#) statements. This enables client programs to handle stored procedures that produce multiple result sets and to obtain the final value of [OUT](#) and [INOUT](#) procedure parameters. See [C API Support for Prepared CALL Statements](#).
- Support for connecting over IPv6. See [IPv6 Support](#).
- Support for binding client programs to a specific IP address at connect time. See [mysql_options\(\)](#).
- Support for specifying connection attributes to pass to the server at connect time. See [mysql_options\(\)](#), and [mysql_options4\(\)](#).

7.3 Connector/C Distribution Contents

Connector/C 6.1 distributions contain the header, library, and utility files necessary to build MySQL client applications that communicate with MySQL Server using the C API.

Distributions are available in binary and source formats. A binary distribution contains the header, library, and utility components discussed following, compiled and ready for use in writing client programs. A source distribution contains the source files required to produce the same headers, libraries, and utilities included in a binary distribution, but you compile them yourself.

Connector/C distributions include these components:

- A set of [.h](#) header files that C applications include at compile time. These files are located in the [include](#) directory.
- Static and dynamic libraries that C applications link to at link time. These libraries are located in the [lib](#) directory. The library names depend on the library type and platform for which a distribution is built:
 - On Unix (and Unix-like) systems, the static library is [libmysqlclient.a](#). The dynamic library is [libmysqlclient.so](#) on most Unix systems and [libmysqlclient.dylib](#) on OS X.
 - On Windows, the static library is [mysqlclient.lib](#) and the dynamic library is [libmysql.dll](#). Windows distributions also include [libmysql.lib](#), a static import library needed for using the dynamic library.

Windows distributions also include a set of debug libraries. These have the same names as the nondebug libraries, but are located in the [lib/debug](#) library.

- Utilities. Connector/C 6.1 includes the following utilities, located in the [bin](#) directory. They are the same as in MySQL Server distributions:
 - [mysql_config](#) displays flags needed to compile C applications to use Connector/C. This utility is a shell script and is included only for Unix systems. See [mysql_config — Display Options for Compiling Clients](#).

- `my_print_defaults` displays the options that are present in option groups within option files. See [my_print_defaults — Display Options from Option Files](#).
- `pererror` displays error messages corresponding to error codes. See [pererror — Explain Error Codes](#).

Connector/C 6.0 distributions are similar to 6.1 distributions, with these exceptions:

- Debug libraries, `my_print_defaults`, and `pererror` are not included.
- `mysql_config` is an executable program that is available on all platforms. However, this version of `mysql_config` is more limited than the shell script version in the types of information it can display.

7.4 Installing Connector/C

Connector/C distributions are available in binary and source formats. Binary distributions are available in native format for many platforms, such as RPM packages for Linux, DMG packages for OS X, and PKG packages for Solaris. Distributions are also available in more generic formats such as Zip archives or compressed `tar` files.

To obtain a distribution, visit [Connector/C downloads](#).

After installing Connector/C, you may need to take additional steps to enable your compiler or linker to find the C API header files and libraries. See [Section 7.4.3, “Postinstallation Steps”](#).

7.4.1 Installing Connector/C from a Binary Distribution

Installers in native package formats are available for many Unix and Unix-like systems, and for Windows. Alternatively, you can install using a distribution in a more generic format such as a Zip archive or compressed `tar` file.

You may need to have `root` or administrator privileges to perform the installation operation.

Installing Connector/C on Unix Using Compressed `tar` Files

On Unix and Unix-like systems, a generic Connector/C binary distribution is packaged as a compressed `tar` file, denoted here as `PACKAGE.tar.gz`. To install a distribution file, unpack it in the intended installation directory using this command:

```
shell> tar zxvf PACKAGE.tar.gz
```

Installing Connector/C on Microsoft Windows

Windows users can choose between two binary packaging formats:

- Windows MSI Installer (`.msi` file): To use the MSI Installer, launch it and follow the prompts in the screens it presents to install Connector/C in the location of your choosing.
- Zip archive without installer (`.zip` file): To use a Zip archive, unpack it in the intended installation directory using [WinZip](#) or another tool that can read `.zip` files.

Installing Connector/C on OS X Using DMG Packages

A OS X native package installer is provided as a DMG (disk image) file. To install a DMG package, double-click the image file, then follow the prompts.

By default, the DMG package installs Connector/C under `/usr/local`, into a dedicated directory that does not conflict with the one used by MySQL Server DMG packages.

Installing Connector/C on Linux Using RPM Packages

There are two Linux RPM packages for Connector/C. Install one or both, depending on the capabilities you require:

- The [shared](#) RPM contains the shared client library. Install this RPM if you intend to compile or run C API applications that depend on the shared client library.
- The [devel](#) RPM contains the header files and the static client library. Install this RPM if you intend to compile C API applications.

RPM packages for Connector/C do not include the [perror](#) or [my_print_defaults](#) utilities.

A Linux RPM package is provided as a file with an [.rpm](#) suffix, denoted here as [PACKAGE.rpm](#). To install a given RPM package, use this command:

```
shell> rpm -i PACKAGE.rpm
```

RPM provides a feature to verify the integrity and authenticity of packages before installing them. To learn more about this feature, see [Verifying Package Integrity Using MD5 Checksums or GnuPG](#).

Installing Connector/C on Solaris Using PKG Packages

A Solaris PKG package is provided as a file with a [.pkg.gz](#) suffix, denoted here as [PACKAGE.pkg.gz](#). To install a PKG package, uncompress it:

```
shell> gunzip PACKAGE.pkg.gz
```

Uncompressing [PACKAGE.pkg.gz](#) produces [PACKAGE.pkg](#). Then use [pkgadd](#) and follow the onscreen prompts:

```
shell> pkgadd -d PACKAGE.pkg
```

By default, the PKG package installs Connector/C under the root path [/opt/mysql](#), into a dedicated directory that does not conflict with the one used by MySQL Server PKG packages. You can change only the installation root path using [pkgadd](#), which can be used to install MySQL in a different Solaris zone. If you need to install in a specific directory, use a binary [tar](#) file distribution.

7.4.2 Installing Connector/C from Source

A Connector/C source distribution is packaged as a compressed [tar](#) file, Zip archive, or RPM package, denoted here as [PACKAGE.tar.gz](#), [PACKAGE.zip](#), or [PACKAGE.src.rpm](#). A source distribution in [tar](#) file or Zip archive format can be used on any supported platform. An RPM package source distribution is intended for RPM-based systems such as Linux.

To unpack a compressed [tar](#) file, use this command in the intended installation directory:

```
shell> tar zxvf PACKAGE.tar.gz
```

After unpacking the distribution, build it using the appropriate instructions for your platform later in this section.

To unpack a Zip archive, use [WinZip](#) or another tool that can read [.zip](#) files. After unpacking the distribution, build it using the appropriate instructions for your platform later in this section.

To install an RPM package, use this command to create binary RPM packages that you can install. If you do not have [rpmbuild](#), use [rpm](#) instead.

```
shell> rpmbuild --rebuild --clean PACKAGE.src.rpm
```

The command should produce binary `shared` and `devel` RPM packages and indicate where it placed them. You can install these packages using the instructions in [Section 7.4.1, “Installing Connector/C from a Binary Distribution”](#).

7.4.2.1 Installing Connector/C from Source on Unix and Unix-Like Systems

If the native compiler toolset for the target platform is available (for example, SunStudio for Solaris), you can use that for compilation. Alternatively, the GNU toolset can be used on all platforms.

You also need [CMake 2.6](#) or newer, which is available from [cmake.org](#).

To build and install the source distribution, use the following procedure:

1. Change location to the top-level directory of the source distribution.
2. Generate the [Makefile](#):

```
shell> cmake -G "Unix Makefiles"
```

Or, for a Debug build:

```
shell> cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug
```

By default, the installation location for Connector/C is `/usr/local/mysql`. To change this location, use the `CMAKE_INSTALL_PREFIX` option to specify a different directory when generating the [Makefile](#). For example:

```
shell> cmake -G "Unix Makefiles" -DCMAKE_INSTALL_PREFIX=/opt/local/mysql
```

For other [CMake](#) options that you might find useful, see [Other Connector/C Build Options](#).

3. Build the project:

```
shell> make
```

4. As `root`, install the Connector/C headers, libraries, and utilities:

```
root-shell> make install
```

7.4.2.2 Installing Connector/C from Source on Microsoft Windows

To build Connector/C on Microsoft Windows, Visual Studio 8 or 9 is recommended. The Express Edition of Visual Studio and other compilers might work, but are untested.

You also need [CMake 2.6](#) or newer, which is available from [cmake.org](#).

To build and install the source distribution, use the following procedure:

1. Set the environment variables for the Visual Studio toolchain. Visual Studio includes a batch file to set these for you, and installs a shortcut in the **Start** menu to open a command prompt with these variables set.
2. Change location to the top-level directory of the source distribution.
3. Generate the [Makefile](#) by entering the following command in a command-prompt window:

```
shell> cmake -G "Visual Studio 9 2008"
```

For other [CMake](#) options that you might find useful, see [Other Connector/C Build Options](#).

The result of the `cmake` command is a project (solution) file, `libmysql.sln`, that you can open with Visual Studio. Alternatively, build from the command line with either of these commands:

```
shell> devenv.com libmysql.sln /build Release
```

```
shell> devenv.com libmysql.sln /build RelWithDebInfo
```

For other versions of Visual Studio or for an `nmake`-based build, use the following command to check which generators can be specified with the `-G` option:

```
shell> cmake --help
```

To compile a Debug build, you must set the `CMake` build type so the correct external library versions are used, then compile using the `Debug` solution configuration:

```
shell> cmake -G "Visual Studio 9 2008" -DCMAKE_BUILD_TYPE=Debug
shell> devenv.com libmysql.sln /build Debug
```

A normal build builds the C API libraries for the `lib` directory. A Debug build additionally builds debug libraries for the `lib/debug` directory. You must use the debug libraries to compile clients built using the debug C runtime.

4. Use the install operation provided by your development environment to install the Connector/C headers, libraries, and utilities. You can also use this `CMake` command:

```
shell> cmake --build . --target INSTALL --config RelWithDebInfo
```

7.4.2.3 Other Connector/C Build Options

The following tables show other options that can be used when building Connector/C from source.

Table 7.1 Build Options for Connector/C 6.1

Build Option	Description
<code>-DWITH_SSL=system</code>	Enable dynamic linking to the system OpenSSL library.
<code>-DWITH_ZLIB=system</code>	Enable dynamic linking to the system Zlib library.

Table 7.2 Build Options for Connector/C 6.0

Build Option	Description
<code>-DWITH_OPENSSL=1</code>	Enable dynamic linking to the system OpenSSL library.
<code>-DWITH_EXTERNAL_ZLIB=1</code>	Enable dynamic linking to the system Zlib library.

7.4.3 Postinstallation Steps

Connector/C binary `.tar.gz` and `.zip` packages unpack into a directory with a name such as `mysql-connector-c-6.1.0-linux-rhel5-x86-64bit`. If you want to work with a simpler name, rename the directory. On Unix, an alternative is to create a symbolic link with a simpler name:

```
shell> ln -s mysql-connector-c-6.1.0-linux-rhel5-x86-64bit connector-c
```

When you build C applications that use Connector/C, if the compiler or linker have trouble finding the Connector/C header files or libraries, you may need to adjust your development tools or runtime environment. See [Building C API Client Programs](#), and [Running C API Client Programs](#).

7.4.4 Testing Connector/C

If you build Connector/C from source, you can use the instructions in this section to test it. The details of the test procedure depend on your Connector/C version, except that a running MySQL server instance must be available regardless of version.

To test Connector/C 6.1:

Use the `mysql_client_test` utility in the `tests` directory. For information about the MySQL Test Framework, see the manual available at <http://dev.mysql.com/doc/mysqltest/2.0/en/>.

To test Connector/C 6.0:

Use the `ctest` command. Before you run the test suite, specify the following environment variables:

- `MYSQL_TEST_HOST`: The host where the MySQL server is running (default `localhost`)
- `MYSQL_TEST_USER`: The user name of the MySQL account to use
- `MYSQL_TEST_PASSWD`: The password of the MySQL account to use
- `MYSQL_TEST_PORT`: The TCP/IP port to connect to
- `MYSQL_TEST_SOCKET`: The socket file to connect to
- `MYSQL_TEST_DB`: The default database to use (default `test`)

To run the test suite, execute `ctest` from the command line:

```
shell> ctest
```

7.5 Building Connector/C Applications

To build C applications that use Connector/C, the connector must be installed. If you need to do that first, see [Section 7.4, "Installing Connector/C"](#).

For instructions on building Connector/C applications, see [Building C API Client Programs](#). To enable your compiler to find the header and library files under the directory where you installed Connector/C, specify the appropriate compile-time options, as indicated in that section.

For binary distributions, the `docs/INFO_BIN` file contains information about the build environment used to compile Connector/C. This may help you select compatible tools for compiling client applications.

Chapter 8 MySQL Connector/C++ Developer Guide

Table of Contents

8.1 Introduction to Connector/C++	1143
8.2 How to Get Connector/C++	1144
8.3 Installing Connector/C++ from a Binary Distribution	1145
8.4 Installing Connector/C++ from Source	1147
8.4.1 Source Installation System Prerequisites	1147
8.4.2 Obtaining and Unpacking a Connector/C++ Source Distribution	1147
8.4.3 Installing Connector/C++ from Source on Unix and Unix-Like Systems	1148
8.4.4 Installing Connector/C++ from Source on Windows	1149
8.4.5 Troubleshooting Connector/C++ Source Installation Problems	1152
8.4.6 Dynamically Linking Connector/C++ Against the MySQL Client Library	1152
8.4.7 Connector/C++ Source-Configuration Options	1153
8.5 Building Connector/C++ Windows Applications with Microsoft Visual Studio	1155
8.6 Building Connector/C++ Linux Applications with NetBeans	1163
8.7 Getting Started with Connector/C++: Usage Examples	1167
8.7.1 Connecting to MySQL	1168
8.7.2 Running a Simple Query	1169
8.7.3 Fetching Results	1169
8.7.4 Using Prepared Statements	1170
8.7.5 Complete Example 1	1170
8.7.6 Complete Example 2	1171
8.8 Connector/C++ Tutorials	1172
8.8.1 Prerequisites and Background Information	1173
8.8.2 Calling Stored Procedures with Statement Objects	1174
8.8.3 Calling Stored Procedures with PreparedStatement Objects	1178
8.9 Connector/C++ Debug Tracing	1181
8.10 Connector/C++ Usage Notes	1182
8.11 Connector/C++ Known Bugs and Issues	1186
8.12 Connector/C++ Support	1187

8.1 Introduction to Connector/C++

MySQL Connector/C++ is a MySQL database connector for C++. It lets you develop C++ applications that connect to the MySQL Server.

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

MySQL Connector/C++ Benefits

MySQL Connector/C++ offers the following benefits for C++ users compared to the MySQL C API (MySQL client library):

- Convenience of pure C++; no C function calls required
- Supports JDBC 4.0, an industry standard API
- Supports the object-oriented programming paradigm
- Reduces development time
- Licensed under the GPL with the FLOSS License Exception
- Available under a commercial license upon request

JDBC Compatibility

Connector/C++ is compatible with the JDBC 4.0 API. Connector/C++ does not implement the entire JDBC 4.0 API, but does feature the following classes:

- [Connection](#)
- [DatabaseMetaData](#)
- [Driver](#)
- [PreparedStatement](#)
- [ResultSet](#)
- [ResultSetMetaData](#)
- [Savepoint](#)
- [Statement](#)

The JDBC 4.0 API defines approximately 450 methods for the classes just mentioned. Connector/C++ implements approximately 80% of these.

Platform Support and Prerequisites

MySQL Connector/C++ requires:

- MySQL 5.1 or higher
- Microsoft Visual Studio 2013 on Windows

To see which platforms are supported, visit the [Connector/C++ downloads page](#).

8.2 How to Get Connector/C++

Connector/C++ distributions are available in binary and source formats. To obtain a distribution, visit the [Connector/C++ downloads page](#).

MySQL Connector/C++ Binary Distributions

Binary distributions are available for these platforms:

Microsoft Windows:

- MSI installer package
- Without installer (a Zip file)

Unix and Unix-like platforms:

- Compressed GNU TAR archive ([tar.gz](#))

MySQL Connector/C++ Source Distributions

Source packages use compressed GNU TAR file ([tar.gz](#)) format and can be used on any supported platform.

MySQL Connector/C++ Source Repository

The latest development sources are available through [Launchpad](#).

The Connector/C++ code repository uses Bazaar. To check out the latest source code, use the [bzip](#) command-line tool:

```
shell> bzip branch lp:~mysql/mysql-connector-cpp/trunk .
```

8.3 Installing Connector/C++ from a Binary Distribution

To obtain a Connector/C++ binary package, visit the [Connector/C++ downloads page](#).

Note

It is important that the tools you use to build your Connector/C++ applications are compatible with the tools used to build Connector/C++ itself. Ideally, build your applications with the same tools that were used to build the Connector/C++ binaries.

All Connector/C++ distributions include a [README](#) file that contains platform-specific notes. At the end of the [README](#) file, you will find the configuration options used to build the binaries. If you experience build-related issues on a platform, it may help to check the settings that were used to build the binary on that platform.

Developers using Microsoft Windows must satisfy the following requirements:

- Microsoft Visual Studio 2013.
- Your applications should use the same linker configuration as Connector/C++. For example, use one of [/MD](#), [/MDd](#), [/MT](#), or [/MTd](#).

To use a different compiler version, release configuration, or runtime library, compile Connector/C++ from source using your desired settings and build your applications using these same settings. To avoid issues, ensure that the three variables of compiler version, runtime library, and runtime linker configuration settings are the same for your applications and Connector/C++ itself.

Archive Package

Unpack the distribution archive into an appropriate directory. If you plan to use a dynamically linked version of Connector/C++, make sure that your system can reference the [libmysqlclient](#) MySQL client library (Connector/C++ is linked against and thus requires the MySQL client library). Consult your operating system documentation on how to modify and expand the search path for libraries. If you cannot modify the library search path, it may help to copy your application, the Connector/C++ library and the MySQL client library into the same directory. Most systems search for libraries in the current directory.

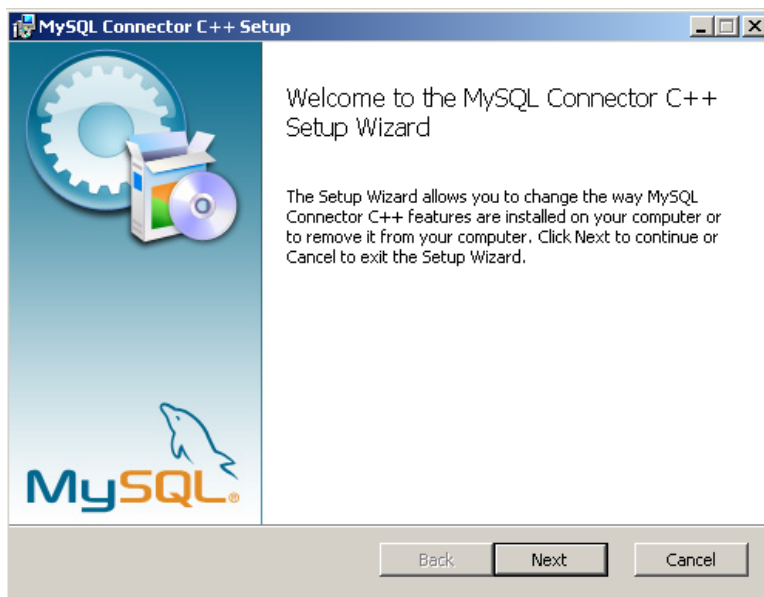
Windows users can choose between two binary packaging formats:

- Windows MSI Installer ([.msi](#) file): To use the MSI Installer, launch it and follow the prompts in the screens it presents to install Connector/C++ in the location of your choosing.
- Zip archive without installer ([.zip](#) file): To use a Zip archive, unpack it in the directory where you intend to install it using [WinZip](#) or another tool that can read [.zip](#) files.

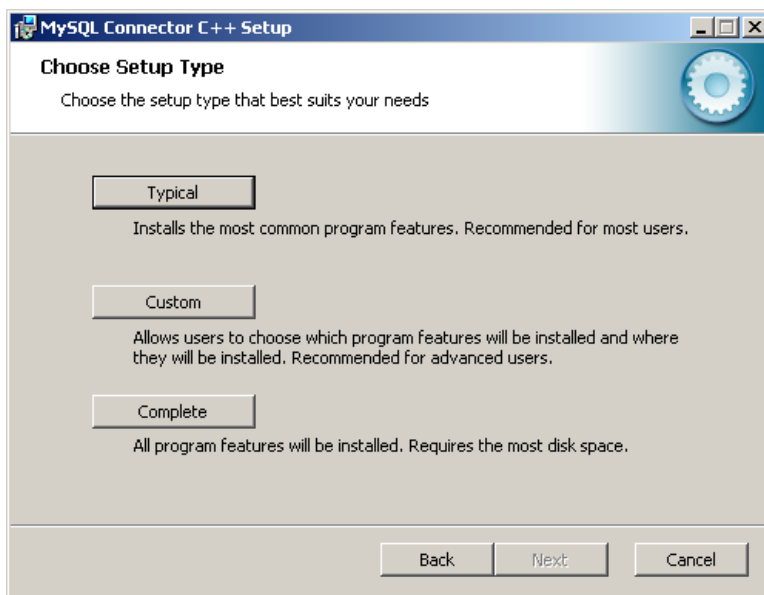
Windows MSI Installer

Using the MSI Installer may be the easiest solution. The MSI Installer does not require any administrative permissions because it simply copies files.

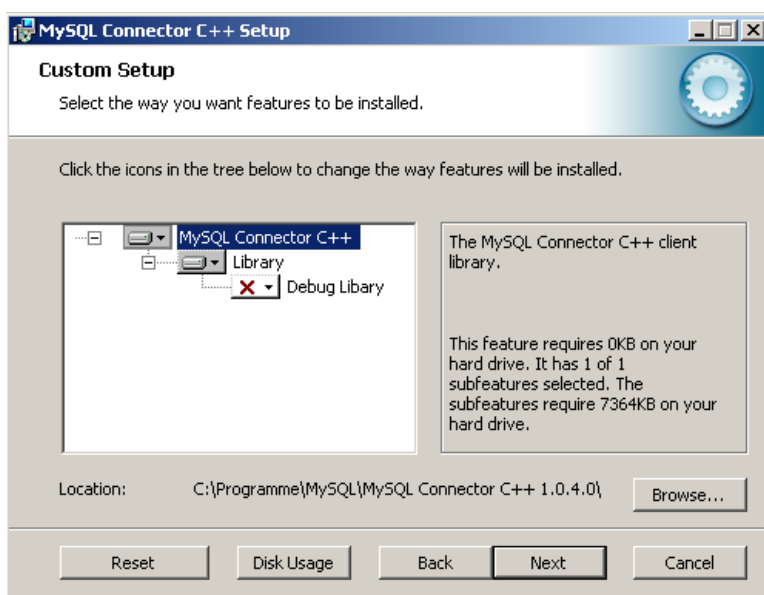
The MSI Installer begins by presenting a welcome screen.

Figure 8.1 MSI Installer Welcome Screen

The MSI Installer overview screen enables you to select the type of installation to perform. The “Typical” installation consists of all required header files and the Release libraries. The “Custom” installation enables you to install additional Debug versions of the connector libraries. The “Complete” installation installs everything in the distribution.

Figure 8.2 MSI Installer Overview Screen

If you select a “Custom” installation, the MSI Installer presents a Custom Setup screen that enables you to select which features to install and where to install them.

Figure 8.3 MSI Installer Custom Setup Screen

8.4 Installing Connector/C++ from Source

This chapter describes how to install Connector/C++ using a source distribution.

8.4.1 Source Installation System Prerequisites

To install Connector/C++ from source, the following system requirements must be satisfied:

- To compile Connector/C++, the MySQL client library (MySQL C API) must be installed. Connector/C++ is based on the MySQL client library and is linked against it.

Typically, the MySQL client library is installed when the MySQL Server is installed. However, check your operating system documentation for other installation options. Alternatively, you can install the client library by installing Connector/C.

You must use either a General Availability version of MySQL 5.7 (5.7.9 or higher) or Connector/C 6.1.8 or higher. Set the `MYSQL_DIR CMake` option appropriately at configuration time as necessary.

- You must have the cross-platform build tool `CMake` 2.6.2, or newer, and `GLib` 2.2.3 or newer. Check the `README` file included with the distribution for platform-specific notes.
- The Boost C++ libraries are required to build Connector/C++ (but not to use it). Boost 1.56.0 or newer must be installed. To obtain Boost and its installation instructions, visit [the official site](#). After Boost is installed, tell the build system where the Boost files are located by defining the `BOOST_ROOT` option when you invoke `CMake`. For example:

```
shell> cmake . -DBOOST_ROOT=/usr/local/boost_1_56_0
```

Adjust the path as necessary to match your installation.

8.4.2 Obtaining and Unpacking a Connector/C++ Source Distribution

To obtain a Connector/C++ source package, visit the [Connector/C++ downloads page](#).

A Connector/C++ source distribution is packaged as a compressed `tar` file or Zip archive, denoted here as `PACKAGE.tar.gz` or `PACKAGE.zip`. A source distribution in `tar` file or Zip archive format can be used on any supported platform.

To unpack a compressed `tar` file, use this command in the intended installation directory:

```
shell> tar zxvf PACKAGE.tar.gz
```

After unpacking the distribution, build it using the appropriate instructions for your platform later in this section.

To unpack a Zip archive, use [WinZip](#) or another tool that can read `.zip` files. After unpacking the distribution, build it using the appropriate instructions for your platform later in this section.

To check out the Connector/C++ code from the source code repository located on GitHub at <https://github.com/mysql/mysql-connector-cpp>, use the following command:

```
shell> git clone https://github.com/mysql/mysql-connector-cpp
```

8.4.3 Installing Connector/C++ from Source on Unix and Unix-Like Systems

To install Connector/C++ from source, your system must satisfy the requirements outlined in [Section 8.4.1, "Source Installation System Prerequisites"](#).

1. Change location to the top-level directory of the source distribution:

```
shell> cd /path/to/mysql-connector-cpp
```

2. Run `CMake` to build a `Makefile`:

```
shell> cmake .
-- Check for working C compiler: /usr/local/bin/gcc
-- Check for working C compiler: /usr/local/bin/gcc -- works
[...]
-- Generating done
-- Build files have been written to: /path/to/mysql-connector-cpp/
```

To use configuration values different from the defaults, use the options described at [Section 8.4.7, "Connector/C++ Source-Configuration Options"](#).

On non-Windows systems, `CMake` checks to see whether the `MYSQL_CONFIG_EXECUTABLE` `CMake` option is set. If not, `CMake` tries to locate `mysql_config` in the default locations.

If you have any problems with the configuration process, check the troubleshooting instructions given in [Section 8.4.5, "Troubleshooting Connector/C++ Source Installation Problems"](#).

3. Use `make` to build Connector/C++. First make sure you have a clean build, then build the connector:

```
shell> make clean
shell> make
[ 1%] Building CXX object »
driver/CMakeFiles/mysqlcppconn.dir/mysql_connection.o
[ 3%] Building CXX object »
driver/CMakeFiles/mysqlcppconn.dir/mysql_constructed_resultset.o
[...]
[100%] Building CXX object examples/CMakeFiles/statement.dir/statement.o
Linking CXX executable statement
```

If all goes well, you will find the Connector/C++ library in the `driver` directory.

4. Install the header and library files:

```
shell> make install
```

Unless you have changed the location in the configuration step, `make install` copies the header files to the directory `/usr/local/include`. The header files copied are `mysql_connection.h` and `mysql_driver.h`.

Again, unless you have specified otherwise, `make install` copies the library files to `/usr/local/lib`. The files copied are the dynamic library `libmysqlcppconn.so`, and the static library `libmysqlcppconn-static.a`. The dynamic library file name extension might differ on your system (for example, `.dylib` on OS X).

After installing Connector/C++, you can carry out a quick test to check the installation. To do this, compile and run one of the example programs, such as `examples/standalone_example.cpp`. This example is discussed in more detail later, but for now, you can use it to test whether the connector has been correctly installed. This procedure assumes that you have a working MySQL Server to which you can connect. It also assumes header and library locations of `/usr/local/include` and `/usr/local/lib`, respectively; adjust these as necessary for your system.

1. Compile the example program. To do this, change location to the `examples` directory and enter this command:

```
shell> g++ -o test_install \
-I/usr/local/include -I/usr/local/include/cppconn \
-Wl,-Bdynamic -lmysqlcppconn standalone_example.cpp
```

2. Make sure the dynamic library which is used in this case can be found at runtime:

```
shell> export LD_LIBRARY_PATH=/usr/local/lib
```

On OS X, try this:

```
shell> export DYLD_LIBRARY_PATH=/usr/local/lib
```

3. Now run the program to test your installation, substituting the appropriate host, user, password, and database arguments for your system:

```
shell> ./test_install localhost root password database
```

You should see output similar to the following:

```
Connector/C++ standalone program example...
... running 'SELECT 'Welcome to Connector/C++' AS _message'
... MySQL replies: Welcome to Connector/C++
... say it again, MySQL
...MySQL replies: Welcome to Connector/C++
... find more at http://www.mysql.com
```

If you see any errors, take note of them and go through the troubleshooting procedures in [Section 8.4.5, “Troubleshooting Connector/C++ Source Installation Problems”](#).

8.4.4 Installing Connector/C++ from Source on Windows

To install Connector/C++ from source, your system must satisfy the requirements outlined in [Section 8.4.1, “Source Installation System Prerequisites”](#).

Note

On Windows, `mysql_config` is not present, so `CMake` attempts to retrieve the location of MySQL from the environment variable `$ENV{MYSQL_DIR}`. If `MYSQL_DIR` is not set, `CMake` checks for MySQL in the following locations:

```
$ENV{ProgramFiles}/MySQL/*/include and $ENV{SystemDrive}/MySQL/*/include.
```

Consult the [CMake](#) manual or check `cmake --help` to find out which build systems are supported by your [CMake](#) version:

```
C:\> cmake --help
cmake version 2.6-patch 2
Usage
[...]
Generators
The following generators are available on this platform:
  Borland Makefiles      = Generates Borland makefiles.
  MSYS Makefiles         = Generates MSYS makefiles.
  MinGW Makefiles        = Generates a make file for use with
                           mingw32-make.
  NMake Makefiles        = Generates NMake makefiles.
  Unix Makefiles          = Generates standard UNIX makefiles.
  Visual Studio 6         = Generates Visual Studio 6 project files.
  Visual Studio 7         = Generates Visual Studio .NET 2002 project
                           files.
  Visual Studio 7 .NET 2003 = Generates Visual Studio .NET 2003 project
                           files.
  Visual Studio 8 2005     = Generates Visual Studio .NET 2005 project
                           files.
  Visual Studio 8 2005 Win64 = Generates Visual Studio .NET 2005 Win64
                           project files.
  Visual Studio 9 2008     = Generates Visual Studio 9 2008 project fil
  Visual Studio 9 2008 Win64 = Generates Visual Studio 9 2008 Win64 proje
                           files.
[...]
```

It is likely that your [CMake](#) binary supports more compilers, known by [CMake](#) as *generators*, than can actually be used to build Connector/C++. We have built the connector using the following generators:

- Microsoft Visual Studio 2013
- NMake

[CMake](#) makes it easy for you to try other compilers. However, you may experience compilation warnings or errors, or linking issues not detected by Visual Studio. Patches are gratefully accepted to fix issues with other compilers.

For troubleshooting and configuration hints, see [Section 8.4.5, "Troubleshooting Connector/C++ Source Installation Problems"](#).

Use these steps to build Connector/C++:

1. Change location to the top-level directory of the source distribution:

```
C:\> cd C:\path_to_mysql_cpp
```

2. Run [CMake](#) to generate build files for your generator.

To use configuration values different from the defaults, use the options described at [Section 8.4.7, "Connector/C++ Source-Configuration Options"](#).

Visual Studio

```
C:\>cmake -G "Visual Studio 12 2013"
-- Check for working C compiler: cl
-- Check for working C compiler: cl -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: cl
-- Check for working CXX compiler: cl -- works
```



```
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- ENV{MYSQL_DIR} =
-- MySQL Include dir: C:/Programs/MySQL/MySQL Server 5.5/include
-- MySQL Library      : C:/Programs/MySQL/MySQL Server 5.5/lib/opt/mysqlclient.lib
-- MySQL Library dir: C:/Programs/MySQL/MySQL Server 5.5/lib/opt
-- MySQL CFLAGS:
-- MySQL Link flags:
-- MySQL Include dir: C:/Programs/MySQL/MySQL Server 5.5/include
-- MySQL Library dir: C:/Programs/MySQL/MySQL Server 5.5/lib/opt
-- MySQL CFLAGS:
-- MySQL Link flags:
-- Configuring cppconn
-- Configuring test cases
-- Looking for isinf
-- Looking for isinf - not found
-- Looking for isinf
-- Looking for isinf - not found.
-- Looking for finite
-- Looking for finite - not found.
-- Configuring C/J junit tests port
-- Configuring examples
-- Configuring done
-- Generating done
-- Build files have been written to: C:\path_to_mysql_cpp
C:\> dir *.sln *.vcproj
[...]
```

19.11.2008	12:16	23.332	MYSQLCPPCONN.sln
[...]			
19.11.2008	12:16	27.564	ALL_BUILD.vcproj
19.11.2008	12:16	27.869	INSTALL.vcproj
19.11.2008	12:16	28.073	PACKAGE.vcproj
19.11.2008	12:16	27.495	ZERO_CHECK.vcproj

NMake

```
C:\> cmake -G "NMake Makefiles"
-- The C compiler identification is MSVC
-- The CXX compiler identification is MSVC
[...]
```

3. Use your compiler to build Connector/C++.

Visual Studio

Open the newly generated project files in the Visual Studio GUI or use a Visual Studio command line to build the driver. The project files contain a variety of different configurations, debug and nondebug versions among them.

For example, to use the command line, execute commands like these to build the distribution and create a Zip package:

```
C:\> devenv MYSQLCPPCONN.sln /build RelWithDebInfo
C:\> cpack -G ZIP --config CPackConfig.cmake -C RelWithDebInfo
CPack: Create package using ZIP
CPack: Install projects
CPack: - Install project: MYSQLCPPCONN
CPack: Create package
CPack: - package: C:/opt/build/mysql-connector-c++-1.1.5/mysql-connector-c++-1.1.7-win32.zip generated
```

NMake

```
C:\>nmake
Microsoft (R) Program Maintenance Utility Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.
Scanning dependencies of target mysqlcppconn
```

```
[ 2%] Building CXX object driver/CMakeFiles/mysqlcppconn.dir/mysql_connection.obj
mysql_connection.cpp
[...]
Linking CXX executable statement.exe
[100%] Built target statement
```

8.4.5 Troubleshooting Connector/C++ Source Installation Problems

If you encounter problems building Connector/C++, try these troubleshooting strategies:

1. For configuration or compilation problems, check your [CMake](#) options (MySQL installation path, debug version, and so forth):

```
shell> cmake -L
[...]
CMAKE_BACKWARDS_COMPATIBILITY:STRING=2.4
CMAKE_BUILD_TYPE:STRING=
CMAKE_INSTALL_PREFIX:PATH=/usr/local
EXECUTABLE_OUTPUT_PATH:PATH=
LIBRARY_OUTPUT_PATH:PATH=
MYSQLCPPCONN_GCOV_ENABLE:BOOL=0
MYSQLCPPCONN_TRACE_ENABLE:BOOL=0
MYSQL_CONFIG_EXECUTABLE:FILEPATH=/usr/local/mysql/bin/mysql_config
```

For example, if your MySQL Server installation path is not `/usr/local/mysql` and you want to build a debug version of the Connector/C++, use this command:

```
shell> cmake \
  -DCMAKE_BUILD_TYPE=Debug \
  -DMYSQL_CONFIG_EXECUTABLE=/path/to/my/mysql/server/bin/mysql_config .
```

2. Use `cmake -L` to verify that your settings were changed:

```
shell> cmake -L
[...]
CMAKE_BACKWARDS_COMPATIBILITY:STRING=2.4
CMAKE_BUILD_TYPE:STRING=Debug
CMAKE_INSTALL_PREFIX:PATH=/usr/local
EXECUTABLE_OUTPUT_PATH:PATH=
LIBRARY_OUTPUT_PATH:PATH=
MYSQLCPPCONN_GCOV_ENABLE:BOOL=0
MYSQLCPPCONN_TRACE_ENABLE:BOOL=0
MYSQL_CONFIG_EXECUTABLE=/path/to/my/mysql/server/bin/mysql_config
```

Proceed by executing a `make clean` command followed by a `make` command, as described previously.

8.4.6 Dynamically Linking Connector/C++ Against the MySQL Client Library

Note

This section refers to dynamic linking of Connector/C++ to the client library, not dynamic linking of applications to Connector/C++. Precompiled binaries of Connector/C++ use static binding with the client library by default.

An application that uses Connector/C++ can be either statically or dynamically linked to the Connector/C++ libraries. Connector/C++ is usually statically linked to the underlying MySQL client library (or to Connector/C; unless otherwise stated, reference to the MySQL client library is also taken to include Connector/C, which is a separately packaged, standalone version of the MySQL client library).

As of Connector/C++ 1.1.0, it is also possible to dynamically link to the underlying MySQL client library, although this capability is not enabled by default. To dynamically link Connector/C++ to the client library, enable `MYSQLCLIENT_STATIC_LINKING` when building Connector/C++ from source:

```

shell> rm CMakeCache.txt
shell> cmake . -DMYSQLCLIENT_STATIC_LINKING=1
shell> make clean
shell> make
shell> make install

```

Now, when creating a connection in your application, Connector/C++ will select and load a client library at runtime. It will choose the client library by searching defined locations and environment variables depending on the host operating system. It is also possible when creating a connection in an application to define an absolute path to the client library to load at runtime. This can be convenient if you have defined a standard location from which you want the client library to be loaded. This is sometimes done to circumvent possible conflicts with other versions of the client library that may be located on the system.

8.4.7 Connector/C++ Source-Configuration Options

Connector/C++ recognizes the [CMake](#) options described in this section.

Table 8.1 Connector/C++ Source-Configuration Option Reference

Formats	Description	Default	Introduced
BOOST_ROOT	The Boost source directory		
CMAKE_BUILD_TYPE	Type of build to produce	RelWithDebInfo	
CMAKE_ENABLE_C++11	Whether to enable C++11 support	OFF	1.1.6
CMAKE_INSTALL_PREFIX	Installation base directory	/usr/local	
MYSQLCLIENT_NO_THREADS	Whether to link against single-threaded MySQL client library	OFF	
MYSQLCLIENT_STATIC_LINKING	Whether to statically link to the MySQL client library	OFF	
MYSQLCPPCONN_GCOV_ENABLE	Whether to enable gcov support	OFF	
MYSQLCPPCON_TRACE_ENABLE	Whether to enable tracing functionality	OFF	
MYSQL_CFLAGS	C compiler flags		
MYSQL_CONFIG_EXECUTABLE	Path to the mysql_config program	\${MYSQL_DIR}/bin/mysql_config	
MYSQL_CXXFLAGS	C++ compiler flags		
MYSQL_CXX_LINKAGE	Whether MySQL client library needs C++ linking	ON	
MYSQL_DIR	MySQL Server or Connector/C installation directory		
MYSQL_EXTRA_LIBRARIES	Extra link libraries		
MYSQL_INCLUDE_DIR	The MySQL header file directory	\${MYSQL_DIR}/include	
MYSQL_LIB_DIR	The MySQL client library directory	\${MYSQL_DIR}/lib	
MYSQL_LINK_FLAGS	Extra link flags		
USE_SERVER_CXXFLAGS	Use MySQL Server CXXFLAGS value rather than system default	OFF	1.1.7

- [-DBOOST_ROOT=dir_name](#)

The directory where the Boost sources are installed.

- `-DCMAKE_BUILD_TYPE=type`

The type of build to produce:

- `RelWithDebInfo`: Enable optimizations and generate debugging information. This is the default Connector/C++ build type.
- `Debug`: Disable optimizations and generate debugging information.
- `-DCMAKE_ENABLE_C++11=bool`

Whether to enable C++11 support. The default is `OFF`.

This option was added in Connector/C++ 1.1.6.

- `-DCMAKE_INSTALL_PREFIX=dir_name`

The installation base directory.

- `-DMYSQLCLIENT_NO_THREADS=bool`

Whether to link against a single-threaded `libmysqlclient` MySQL client library. In MySQL 5.5 and higher, the value of this option makes no difference because `libmysqlclient` is always multi-threaded.

- `-DMYSQLCLIENT_STATIC_LINKING=bool`

Whether to statically link to the MySQL client library. The default is `OFF` (use dynamic linking to the client library). For more information, see [Section 8.4.6, “Dynamically Linking Connector/C++ Against the MySQL Client Library”](#).

- `-DMYSQLCPPCONN_GCOV_ENABLE=bool`

Whether to enable `gcov` support.

- `-DMYSQLCPPCON_TRACE_ENABLE=VALUE_TYPE`

Whether to enable tracing functionality. For information about tracing, see [Section 8.9, “Connector/C++ Debug Tracing”](#).

- `-DMYSQL_CFLAGS=flags`

C compiler flags.

- `-DMYSQL_CONFIG_EXECUTABLE=file_name`

The path to the `mysql_config` program.

On non-Windows systems, `CMake` checks to see whether the `MYSQL_CONFIG_EXECUTABLE` `CMake` option is set. If not, `CMake` tries to locate `mysql_config` in the default locations.

- `-DMYSQL_CXXFLAGS=flags`

C++ compiler flags.

- `-DMYSQL_CXX_LINKAGE=bool`

Whether the MySQL client library needs C++ linking.

- `-DMYSQL_DIR=dir_name`

The directory where MySQL Server or Connector/C is installed.

- `-DMYSQL_EXTRA_LIBRARIES=flags`

Any needed additional link libraries.

- `-DMYSQL_INCLUDE_DIR=dir_name`

The directory where the MySQL header files are installed.

- `-DMYSQL_LIB_DIR=dir_name`

The directory where the MySQL client library is installed.

- `-DMYSQL_LINK_FLAGS=flags`

Any needed additional link flags.

- `-DUSE_SERVER_CXXFLAGS=bool`

Use MySQL Server `CXXFLAGS` values rather than the system default value.

This option was added in Connector/C++ 1.1.7.

8.5 Building Connector/C++ Windows Applications with Microsoft Visual Studio

Connector/C++ is available as a static or dynamic library to use with your application. This section describes how to link the library to your application.

Note

To avoid potential crashes, the build configuration of Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of Connector/C++ with a debug build of the client application.

Static Library

The Connector/C++ static library file is `mysqlcppconn-static.lib`. You link this library statically with your application. Also link against the files `libmysql.dll` and `libmysql.lib`. At runtime, the application will require access to `libmysql.dll`.

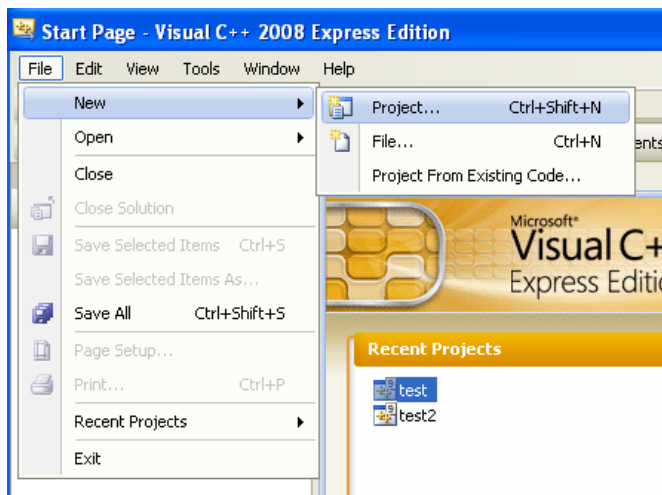
Dynamic Library

The Connector/C++ dynamic library file is `mysqlcppconn.dll`. To build your client application, link it with the file `mysqlcppconn.lib`. At runtime, the application will require access to the files `mysqlcppconn.dll` and `libmysql.dll`.

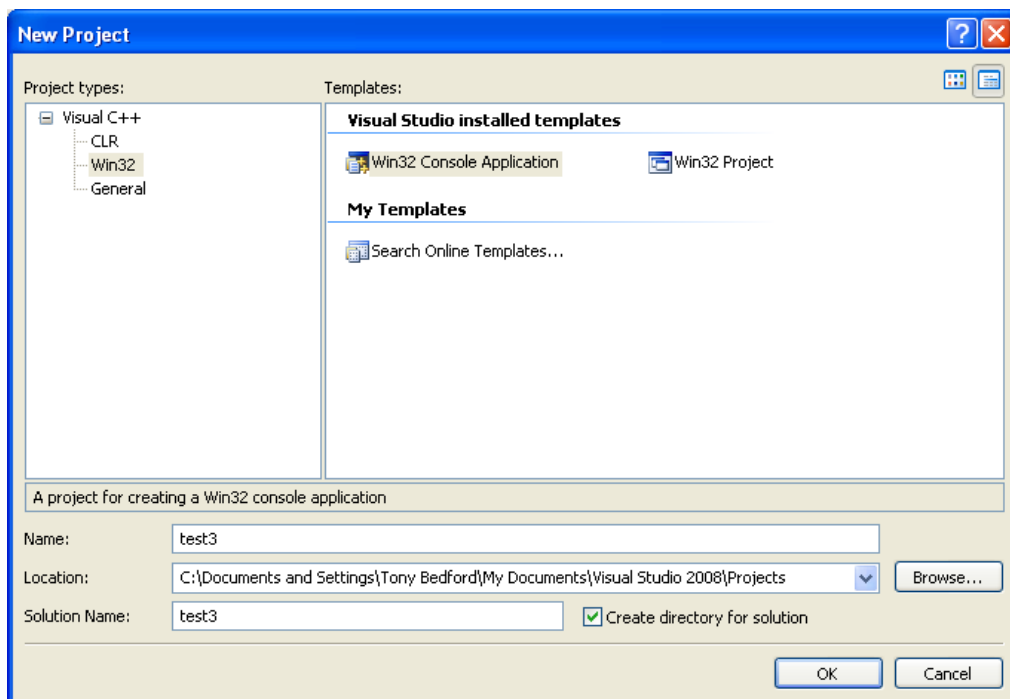
Building a Connector/C++ Application with Microsoft Visual Studio

The initial steps for building an application to use either the static or dynamic library are the same. Some additional steps vary, depend on whether you are building your application to use the [static](#) or [dynamic](#) library.

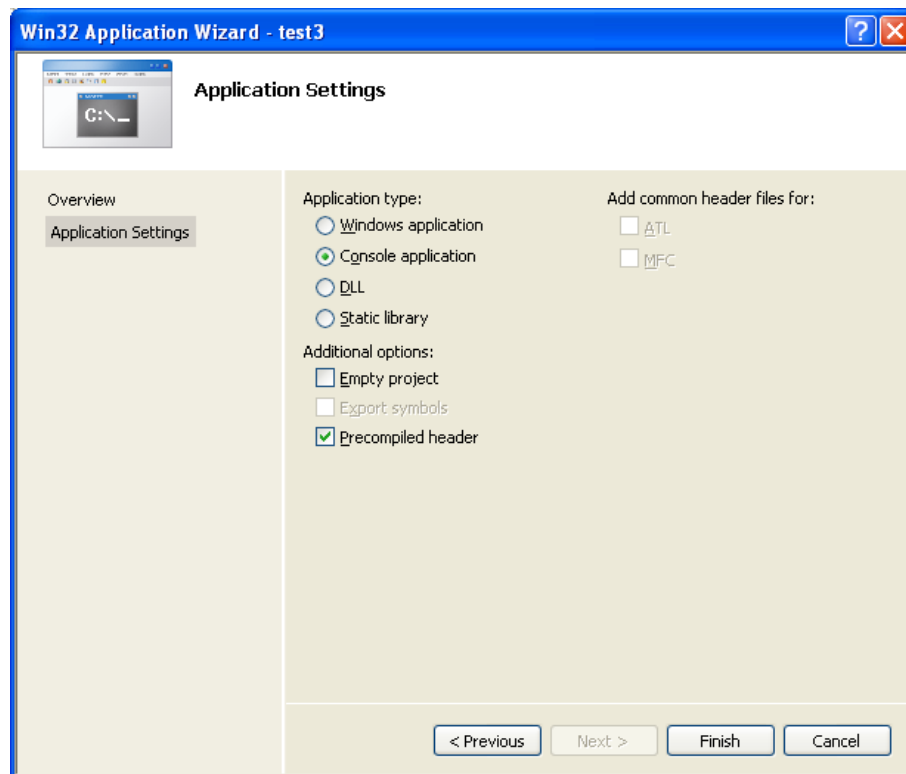
1. Select **File, New, Project** from the main menu.

Figure 8.4 Creating a New Project

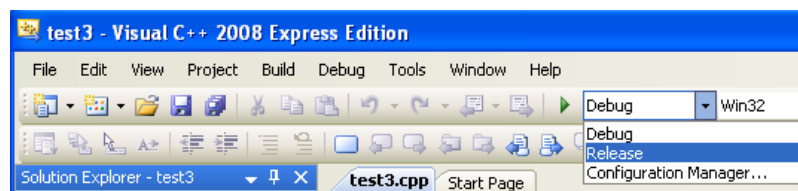
2. In the wizard, select **Visual C++, Win32**. From **Visual Studio Installed Templates**, select the application type **Win32 Console Application**. Enter a name for the application, then click **OK**, to move to the Win32 Application Wizard.

Figure 8.5 The New Project Dialog Box

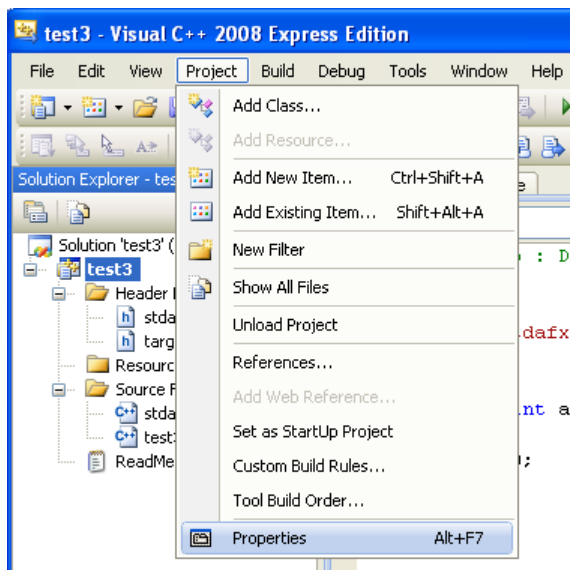
3. In the Win32 Application Wizard, click **Application Settings** and ensure the defaults are selected. The radio button **Console application** and the check box **Precompiled headers** are selected. Click **Finish** to close the wizard.

Figure 8.6 The Win32 Application Wizard

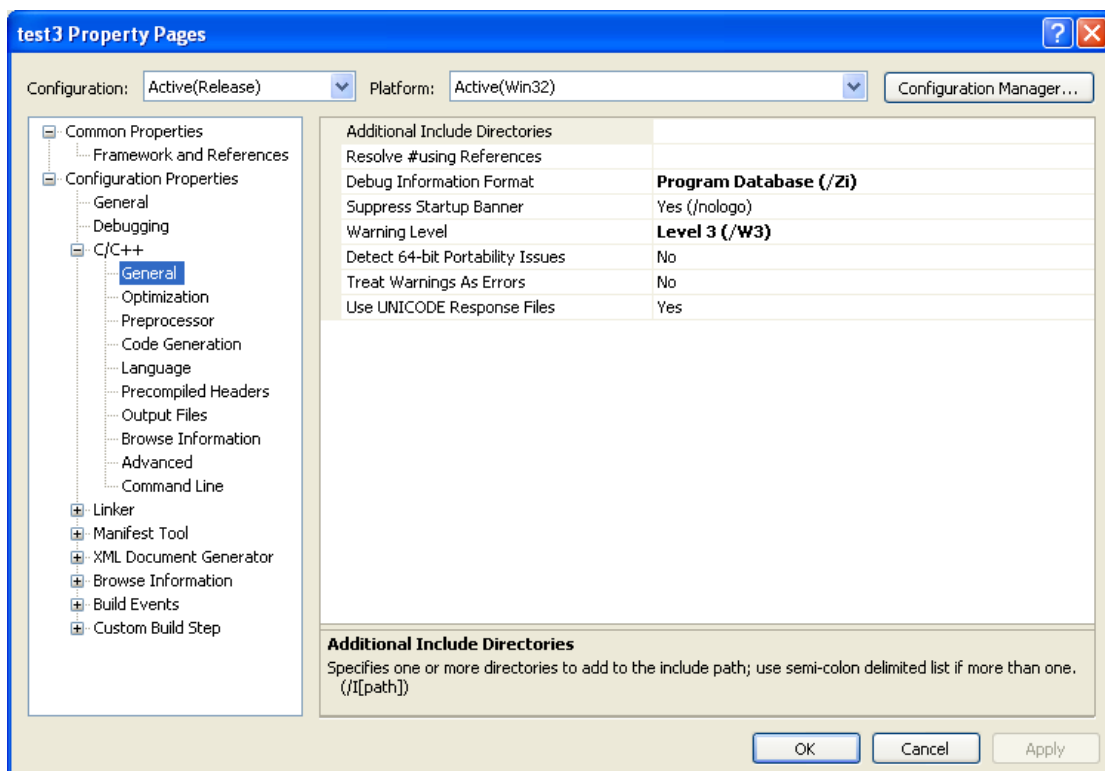
4. From the drop down list box on the toolbar, change from the default **Debug** build to the **Release** build.

Figure 8.7 Selecting the Release Build

5. From the main menu select **Project, Properties**. This can also be accessed using the hot key **ALT + F7**.

Figure 8.8 Selecting Project Properties from the Main Menu

6. Under **Configuration Properties**, open the tree view.
7. Select **C++, General** in the tree view.

Figure 8.9 Setting Properties

8. Ensure that Visual Studio can find the MySQL include directory. This directory includes header files that can optionally be installed when installing MySQL Server.

Figure 8.10 MySQL Include Directory

```

C:\mysql\include>dir
Volume in drive C has no label.
Volume Serial Number is 9484-B7C3

Directory of C:\mysql\include

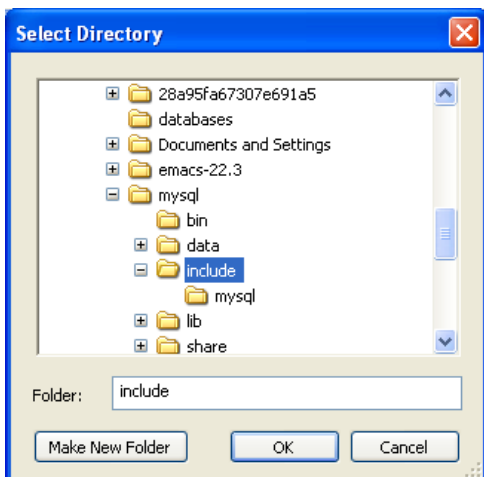
10/12/2008  13:56    <DIR>          .
10/12/2008  13:56    <DIR>          ..
15/11/2008  06:53             10,935  config-win.h
15/11/2008  06:53             4,058  decimal.h
15/11/2008  06:53             3,854  errmsg.h
15/11/2008  06:53             6,954  keycache.h
15/11/2008  06:53             2,819  libmysql.def
10/12/2008  13:56    <DIR>          mysql
15/11/2008  06:53            33,654  mysql.h
15/11/2008  06:53            23,155  mysqld_ename.h
15/11/2008  06:53            23,212  mysqld_error.h
15/11/2008  06:53            20,174  mysql_com.h
15/11/2008  06:53             1,130  mysql_embed.h
15/11/2008  06:53             2,097  mysql_time.h
15/11/2008  06:53             860   mysql_version.h
15/11/2008  06:53             1,804  my_alloc.h
15/11/2008  06:53             1,941  my_attribute.h
15/11/2008  06:53             4,576  my_debug.h
15/11/2008  06:53             3,460  my_dir.h
15/11/2008  06:53             3,278  my_getopt.h
15/11/2008  06:53            50,497  my_global.h
15/11/2008  06:53             1,488  my_list.h
15/11/2008  06:53             3,656  my_net.h
15/11/2008  06:53             1,542  my_no_pthread.h
15/11/2008  06:53            26,468  my_pthread.h
15/11/2008  06:53            40,097  my_sys.h
15/11/2008  06:53             2,677  my_xml.h
15/11/2008  06:53            22,584  mctype.h
15/11/2008  06:53             8,256  mstring.h
15/11/2008  06:53            1,957  sql_common.h
15/11/2008  06:53            12,484  sql_state.h
15/11/2008  06:53             1,006  sslopt-case.h
15/11/2008  06:53            2,205  sslopt-longopts.h
15/11/2008  06:53             1,107  sslopt-vars.h
15/11/2008  06:53             1,501  typelib.h
               32 File(s)              325,486 bytes
                3 Dir(s)      4,091,248,640 bytes free

C:\mysql\include>

```

9. In the **Additional Include Directories** text field, add the MySQL `include/` directory.

Figure 8.11 Select Directory Dialog



10. Also set the location of additional libraries that Visual Studio needs to build the application. These are located in the MySQL `lib/opt` directory, a subdirectory of the MySQL Server installation directory.

Figure 8.12 Typical Contents of MySQL lib/opt Directory

```

C:\mysql\lib\opt>dir
Volume in drive C has no label.
Volume Serial Number is 9484-B7C3

Directory of C:\mysql\lib\opt

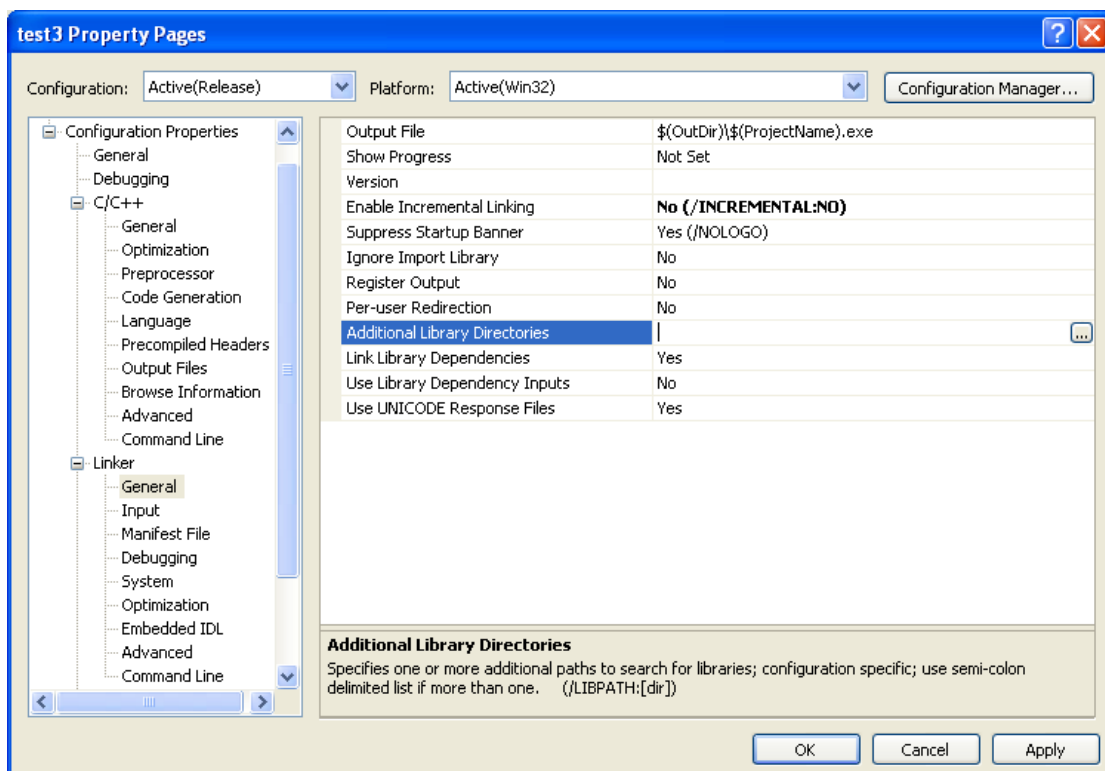
10/12/2008  13:56    <DIR>        .
10/12/2008  13:56    <DIR>        ..
15/11/2008  06:53             34,876  libmysql.lib
15/11/2008  06:53        6,590,464  libmysql.pdb
15/11/2008  06:53        7,710,426  mysqlclient.lib
15/11/2008  06:53        397,312   mysqlclient.pdb
15/11/2008  06:53        2,675,708   mysys.lib
15/11/2008  06:53        135,168     mysys.pdb
15/11/2008  06:53        235,420     regex.lib
15/11/2008  06:53         77,824     regex.pdb
15/11/2008  06:53        2,566,162   strings.lib
15/11/2008  06:53         86,016     strings.pdb
15/11/2008  06:53        202,226     zlib.lib
15/11/2008  06:53         69,632     zlib.pdb
                12 File(s)          20,781,234 bytes
                2 Dir(s)          4,091,240,448 bytes free

C:\mysql\lib\opt>_

```

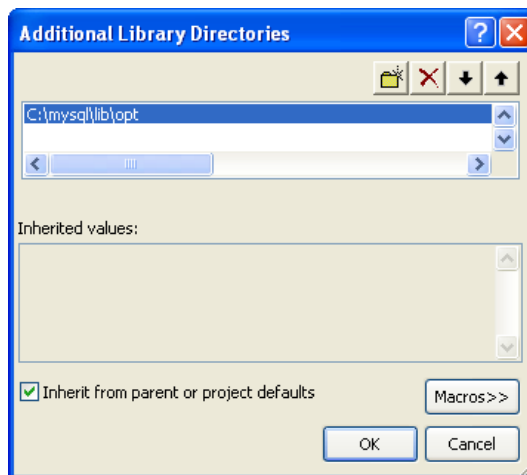
11. In the tree view, open **Linker**, **General**, **Additional Library Directories**.

Figure 8.13 Additional Library Directories



12. Add the **lib/opt** directory into the **Additional Library Directories** text field. This enables the library file **libmysql.lib** to be found.

Figure 8.14 Additional Library Directories Dialog

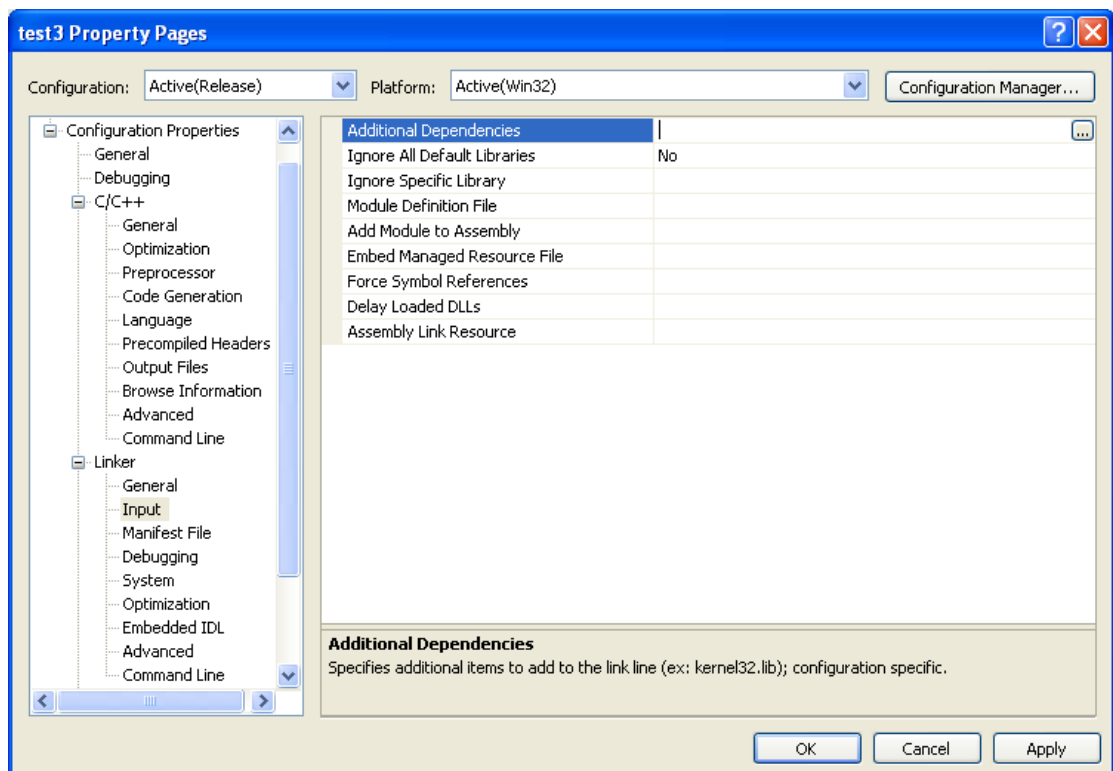


Static Build

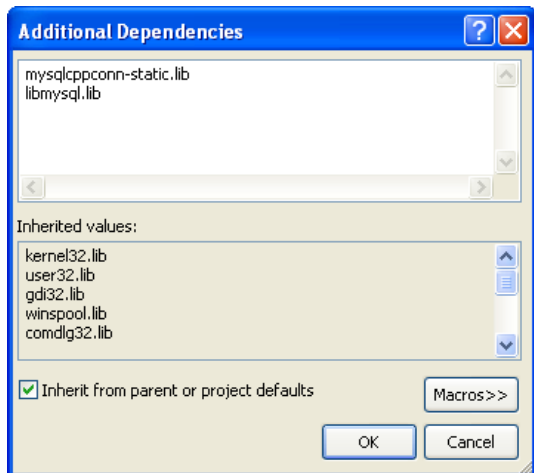
The remaining steps depend on whether you are building an application to use the Connector/C++ static or dynamic library. If you are building your application to use the dynamic library, see [Dynamic Build](#). If you are building your application to use the static library, carry out the following steps:

1. Open **Linker, Input, Additional Dependencies**.

Figure 8.15 Additional Dependencies

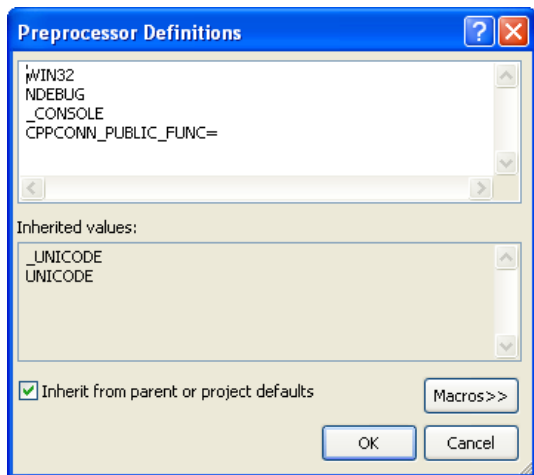


2. Enter `mysqlcppconn-static.lib` and `libmysql.lib`.

Figure 8.16 Adding Additional Dependencies

3. By default `CPPCONN_PUBLIC_FUNC` is defined to declare functions to be compatible with an application that calls a DLL. If you are building an application to call the static library, ensure that function prototypes are compatible with this. In this case, define `CPPCONN_PUBLIC_FUNC` to be an empty string, so that functions are declared with the correct prototype.

In the **Project, Properties** tree view, under **C++, Preprocessor**, enter `CPPCONN_PUBLIC_FUNC=` into the **Preprocessor Definitions** text field.

Figure 8.17 Setting the CPPCONN_PUBLIC_FUNC Define**Note**

Make sure you enter `CPPCONN_PUBLIC_FUNC=` and not `CPPCONN_PUBLIC_FUNC`, so that it is defined as an empty string.

Dynamic build

If you are building an application to use the Connector/C++ dynamically linked library, carry out these steps:

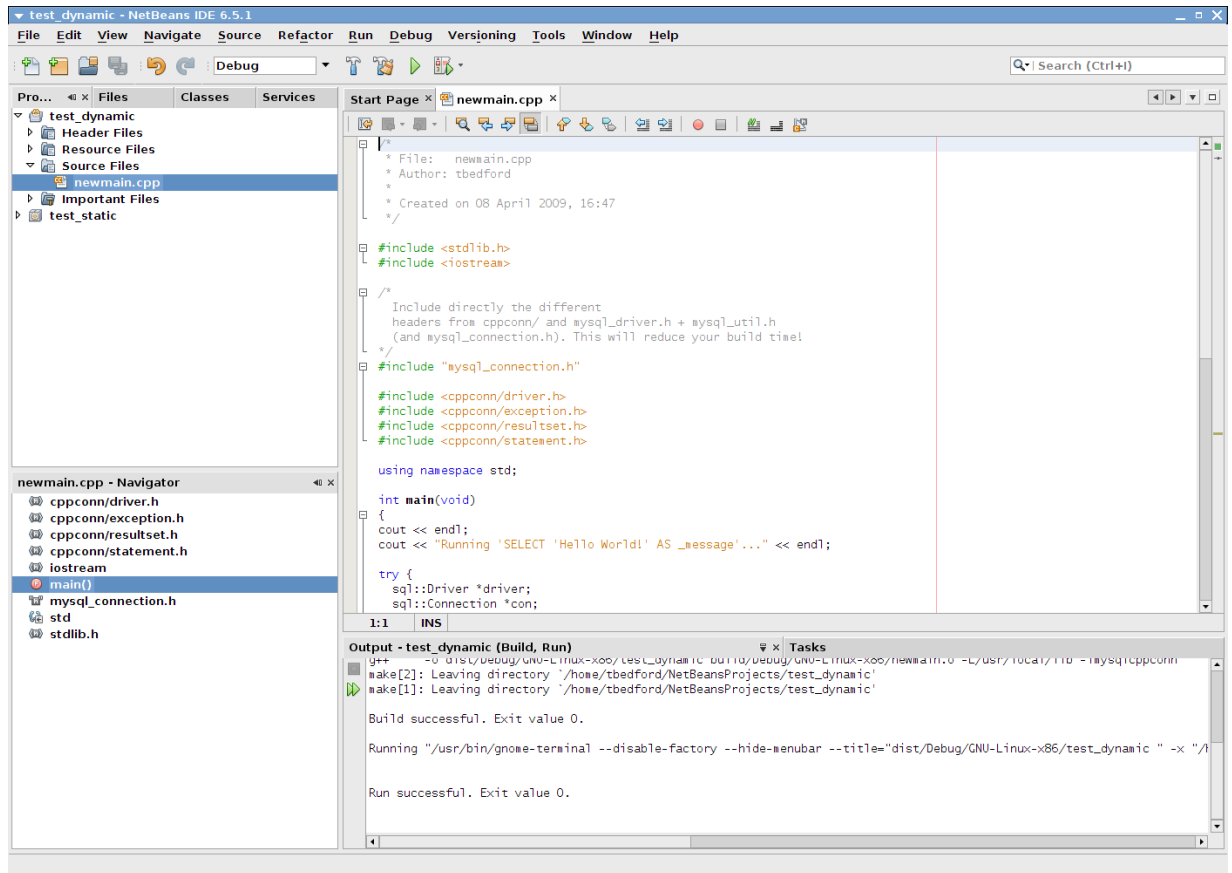
1. Under **Linker, Input**, add `mysqlcppconn.lib` into the **Additional Dependencies** text field.
2. `mysqlcppconn.dll` must be in the same directory as the application executable, or somewhere on the system's path, so that the application can access the Connector/C++ Dynamic Linked Library at runtime.

Copy `mysqlcppconn.dll` to the same directory as the application. Alternatively, extend the `PATH` environment variable using `SET PATH=%PATH%;C:\path\to\cpp`. Alternatively, you can copy `mysqlcppconn.dll` to the Windows installation Directory, typically `c:\windows`.

8.6 Building Connector/C++ Linux Applications with NetBeans

This section describes how to build Connector/C++ applications for Linux using the NetBeans IDE.

Figure 8.18 The NetBeans IDE



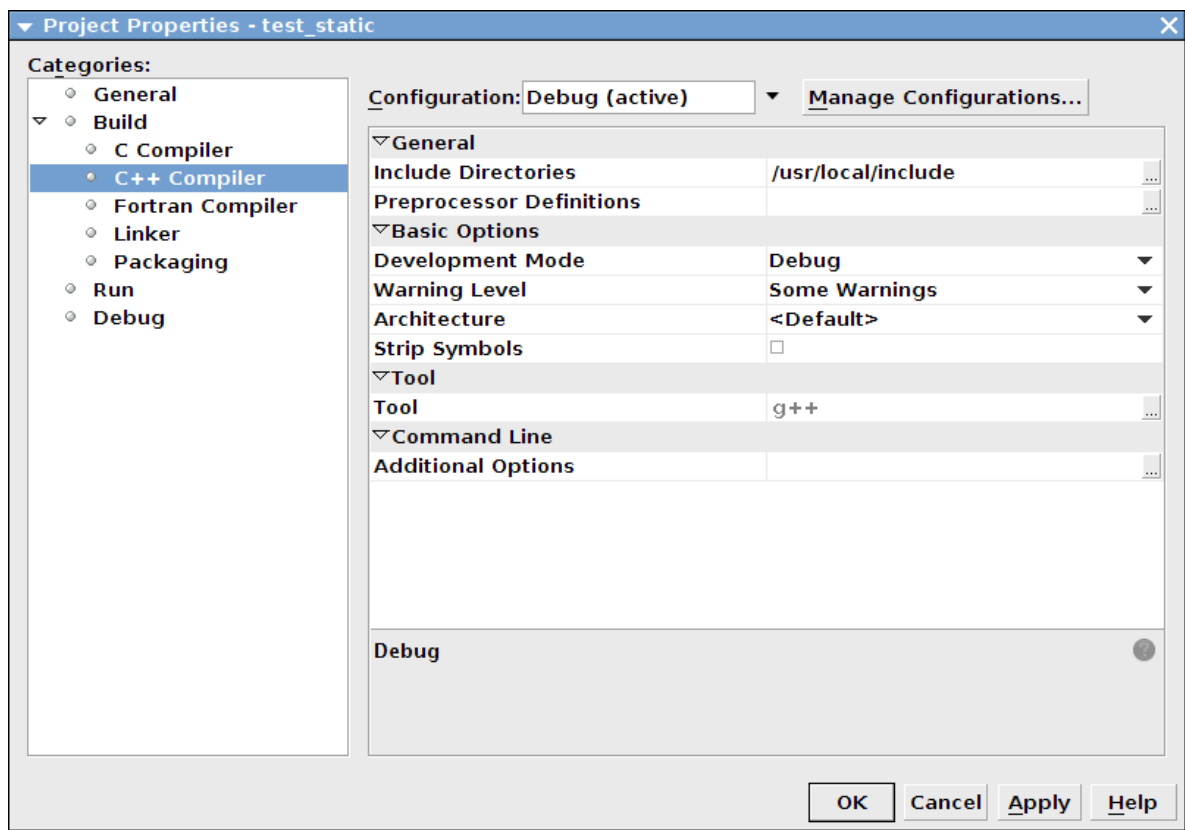
Note

To avoid potential crashes, the build configuration of Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of Connector/C++ with a debug build of the client application.

1. Create a new project. Select **File, New Project**. Choose a **C/C++ Application** and click **Next**.
2. Give the project a name and click **Finish**. A new project is created.
3. In the **Projects** tab, right-click **Source Files** and select **New**, then **Main C++ File...**
4. Change the filename, or simply select the defaults and click **Finish** to add the new file to the project.
5. Now add some working code to your main source file. Explore your Connector/C++ installation and navigate to the `examples` directory.
6. Select a suitable example, such as `standalone_example_docs1.cpp`. Copy all the code in this file, and use it to replace the code in your existing main source file. Amend the code to reflect the connection properties required for your test database. You now have a working example that will access a MySQL database using Connector/C++.

7. At this point, NetBeans shows some errors in the source code. Direct NetBeans to the necessary header files to include. Select **File, Project Properties** from the main menu.
8. In the **Categories:** tree view panel, navigate to **Build, C++ Compiler**.
9. In the **General** panel, select **Include Directories**.
10. Click the ... button.
11. Click **Add**, then navigate to the directory where the Connector/C++ header files are located. This is `/usr/local/include` unless you have installed the files to a different location. Click **Select**. Click **OK**.

Figure 8.19 Setting the Header Include Directory



12. Click **OK** again to close the **Project Properties** dialog.

At this point, you have created a NetBeans project containing a single C++ source file. You have also ensured that the necessary include files are accessible. Before continuing, decide whether your project is to use the Connector/C++ static or dynamic library. The project settings are slightly different in each case, because you link against a different library.

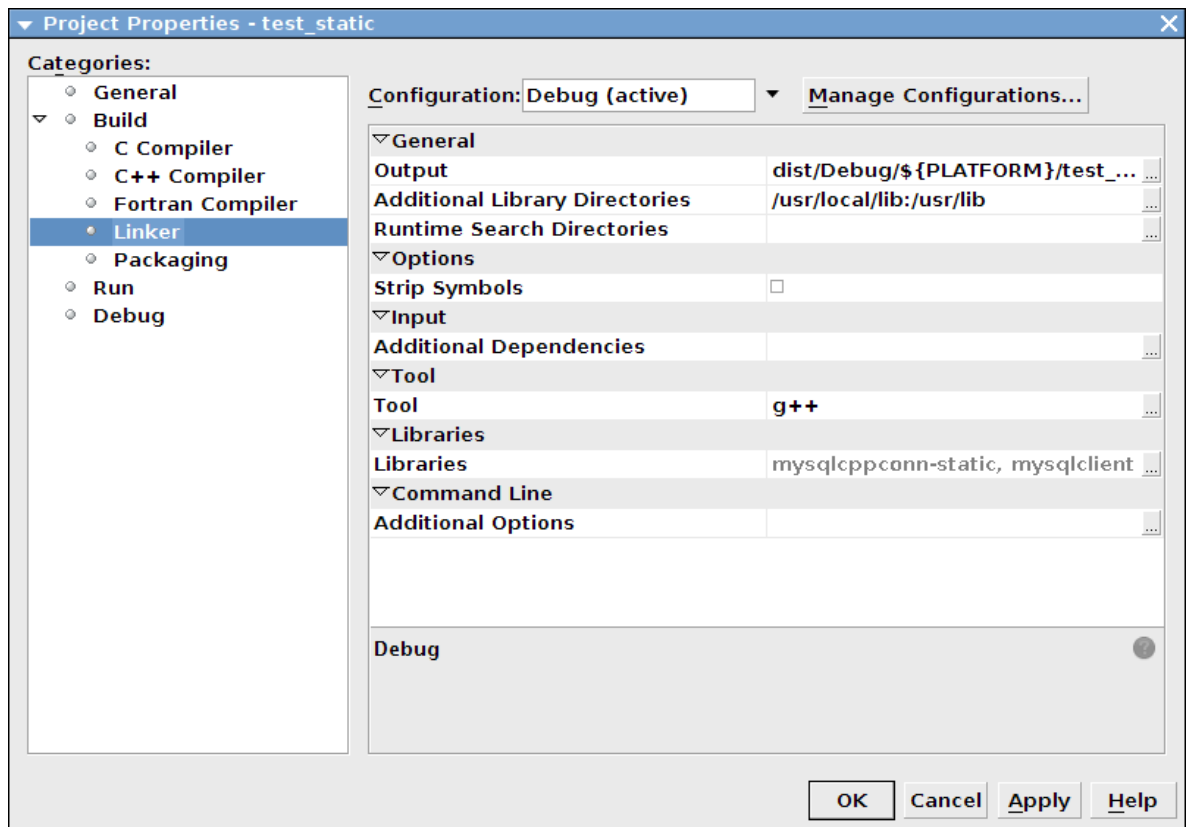
Using the Static Library

To use the static Connector/C++ library, link against two library files, `libmysqlcppconn-static.a` and `libmysqlclient.a`. The locations of the files depend on your setup, but typically the former are in `/usr/local/lib` and the latter in `/usr/lib`. The file `libmysqlclient.a` is not part of Connector/C++, but is the MySQL client library file distributed with MySQL Server. (Remember, the MySQL client library is an optional component as part of the MySQL Server installation process.) The MySQL client library is also available as part of the Connector/C distribution.

1. Set the project to link the necessary library files. Select **File, Project Properties** from the main menu.

2. In the **Categories:** tree view, navigate to **Linker**.
3. In the **General** panel, select **Additional Library Directories**. Click the ... button.
4. Select and add the `/usr/lib` and `/usr/local/lib` directories.
5. In the same panel, add the two library files required for static linking as discussed earlier. The properties panel should then look similar to the following screenshot.

Figure 8.20 Setting the Static Library Directories and File Names

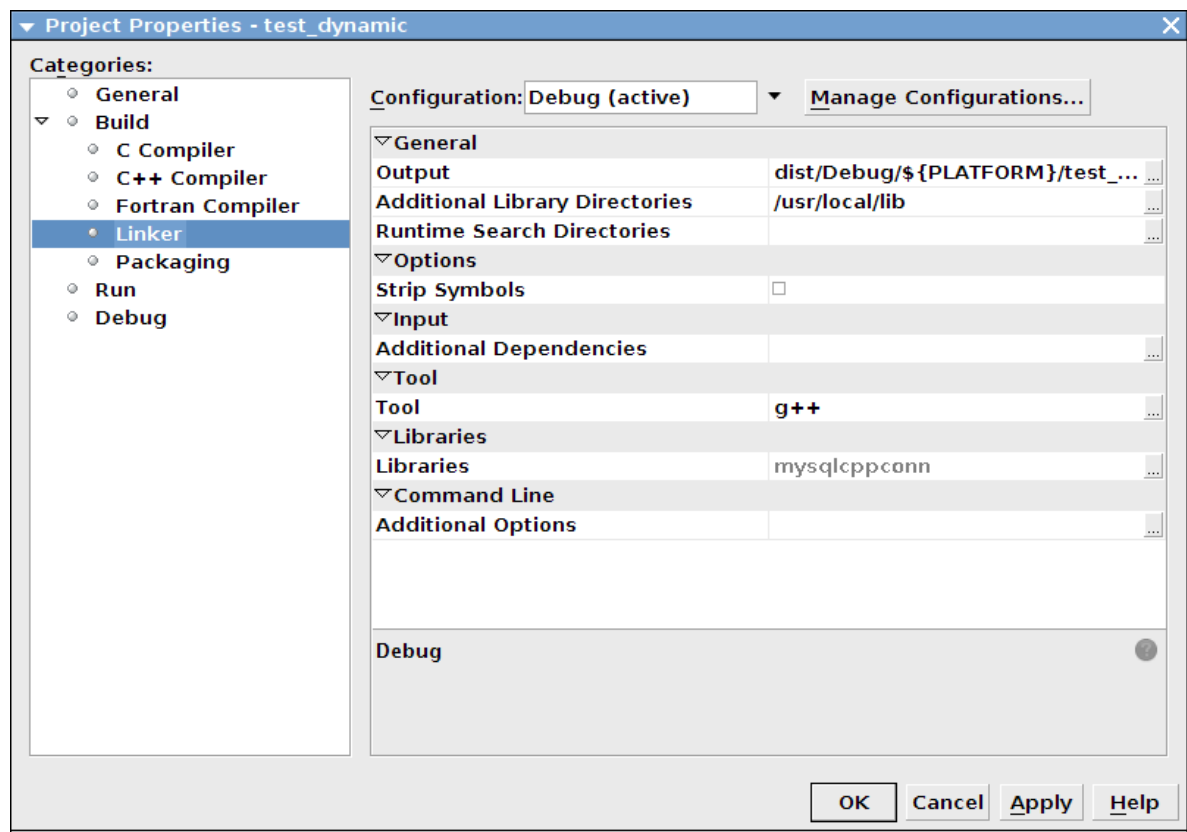


6. Click **OK** to close the **Project Properties** dialog.

Using the Dynamic Library

To use the Connector/C++ dynamic library, link your project with a single library file, `libmysqlcppconn.so`. The location of this file depends on how you configured your installation of Connector/C++, but typically is `/usr/local/lib`.

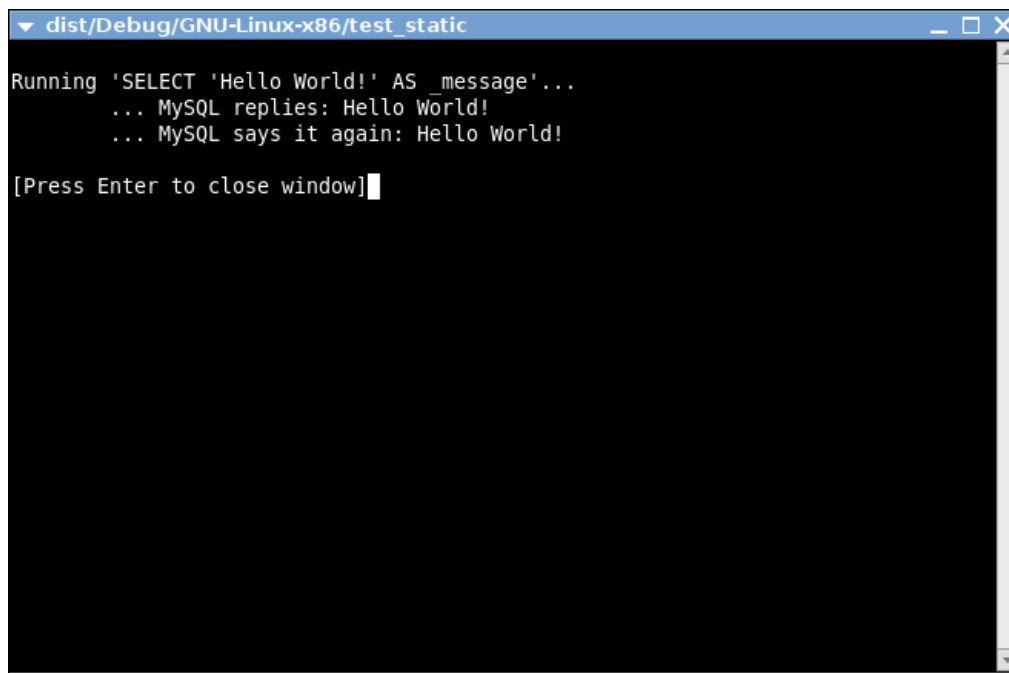
1. Set the project to link the necessary library file. Select **File, Project Properties** from the main menu.
2. In the **Categories:** tree view, navigate to **Linker**.
3. In the **General** panel, select **Additional Library Directories**. Click the ... button.
4. Select and add the `/usr/local/lib` directories.
5. In the same panel, add the library file required for static linking as discussed earlier. The properties panel should look similar to the following screenshot.

Figure 8.21 Setting the Dynamic Library Directory and File Name

6. Click OK to close the Project Properties dialog.

After configuring your project, build it by selecting **Run, Build Main Project** from the main menu. You then run the project using **Run, Run Main Project**.

On running the application, you should see a screen similar to the following (this is actually the static version of the application shown):

Figure 8.22 The Example Application Running

```
dist/Debug/GNU-Linux-x86/test_static
Running 'SELECT 'Hello World!' AS _message'...
... MySQL replies: Hello World!
... MySQL says it again: Hello World!

[Press Enter to close window]
```

Note

The preceding settings and procedures were carried out for the default [Debug](#) configuration. To create a [Release](#) configuration, select that configuration before setting the Project Properties.

8.7 Getting Started with Connector/C++: Usage Examples

Source distributions of Connector/C++ include an [examples](#) directory that contains usage examples that explain how to use the following classes:

- [Connection](#)
- [Driver](#)
- [PreparedStatement](#)
- [ResultSet](#)
- [ResultSetMetaData](#)
- [Statement](#)

The examples cover:

- Using the [Driver](#) class to connect to MySQL
- Creating tables, inserting rows, fetching rows using (simple) statements
- Creating tables, inserting rows, fetching rows using prepared statements
- Hints for working around prepared statement limitations
- Accessing result set metadata

Several examples in this document are only code snippets, not complete programs. These snippets provide a brief overview on the API. For complete programs, check the [examples](#) directory of your Connector/C++ installation. Please also read the [README](#) file in that directory. To test the example

code, edit the `examples.h` file in the `examples` directory to add your connection information, then rebuild the code by issuing a `make` command.

The example programs in the `examples` directory include:

- `connect.cpp`:
How to create a connection, insert data, and handle exceptions.
- `connection_meta_schemaobj.cpp`:
How to obtain metadata associated with a connection object, such as a list of tables or databases, the MySQL version, or the connector version.
- `debug_output.cpp`:
How to activate and deactivate the Connector/C++ debug protocol.
- `exceptions.cpp`:
A closer look at the exceptions thrown by the connector and how to fetch error information.
- `prepared_statements.cpp`:
How to execute Prepared Statements, including an example showing how to handle SQL statements that cannot be prepared by the MySQL Server.
- `resultset.cpp`:
How to use a cursor to fetch data and iterate over a result set.
- `resultset_meta.cpp`:
How to obtain metadata associated with a result set, such as the number of columns and column types.
- `resultset_types.cpp`:
Result sets returned from metadata methods. (This is more a test than an example.)
- `standalone_example.cpp`:
Simple standalone program not integrated into regular `CMake` builds.
- `statements.cpp`:
How to execute SQL statements without using Prepared Statements.
- `cpp_trace_analyzer.cpp`:
This example shows how to filter the output of the `debug trace`. Please see the inline comments for further documentation. This script is unsupported.

8.7.1 Connecting to MySQL

To establish a connection to MySQL Server, retrieve an instance of `sql::Connection` from a `sql::mysql::MySQL_Driver` object. A `sql::mysql::MySQL_Driver` object is returned by `sql::mysql::get_mysql_driver_instance()`.

```
sql::mysql::MySQL_Driver *driver;  
sql::Connection *con;  
driver = sql::mysql::get_mysql_driver_instance();  
con = driver->connect("tcp://127.0.0.1:3306", "user", "password");  
delete con;
```

Make sure that you free `con`, the `sql::Connection` object, as soon as you do not need it any more. But do not explicitly free `driver`, the connector object. Connector/C++ takes care of freeing that.

Note

`get_mysql_driver_instance()` calls `get_driver_instance()`, which is not thread-safe. Either avoid invoking these methods from within multiple threads at once, or surround the calls with a mutex to prevent simultaneous execution in multiple threads.

These methods can be used to check the connection state or reconnect:

- `sql::Connection::isValid()` checks whether the connection is alive
- `sql::Connection::reconnect()` reconnects if the connection has gone down

8.7.2 Running a Simple Query

To run simple queries, you can use the `sql::Statement::execute()`, `sql::Statement::executeQuery()`, and `sql::Statement::executeUpdate()` methods. Use the method `sql::Statement::execute()` if your query does not return a result set or if your query returns more than one result set. See the [examples](#) directory for more information.

```
sql::mysql::MySQL_Driver *driver;
sql::Connection *con;
sql::Statement *stmt;
driver = sql::mysql::get_mysql_driver_instance();
con = driver->connect("tcp://127.0.0.1:3306", "user", "password");
stmt = con->createStatement();
stmt->execute("USE " EXAMPLE_DB);
stmt->execute("DROP TABLE IF EXISTS test");
stmt->execute("CREATE TABLE test(id INT, label CHAR(1))");
stmt->execute("INSERT INTO test(id, label) VALUES (1, 'a')");
delete stmt;
delete con;
```

Note

You must free the `sql::Statement` and `sql::Connection` objects explicitly using `delete`.

8.7.3 Fetching Results

The API for fetching result sets is identical for (simple) statements and prepared statements. If your query returns one result set, use `sql::Statement::executeQuery()` or `sql::PreparedStatement::executeQuery()` to run your query. Both methods return `sql::ResultSet` objects. By default, Connector/C++ buffers all result sets on the client to support cursors.

```
// ...
sql::Connection *con;
sql::Statement *stmt;
sql::ResultSet *res;
// ...
stmt = con->createStatement();
// ...
res = stmt->executeQuery("SELECT id, label FROM test ORDER BY id ASC");
while (res->next()) {
    // You can use either numeric offsets...
    cout << "id = " << res->getInt(1); // getInt(1) returns the first column
    // ... or column names for accessing results.
    // The latter is recommended.
    cout << ", label = '" << res->getString("label") << "'" << endl;
```

```

}
delete res;
delete stmt;
delete con;

```

Note

In the preceding code snippet, column indexing starts from 1.

Note

You must free the `sql::Statement`, `sql::Connection`, and `sql::ResultSet` objects explicitly using `delete`.

Cursor usage is demonstrated in the examples contained in the download package.

8.7.4 Using Prepared Statements

If you are not familiar with Prepared Statements in MySQL, take a look at the source code comments and explanations in the file [examples/prepared_statement.cpp](#).

`sql::PreparedStatement` is created by passing an SQL query to `sql::Connection::prepareStatement()`. As `sql::PreparedStatement` is derived from `sql::Statement`, you will feel familiar with the API once you have learned how to use (simple) statements (`sql::Statement`). For example, the syntax for fetching results is identical.

```

// ...
sql::Connection *con;
sql::PreparedStatement *prep_stmt
// ...
prep_stmt = con->prepareStatement("INSERT INTO test(id, label) VALUES (?, ?)");
prep_stmt->setInt(1, 1);
prep_stmt->setString(2, "a");
prep_stmt->execute();
prep_stmt->setInt(1, 2);
prep_stmt->setString(2, "b");
prep_stmt->execute();
delete prep_stmt;
delete con;

```

Note

You must free the `sql::PreparedStatement` and `sql::Connection` objects explicitly using `delete`.

8.7.5 Complete Example 1

The following code shows a complete example of how to use Connector/C++.

```

/* Copyright 2008, 2010, Oracle and/or its affiliates. All rights reserved.
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; version 2 of the License.
There are special exceptions to the terms and conditions of the GPL
as it is applied to this software. View the full text of the
exception in file EXCEPTIONS-CONNECTOR-C++ in the directory of this
software distribution.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
*/

```

```

/* Standard C++ includes */
#include <stdlib.h>
#include <iostream>
/*
   Include directly the different
   headers from cppconn/ and mysql_driver.h + mysql_util.h
   (and mysql_connection.h). This will reduce your build time!
*/
#include "mysql_connection.h"
#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>
using namespace std;
int main(void)
{
    cout << endl;
    cout << "Running 'SELECT 'Hello World!' »
        AS _message'..." << endl;
    try {
        sql::Driver *driver;
        sql::Connection *con;
        sql::Statement *stmt;
        sql::ResultSet *res;
        /* Create a connection */
        driver = get_driver_instance();
        con = driver->connect("tcp://127.0.0.1:3306", "root", "root");
        /* Connect to the MySQL test database */
        con->setSchema("test");
        stmt = con->createStatement();
        res = stmt->executeQuery("SELECT 'Hello World!' AS _message");
        while (res->next()) {
            cout << "\t... MySQL replies: ";
            /* Access column data by alias or column name */
            cout << res->getString("_message") << endl;
            cout << "\t... MySQL says it again: ";
            /* Access column data by numeric offset, 1 is the first column */
            cout << res->getString(1) << endl;
        }
        delete res;
        delete stmt;
        delete con;
    } catch (sql::SQLException &e) {
        cout << "# ERR: SQLException in " << __FILE__;
        cout << "(" << __FUNCTION__ << ") on line " <<
            << __LINE__ << endl;
        cout << "# ERR: " << e.what();
        cout << " (MySQL error code: " << e.getErrorCode();
        cout << ", SQLState: " << e.getSQLState() << ") " << endl;
    }
    cout << endl;
    return EXIT_SUCCESS;
}

```

8.7.6 Complete Example 2

The following code shows a complete example of how to use Connector/C++.

```

/* Copyright 2008, 2010, Oracle and/or its affiliates. All rights reserved.
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; version 2 of the License.
There are special exceptions to the terms and conditions of the GPL
as it is applied to this software. View the full text of the
exception in file EXCEPTIONS-CONNECTOR-C++ in the directory of this
software distribution.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU General Public License

```

```
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
*/
/* Standard C++ includes */
#include <stdlib.h>
#include <iostream>
/*
   Include directly the different
   headers from cppconn/ and mysql_driver.h + mysql_util.h
   (and mysql_connection.h). This will reduce your build time!
*/
#include "mysql_connection.h"
#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>
#include <cppconn/prepared_statement.h>
using namespace std;
int main(void)
{
    cout << endl;
    cout << "Let's have MySQL count from 10 to 1..." << endl;
    try {
        sql::Driver *driver;
        sql::Connection *con;
        sql::Statement *stmt;
        sql::ResultSet *res;
        sql::PreparedStatement *pstmt;
        /* Create a connection */
        driver = get_driver_instance();
        con = driver->connect("tcp://127.0.0.1:3306", "root", "root");
        /* Connect to the MySQL test database */
        con->setSchema("test");
        stmt = con->createStatement();
        stmt->execute("DROP TABLE IF EXISTS test");
        stmt->execute("CREATE TABLE test(id INT)");
        delete stmt;
        /* '?' is the supported placeholder syntax */
        pstmt = con->prepareStatement("INSERT INTO test(id) VALUES (?)");
        for (int i = 1; i <= 10; i++) {
            pstmt->setInt(1, i);
            pstmt->executeUpdate();
        }
        delete pstmt;
        /* Select in ascending order */
        pstmt = con->prepareStatement("SELECT id FROM test ORDER BY id ASC");
        res = pstmt->executeQuery();
        /* Fetch in reverse = descending order! */
        res->afterLast();
        while (res->previous())
            cout << "\t... MySQL counts: " << res->getInt("id") << endl;
        delete res;
        delete pstmt;
        delete con;
    } catch (sql::SQLException &e) {
        cout << "# ERR: SQLException in " << __FILE__;
        cout << "(" << __FUNCTION__ << ") on line " <<
            << __LINE__ << endl;
        cout << "# ERR: " << e.what();
        cout << " (MySQL error code: " << e.getErrorCode();
        cout << ", SQLState: " << e.getSQLState() << "
            " )" << endl;
    }
    cout << endl;
    return EXIT_SUCCESS;
}
```

8.8 Connector/C++ Tutorials

The following tutorials illustrate various aspects of using MySQL Connector/C++. Also consult the examples in [Section 8.7, "Getting Started with Connector/C++: Usage Examples"](#).

8.8.1 Prerequisites and Background Information

This section describes the prerequisites that must be satisfied before you work through the remaining tutorial sections, and shows how to set up the framework code that serves as the basis for the tutorial applications.

These tutorials refer to tables and sample data from the [world](#) database, which you can download from the “Example Databases” section of the [MySQL Documentation](#) page.

Each tutorial application uses a framework consisting of the following code. The examples vary at the line that says `/* INSERT TUTORIAL CODE HERE! */` within the `try` block, which is replaced for each application with the application-specific code.

```
#include <stdlib.h>
#include <iostream>
#include <sstream>
#include <stdexcept>
/* uncomment for applications that use vectors */
/*#include <vector>*/
#include "mysql_connection.h"
#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>
#include <cppconn/prepared_statement.h>
#define EXAMPLE_HOST "localhost"
#define EXAMPLE_USER "worlduser"
#define EXAMPLE_PASS "worldpass"
#define EXAMPLE_DB "world"
using namespace std;
int main(int argc, const char **argv)
{
    string url(argc >= 2 ? argv[1] : EXAMPLE_HOST);
    const string user(argc >= 3 ? argv[2] : EXAMPLE_USER);
    const string pass(argc >= 4 ? argv[3] : EXAMPLE_PASS);
    const string database(argc >= 5 ? argv[4] : EXAMPLE_DB);
    cout << "Connector/C++ tutorial framework..." << endl;
    cout << endl;
    try {
        /* INSERT TUTORIAL CODE HERE! */
    } catch (sql::SQLException &e) {
        /*
         * MySQL Connector/C++ throws three different exceptions:
         * - sql::MethodNotImplementedException (derived from sql::SQLException)
         * - sql::InvalidArgumentException (derived from sql::SQLException)
         * - sql::SQLException (derived from std::runtime_error)
         */
        cout << "# ERR: SQLException in " << __FILE__ << endl;
        cout << "(" << __FUNCTION__ << ")" on line " << __LINE__ << endl;
        /* what() (derived from std::runtime_error) fetches error message */
        cout << "# ERR: " << e.what();
        cout << " (MySQL error code: " << e.getErrorCode();
        cout << ", SQLState: " << e.getSQLState() << " )" << endl;
        return EXIT_FAILURE;
    }
    cout << "Done." << endl;
    return EXIT_SUCCESS;
}
```

To try the framework code as a standalone program, use this procedure:

1. Copy and paste the framework code to a file such as `framework.cpp`. Edit the `#define` statements to reflect your connection parameters (server, user, password, database). Also, because the file contains those parameters, set its access mode to be readable only to yourself.

2. Compile the framework. For example, on OS X, the command might look like this (enter the command on one line):

```
shell> g++ -o framework
-I/usr/local/include -I/usr/local/include/cppconn
-lmysqlcppconn framework.cpp
```

Adapt the command as necessary for your system. A similar command is needed for the tutorial applications that follow.

3. To run the framework, enter the following:

```
shell> ./framework
```

You will see a simple message:

```
Connector/C++ tutorial framework...
Done.
```

You are now ready to continue to the tutorials.

8.8.2 Calling Stored Procedures with [Statement](#) Objects

A stored procedure can be called using a [Statement](#) or [PreparedStatement](#) object. This section shows how to call stored procedures using [Statement](#) objects. To see how to use [PreparedStatement](#) objects, see [Section 8.8.3, "Calling Stored Procedures with PreparedStatement Objects"](#).

The following list describes different types of stored procedures that you can construct and call, along with example stored procedures that illustrate each type:

1. A stored procedure that returns no result. For example, such a stored procedure can log non-critical information, or change database data in a straightforward way.

The following procedure adds a country to the [world](#) database, but does not return a result:

```
CREATE PROCEDURE add_country (IN country_code CHAR(3),
                             IN country_name CHAR(52),
                             IN continent_name CHAR(30))
BEGIN
    INSERT INTO Country(Code, Name, Continent)
    VALUES (country_code, country_name, continent_name);
END;
```

2. A stored procedure that returns one or more values using output parameters. For example, such a procedure can indicate success or failure, or retrieve and return data items.

The following procedures use an output parameter to return the population of a specified country or continent, or the entire world:

```
CREATE PROCEDURE get_pop (IN country_name CHAR(52),
                         OUT country_pop BIGINT)
BEGIN
    SELECT Population INTO country_pop FROM Country
    WHERE Name = country_name;
END;
```

```
CREATE PROCEDURE get_pop_continent (IN continent_name CHAR(30),
                                    OUT continent_pop BIGINT)
BEGIN
```



```
SELECT SUM(Population) INTO continent_pop FROM Country
WHERE Continent = continent_name;
END;
```

```
CREATE PROCEDURE get_pop_world (OUT world_pop BIGINT)
BEGIN
  SELECT SUM(Population) INTO world_pop FROM Country;
END;
```

3. A stored procedure that returns one or more result sets. The procedure can execute one or more queries, each of which returns an arbitrary number of rows. Your application loops through each result set to display, transform, or otherwise process each row in it.

This procedure returns several result sets:

```
CREATE PROCEDURE get_data ()
BEGIN
  SELECT Code, Name, Population, Continent FROM Country
  WHERE Continent = 'Oceania' AND Population < 10000;
  SELECT Code, Name, Population, Continent FROM Country
  WHERE Continent = 'Europe' AND Population < 10000;
  SELECT Code, Name, Population, Continent FROM Country
  WHERE Continent = 'North America' AND Population < 10000;
END;
```

Enter and test the stored procedures manually to ensure that they will be available to your C++ applications. (Select **world** as the default database before you create them.) You are now ready to start writing applications using Connector/C++ that call stored procedures.

8.8.2.1 Using a **Statement** for a Stored Procedure That Returns No Result

This example shows how to call a stored procedure that returns no result set.

1. Make a copy of the tutorial framework code:

```
shell> cp framework.cpp sp_scenario1.cpp
```

2. Add the following code to the **try** block of the tutorial framework:

```
sql::Driver* driver = get_driver_instance();
std::auto_ptr<sql::Connection> con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr<sql::Statement> stmt(con->createStatement());
// We need not check the return value explicitly. If it indicates
// an error, Connector/C++ generates an exception.
stmt->execute("CALL add_country('ATL', 'Atlantis', 'North America')");
```

3. Compile the program as described in [Section 8.8.1, “Prerequisites and Background Information”](#).
4. Run the program:

```
shell> ./sp_scenario1
```

5. Using the **mysql** command-line client or other suitable program, check the **world** database to determine that it has been updated correctly. You can use this query:

```
mysql> SELECT Code, Name, Continent FROM Country WHERE Code='ATL';
+-----+-----+-----+
```

Code	Name	Continent
ATL	Atlantis	North America

The code in this application simply invokes the `execute` method, passing to it a statement that calls the stored procedure. The procedure itself returns no value, although it is important to note there is always a return value from the `CALL` statement; this is the `execute` status. Connector/C++ handles this status for you, so you need not handle it explicitly. If the `execute` call fails for some reason, it raises an exception that the `catch` block handles.

8.8.2.2 Using a **Statement** for a Stored Procedure That Returns an Output Parameter

This example shows how to handle a stored procedure that returns an output parameter.

1. Make a copy of the tutorial framework code:

```
shell> cp framework.cpp sp_scenario2.cpp
```

2. Add the following code to the `try` block of the tutorial framework:

```
sql::Driver* driver = get_driver_instance();
std::auto_ptr<sql::Connection> con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr<sql::Statement> stmt(con->createStatement());
stmt->execute("CALL get_pop('Uganda', @pop)");
std::auto_ptr<sql::ResultSet> res(stmt->executeQuery("SELECT @pop AS _reply"));
while (res->next())
    cout << "Population of Uganda: " << res->getString("_reply") << endl;
stmt->execute("CALL get_pop_continent('Asia', @pop)");
res.reset(stmt->executeQuery("SELECT @pop AS _reply"));
while (res->next())
    cout << "Population of Asia: " << res->getString("_reply") << endl;
stmt->execute("CALL get_pop_world(@pop)");
res.reset(stmt->executeQuery("SELECT @pop AS _reply"));
while (res->next())
    cout << "Population of World: " << res->getString("_reply") << endl;
```

3. Compile the program as described in [Section 8.8.1, "Prerequisites and Background Information"](#).
4. Run the program:

```
shell> ./sp_scenario2
Connector/C++ tutorial framework...
Population of Uganda: 21778000
Population of Asia: 3705025700
Population of World: 6078749450
Done.
```

In this scenario, each stored procedure sets the value of an output parameter. This is not returned directly to the `execute` method, but needs to be obtained using a subsequent query. If you were executing the SQL statements directly, you might use statements similar to these:

```
CALL get_pop('Uganda', @pop);
SELECT @pop;
CALL get_pop_continent('Asia', @pop);
SELECT @pop;
CALL get_pop_world(@pop);
SELECT @pop;
```

In the C++ code, a similar sequence is carried out for each procedure call:

1. Execute the [CALL](#) statement.
2. Obtain the output parameter by executing an additional query. The query produces a [ResultSet](#) object.
3. Retrieve the data using a [while](#) loop. The simplest way to do this is to use a [getString](#) method on the [ResultSet](#), passing the name of the variable to access. In this example [_reply](#) is used as a placeholder for the variable and therefore is used as the key to access the correct element of the result dictionary.

Although the query used to obtain the output parameter returns only a single row, it is important to use the [while](#) loop to catch more than one row, to avoid the possibility of the connection becoming unstable.

8.8.2.3 Using a [Statement](#) for a Stored Procedure That Returns a Result Set

This example shows how to handle result sets produced by a stored procedure.

Note

This scenario requires MySQL 5.5.3 or higher. The client/server protocol does not support fetching multiple result sets from stored procedures prior to 5.5.3.

1. Make a copy of the tutorial framework code:

```
shell> cp framework.cpp sp_scenario3.cpp
```

2. Add the following code to the [try](#) block of the tutorial framework:

```
sql::Driver* driver = get_driver_instance();
std::auto_ptr<sql::Connection> con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr<sql::Statement> stmt(con->createStatement());
stmt->execute("CALL get_data()");
std::auto_ptr< sql::ResultSet > res;
do {
    res.reset(stmt->getResultSet());
    while (res->next()) {
        cout << "Name: " << res->getString("Name")
            << " Population: " << res->getInt("Population")
            << endl;
    }
} while (stmt->getMoreResults());
```

3. Compile the program as described in [Section 8.8.1, "Prerequisites and Background Information"](#).
4. Run the program:

```
shell> ./sp_scenario3
Connector/C++ tutorial framework...
Name: Cocos (Keeling) Islands Population: 600
Name: Christmas Island Population: 2500
Name: Norfolk Island Population: 2000
Name: Niue Population: 2000
Name: Pitcairn Population: 50
Name: Tokelau Population: 2000
Name: United States Minor Outlying Islands Population: 0
Name: Svalbard and Jan Mayen Population: 3200
Name: Holy See (Vatican City State) Population: 1000
Name: Anguilla Population: 8000
Name: Atlantis Population: 0
Name: Saint Pierre and Miquelon Population: 7000
```

Done.

The code is similar to the examples shown previously. The code of particular interest here is:

```
do {
    res.reset(stmt->getResultSet());
    while (res->next()) {
        cout << "Name: " << res->getString("Name")
            << " Population: " << res->getInt("Population")
            << endl;
    }
} while (stmt->getMoreResults());
```

The `CALL` is executed as before, but this time the results are returned into multiple `ResultSet` objects because the stored procedure executes multiple `SELECT` statements. In this example, the output shows that three result sets are processed, because there are three `SELECT` statements in the stored procedure. Each result set returns more than one row.

The results are processed using this code pattern:

```
do {
    Get Result Set
    while (Get Result) {
        Process Result
    }
} while (Get More Result Sets);
```

Note

Use this pattern even if the stored procedure executes only a single `SELECT` and produces only one result set. This is a requirement of the underlying protocol.

8.8.3 Calling Stored Procedures with `PreparedStatement` Objects

This section shows how to call stored procedures using prepared statements. It is recommended that, before working through it, you first work through the previous tutorial [Section 8.8.2, "Calling Stored Procedures with Statement Objects"](#). That section shows the stored procedures required by the applications in this section.

8.8.3.1 Using a `PreparedStatement` for a Stored Procedure That Returns No Result

This example shows how to call a stored procedure that returns no result set.

1. Make a copy of the tutorial framework code:

```
shell> cp framework.cpp ps_scenario1.cpp
```

2. Add the following code to the `try` block of the tutorial framework:

```
vector<string> code_vector;
code_vector.push_back("SLD");
code_vector.push_back("DSN");
code_vector.push_back("ATL");
vector<string> name_vector;
name_vector.push_back("Sealand");
name_vector.push_back("Disneyland");
```

```
name_vector.push_back("Atlantis");
vector<string> cont_vector;
cont_vector.push_back("Europe");
cont_vector.push_back("North America");
cont_vector.push_back("Oceania");
sql::Driver * driver = get_driver_instance();
std::auto_ptr< sql::Connection > con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr< sql::PreparedStatement > pstmt;
pstmt.reset(con->prepareStatement("CALL add_country(?,?,?)"));
for (int i=0; i<3; i++)
{
    pstmt->setString(1,code_vector[i]);
    pstmt->setString(2,name_vector[i]);
    pstmt->setString(3,cont_vector[i]);
    pstmt->execute();
}
```

Also, uncomment `#include <vector>` near the top of the code, because vectors are used to store sample data.

3. Compile the program as described in [Section 8.8.1, "Prerequisites and Background Information"](#).
4. Run the program:

```
shell> ./ps_scenario1
```

5. You can check whether the database has been updated correctly by using this query:

```
mysql> SELECT Code, Name, Continent FROM Country
-> WHERE Code IN('DSN','ATL','SLD');
```

Code	Name	Continent
ATL	Atlantis	Oceania
DSN	Disneyland	North America
SLD	Sealand	Europe

The code is relatively simple, as no processing is required to handle result sets. The procedure call, `CALL add_country(?,?,?)`, is made using placeholders for input parameters denoted by `'?'`. These placeholders are replaced by the appropriate data values using the `PreparedStatement` object's `setString` method. The `for` loop is set up to iterate 3 times, as there are three data sets in this example. The same `PreparedStatement` is executed three times, each time with different input parameters.

8.8.3.2 Using a `PreparedStatement` for a Stored Procedure That Returns an Output Parameter

This example shows how to handle a stored procedure that returns an output parameter.

1. Make a copy of the tutorial framework code:

```
shell> cp framework.cpp ps_scenario2.cpp
```

2. Add the following code to the `try` block of the tutorial framework:

```
vector<string> cont_vector;
cont_vector.push_back("Europe");
cont_vector.push_back("North America");
cont_vector.push_back("Oceania");
sql::Driver * driver = get_driver_instance();
```

```
std::auto_ptr< sql::Connection > con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr< sql::Statement > stmt(con->createStatement());
std::auto_ptr< sql::PreparedStatement > pstmt;
std::auto_ptr< sql::ResultSet > res;
pstmt.reset(con->prepareStatement("CALL get_pop_continent(?,@pop)"));
for (int i=0; i<3; i++)
{
    pstmt->setString(1,cont_vector[i]);
    pstmt->execute();
    res.reset(stmt->executeQuery("SELECT @pop AS _population"));
    while (res->next())
        cout << "Population of "
            << cont_vector[i]
            << " is "
            << res->getString("_population") << endl;
}
```

Also, uncomment `#include <vector>` near the top of the code, because vectors are used to store sample data.

3. Compile the program as described in [Section 8.8.1, “Prerequisites and Background Information”](#).
4. Run the program:

```
shell> ./ps_scenario2
Connector/C++ tutorial framework...
Population of Europe is 730074600
Population of North America is 482993000
Population of Oceania is 30401150
Done.
```

In this scenario a `PreparedStatement` object is created that calls the `get_pop_continent` stored procedure. This procedure takes an input parameter, and also returns an output parameter. The approach used is to create another statement that can be used to fetch the output parameter using a `SELECT` query. Note that when the `PreparedStatement` is created, the input parameter to the stored procedure is denoted by `'?'`. Prior to execution of the prepared statement, it is necessary to replace this placeholder by an actual value. This is done using the `setString` method:

```
pstmt->setString(1,cont_vector[i]);
```

Although the query used to obtain the output parameter returns only a single row, it is important to use the `while` loop to catch more than one row, to avoid the possibility of the connection becoming unstable.

8.8.3.3 Using a `PreparedStatement` for a Stored Procedure That Returns a Result Set

This example shows how to handle result sets produced by a stored procedure.

Note

This scenario requires MySQL 5.5.3 or higher. The client/server protocol does not support fetching multiple result sets from stored procedures prior to 5.5.3.

1. Make a copy of the tutorial framework code:

```
shell> cp framework.cpp ps_scenario3.cpp
```

2. Add the following code to the `try` block of the tutorial framework:

```

sql::Driver * driver = get_driver_instance();
std::auto_ptr< sql::Connection > con(driver->connect(url, user, pass));
con->setSchema(database);
std::auto_ptr< sql::PreparedStatement > pstmt;
std::auto_ptr< sql::ResultSet > res;
pstmt.reset(con->prepareStatement("CALL get_data()"));
res.reset(pstmt->executeQuery());
for(;;)
{
    while (res->next()) {
        cout << "Name: " << res->getString("Name")
              << " Population: " << res->getInt("Population")
              << endl;
    }
    if (pstmt->getMoreResults())
    {
        res.reset(pstmt->getResultSet());
        continue;
    }
    break;
}

```

3. Compile the program as described in [Section 8.8.1, "Prerequisites and Background Information"](#).
4. Run the program:

```
shell> ./ps_scenario3
```

5. Make a note of the output generated.

The code executes the stored procedure using a [PreparedStatement](#) object. The standard [do/while](#) construct is used to ensure that all result sets are fetched. The returned values are fetched from the result sets using the [getInt](#) and [getString](#) methods.

8.9 Connector/C++ Debug Tracing

Although a debugger can be used to debug a specific instance of your application, you may find it beneficial to enable the debug traces of the connector. Some problems happen randomly which makes them difficult to debug using a debugger. In such cases, debug traces and protocol files are more useful because they allow you to trace the activities of all instances of your program.

DTrace is a very powerful technology to trace any application without having to develop an extra trace module for your application. DTrace is currently available on Solaris, OS X, and FreeBSD.

Connector/C++ can write two trace files:

1. A trace file generated by the MySQL client library
2. A trace file generated internally by Connector/C++

The first trace file can be generated by the underlying MySQL client library ([libmysqlclient](#)). To enable this trace, the connector calls the [mysql_debug\(\)](#) C API function internally. Because only debug versions of the MySQL client library are capable of writing a trace file, compile Connector/C++ against a debug version of the library to use this trace. The trace shows the internal function calls and the addresses of internal objects as shown here:

```

>mysql_stmt_init
| >_mymalloc
| | enter: Size: 816
| | exit: ptr: 0x68e7b8
| <_mymalloc | >init_alloc_root
| | enter: root: 0x68e7b8
| | >_mymalloc

```

```
| | | enter: Size: 2064
| | | exit: ptr: 0x68eb28
[...]
```

The second trace is the Connector/C++ internal trace. It is available with debug and nondebug builds of the connector as long as you have enabled the tracing module at compile time. By default, tracing functionality is not available and calls to trace functions are removed by the preprocessor. To enable the tracing module, use `cmake -DMYSQLCPPCONN_TRACE_ENABLE=1`.

Compiling the connector with tracing functionality enabled causes two additional tracing function calls per each connector function call. For example:

```
| INF: Tracing enabled
| <MySQL_Connection::setClientOption
| >MySQL_Prepared_Statement::setInt
| INF: this=0x69a2e0
| >MySQL_Prepared_Statement::checkClosed
| <MySQL_Prepared_Statement::checkClosed
| <MySQL_Prepared_Statement::setInt
[...]
```

Run your own benchmark to find out how much this will impact the performance of your application.

A simple test using a loop running 30,000 `INSERT` SQL statements showed no significant real-time impact. The two variants of this application using a trace enabled and trace disabled version of the connector performed equally well. The runtime measured in real time was not significantly impacted as long as writing a debug trace was not enabled. However, there will be a difference in the time spent in the application. When writing a debug trace, the I/O subsystem may become a bottleneck.

In summary, use connector builds with tracing enabled carefully. Trace-enabled versions may cause higher CPU usage even if the overall runtime of your application is not impacted significantly.

The example from `examples/debug_output.cpp` demonstrates how to activate the debug traces in your program. Currently they can only be activated through API calls. The traces are controlled on a per-connection basis. You can use the `setClientOption()` method of a connection object to activate and deactivate trace generation. The MySQL client library trace always writes its trace to a file, whereas the connector writes protocol messages to the standard output.

```
sql::Driver *driver;
int on_off = 1;
/* Using the Driver to create a connection */
driver = get_driver_instance();
std::auto_ptr< sql::Connection > con(driver->connect(host, user, pass));
/*
Activate debug trace of the MySQL client library (C API)
Only available with a debug build of the MySQL client library!
*/
con->setClientOption("libmysql_debug", "d:t:0,client.trace");
/*
Connector/C++ tracing is available if you have compiled the
driver using cmake -DMYSQLCPPCONN_TRACE_ENABLE=BOOL=1
*/
con->setClientOption("clientTrace", &on_off);
```

8.10 Connector/C++ Usage Notes

Connector/C++ is compatible with the JDBC 4.0 API. See the [JDBC overview](#) for information on JDBC 4.0. Please also check the `examples` directory of the download package.

- The Connector/C++ `sql::DataType` class defines the following JDBC standard data types: `UNKNOWN`, `BIT`, `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INTEGER`, `BIGINT`, `REAL`, `DOUBLE`, `DECIMAL`,

NUMERIC, CHAR, BINARY, VARCHAR, VARBINARY, LONGVARCHAR, LONGVARBINARY, TIMESTAMP, DATE, TIME, GEOMETRY, ENUM, SET, SQLNULL.

Connector/C++ does not support the following JDBC standard data types: ARRAY, BLOB, CLOB, DISTINCT, FLOAT, OTHER, REF, STRUCT.

- `DatabaseMetaData::supportsBatchUpdates()` returns `true` because MySQL supports batch updates in general. However, the Connector/C++ API provides no API calls for batch updates.
- Two non-JDBC methods let you fetch and set unsigned integers: `getUInt64()` and `getUInt()`. These are available for `ResultSet` and `PreparedStatement`:
 - `ResultSet::getUInt64()`
 - `ResultSet::getUInt()`
 - `PreparedStatement::setUInt64()`
 - `PreparedStatement::setUInt()`

The corresponding `getLong()` and `setLong()` methods have been removed.

- The `DatabaseMetaData::getColumnNames()` method has 23 columns in its result set, rather than the 22 columns defined by JDBC. The first 22 columns are as described in the JDBC documentation, but column 23 is new:

23. `IS_AUTOINCREMENT`: A string which is "YES" if the column is an auto-increment column, "NO" otherwise.

- Connector/C++ may return different metadata for the same column, depending on the method you call.

Suppose that you have a column that accepts a character set and a collation in its specification and you specify a binary collation, such as:

```
VARCHAR(20) CHARACTER SET utf8 COLLATE utf8_bin
```

The server sets the `BINARY` flag in the result set metadata of this column. The `ResultSetMetaData::getColumnTypeName()` method uses the metadata and reports, due to the `BINARY` flag, that the column type name is `BINARY`, as illustrated here:

```
mysql> CREATE TABLE varbin (a VARCHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
Query OK, 0 rows affected (0.00 sec)
mysql> select * from varbin;
Field 1: `a`
Catalog: `def`
Database: `test`
Table: `varbin`
Org_table: `varbin`
Type: VAR_STRING
Collation: latin1_swedish_ci (8)
Length: 20
Max_length: 0
Decimals: 0
Flags: BINARY
0 rows in set (0.00 sec)
mysql> SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='varbin'\G
***** 1. row *****
TABLE_CATALOG: NULL
TABLE_SCHEMA: test
TABLE_NAME: varbin
COLUMN_NAME: a
ORDINAL_POSITION: 1
COLUMN_DEFAULT: NULL
```

```

        IS_NULLABLE: YES
        DATA_TYPE: varchar
CHARACTER_MAXIMUM_LENGTH: 20
CHARACTER_OCTET_LENGTH: 60
        NUMERIC_PRECISION: NULL
        NUMERIC_SCALE: NULL
CHARACTER_SET_NAME: utf8
COLLATION_NAME: utf8_bin
        COLUMN_TYPE: varchar(20)
        COLUMN_KEY:
        EXTRA:
        PRIVILEGES: select,insert,update,references
        COLUMN_COMMENT:
1 row in set (0.01 sec)

```

However, `INFORMATION_SCHEMA` gives no hint in its `COLUMNS` table that metadata will contain the `BINARY` flag. `DatabaseMetaData::getColumnns()` uses `INFORMATION_SCHEMA` and will report the type name `VARCHAR` for the same column. It also returns a different type code.

- When inserting or updating `BLOB` or `TEXT` columns, Connector/C++ developers are advised not to use `setString()`. Instead, use the dedicated `setBlob()` API function.

The use of `setString()` can cause a `Packet too large` error message. The error occurs if the length of the string passed to the connector using `setString()` exceeds `max_allowed_packet` (minus a few bytes reserved in the protocol for control purposes). This situation is not handled in Connector/C++, because it could lead to security issues, such as extremely large memory allocation requests due to malevolently long strings.

If `setBlob()` is used, this problem does not arise because `setBlob()` takes a streaming approach based on `std::istream`. When sending the data from the stream to MySQL Server, Connector/C++ splits the stream into chunks appropriate for MySQL Server using the current `max_allowed_packet` setting.

Caution

When using `setString()`, it is not possible to set `max_allowed_packet` to a value large enough for the string prior to passing it to Connector/C++. That configuration option cannot be changed within a session.

This difference from the JDBC specification ensures that Connector/C++ is not vulnerable to memory flooding attacks.

- In general, Connector/C++ works with MySQL 5.0, but it is not completely supported. Some methods may not be available when connecting to MySQL 5.0. This is because the Information Schema is used to obtain the requested information. There are no plans to improve the support for 5.0 because the current GA version of MySQL Server is 5.6. Connector/C++ is primarily targeted at the MySQL Server GA version that is available on its release.

The following methods throw a `sql::MethodNotImplemented` exception when you connect to a MySQL server earlier than 5.1:

- `DatabaseMetaData::getCrossReference()`
- `DatabaseMetaData::getExportedKeys()`
- Connector/C++ includes a `Connection::getClientOption()` method that is not included in the JDBC API specification. The prototype is:

```
void getClientOption(const std::string & optionName, void * optionValue)
```

The method can be used to check the value of connection properties set when establishing a database connection. The values are returned through the `optionValue` argument passed to the method with the type `void *`.

Currently, `getClientOption()` supports fetching the `optionValue` of the following options:

- `metadataUseInfoSchema`
- `defaultStatementResultType`
- `defaultPreparedStatementResultType`

The `metadataUseInfoSchema` connection option controls whether to use the `Information_Schemata` for returning the metadata of `SHOW` statements:

- For `metadataUseInfoSchema`, interpret the `optionValue` argument as a boolean upon return.
- For `defaultStatementResultType` and `defaultPreparedStatementResultType`, interpret the `optionValue` argument as an integer upon return.

The connection property can be set either when establishing the connection through the connection property map, or using `void Connection::setClientOption(const std::string &optionName, const void *optionValue)` where `optionName` is assigned the value `metadataUseInfoSchema`.

Some examples:

```
bool isInfoSchemaUsed;
conn->getClientOption("metadataUseInfoSchema", (void *) &isInfoSchemaUsed);
int defaultStmtResType;
int defaultPStmtResType;
conn->getClientOption("defaultStatementResultType", (void *) &defaultStmtResType);
conn->getClientOption("defaultPreparedStatementResultType", (void *) &defaultPStmtResType);
```

- To get and set MySQL session variables, Connector/C++ supports the following `MySQL_Connection` methods, which are not found in the JDBC API standard:

```
std::string MySQL_Connection::getSessionVariable(const std::string &varname)
```

```
void MySQL_Connection::setSessionVariable(const std::string &varname, const std::string &value)
```

`getSessionVariable()` is equivalent to executing the following and fetching the first return value:

```
SHOW SESSION VARIABLES LIKE 'var_name'
```

You can use the `%` and `_` SQL pattern characters in `var_name`.

`setSessionVariable()` is equivalent to executing:

```
SET SESSION var_name = value
```

- Fetching the value of a column can sometimes return different values depending on whether the call is made from a Statement or Prepared Statement. This is because the protocol used to communicate with the server differs depending on whether a Statement or Prepared Statement is used.

To illustrate this, consider the case where a column has been defined as type `BIGINT`. The most negative `BIGINT` value is then inserted into the column. If a Statement and Prepared Statement are created that perform a `getUInt64()` call, then the results will be different in each case. The Statement returns the maximum positive value for `BIGINT`. The Prepared Statement returns 0.

The difference results from the fact that Statements use a text protocol, and Prepared Statements use a binary protocol. With the binary protocol in this case, a binary value is returned from the server that can be interpreted as an `int64`. In the preceding scenario, a very large negative value

is fetched with `getUInt64()`, which fetches unsigned integers. Because the large negative value cannot be sensibly converted to an unsigned value, 0 is returned.

In the case of the Statement, which uses the text protocol, values are returned from the server as strings, and then converted as required. When a string value is returned from the server in the preceding scenario, the large negative value must be converted by the runtime library function `strtoul()`, which `getUInt64()` calls. The behavior of `strtoul()` is dependent upon the specific runtime and host operating system, so the results can be platform dependent. In the case, given a large positive value was actually returned.

Although it is very rare, there are some cases where Statements and Prepared Statements can return different values unexpectedly, but this usually only happens in extreme cases such as the one mentioned.

- The JDBC documentation [lists many fields](#) for the `DatabaseMetaData` class. JDBC also appears to [define certain values](#) for those fields. However, Connector/C++ does not define certain values for those fields. Internally enumerations are used and the compiler determines the values to assign to a field.

To compare a value with the field, use code such as the following, rather than making assumptions about specific values for the attribute:

```
// dbmeta is an instance of DatabaseMetaData
if (myvalue == dbmeta->attributeNoNulls) {
    ...
}
```

Usually `myvalue` will be a column from a result set holding metadata information. Connector/C++ does not guarantee that `attributeNoNulls` is 0. It can be any value.

- When programming stored procedures, JDBC has available an extra class, an extra abstraction layer for callable statements, the `CallableStatement` class. As this class is not present in Connector/C++, use the methods from the `Statement` and `PreparedStatement` classes to execute a stored procedure using `CALL`.

8.11 Connector/C++ Known Bugs and Issues

Please report bugs through the MySQL Bug System. See [How to Report Bugs or Problems](#).

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

- When linking against a static library for 1.0.3 on Windows, define `CPPDBC_PUBLIC_FUNC` either in the compiler options (preferable) or with `/D "CPPCONN_PUBLIC_FUNC="`. You can also explicitly define it in your code by placing `#define CPPCONN_PUBLIC_FUNC` before the header inclusions.
- Generally speaking, C++ library binaries are less portable than C library binaries. Issues can be caused by name mangling, different Standard Template Library (STL) versions, and using different compilers and linkers for linking against the libraries than were used for building the library itself.

Even a small change in the compiler version can cause problems. If you obtain error messages that you suspect are related to binary incompatibilities, build Connector/C++ from source, using the same compiler and linker that you use to build and link your application.

Due to the variations between Linux distributions, compiler and linker versions and STL versions, it is not possible to provide binaries for each and every possible configuration. However, the Connector/C++ binary distributions contain a [README](#) file that describes the environment and settings used to build the binary versions of the libraries.

- To avoid potential crashes, the build configuration of Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of Connector/C++ with a debug build of the client application.

8.12 Connector/C++ Support

For general discussion of Connector/C++, please use the [C/C++ community forum](#) or join the [Connector/C++ mailing list](#).

Bugs can be reported at the [MySQL bug Web site](#).

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

For Licensing questions, and to purchase MySQL Products and Services, please see <http://www.mysql.com/buy-mysql/>.

Appendix A Connector Release Notes

For notes detailing changes made in each Connector, see the following documents:

- [Connector/J](#)
- [Connector/ODBC](#)
- [Connector/Net](#) (includes MySQL Visual Studio Plugin release notes)
- [Connector/Python](#)
- [Connector/C](#)
- [Connector/C++](#)

