Connectors and APIs

Connectors and APIs Manual

Abstract

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

Document generated on: 2009-06-03 (revision: 15169)

If you want 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, CHM, and PDF formats, see MySQL Documentation Library.

Connectors

This chapter describes MySQL Connectors, drivers that provide connectivity to the MySQL server for client programs. There are currently five MySQL Connectors:

- Connector/ODBC provides driver support for connecting to a MySQL server using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix and Mac OS X platforms.
- Connector/NET enables developers to create .NET applications that use data stored in a MySQL database. Connector/NET implements a fully-functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that want to use Connector/NET can be written in any of the supported .NET languages.
- 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 within Visual Studio to create and edit objects within a MySQL database.
- Connector/J provides driver support for connecting to MySQL from a Java application using the standard Java Database Connectivity (JDBC) API.
- Connector/MXJ is a tool that enables easy deployment and management of MySQL server and database through your Java application.
- Connector/PHP is a Windows-only connector for PHP that provides the `mysql` and `mysqli` extensions for use with MySQL 5.0.18 and later.

For information on connecting to a MySQL server using other languages and interfaces than those detailed above, including Perl, Python and PHP for other platforms and environments, please refer to the APIs and Libraries chapter.

Chapter 1. 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 3.51 and Connector/ODBC 5.1. Both releases provide an ODBC compliant interface to MySQL Server.

MySQL Connector/ODBC provides both driver-manager based and native interfaces to the MySQL database, which full support for MySQL functionality, including stored procedures, transactions and, with Connector/ODBC 5.1, 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 part of this reference assumes a good working knowledge of C, general DBMS knowledge, and finally, but not least, 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 Mac OS X you can use the native MySQL network or named pipe to communicate with your MySQL database. You may need Connector/ODBC for Unix or Mac 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.

Key topics:

- For help installing Connector/ODBC see Section 1.3, “Connector/ODBC Installation”.
- For information on the configuration options, see Section 1.4.2, “Connector/ODBC Connection Parameters”.
- For more information on connecting to a MySQL database from a Windows host using Connector/ODBC see Section 1.5.2, “Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC”.
- If you want to use Microsoft Access as an interface to a MySQL database using Connector/ODBC see Section 1.5.4, “Using Connector/ODBC with Microsoft Access”.
- General tips on using Connector/ODBC, including obtaining the last auto-increment ID see Section 1.7.1, “Connector/ODBC General Functionality”.
- For tips and common questions on using Connector/ODBC with specific application see Section 1.7.2, “Connector/ODBC Application Specific Tips”.
- For a general list of Frequently Asked Questions see Section 1.7.3, “Connector/ODBC Errors and Resolutions (FAQ)”.
- Additional support when using Connector/ODBC is available, see Section 1.8, “Connector/ODBC Support”.

> **MySQL Enterprise**
> MySQL Enterprise subscribers will find more information about MySQL and ODBC in the Knowledge Base articles about ODBC. Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information, see http://www.mysql.com/products/enterprise/advisors.html.

1.1. Connector/ODBC Versions

There are currently two version of Connector/ODBC available:

- Connector/ODBC 5.1, currently in GA status, is a partial rewrite of the of the 3.51 code base and is designed to work with all versions of MySQL from 4.1. Connector/ODBC 5.1 will be a complete implementation of the ODBC Core interface,plus more Level 1 and Level 2 functionality of the ODBC specification than that currently supported by Connector/ODBC 3.51. See Section 1.2.1, “Connector/ODBC Roadmap”.

  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` datatype, and support for Unicode login, password and DSN configurations. For more information,. see Microsoft Knowledgebase Article #716246.
  - Support for the `SQL_NUMERIC_STRUCT` datatype, 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 is the current release of the 32-bit ODBC driver, also known as the MySQL ODBC 3.51 driver. Connector/ODBC 3.51 has support for ODBC 3.5x specification level 1 (complete core API + level 2 features) in order to continue to provide all functionality of ODBC for accessing MySQL.

The manual for versions of Connector/ODBC older than 3.51 can be located in the corresponding binary or source distribution. Please note that versions of Connector/ODBC earlier than the 3.51 revision were not fully compliant with the ODBC specification.

Note

Development on Connector/ODBC 5.0 was stopped due to development issues. Connector/ODBC 5.1 is now the current development release.

Note

From this section onward, the primary focus of this guide is the Connector/ODBC 3.51 and Connector/ODBC 5.1 drivers.

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 format is different.

1.2. Connector/ODBC Introduction

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 allows 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.

1.2.1. Connector/ODBC Roadmap

Connector/ODBC 5.1 is currently in development and will be a complete implementation of the ODBC Core interface,plus more Level 1 and Level 2 functionality of the ODBC specification than that currently supported by Connector/ODBC 3.51.

The following functionality was added or changed as part of 5.1:

- Add support for SQL_NUMERIC_STRUCT: MSDN Article 714556.
- Replace installer library with new implementation (from v5 tree).
- Implement native Windows setup library.
- Implement SQLCancel() (Bug#15601): MSDN Article 714112.

The following functionality will be added in a version after 5.1:

- Implement native Mac OS X setup library.
- Replace OPTIONS flags with individual DSN settings (but support OPTIONS for backwards-compatibility).
- Fix support for SQLBIGINT (Bug#28887): MSDN Article 714121.
- Make diagnostics support standards-compliant: MSDN Article 711021.

- Add support for SQL_ATTR_METADATA_ID: MSDN Article 716447.
- Implement SQLBrowseConnect(): MSDN Article 714565, MSDN Article 712446.
- Implement arrays of parameters: MSDN Article 711818.

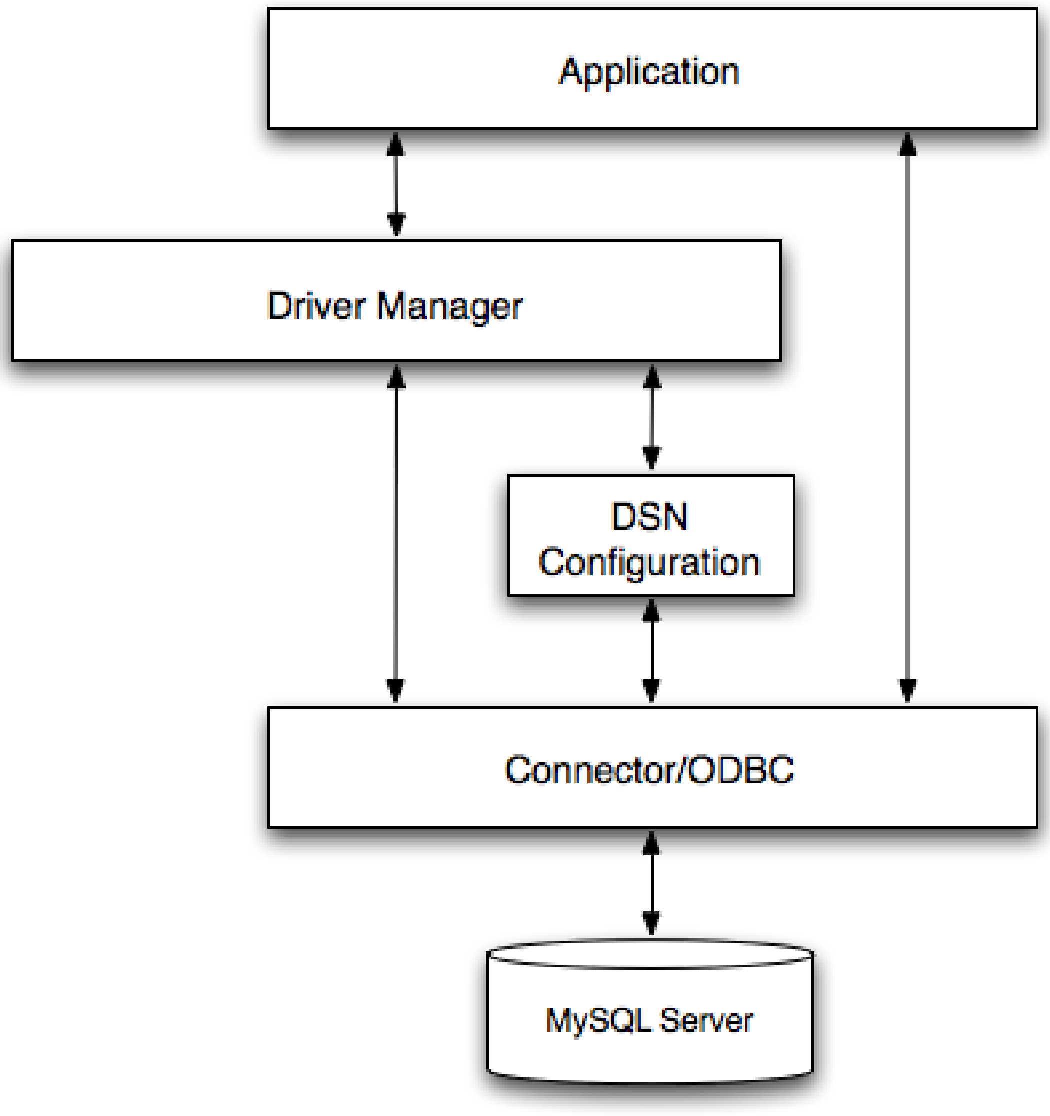
1.2.2. General Information About ODBC and Connector/ODBC

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 1.6.1, "Connector/ODBC API Reference". For general information about ODBC, see http://support.microsoft.com/kb/110093.

1.2.2.1. 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 uses the 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 1.4, “Connector/ODBC Configuration”.

- **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).

1.2.2.2. 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.

Both Windows and Mac OS X include ODBC driver managers with the operating system. Most ODBC Driver Manager implementations also include an administration application that makes the configuration of DSN and drivers easier. Examples and information on these managers, including Unix ODBC driver managers are listed below:

- Microsoft Windows ODBC Driver Manager (`odbc32.dll`), http://support.microsoft.com/kb/110093.
- Mac OS X includes `ODBC Administrator`, a GUI application that provides a simpler configuration mechanism for the Unix 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. http://www.apple.com.
- `unixODBC` Driver Manager for Unix (`libodbc.so`). See http://www.unixodbc.org, for more information. The `unixODBC` Driver Manager includes the Connector/ODBC driver 3.51 in the installation package, starting with version `unixODBC` 2.1.2.
- `iODBC` ODBC Driver Manager for Unix (`libiodbc.so`), see http://www.iodbc.org, for more information.

1.3. Connector/ODBC Installation

You can install the Connector/ODBC drivers using two different methods, a binary installation and a source installation. The binary installation is the easiest and most straightforward method of installation. Using the source installation methods should only be necessary on 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.

Where to Get Connector/ODBC

Sun Microsystems, Inc distributes its MySQL products under the General Public License (GPL). 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/.

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/.

Supported Platforms

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

- Windows 95, 98, Me, NT, 2000, XP, and 2003
- All Unix-like Operating Systems, including: AIX, Amiga, BSDI, DEC, FreeBSD, HP-UX 10/11, Linux, NetBSD, OpenBSD, OS/2, SGI Irix, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64 Unix
- Mac OS X and Mac OS X Server

Using a binary distribution offers the most straightforward method for installing Connector/ODBC. If you want more control over the driver, the installation location and or to customize elements of the driver you will need to build and install from the source.

If a binary distribution is not available for a particular platform 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>`, so that it becomes available for other users.

Note

On all non-Windows platforms except Mac 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` and you will need to recompile the driver against `iODBC` if you wish to use them together. For further information see Section 1.2.2.2, "ODBC Driver Managers".

For further instructions:

Platform	Binary	Source
Windows	Installation Instructions	Build Instructions
Unix/Linux	Installation Instructions	Build Instructions
Mac OS X	Installation Instructions	

1.3.1. Installing Connector/ODBC from a Binary Distribution on Windows

Before installing the Connector/ODBC drivers on Windows you should ensure that 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.

There are three available distribution types to use when installing for Windows. The contents in each case are identical, it is only the installation method which is different.

- Zipped installer consists of a Zipped package containing a standalone installation application. To install from this package, you must unzip the installer, and then run the installation application. See Section 1.3.1.1, "Installing the Windows Connector/ODBC Driver using an installer" to complete the installation.
- MSI installer, an installation file that can be used with the installer included in Windows 2000, Windows XP and Windows Server 2003. See Section 1.3.1.1, "Installing the Windows Connector/ODBC Driver using an installer" to complete the installation.
- Zipped DLL package, containing the DLL files that need must be manually installed. See Section 1.3.1.2, "Installing the Windows Connector/ODBC Driver using the Zipped DLL package" to complete the installation.

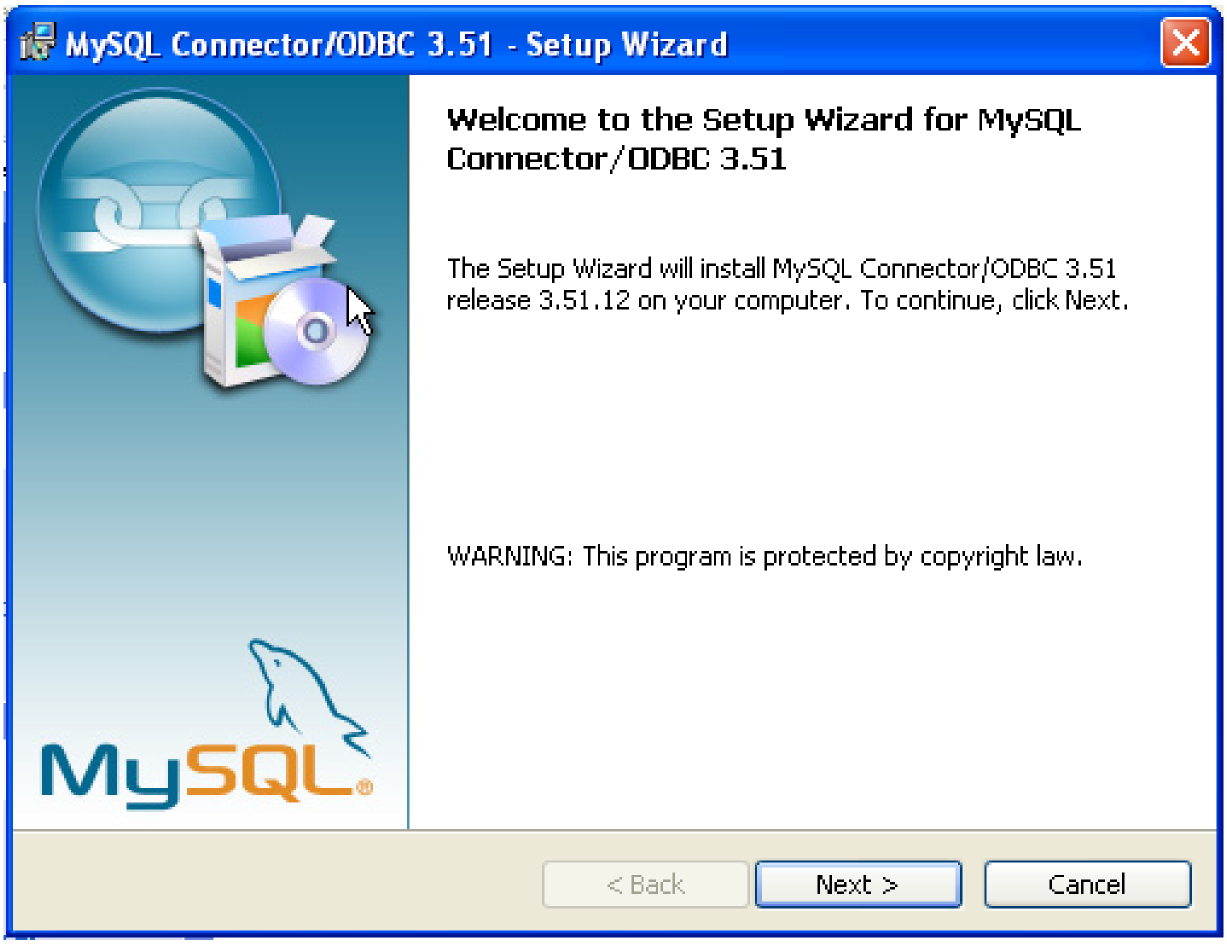
> **Note**
>
> An OLEDB/ODBC driver for Windows 64-bit is available from Microsoft Downloads.

1.3.1.1. Installing the Windows Connector/ODBC Driver using an installer

The installer packages offer a very simple method for installing the Connector/ODBC drivers. If you have downloaded the zipped installer then you must extract the installer application. The basic installation process is identical for both installers.

You should follow these steps to complete the installation:

1. Double click on the standalone installer that you extracted, or the MSI file you downloaded.
2. The MySQL Connector/ODBC 3.51 - Setup Wizard will start. Click the NEXT button to begin the installation process.



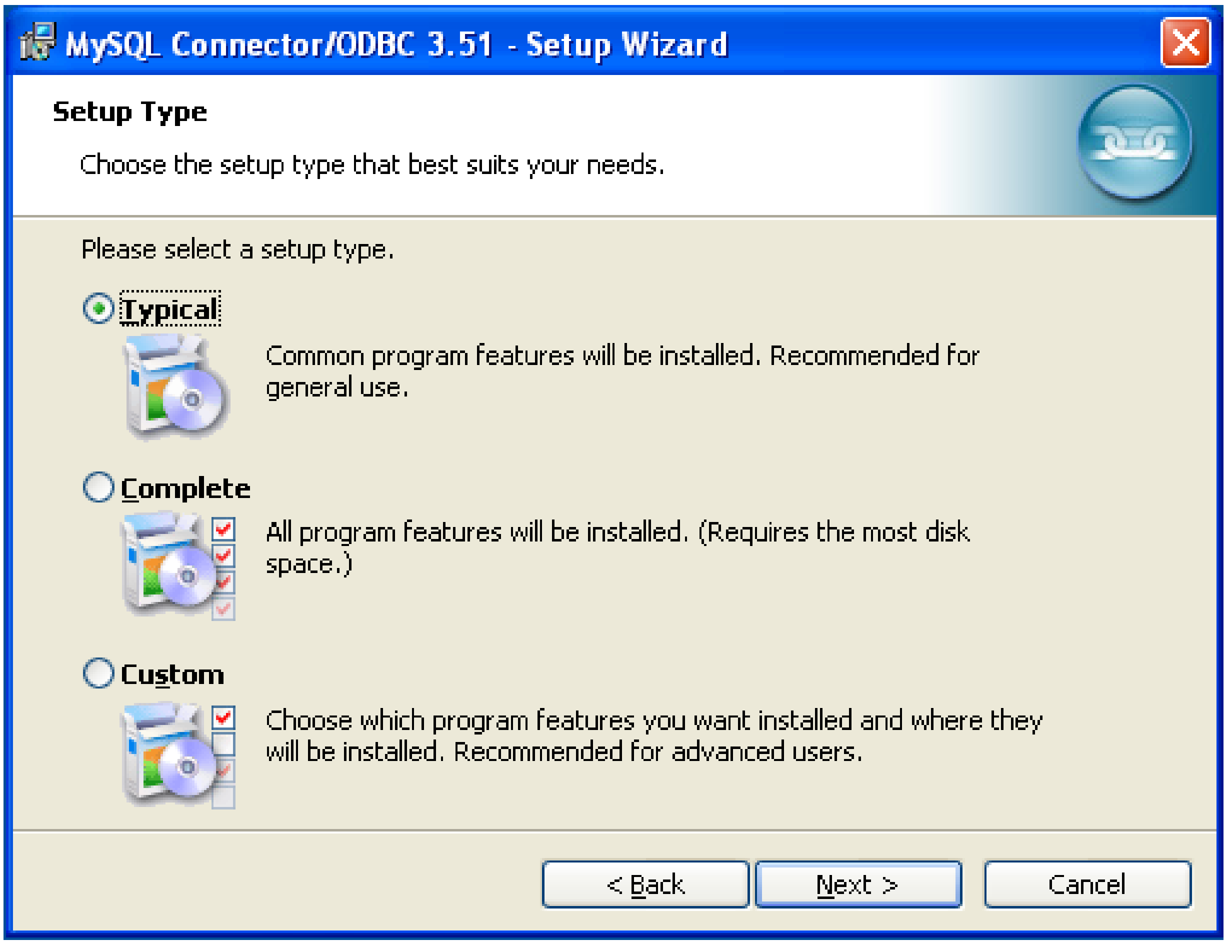
Welcome to the Setup Wizard for MySQL Connector/ODBC 3.51

The Setup Wizard will install MySQL Connector/ODBC 3.51 release 3.51.12 on your computer. To continue, click Next.

WARNING: This program is protected by copyright law.

3. You will need to choose the installation type. The Typical installation provides the standard files you will need to connect to a MySQL database using ODBC. The Complete option installs all the available files, including debug and utility components. It is recommended you choose one of these two options to complete the installation. If choose one of these methods, click NEXT and then proceed to step 5.

   You may also choose a Custom installation, which enables you to select the individual components that you want to install. You have chosen this method, click NEXT and then proceed to step 4.



MySQL Connector/ODBC 3.51 - Setup Wizard

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

[✓ Typical]

Common program features will be installed. Recommended for general use.

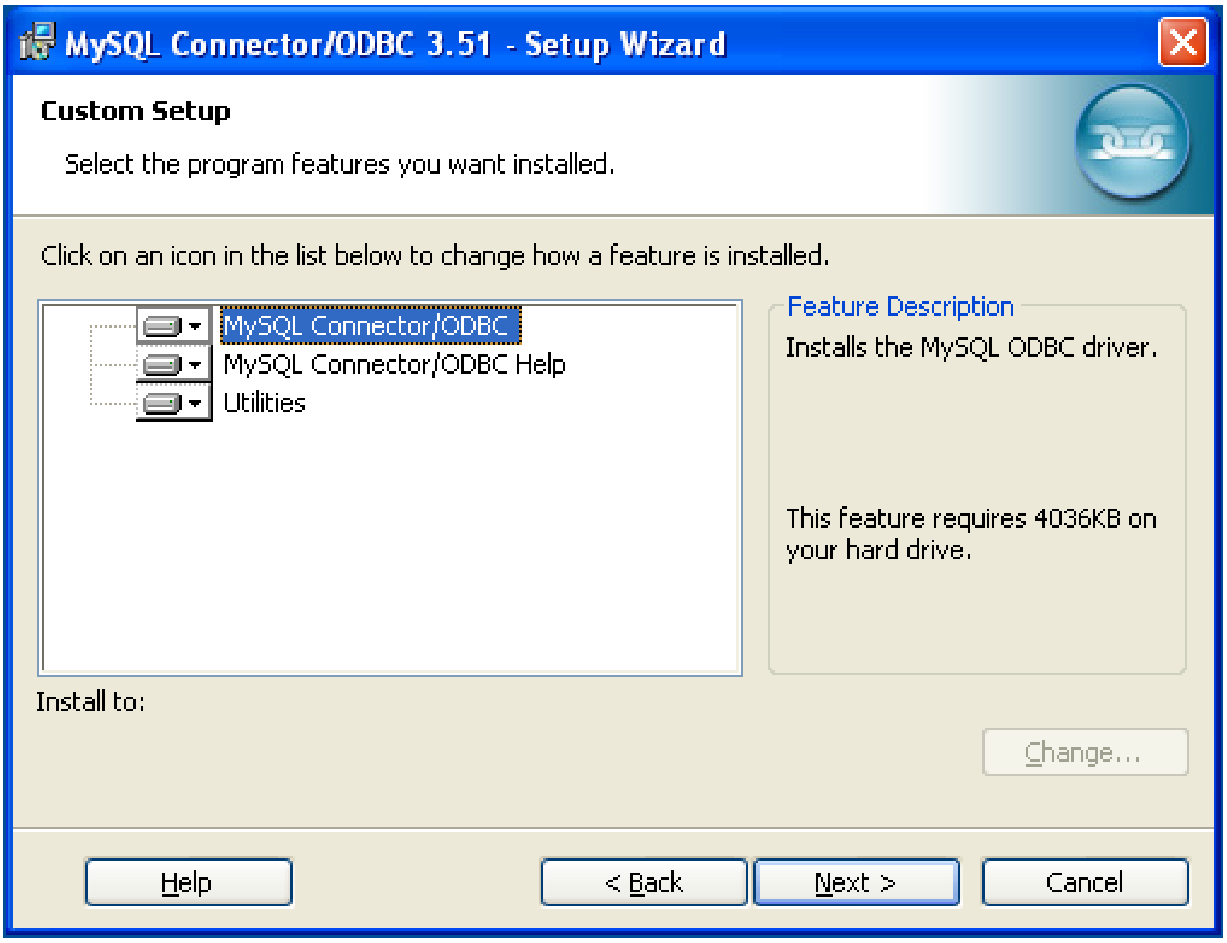
Complete

All program features will be installed. (Requires the most disk space.)

Custom

Choose which program features you want installed and where they will be installed. Recommended for advanced users.

4. If you have chosen a custom installation, use the pop-ups to select which components to install and then click NEXT to install the necessary files.


MySQL Connector/ODBC 3.51 - Setup Wizard

Custom Setup

Select the program features you want installed.

Click on an icon in the list below to change how a feature is installed.

- MySQL Connector/ODBC
- MySQL Connector/ODBC Help
- Utilities

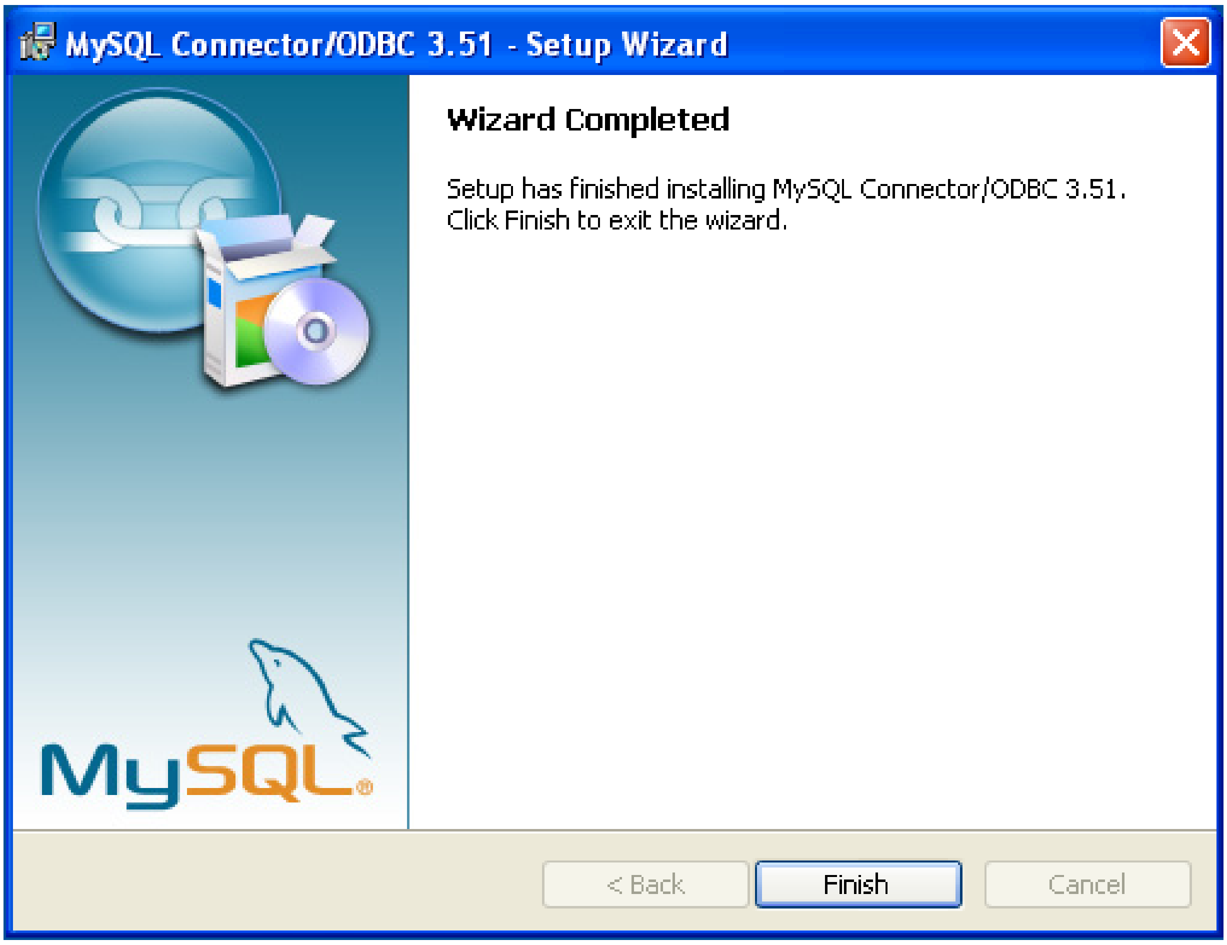
Feature Description

Installs the MySQL ODBC driver.

This feature requires 4036KB on your hard drive.

Install to:

5. Once the files have copied to your machine, the installation is complete. Click FINISH to exit the installer.



Wizard Completed

Setup has finished installing MySQL Connector/ODBC 3.51.
Click Finish to exit the wizard.

Now the installation is complete, you can continue to configure your ODBC connections using Section 1.4, “Connector/ODBC Configuration”.

1.3.1.2. Installing the Windows Connector/ODBC Driver using the Zipped DLL package

If you have downloaded the Zipped DLL package then you must install the individual files required for Connector/ODBC operation manually. Once you have unzipped the installation files, you can either perform this operation by hand, executing each statement individually, or you can use the included Batch file to perform an installation to the default 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 will copy the necessary files into the default location, and then register the Connector/ODBC driver with the Windows ODBC manager.

If you want to copy the files to an alternative location - for example, to run or test different versions of the Connector/ODBC driver on the same machine, then you must copy the files by hand. It is however not recommended to install these files in a non-standard location. To copy the files by hand to the default installation location 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 is to copy them into the default Windows system directory `\Windows\System32`:

```
C:\> copy lib\myodbc3S.dll \Windows\System32
C:\> copy lib\myodbc3S.lib \Windows\System32
C:\> copy lib\myodbc3.dll \Windows\System32
C:\> copy lib\myodbc3.lib \Windows\System32
```

5. Copy the Connector/ODBC tools. These must be placed into a directory that is in the system `PATH`. The default is to install these into the Windows system directory `\Windows\System32`:

```
C:\> copy bin\myodbc3i.exe \Windows\System32
C:\> copy bin\myodbc3m.exe \Windows\System32
C:\> copy bin\myodbc3c.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, you must register the Connector/ODBC driver with the ODBC manager:

```
C:\> myodbc3i -a -d -t"MySQL ODBC 3.51 Driver;\
  DRIVER=myodbc3.dll;SETUP=myodbc3S.dll"
```

   You must change the references to the DLL files and command location in the above statement if you have not installed these files into the default location.

1.3.2. Installing Connector/ODBC from a Binary Distribution on Unix

There are two methods available for installing Connector/ODBC on Unix from a binary distribution. For most Unix environments you will need to use the tarball distribution. For Linux systems, there is also an RPM distribution available.

Note

To install Connector/ODBC 5.1 on Unix you require unixODBC 2.2.12 or later to be installed.

1.3.2.1. 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 that demonstrate the process using the Linux version of the tarball:

```
shell> su root
shell> gunzip mysql-connector-odbc-3.51.11-i686-pc-linux.tar.gz
shell> tar xvf mysql-connector-odbc-3.51.11-i686-pc-linux.tar
shell> cd mysql-connector-odbc-3.51.11-i686-pc-linux
```

Read the installation instructions in the `INSTALL` file and execute these commands.

Then proceed on to Section 1.4.5, “Configuring a Connector/ODBC DSN on Unix”, to configure the DSN for Connector/ODBC. For more information, refer to the `INSTALL` file that comes with your distribution.

1.3.2.2. 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-3.51.12.i386.rpm
```

If the driver exists, upgrade it like this:

```
shell> su root
shell> rpm -Uvh mysql-connector-odbc-3.51.12.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. Proceed onto Section 1.4.5, "Configuring a Connector/ODBC DSN on Unix".

To **uninstall** the driver, become `root` and execute an `rpm` command:

```
shell> su root
shell> rpm -e mysql-connector-odbc
```

1.3.3. Installing Connector/ODBC from a Binary Distribution on Mac OS X

Mac 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 enables you to connect to MySQL databases on any platform through the ODBC interface. You should only need to install the Connector/ODBC driver when your application requires an ODBC interface. Applications that require or can use ODBC (and therefore the Connector/ODBC driver) include ColdFusion, Filemaker Pro, 4th Dimension and many other applications.

Mac OS X includes its own ODBC manager, based on the `iODBC` manager. Mac OS X includes an administration tool that provides easier administration of ODBC drivers and configuration, updating the underlying `iODBC` configuration files.

The method for installing Connector/ODBC on Mac OS X depends on the version on Connector/ODBC you are using. For Connector/ODBC 3.51.14 and later, the package is provided as a compressed tar archive that you must manually install. For Connector/ODBC 3.51.13 and earlier the software was provided on a compressed disk image (`.dmg`) file and included an installer.

In either case, the driver is designed to work with the iODBC driver manager included with Mac OS X.

To install Connector/ODBC 3.51.14 and later:

1. Download the installation file. Note that versions are available for both PowerPC and Intel platforms.

2. Extract the archive:

   ```
   shell> tar zxf mysql-connector-odbc-3.51.16-osx10.4-x86-32bit.tar.gz
   ```

3. The directory created will contain two subdirectories, `lib` and `bin`. You need to copy these to a suitable location such as `/usr/local`:

   ```
   shell> cp bin/* /usr/local/bin
   shell> cp lib/* /usr/local/lib
   ```

4. Finally, you must register the driver with iODBC using the `myodbc3i` tool you just installed:

   ```
   shell> myodbc3i -a -d -t"MySQL ODBC 3.51 Driver;Driver=/usr/local/lib/libmyodbc3.so;Setup=/usr/local/lib/libmyodbc
   ```

You can verify the installed drivers either by using the ODBC Administrator application or the `myodbc3i` utility:

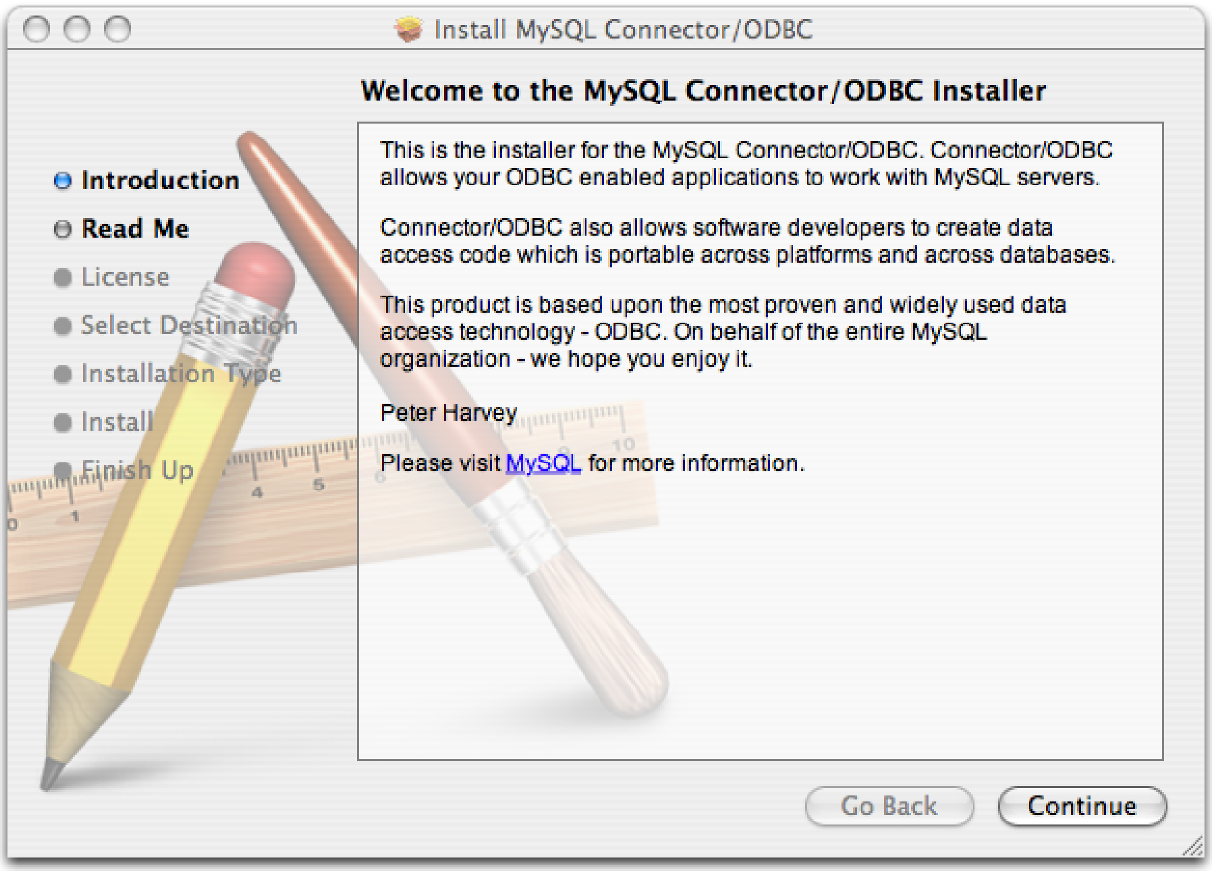
```
shell> myodbc3i -q -d
```

To install Connector/ODBC 3.51.13 and earlier, follow these steps:

1. Download the file to your computer and double-click on the downloaded image file.

2. Within the disk image you will find an installer package (with the `.pkg` extension). Double click on this file to start the Mac OS X installer.

3. You will be presented with the installer welcome message. Click the CONTINUE button to begin the installation process.



Welcome to the MySQL Connector/ODBC Installer

This is the installer for the MySQL Connector/ODBC. Connector/ODBC allows your ODBC enabled applications to work with MySQL servers.

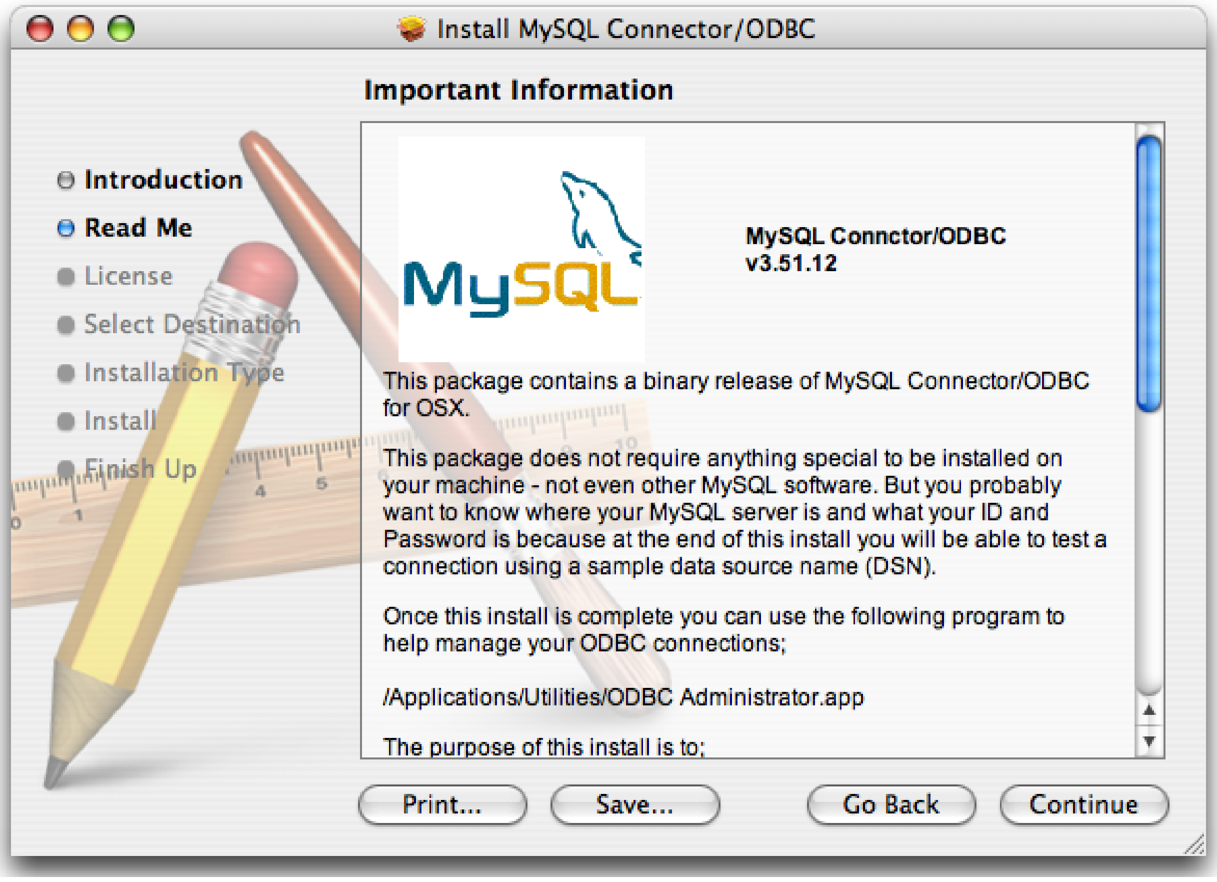
Connector/ODBC also allows software developers to create data access code which is portable across platforms and across databases.

This product is based upon the most proven and widely used data access technology - ODBC. On behalf of the entire MySQL organization - we hope you enjoy it.

Peter Harvey

Please visit MySQL for more information.

4. Please take the time to read the Important Information as it contains guidance on how to complete the installation process. Once you have read the notice and collected the necessary information, click CONTINUE.



Important Information

MySQL Connctor/ODBC
v3.51.12

This package contains a binary release of MySQL Connector/ODBC for OSX.

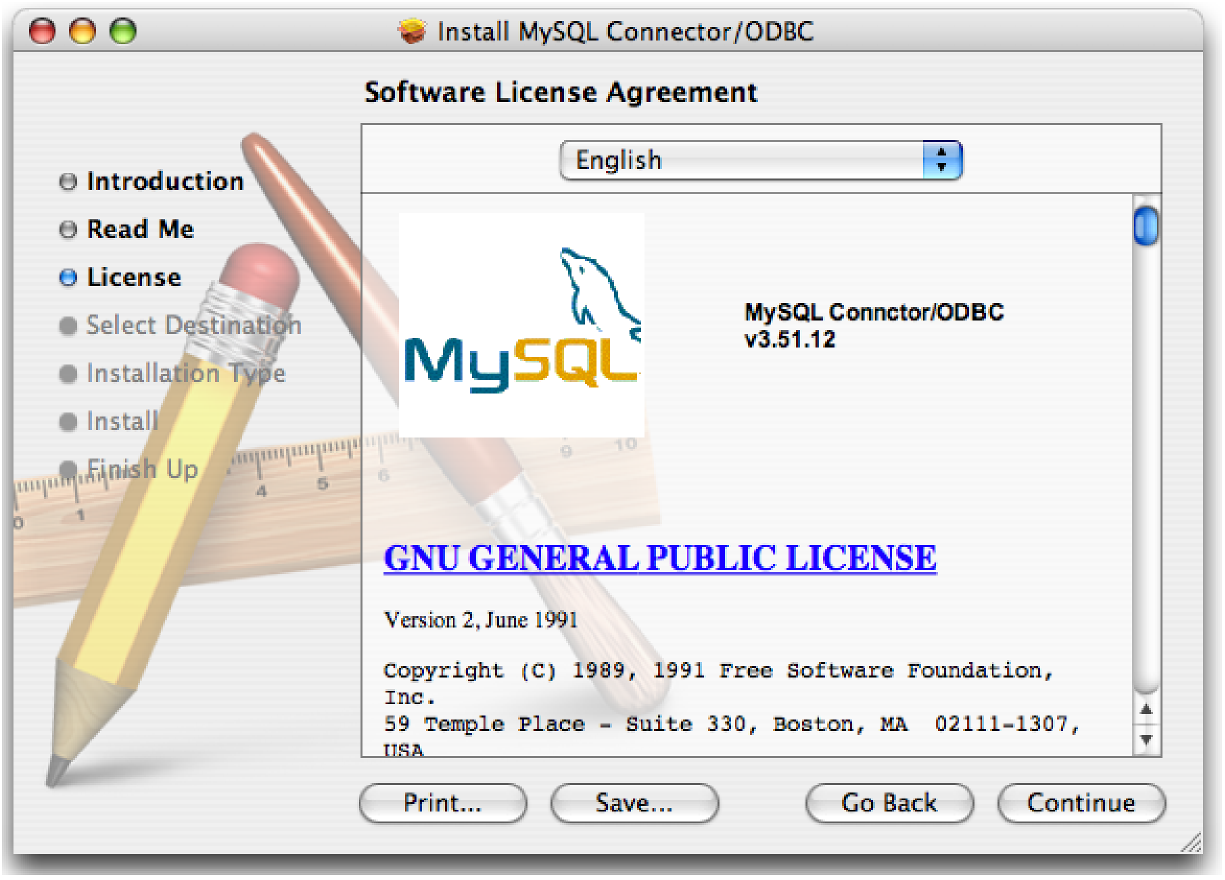
This package does not require anything special to be installed on your machine - not even other MySQL software. But you probably want to know where your MySQL server is and what your ID and Password is because at the end of this install you will be able to test a connection using a sample data source name (DSN).

Once this install is complete you can use the following program to help manage your ODBC connections;

/Applications/Utilities/ODBC Administrator.app

The purpose of this install is to;

5. Connector/ODBC drivers are made available under the GNU General Public License. Please read the license if you are not familiar with it before continuing installation. Click CONTINUE to approve the license (you will be asked to confirm that decision) and continue the installation.



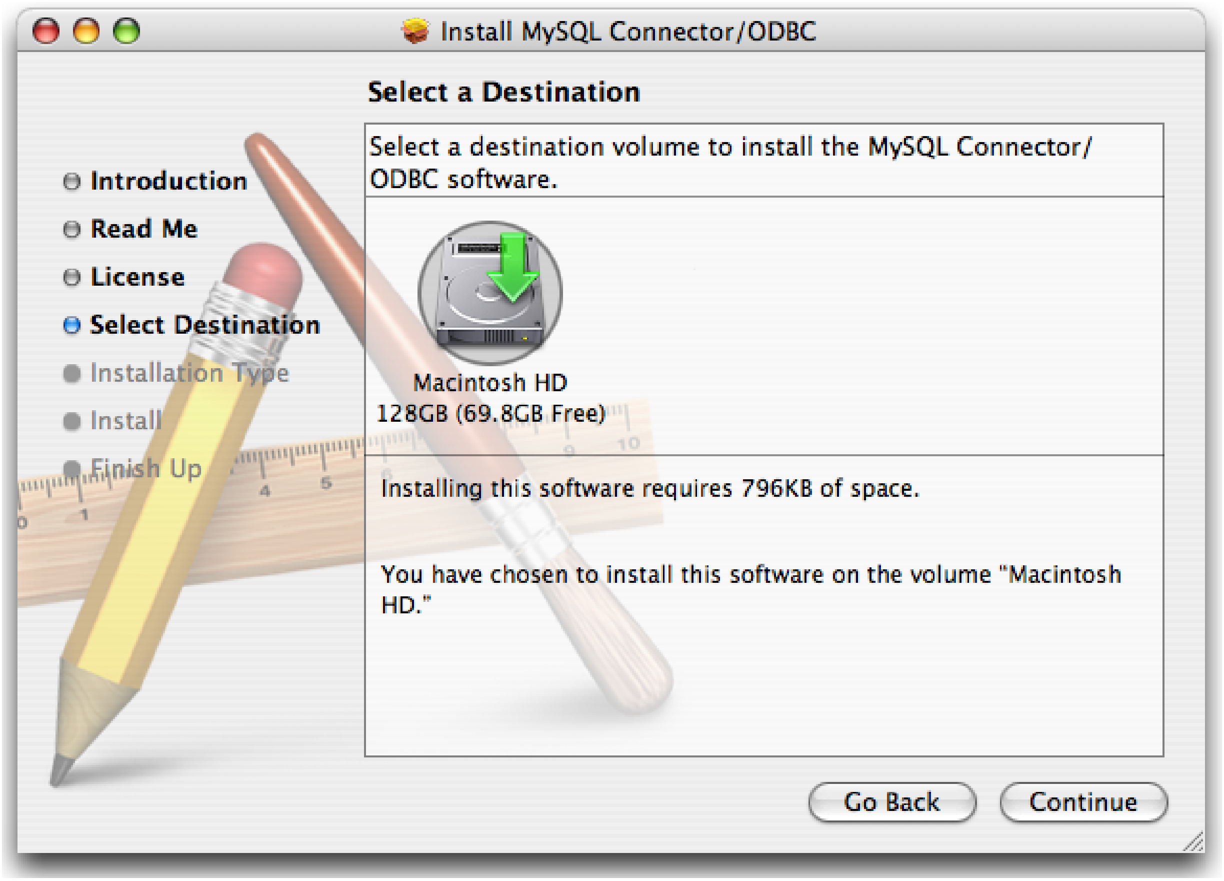
MySQL Connctor/ODBC
v3.51.12

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

6. Choose a location to install the Connector/ODBC drivers and the ODBC Administrator application. You must install the files onto a drive with an operating system and you may be limited in the choices available. Select the drive you want to use, and then click CONTINUE.



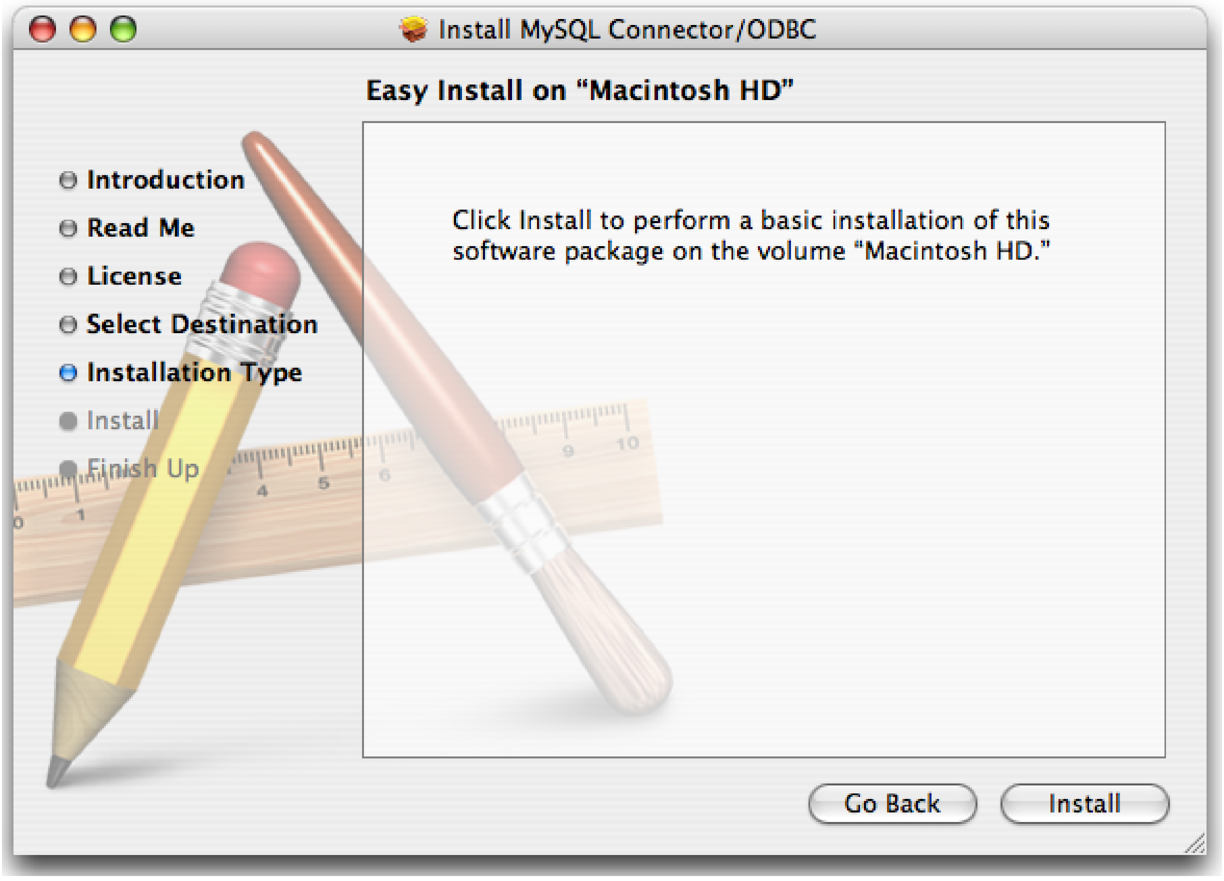
Select a Destination

Select a destination volume to install the MySQL Connector/ ODBC software.

Installing this software requires 796KB of space.

You have chosen to install this software on the volume "Macintosh HD."

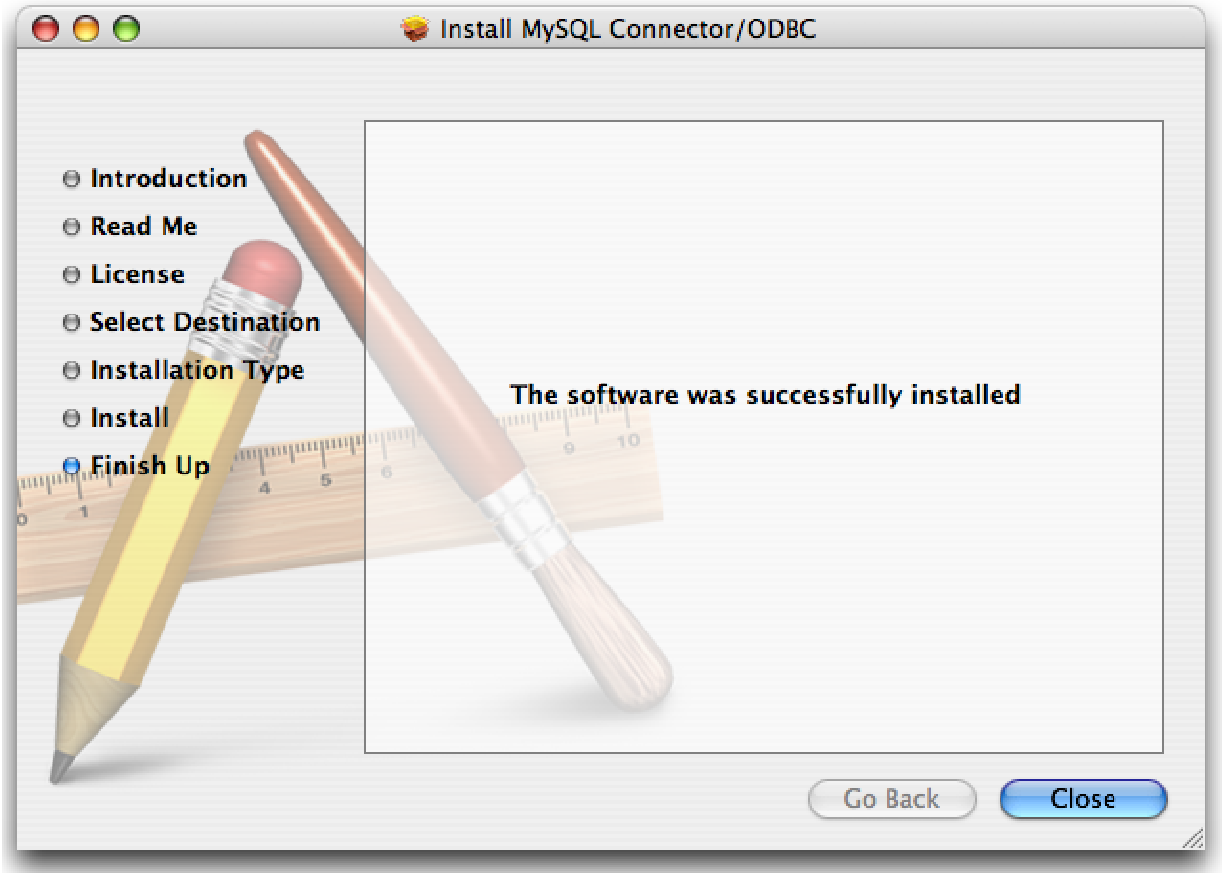
7. The installer will automatically select the files that need to be installed on your machine. Click INSTALL to continue. The installer will copy the necessary files to your machine. A progress bar will be shown indicating the installation progress.



Easy Install on "Macintosh HD"

Click Install to perform a basic installation of this software package on the volume "Macintosh HD."

8. When installation has been completed you will get a window like the one shown below. Click CLOSE to close and quit the installer.



The software was successfully installed

1.3.4. Installing Connector/ODBC from a Source Distribution on Windows

You should only need to install Connector/ODBC from source on Windows if you want to change or modify the source or installation. If you are unsure whether to install from source, please use the binary installation detailed in Section 1.3.1, "Installing Connector/ODBC from a Binary Distribution on Windows".

Installing 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.
- Compatible `make` tool. Microsoft's `nmake` is used in the examples in this section.
- 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 new 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/.

1.3.4.1. Building Connector/ODBC 3.51

Connector/ODBC source distributions include `Makefiles` that require the `nmake` or other `make` utility. In the distribution, you can find `Makefile` for building the release version and `Makefile_debug` for building debugging versions of the driver libraries and DLLs.

To build the driver, use this procedure:

1. Download and extract the sources to a folder, then change directory into that folder. The following command assumes the folder is named `myodbc3-src`:

```
C:\> cd myodbc3-src
```

2. Edit `Makefile` to specify the correct path for the MySQL client libraries and header files. Then use the following commands to build and install the release version:

   ```
   C:\> nmake -f Makefile
   C:\> nmake -f Makefile install
   ```

   `nmake -f Makefile` builds the release version of the driver and places the binaries in subdirectory called `Release`.

   `nmake -f Makefile install` installs (copies) the driver DLLs and libraries (`myodbc3.dll`, `myodbc3.lib`) to your system directory.

3. To build the debug version, use `Makefile_Debug` rather than `Makefile`, as shown below:

   ```
   C:\> nmake -f Makefile_debug
   C:\> nmake -f Makefile_debug install
   ```

4. You can clean and rebuild the driver by using:

   ```
   C:\> nmake -f Makefile clean
   C:\> nmake -f Makefile install
   ```

Note

- Make sure to specify the correct MySQL client libraries and header files path in the Makefiles (set the `MYSQL_LIB_PATH` and `MYSQL_INCLUDE_PATH` variables). The default header file path is assumed to be `C:\mysql\include`. The default library path is assumed to be `C:\mysql\lib\opt` for release DLLs and `C:\mysql\lib\debug` for debug versions.
- For the complete usage of `nmake`, visit http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vcce4/html/evgrfRunningNMAKE.asp.
- If you are using the Subversion tree for compiling, all Windows-specific `Makefiles` are named as `Win_Makefile*`.

1.3.4.2. Testing

After the driver libraries are copied/installed to the system directory, you can test whether the libraries are properly built by using the samples provided in the `samples` subdirectory:

```
C:\> cd samples
C:\> nmake -f Makefile all
```

1.3.5. Installing 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` 2.95.2 or later, SGI C++, and SunPro C++ are some of the compilers that are known to work.
- A good `make` program. GNU `make` is always recommended and is sometimes required.
- 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 new 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/.

  If you have built your own MySQL server and/or client libraries from source then you must have used the `--enable-thread-safe-client` option to `configure` when the libraries were built.

  You should also ensure that the `libmysqlclient` library were built and installed as a shared library.
- A compatible ODBC manager must be installed. Connector/ODBC is known to work with the `iODBC` and `unixODBC` managers. See Section 1.2.2.2, “ODBC Driver Managers”, for more information.

- If you are using a character set that isn't compiled into the MySQL client library then you need to 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, you then have to run `configure` and build the library using `make`.

1.3.5.1. Typical `configure` Options

The `configure` script gives you a great deal of control over how you configure your Connector/ODBC build. Typically you do this using options on the `configure` command line. You can also affect `configure` using certain environment variables. For a list of options and environment variables supported by `configure`, run this command:

```
shell> ./configure --help
```

Some of the more commonly used `configure` options are described here:

1. To compile Connector/ODBC, you need to supply the MySQL client include and library files path using the `--with-mysql-path=DIR` option, where *`DIR`* is the directory where MySQL is installed.

   MySQL compile options can be determined by running `DIR/bin/mysql_config`.

2. Supply the standard header and library files path for your ODBC Driver Manager (`iODBC` or `unixODBC`).

   - If you are using `iODBC` and `iODBC` is not installed in its default location (`/usr/local`), you might have to use the `--with-iodbc=DIR` option, where *`DIR`* is the directory where `iODBC` is installed.

     If the `iODBC` headers do not reside in `DIR/include`, you can use the `--with-iodbc-includes=INCDIR` option to specify their location.

     The applies to libraries. If they are not in `DIR/lib`, you can use the `--with-iodbc-libs=LIBDIR` option.

   - If you are using `unixODBC`, use the `--with-unixODBC=DIR` option (case sensitive) to make `configure` look for `unixODBC` instead of `iODBC` by default, *`DIR`* is the directory where `unixODBC` is installed.

     If the `unixODBC` headers and libraries aren't located in `DIR/include` and `DIR/lib`, use the `--with-unixODBC-includes=INCDIR` and `--with-unixODBC-libs=LIBDIR` options.

3. You might want to specify an installation prefix other than `/usr/local`. For example, to install the Connector/ODBC drivers in `/usr/local/odbc/lib`, use the `--prefix=/usr/local/odbc` option.

The final configuration command looks something like this:

```
shell> ./configure --prefix=/usr/local \
         --with-iodbc=/usr/local \
         --with-mysql-path=/usr/local/mysql
```

1.3.5.2. Additional configure Options

There are a number of other options that you need, or want, to set when configuring the Connector/ODBC driver before it is built.

- To link the driver with MySQL thread safe client libraries `libmysqlclient_r.so` or `libmysqlclient_r.a`, you must specify the following `configure` option:

  ```
  --enable-thread-safe
  ```

  and can be disabled (default) using

  ```
  --disable-thread-safe
  ```

  This option enables the building of the driver thread-safe library `libmyodbc3_r.so` from by linking with MySQL thread-safe client library `libmysqlclient_r.so` (The extensions are OS dependent).

  If the compilation with the thread-safe option fails, it may be because the correct thread-libraries on the system could not be located. You should set the value of `LIBS` to point to the correct thread library for your system.

```
LIBS="-lpthread" ./configure ..
```

- You can enable or disable the shared and static versions of Connector/ODBC using these options:

  ```
  --enable-shared[=yes/no]
  --disable-shared
  --enable-static[=yes/no]
  --disable-static
  ```

- By default, all the binary distributions are built as non-debugging versions (configured with `--without-debug`).

  To enable debugging information, build the driver from source distribution and use the `--with-debug` option when you run `configure`.

- This option is available only for source trees that have been obtained from the Subversion repository. This option does not apply to the packaged source distributions.

  By default, the driver is built with the `--without-docs` option. If you would like the documentation to be built, then execute `configure` with:

  ```
  --with-docs
  ```

1.3.5.3. Building and Compilation

To build the driver libraries, you have to just execute `make`.

```
shell> make
```

If any errors occur, correct them and continue the build process. If you aren't able to build, then send a detailed email to `<myodbc@lists.mysql.com>` for further assistance.

1.3.5.4. Building Shared Libraries

On most platforms, MySQL does not build or support `.so` (shared) client libraries by default. This is based on our experience of problems when building shared libraries.

In cases like this, you have to download the MySQL distribution and configure it with these options:

```
--without-server --enable-shared
```

To build shared driver libraries, you must specify the `--enable-shared` option for `configure`. By default, `configure` does not enable this option.

If you have configured with the `--disable-shared` option, you can build the `.so` file from the static libraries using the following commands:

```
shell> cd mysql-connector-odbc-3.51.01
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
          $CC -bundle -flat_namespace -undefined error \
          -o .libs/libmyodbc3-3.51.01.so \
          catalog.o connect.o cursor.o dll.o error.o execute.o \
          handle.o info.o misc.o myodbc3.o options.o prepare.o \
          results.o transact.o utility.o \
          -L/usr/local/mysql/lib/mysql/ \
          -L/usr/local/iodbc/lib/ \
          -lz -lc -lmysqlclient -liodbcinst
```

Make sure to change `-liodbcinst` to `-lodbcinst` if you are using `unixODBC` instead of `iODBC`, and configure the library paths accordingly.

This builds and places the `libmyodbc3-3.51.01.so` file in the `.libs` directory. Copy this file to the Connector/ODBC library installation directory (`/usr/local/lib` (or the `lib` directory under the installation directory that you supplied with the `--prefix`).

```
shell> cd .libs
shell> cp libmyodbc3-3.51.01.so /usr/local/lib
shell> cd /usr/local/lib
shell> ln -s libmyodbc3-3.51.01.so libmyodbc3.so
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
          $CC -bundle -flat_namespace -undefined error
          -o .libs/libmyodbc3_r-3.51.01.so
          catalog.o connect.o cursor.o dll.o error.o execute.o
          handle.o info.o misc.o myodbc3.o options.o prepare.o
          results.o transact.o utility.o
          -L/usr/local/mysql/lib/mysql/
          -L/usr/local/iodbc/lib/
          -lz -lc -lmysqlclient_r -liodbcinst
```

1.3.5.5. Installing Driver Libraries

To install the driver libraries, execute the following command:

```
shell> make install
```

That command installs one of the following sets of libraries:

For Connector/ODBC 3.51:

- `libmyodbc3.so`
- `libmyodbc3-3.51.01.so`, where 3.51.01 is the version of the driver
- `libmyodbc3.a`

For thread-safe Connector/ODBC 3.51:

- `libmyodbc3_r.so`
- `libmyodbc3-3_r.51.01.so`
- `libmyodbc3_r.a`

For more information on build process, refer to the `INSTALL` file that comes with the source distribution. Note that if you are trying to use the `make` from Sun, you may end up with errors. On the other hand, GNU `gmake` should work fine on all platforms.

1.3.5.6. Testing Connector/ODBC on Unix

To run the basic samples provided in the distribution with the libraries that you built, use the following command:

```
shell> make test
```

Before running the tests, create the DSN 'myodbc3' in `odbc.ini` and set the environment variable `ODBCINI` to the correct `odbc.ini` file; and MySQL server is running. You can find a sample `odbc.ini` with the driver distribution.

You can even modify the `samples/run-samples` script to pass the desired DSN, UID, and PASSWORD values as the command-line arguments to each sample.

1.3.5.7. Building Connector/ODBC from Source on Mac OS X

To build the driver on Mac 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 Mac 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 -lmysqlclienti_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.

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
```

1.3.5.8. Building Connector/ODBC from Source on HP-UX

To build the driver on HP-UX 10.x or 11.x, make use of the following `configure` example:

If using `cc`:

```
shell> CC="cc" \
          CFLAGS="+z" \
          LDFLAGS="-Wl,+b:-Wl,+s" \
          ./configure --prefix=/usr/local
          --with-unixodbc=/usr/local
          --with-mysql-path=/usr/local/mysql/lib/mysql
          --enable-shared
          --enable-thread-safe
```

If using `gcc`:

```
shell> CC="gcc" \
          LDFLAGS="-Wl,+b:-Wl,+s" \
          ./configure --prefix=/usr/local
          --with-unixodbc=/usr/local
          --with-mysql-path=/usr/local/mysql
          --enable-shared
          --enable-thread-safe
```

Once the driver is built, cross-check its attributes using `chatr .libs/libmyodbc3.sl` to determine whether you need to have set the MySQL client library path using the `SHLIB_PATH` environment variable. For static versions, ignore all shared-library options and run `configure` with the `--disable-shared` option.

1.3.5.9. Building Connector/ODBC from Source on AIX

To build the driver on AIX, 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-thread-safe
```

Note

For more information about how to build and set up the static and shared libraries across the different platforms refer to ' Using static and shared libraries across platforms'.

1.3.6. Installing Connector/ODBC from the Development Source Tree

Caution

You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL

Connector/ODBC up and running on your system, you should use a standard release distribution.

To be able to access the Connector/ODBC source tree, you must have Subversion installed. Subversion is freely available from http://subversion.tigris.org/.

To build from the source trees, you need the following tools:

- autoconf 2.52 (or newer)
- automake 1.4 (or newer)
- libtool 1.4 (or newer)
- m4

The most recent development source tree is available from our public Subversion trees at http://dev.mysql.com/tech-resources/sources.html.

To checkout 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> svn co http://svn.mysql.com/svnpublic/connector-odbc3
```

You should now have a copy of the entire Connector/ODBC source tree in the directory `connector-odbc3`. To build from this source tree on Unix or Linux follow these steps:

```
shell> cd connector-odbc3
shell> aclocal
shell> autoheader
shell> autoconf
shell> automake;
shell> ./configure  # Add your favorite options here
shell> make
```

For more information on how to build, refer to the `INSTALL` file located in the same directory. For more information on options to `configure`, see Section 1.3.5.1, "Typical `configure` Options"

When the build is done, run `make install` to install the Connector/ODBC 3.51 driver on your system.

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 1.3.4, "Installing Connector/ODBC from a Source Distribution on Windows".

After the initial checkout operation to get the source tree, you should run `svn update` periodically update your source according to the latest version.

1.4. Connector/ODBC Configuration

Before you connect to a MySQL database using the Connector/ODBC driver you must configure an ODBC *Data Source Name*. 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.

1.4.1. 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.

1.4.2. Connector/ODBC Connection Parameters

You can specify the parameters in the following tables for Connector/ODBC when configuring a DSN. Users on Windows can use the Options and Advanced panels when configuring a DSN to set these parameters; see the table for information on which options relate to which fields and checkboxes. On Unix and Mac 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.

Parameter	Default Value	Comment
`user`	ODBC	The user name used to connect to MySQL.
`uid`	ODBC	Synonymous with `user`. Added in 3.51.16.
`server`	`localhost`	The host name of the MySQL server.
`database`		The default database.
`option`	0	Options that specify how Connector/ODBC should work. See below.
`port`	3306	The TCP/IP port to use if `server` is not `localhost`.
`initstmt`		Initial statement. A statement to execute when connecting to MySQL. In version 3.51 the parameter is called `stmt`. Note, the driver supports the initial statement being executed only at the time of the initial connection.
`password`		The password for the `user` account on `server`.
`pwd`		Synonymous with `password`. Added in 3.51.16.
`socket`		The Unix socket file or Windows named pipe to connect to if `server` is `localhost`.
`sslca`		The path to a file with a list of trust SSL CAs. Added in 3.51.16.
`sslcapath`		The path to a directory that contains trusted SSL CA certificates in PEM format. Added in 3.51.16.
`sslcert`		The name of the SSL certificate file to use for establishing a secure connection. Added in 3.51.16.
`sslcipher`		A list of allowable 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.
`charset`		The character set to use for the connection. Added in 3.51.17.
`sslverify`		If set to 1, the SSL certificate will be verified when used with the MySQL connection. If not set, then the default behaviour is to ignore SSL certificate verification.
`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. This option was added in Connector/ODBC 3.51.27.

Parameter	Default Value	Comment
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. This option was added in Connector/ODBC 3.51.27.

Note

The SSL configuration parameters can also be automatically loaded from a my.ini or my.cnf file.

The option argument is used to tell Connector/ODBC that the client isn't 100% ODBC compliant. On Windows, you normally select options by toggling the checkboxes in the connection screen, but you can also select them in the option argument. The following options are listed in the order in which they appear in the Connector/ODBC connect screen.

Value	Flagname	GUI Option	Description
1	FLAG_FIELD_LENGTH	Do not Optimize Column Width	The client cannot handle that Connector/ODBC returns the real width of a column. This option was removed in 3.51.18.
2	FLAG_FOUND_ROWS	Return Matching Rows	The client cannot handle that 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 to get this to work.
4	FLAG_DEBUG	Trace Driver Calls To myodbc.log	Make a debug log in C:\myodbc.log on Windows, or /tmp/myodbc.log on Unix variants. This option was removed in Connector/ODBC 3.51.18.
8	FLAG_BIG_PACKETS	Allow Big Results	Do not set any packet limit for results and bind parameters. Without this option, parameter binding will be truncated to 255 characters.
16	FLAG_NO_PROMPT	Do not Prompt Upon Connect	Do not prompt for questions even if driver would like to prompt.
32	FLAG_DYNAMIC_CURSOR	Enable Dynamic Cursor	Enable or disable the dynamic cursor support.
64	FLAG_NO_SCHEMA	Ignore # in Table Name	Ignore use of database name in *db_name.tbl_name.col_name*.
128	FLAG_NO_DEFAULT_CURSOR	User Manager Cursors	Force use of ODBC manager cursors (experimental).
256	FLAG_NO_LOCALE	Do not Use Set Locale	Disable the use of extended fetch (experimental).
512	FLAG_PAD_SPACE	Pad Char To Full Length	Pad CHAR columns to full column length.
1024	FLAG_FULL_COLUMN_NAMES	Return Table Names for SQLDescribeCol	SQLDescribeCol() returns fully qualified column names.
2048	FLAG_COMPRESSED_PROTO	Use Compressed Protocol	Use the compressed client/server protocol.
4096	FLAG_IGNORE_SPACE	Ignore Space After Function Names	Tell server to ignore space after function name and before “(” (needed by PowerBuilder). This makes all function names keywords.
8192	FLAG_NAMED_PIPE	Force Use of Named Pipes	Connect with named pipes to a mysqld server running on NT.
16384	FLAG_NO_BIGINT	Change BIGINT Columns to Int	Change BIGINT columns to INT columns (some applications cannot handle BIGINT).
32768	FLAG_NO_CATALOG	No Catalog	Forces results from the catalog functions, such as SQLTables, to always return NULL and the driver to report that catalogs are not supported.
65536	FLAG_USE_MYCNF	Read Options From my.cnf	Read parameters from the [client] and [odbc] groups from my.cnf.
131072	FLAG_SAFE	Safe	Add some extra safety checks.
262144	FLAG_NO_TRANSACTIONS	Disable transactions	Disable transactions.
524288	FLAG_LOG_QUERY	Save queries to myod-	Enable query logging to c:\myodbc.sql(/

		bc.sql	tmp/myodbc.sql) file. (Enabled only in debug mode.)
1048576	FLAG_NO_CACHE	Do not Cache Result (forward only cursors)	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.
2097152	FLAG_FORWARD_CURSOR	Force Use Of Forward Only Cursors	Force the use of Forward-only cursor type. In case of applications setting the default static/dynamic cursor type, and one wants the driver to use non-cache result sets, then this option ensures the forward-only cursor behavior.
4194304	FLAG_AUTO_RECONNECT	Enable auto-reconnect.	Enables auto-reconnection functionality. You should not use this option with transactions, since a auto reconnection during a incomplete transaction may cause corruption. Note that an auto-reconnected connection will not inherit the same settings and environment as the original. This option was added in Connector/ODBC 3.51.13.
8388608	FLAG_AUTO_IS_NULL	Flag Auto Is Null	When FLAG_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 FLAG_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 behaviour. Thus, omitting the flag disables the compatibility option and forces SQL standard behaviour. See IS NULL. This option was added in Connector/ODBC 3.51.13.
16777216	FLAG_ZERO_DATE_TO_MIN	Flag Zero Date to Min	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. This option was added in Connector/ODBC 3.51.17.
33554432	FLAG_MIN_DATE_TO_ZERO	Flag Min Date to Zero	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. This option was added in Connector/ODBC 3.51.17.
67108864	FLAG_MULTI_STATEMENTS	Allow multiple statements	Enables support for batched statements. This option was added in Connector/ODBC 3.51.18.
134217728	FLAG_COLUMN_SIZE_S32	Limit column size to 32-bit value	Limits the column size to a signed 32-bit value to prevent problems with larger column sizes in applications that do not support them. This option is automatically enabled when working with ADO applications. This option was added in Connector/ODBC 3.51.22.
268435456	FLAG_NO_BINARY_RESULT	Always handle binary function results as character data	When set this option disables charset 63 for columns with an empty org_table. This option was added in Connector/ODBC 3.51.26.

To select multiple options, add together their values. For example, setting `option` to 12 (4+8) gives you debugging without packet limits.

The following table shows some recommended `option` values for various configurations.

Configuration	Option Value
Microsoft Access, Visual Basic	3
Driver trace generation (Debug mode)	4
Microsoft Access (with improved DELETE queries)	35
Large tables with too many rows	2049

Sybase PowerBuilder	135168
Query log generation (Debug mode)	524288
Generate driver trace as well as query log (Debug mode)	524292
Large tables with no-cache results	3145731

1.4.3. Configuring a Connector/ODBC DSN on Windows

The `ODBC Data Source Administrator` within Windows enables you to 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**
>
> Because it is possible to create DSN using either the 32-bit or 64-bit driver, but using the same DNS identifier, it is advisable to include the driver being used within the DSN identifier. This will help you to identify the DSN when using it from applications such as Excel that are only compatible with the 32-bit driver. For example, you might add `Using32bitCODBC` to the DSN identifier for the 32-bit interface and `Using64bitCODBC` for those using the 64-bit Connector/ODBC driver.

1. On the `Start` menu, choose `Administrative Tools`, and then click `Data Sources (ODBC)`.

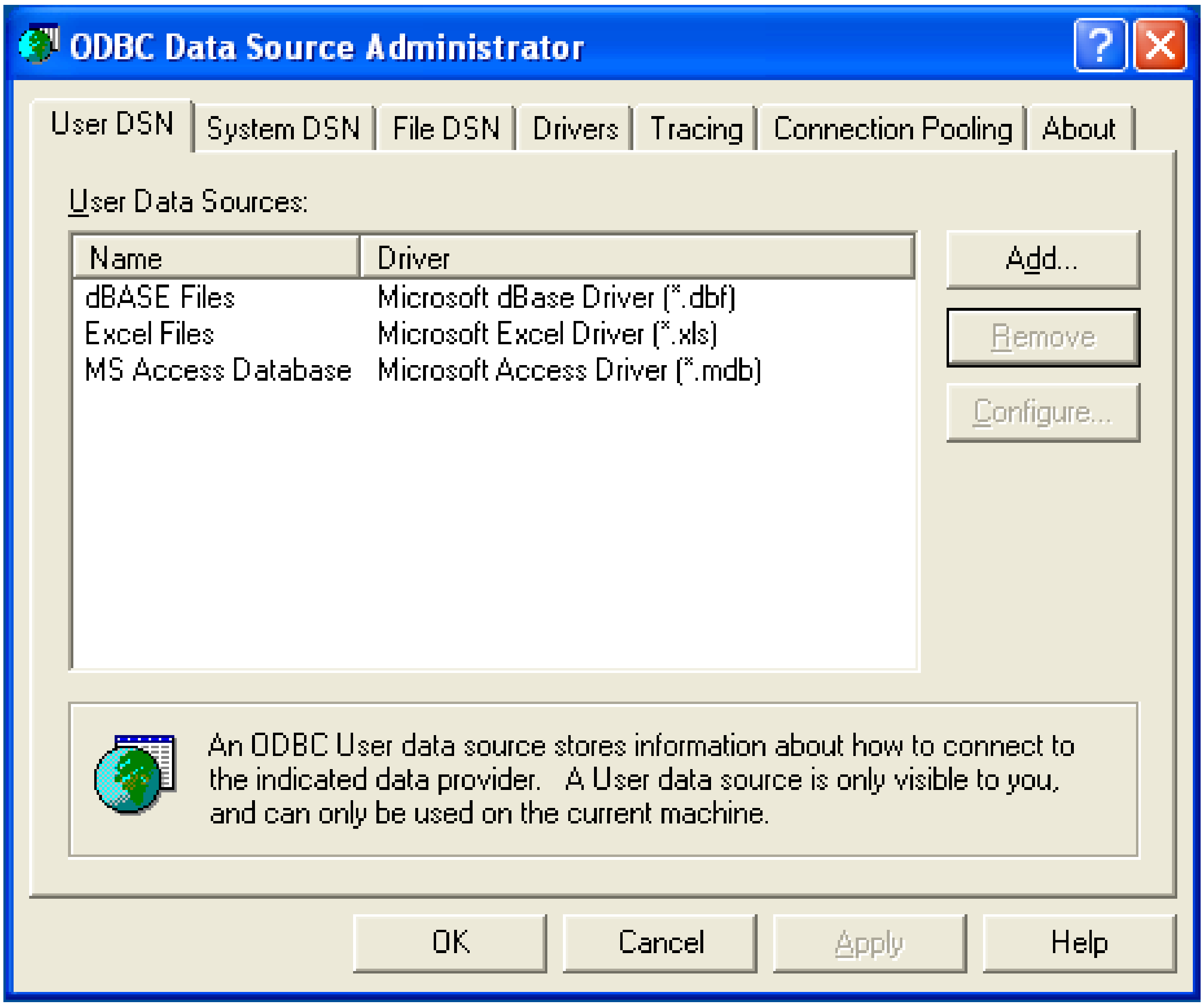
To open the `ODBC Data Source Administrator` in Windows 2000 Server or Windows 2000 Professional:

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:



Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
MS Access Database	Microsoft Access Driver (*.mdb)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

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 on 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 you may want to permanently add 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 5.1 versions of the Connector/ODBC driver are different, although the fields and information that you need to enter remain the same.

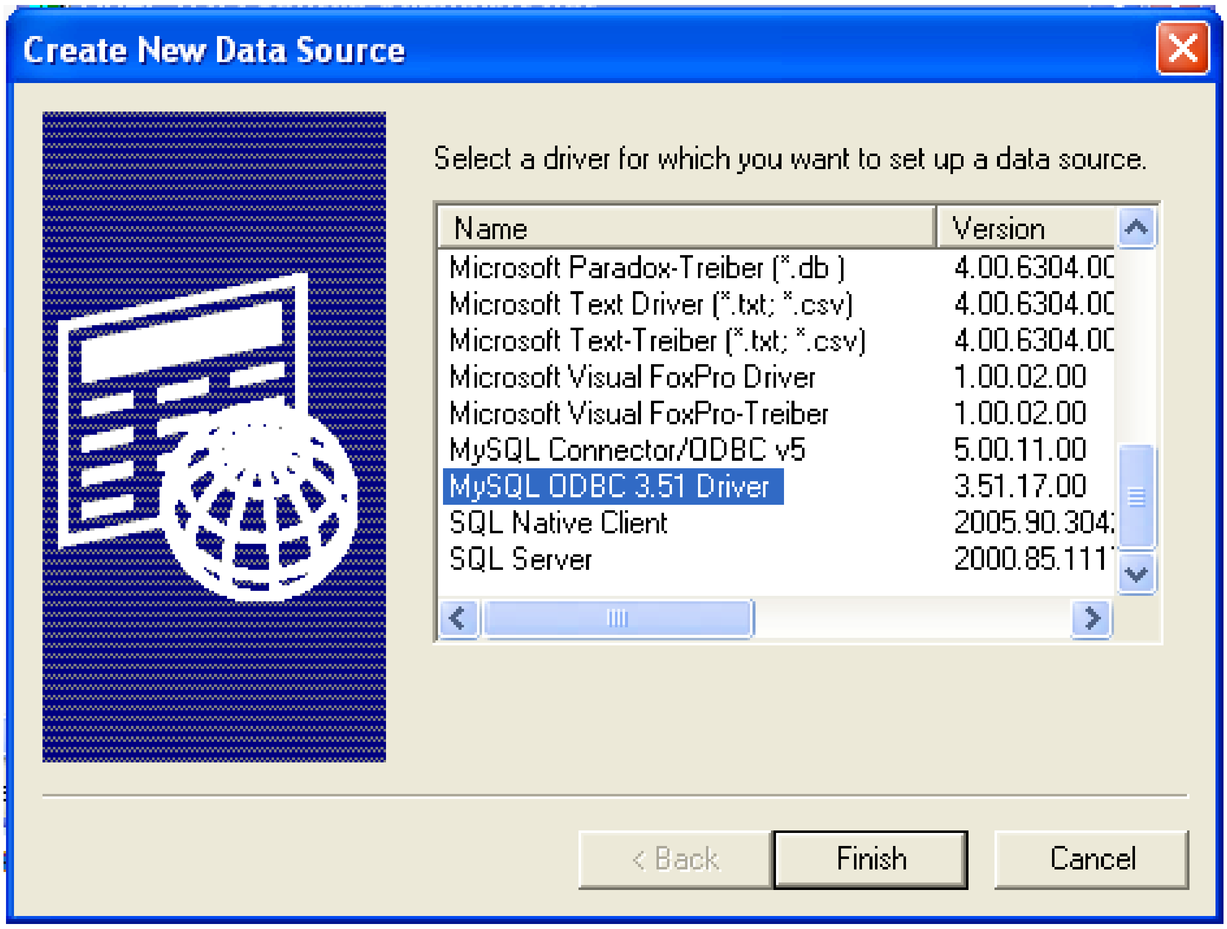
To configure a DSN using Connector/ODBC 3.51.x or Connector/ODBC 5.1.0, see Section 1.4.3.1, “Configuring a Connector/ODBC 3.51 DSN on Windows”.

To configure a DSN using Connector/ODBC 5.1.1 or later, see Section 1.4.3.2, “Configuring a Connector/ODBC 5.1 DSN on Windows”.

1.4.3.1. Configuring a Connector/ODBC 3.51 DSN on Windows

To add and configure a new Connector/ODBC data source on Windows, use the `ODBC Data Source Administrator`:

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. You will need to select the ODBC driver for this DSN.



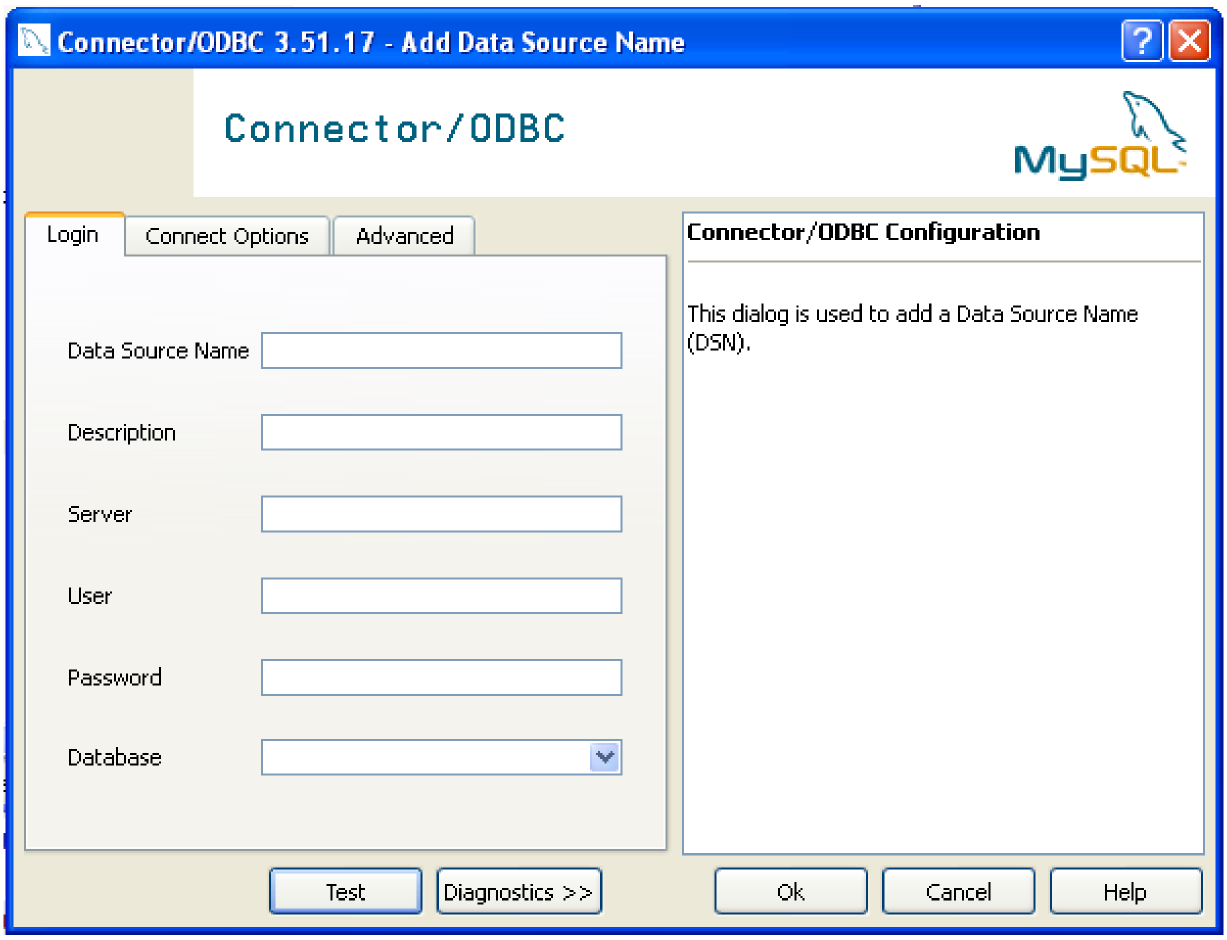
Create New Data Source

Select a driver for which you want to set up a data source.

Name	Version
Microsoft Paradox-Treiber (*.db)	4.00.6304.00
Microsoft Text Driver (*.txt; *.csv)	4.00.6304.00
Microsoft Text-Treiber (*.txt; *.csv)	4.00.6304.00
Microsoft Visual FoxPro Driver	1.00.02.00
Microsoft Visual FoxPro-Treiber	1.00.02.00
MySQL Connector/ODBC v5	5.00.11.00
MySQL ODBC 3.51 Driver	3.51.17.00
SQL Native Client	2005.90.304
SQL Server	2000.85.111

   Select `MySQL ODBC 3.51 Driver`, then click FINISH.

4. You now need to configure the specific fields for the DSN you are creating through the `Add Data Source Name` dialog.



Data Source Name
Description
Server
User
Password
Database

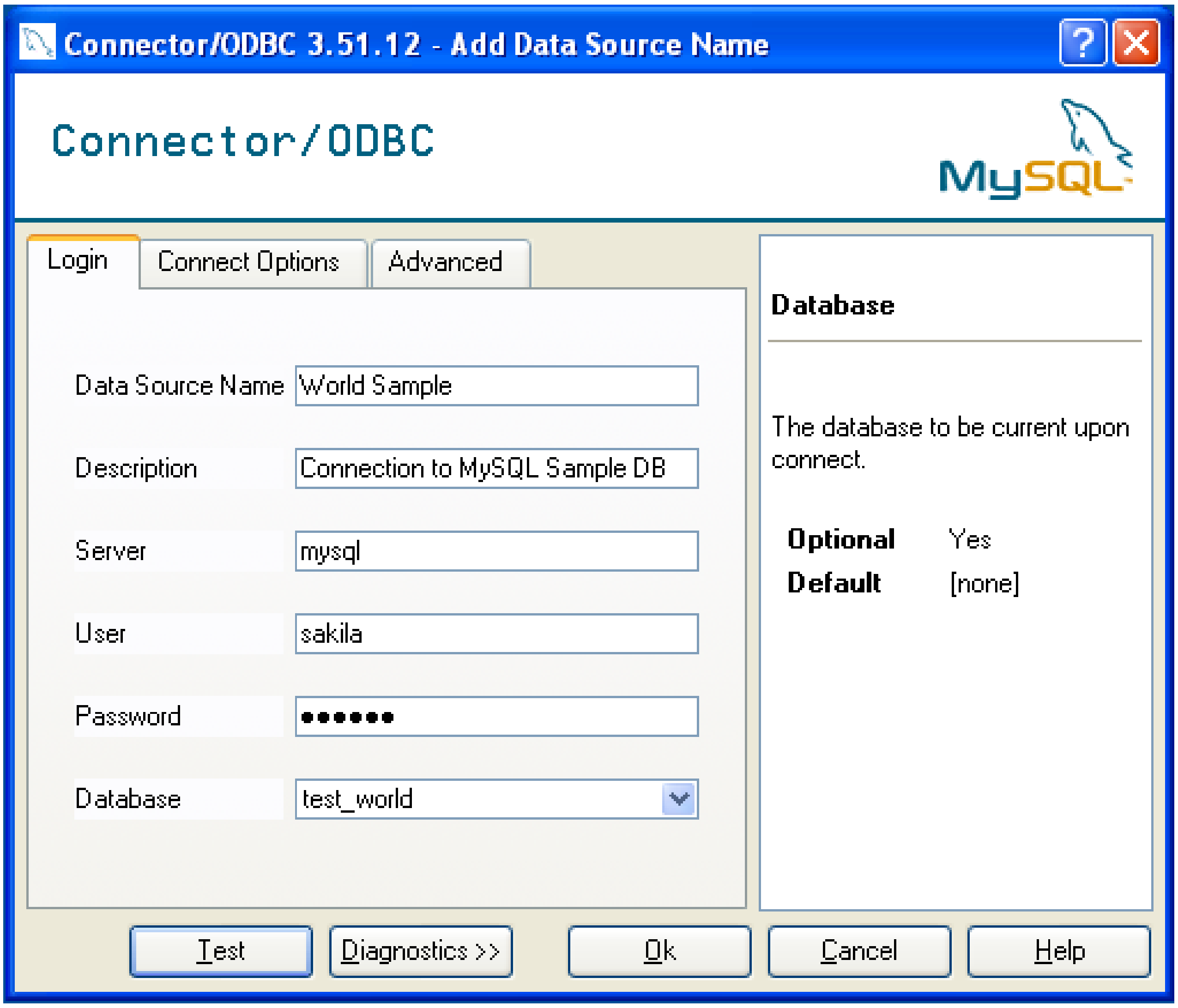
Connector/ODBC Configuration

This dialog is used to add a Data Source Name (DSN).

In the **Data Source Name** box, enter the name of the data source you want 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 that you want 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. Click OK to save the DSN.

A completed DSN configuration may look like this:



Data Source Name	World Sample
Description	Connection to MySQL Sample DB
Server	mysql
User	sakila
Password	●●●●●●
Database	test_world

Database

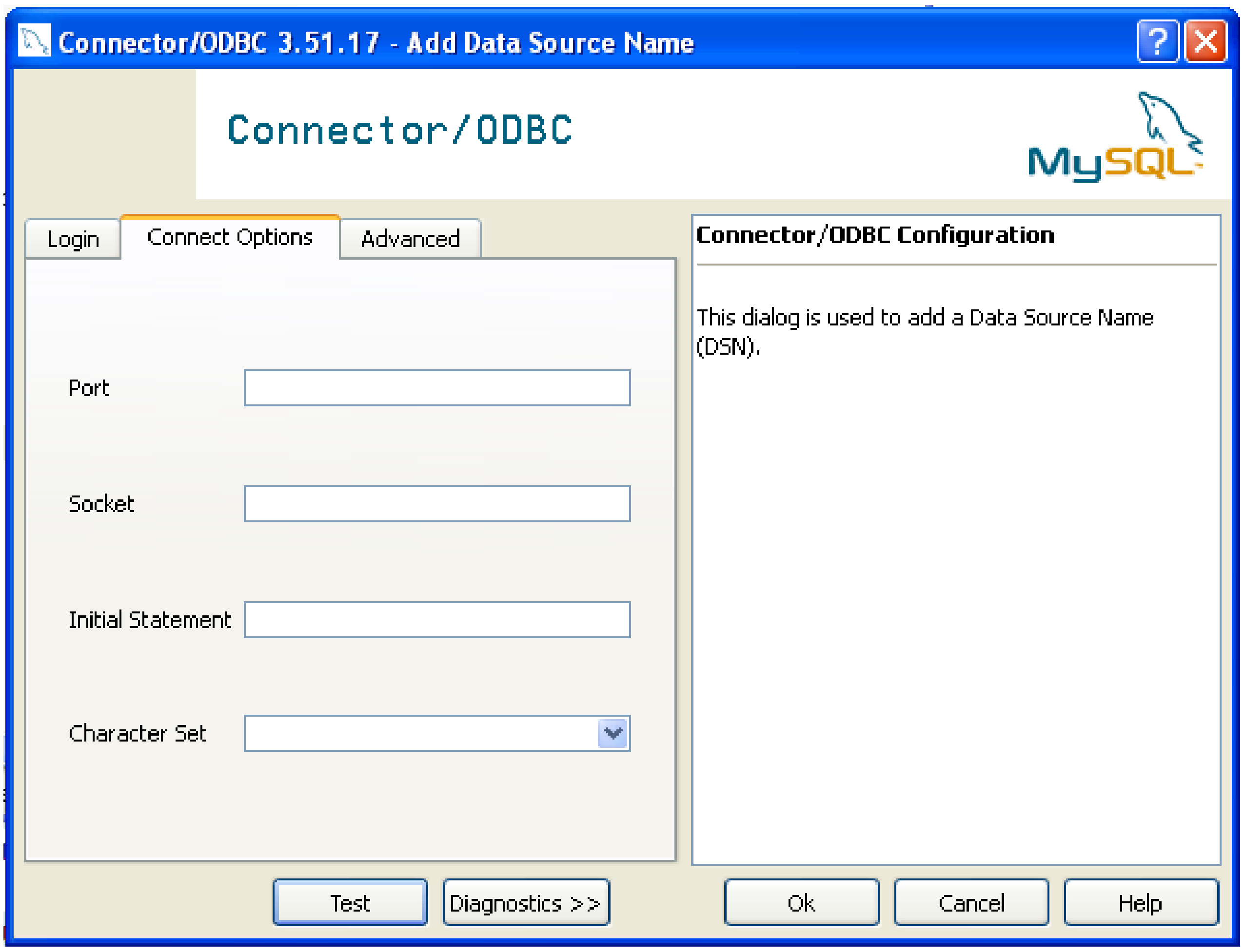
The database to be current upon connect.

Optional	Yes
Default	[none]

You can verify the connection using the parameters you have entered by clicking the TEST button. If the connection could be made successfully, you will be notified with a `Success; connection was made!` dialog.

If the connection failed, you can obtain more information on the test and why it may have failed by clicking the DIAGNOSTICS... button to show additional error messages.

You can configure a number of options for a specific DSN by using either the CONNECT OPTIONS or ADVANCED tabs in the DSN configuration dialog.



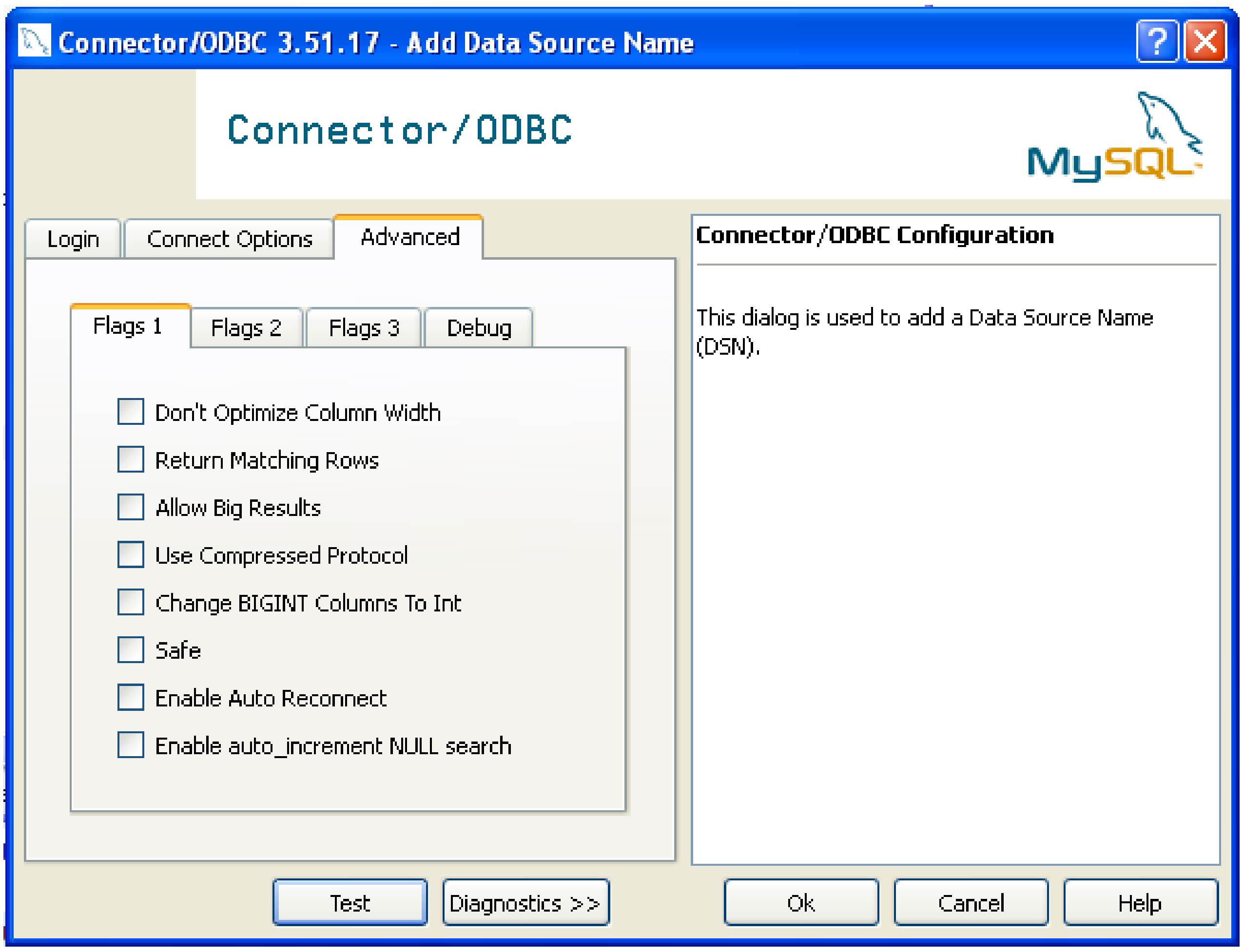
Connector/ODBC Configuration

This dialog is used to add a Data Source Name (DSN).

The three options you can configure are:

- **Port** sets the TCP/IP port number to use when communicating with MySQL. Communication with MySQL uses port 3306 by default. If your server is configured to use a different TCP/IP port, you must specify that port number here.
- **Socket** sets the name or location of a specific socket or Windows pipe to use when communicating with MySQL.
- **Initial Statement** defines an SQL statement that will be executed when the connection to MySQL is opened. You can use this to set MySQL options for your connection, such as disabling autocommit.
- **Character Set** is a pop-up list from which you can select the default character set to be used with this connection. The Character Set option was added in 3.5.17.

The Advanced tab enables you to configure Connector/ODBC connection parameters. Refer to Section 1.4.2, "Connector/ODBC Connection Parameters", for information about the meaning of these options.


Connector/ODBC 3.51.17 - Add Data Source Name

Connector/ODBC

Login | Connect Options | Advanced

Flags 1 | Flags 2 | Flags 3 | Debug

- [ ] Don't Optimize Column Width
- [ ] Return Matching Rows
- [ ] Allow Big Results
- [ ] Use Compressed Protocol
- [ ] Change BIGINT Columns To Int
- [ ] Safe
- [ ] Enable Auto Reconnect
- [ ] Enable auto_increment NULL search

Connector/ODBC Configuration

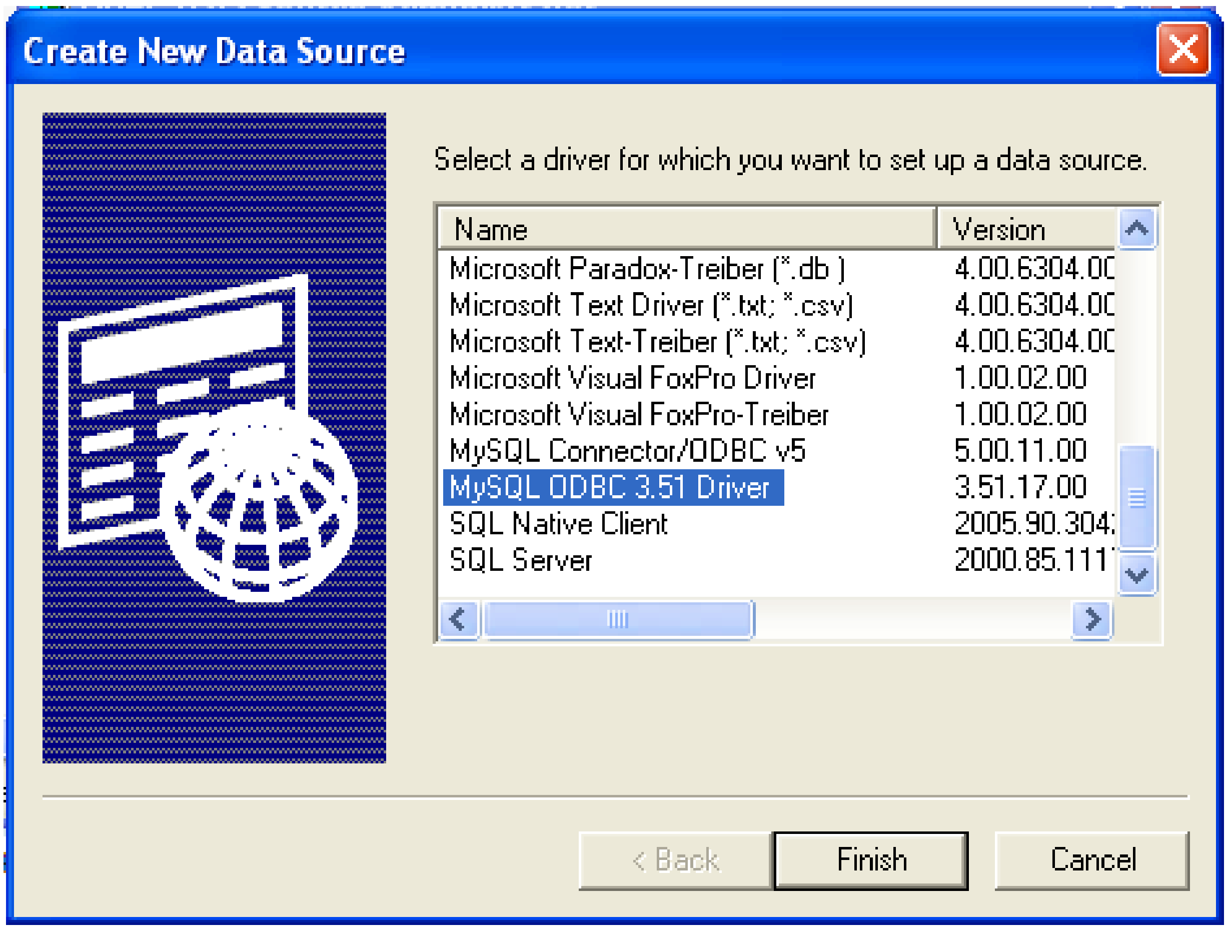
This dialog is used to add a Data Source Name (DSN).

1.4.3.2. Configuring a Connector/ODBC 5.1 DSN on Windows

The DSN configuration when using Connector/ODBC 5.1.1 and later has a slightly different layout. Also, due to the native Unicode support within Connector/ODBC 5.1, you no longer need to specify the initial character set to be used with your connection.

To configure a DSN using the Connector/ODBC 5.1.1 or later driver:

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. You will need to select the ODBC driver for this DSN.

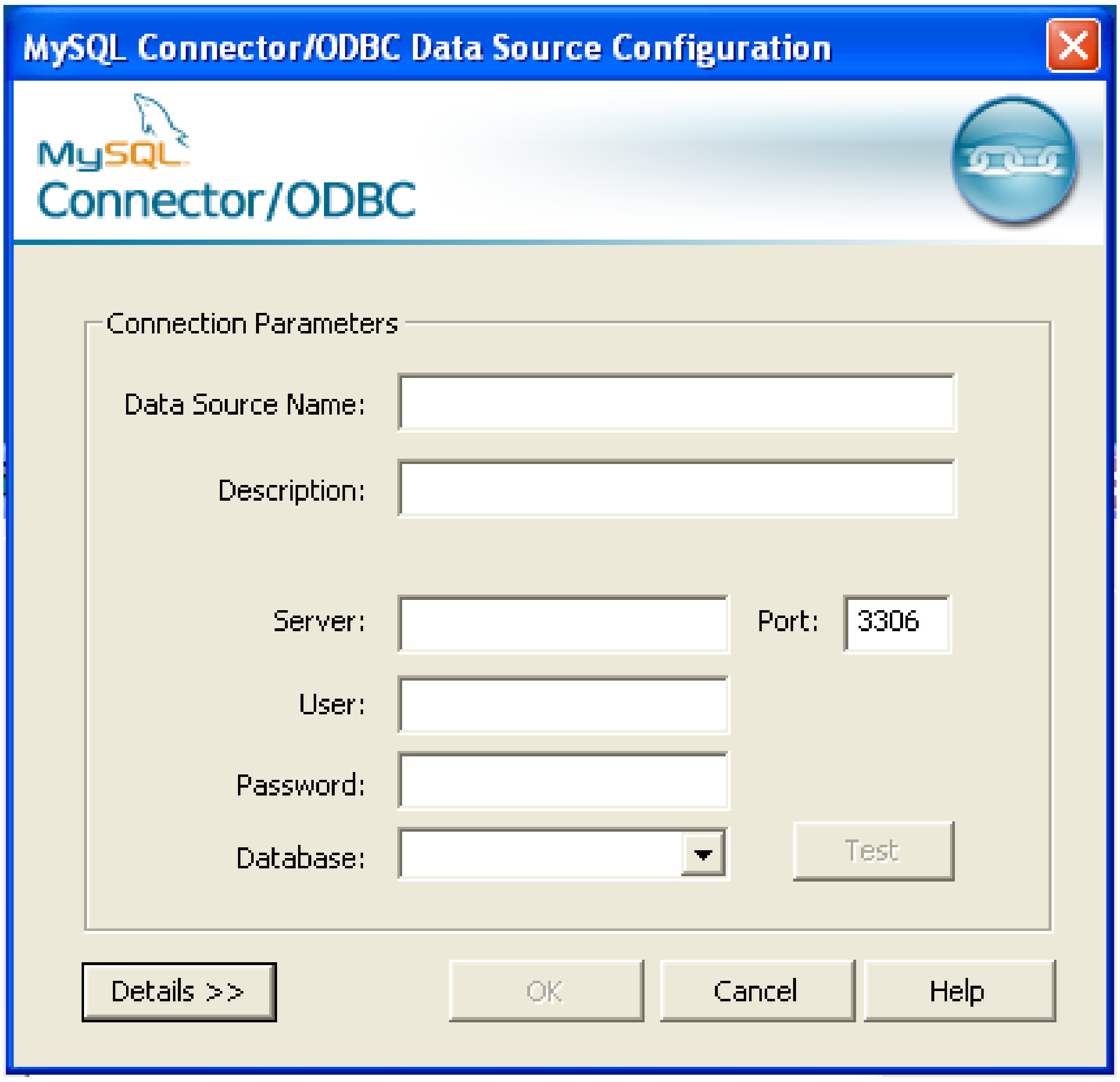


Select a driver for which you want to set up a data source.

Name	Version
Microsoft Paradox-Treiber (*.db)	4.00.6304.0C
Microsoft Text Driver (*.txt; *.csv)	4.00.6304.0C
Microsoft Text-Treiber (*.txt; *.csv)	4.00.6304.0C
Microsoft Visual FoxPro Driver	1.00.02.00
Microsoft Visual FoxPro-Treiber	1.00.02.00
MySQL Connector/ODBC v5	5.00.11.00
MySQL ODBC 3.51 Driver	3.51.17.00
SQL Native Client	2005.90.304
SQL Server	2000.85.111

Select `MySQL ODBC 5.1 Driver`, then click FINISH.

4. You now need to configure the specific fields for the DSN you are creating through the `Connection Parameters` dialog.



In the **Data Source Name** box, enter the name of the data source you want 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 that you want 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.

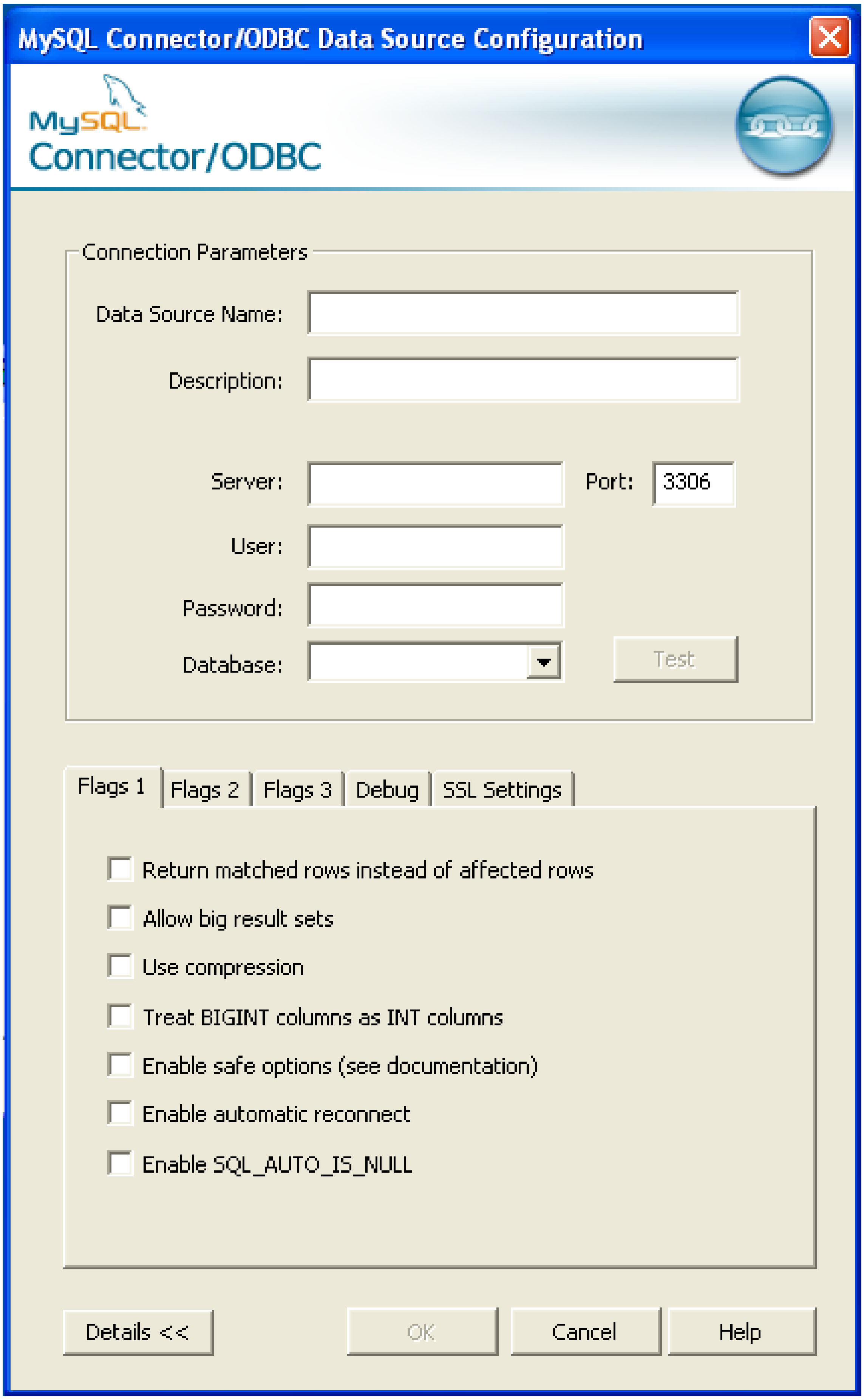
9. The **Database** pop-up should automatically populate with the list of databases that the user has permissions to access.

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.

You can verify the connection using the parameters you have entered by clicking 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.



MySQL Connector/ODBC Data Source Configuration

MySQL Connector/ODBC

Connection Parameters

Data Source Name:			
Description:			
Server:		Port:	3306
User:			
Password:			
Database:			Test

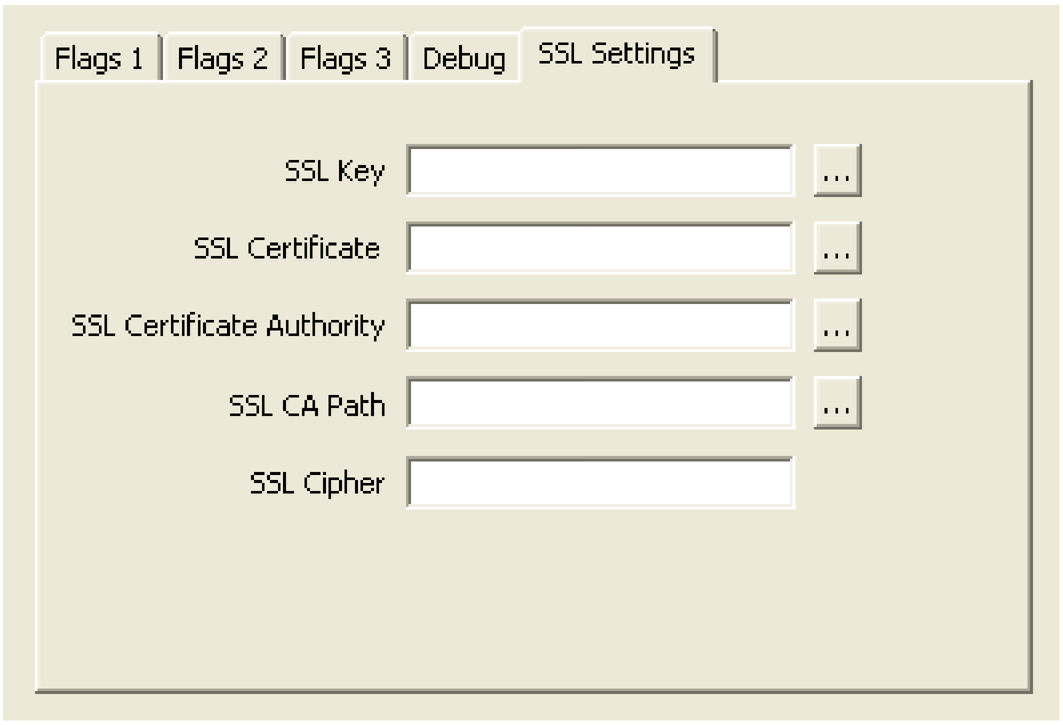
Flags 1 | Flags 2 | Flags 3 | Debug | SSL Settings

- [ ] 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

Details << | OK | Cancel | Help

The **DETAILS** button opens a tabbed display which allows you to 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 1.4.2, "Connector/ODBC Connection Parameters".
- **DEBUG** allows you to enable ODBC debugging to record the queries you execute through the DSN to the `myodbc.sql` file. For more information, see Section 1.4.8, "Getting an ODBC Trace File".
- **SSL SETTINGS** configures the additional options required for using the Secure Sockets Layer (SSL) when communicating with MySQL server. Note that you must have enabled SSL and configured the MySQL server with suitable certificates to communicate over SSL.



The ADVANCED tab enables you to configure Connector/ODBC connection parameters. Refer to Section 1.4.2, "Connector/ODBC Connection Parameters", for information about the meaning of these options.

1.4.3.3. Errors and Debugging

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 `ctl3d32.dll` in your system directory.

- On Windows, the default `myodbc3.dll` is compiled for optimal performance. If you want to debug Connector/ODBC 3.51 (for example, to enable tracing), you should instead use `myodbc3d.dll`. To install this file, copy `myodbc3d.dll` over the installed `myodbc3.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. Note that the `myodbc3d.dll` isn't included in Connector/ODBC 3.51.07 through 3.51.11. If you are using one of these versions, you should copy that DLL from a previous version (for example, 3.51.06).

1.4.4. Configuring a Connector/ODBC DSN on Mac OS X

To configure a DSN on Mac OS X you can either use the `myodbc3i` utility, edit the `odbc.ini` file within the `Library/ODBC` directory of the user or the should use the ODBC Administrator. If you have Mac OS X 10.2 or earlier, refer to Section 1.4.5, "Configuring a Connector/ODBC DSN on Unix". Select whether you want to create a User DSN or a System DSN. If you want to add a System DSN, you may need to authenticate with the system. You must click the padlock and enter a user and password with administrator privileges.

For correct operation of ODBC Administrator, you should 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 have worked but not generated the correct entry.

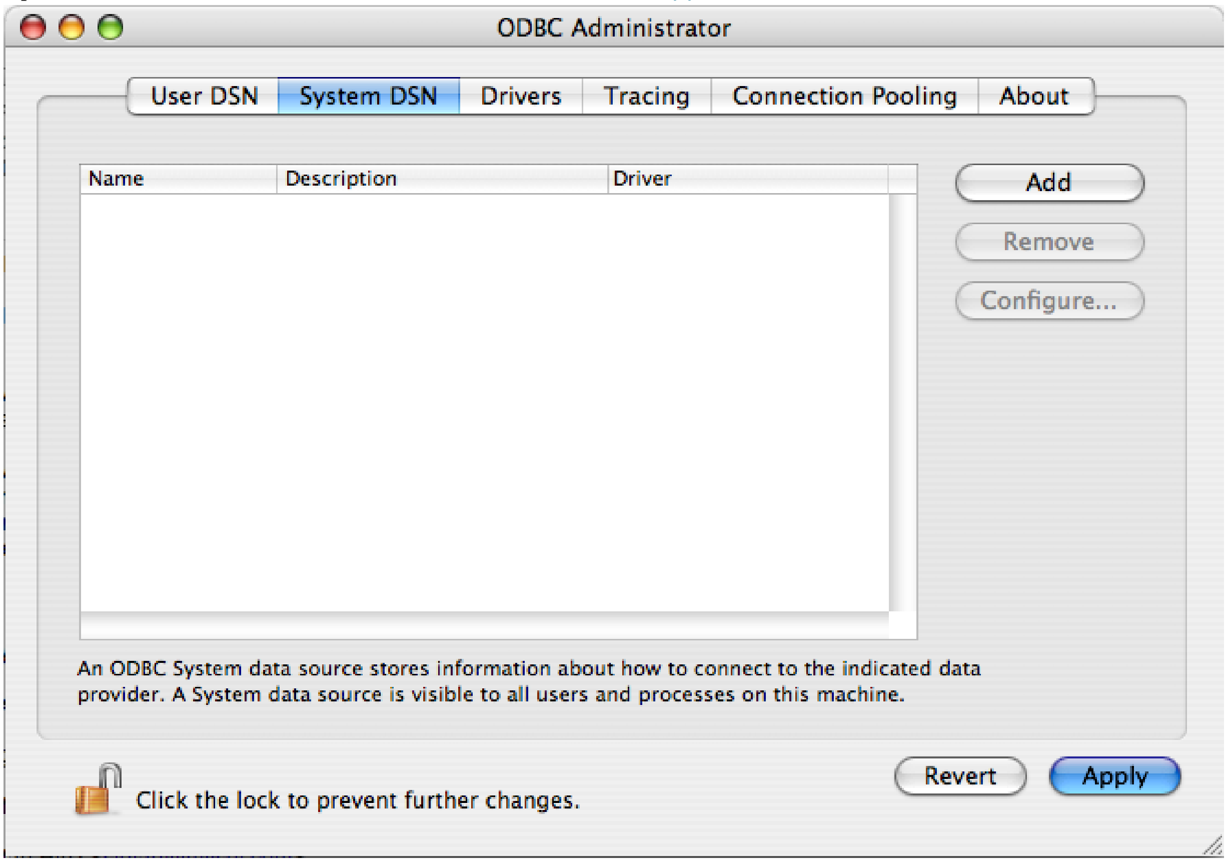
> **Warning**
>
> There are known issues with the OS X ODBC Administrator and Connector/ODBC that may prevent you from creating a DSN using this method. In this case you should use the command-line or edit the `odbc.ini` file directly. Note that existing DSNs or those that you create via the `myodbc3i` or `myodbc-installer`tool can still be checked and edited using ODBC Administrator.

To create a DSN using the `myodbc3i` utility, you need only specify the DSN type and the DSN connection string. For example:

```
shell> myodbc3i -a -s -t"DSN=mydb;DRIVER=MySQL ODBC 3.51 Driver;SERVER=mysql;USER=username;PASSWORD=pass"
```

To use ODBC Administrator:

1. Open the ODBC Administrator from the `Utilities` folder in the `Applications` folder.



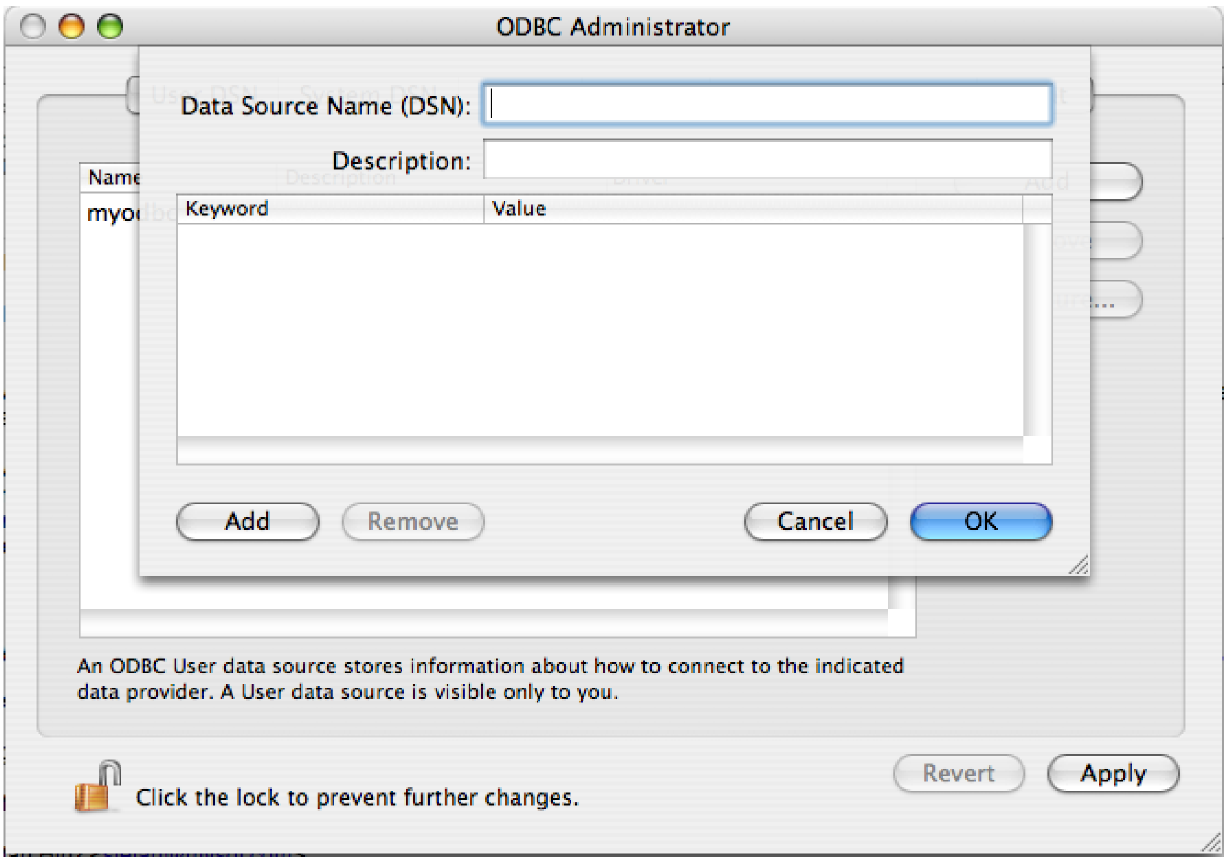
An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users and processes on this machine.

Click the lock to prevent further changes.

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.



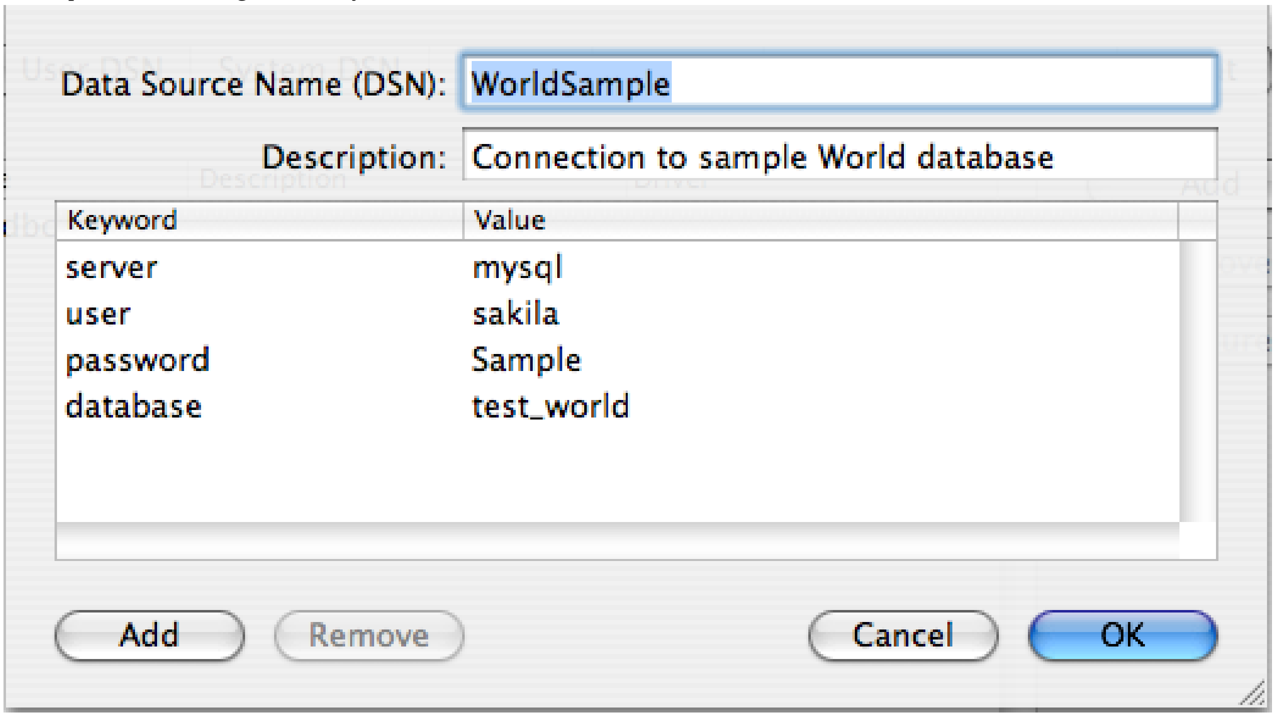
An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is visible only to you.

Click the lock to prevent further changes.

5. Click ADD to add a new keyword/value pair to the panel. You should configure at least four pairs to specify the `server`, `username`, `password` and `database` connection parameters. See Section 1.4.2, “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

You can configure additional ODBC options to your DSN by adding further keyword/value pairs and setting the corresponding values. See Section 1.4.2, "Connector/ODBC Connection Parameters".

1.4.5. 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 1.4.2, "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
```

1.4.6. 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 3.51:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\
                   SERVER=localhost;\
                   DATABASE=test;\
                   USER=venu;\
                   PASSWORD=venu;\
                   OPTION=3;"
```

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 3.51 Driver};"
                   "SERVER=localhost;"
                   "DATABASE=test;"
                   "USER=venu;"
                   "PASSWORD=venu;"
```

```
                  "OPTION=3;"
```

Note. Note that on Mac OS X you may need to specify the full path to the Connector/ODBC driver library.

Refer to the Section 1.4.2, "Connector/ODBC Connection Parameters", for the list of connection parameters that can be supplied.

1.4.7. 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.

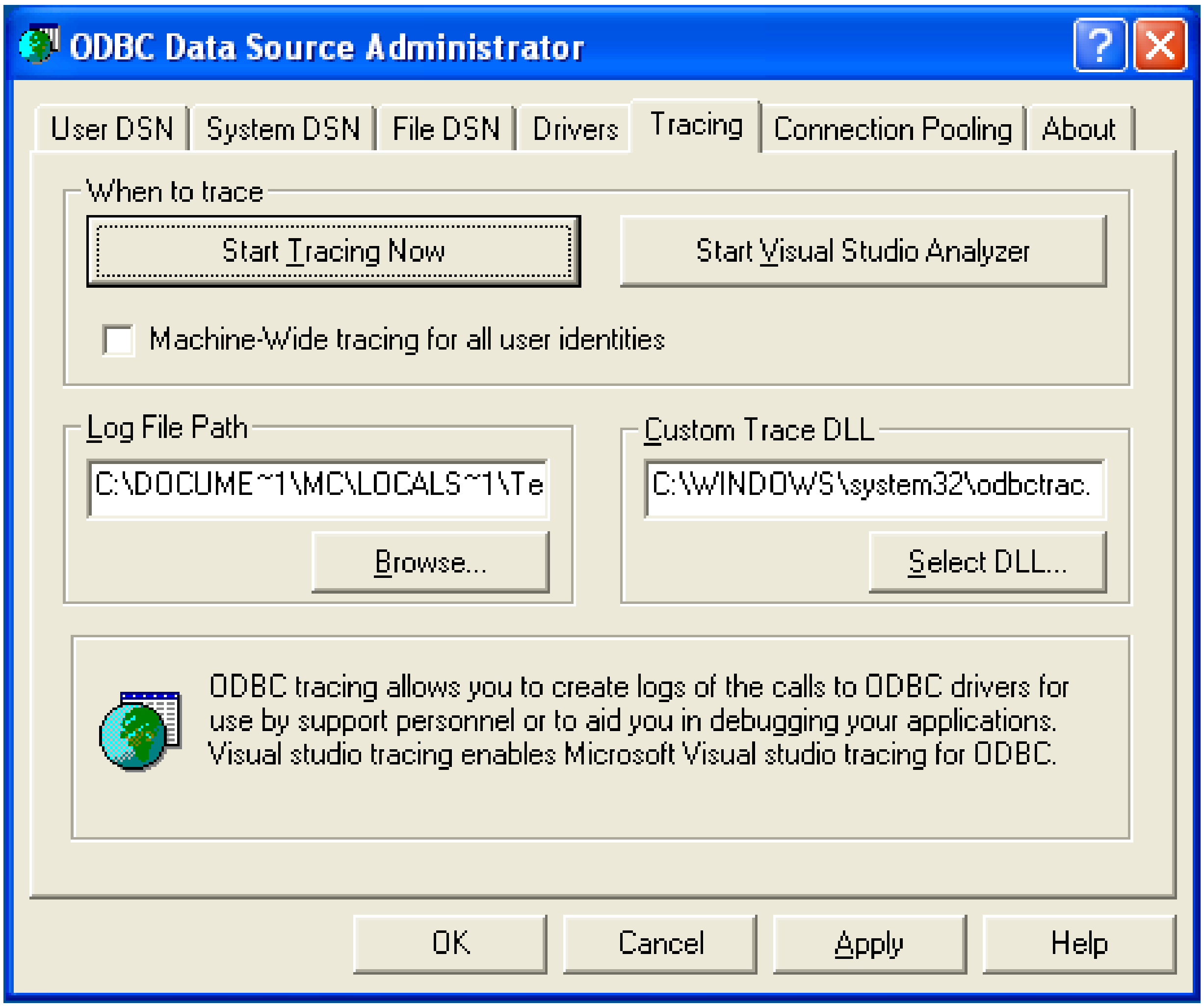
1.4.8. Getting an ODBC Trace File

If you encounter difficulties or problems with Connector/ODBC, you should 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, Mac OS X and Unix.

1.4.8.1. Enabling ODBC Tracing on Windows

To enable the trace option on Windows:

1. The `Tracing` tab of the ODBC Data Source Administrator dialog box enables you to configure the way ODBC function calls are traced.



Machine-Wide tracing for all user identities: [ ]

Log File Path: C:\DOCUME~1\MC\LOCALS~1\Te

Custom Trace DLL: C:\WINDOWS\system32\odbctrac.

ODBC tracing allows you to create logs of the calls to ODBC drivers for use by support personnel or to aid you in debugging your applications. Visual studio tracing enables Microsoft Visual studio tracing for ODBC.

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.

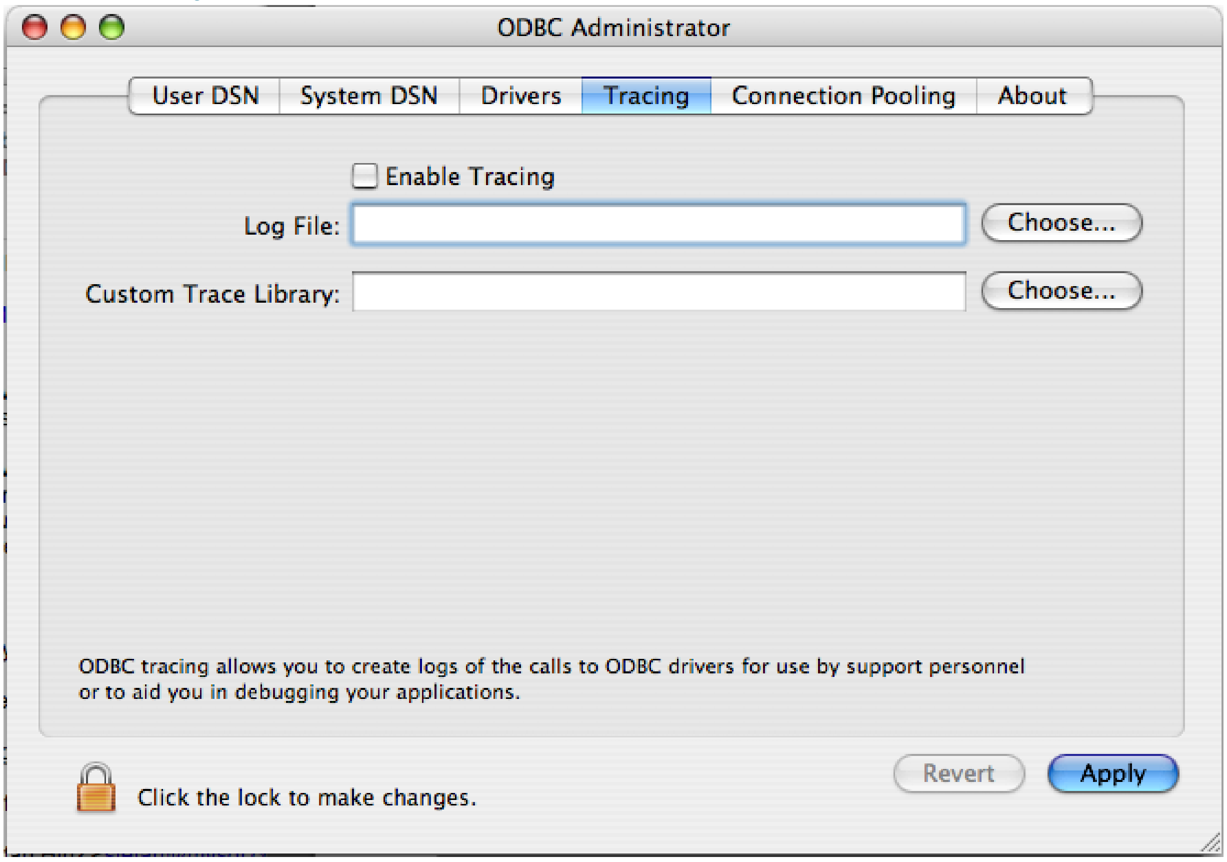
4. Tracing ceases only after you click `Stop Tracing Now`. Remember that while tracing is on, the log file continues to increase in size and that tracing affects the performance of all your ODBC applications.

1.4.8.2. Enabling ODBC Tracing on Mac OS X

To enable the trace option on Mac OS X 10.3 or later you should use the `Tracing` tab within ODBC Administrator .

1. Open the ODBC Administrator.

2. Select the `Tracing` tab.



ODBC tracing allows you to create logs of the calls to ODBC drivers for use by support personnel or to aid you in debugging your applications.

3. Select the `Enable Tracing` checkbox.

4. Enter the location where you want to save the Tracing log. If you want to append information to an existing log file, click the CHOOSE... button.

1.4.8.3. Enabling ODBC Tracing on Unix

To enable the trace option on Mac OS X 10.2 (or earlier) or Unix you must add the `trace` option to the ODBC configuration:

1. On Unix, you need to 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.

1.4.8.4. 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 1.4.3.3, “Errors and Debugging”.

   On Mac OS X, Unix, or if you are using DSN-Less connection, then you need to 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 1.8.1, “Connector/ODBC Community Support”.

1.5. Connector/ODBC Examples

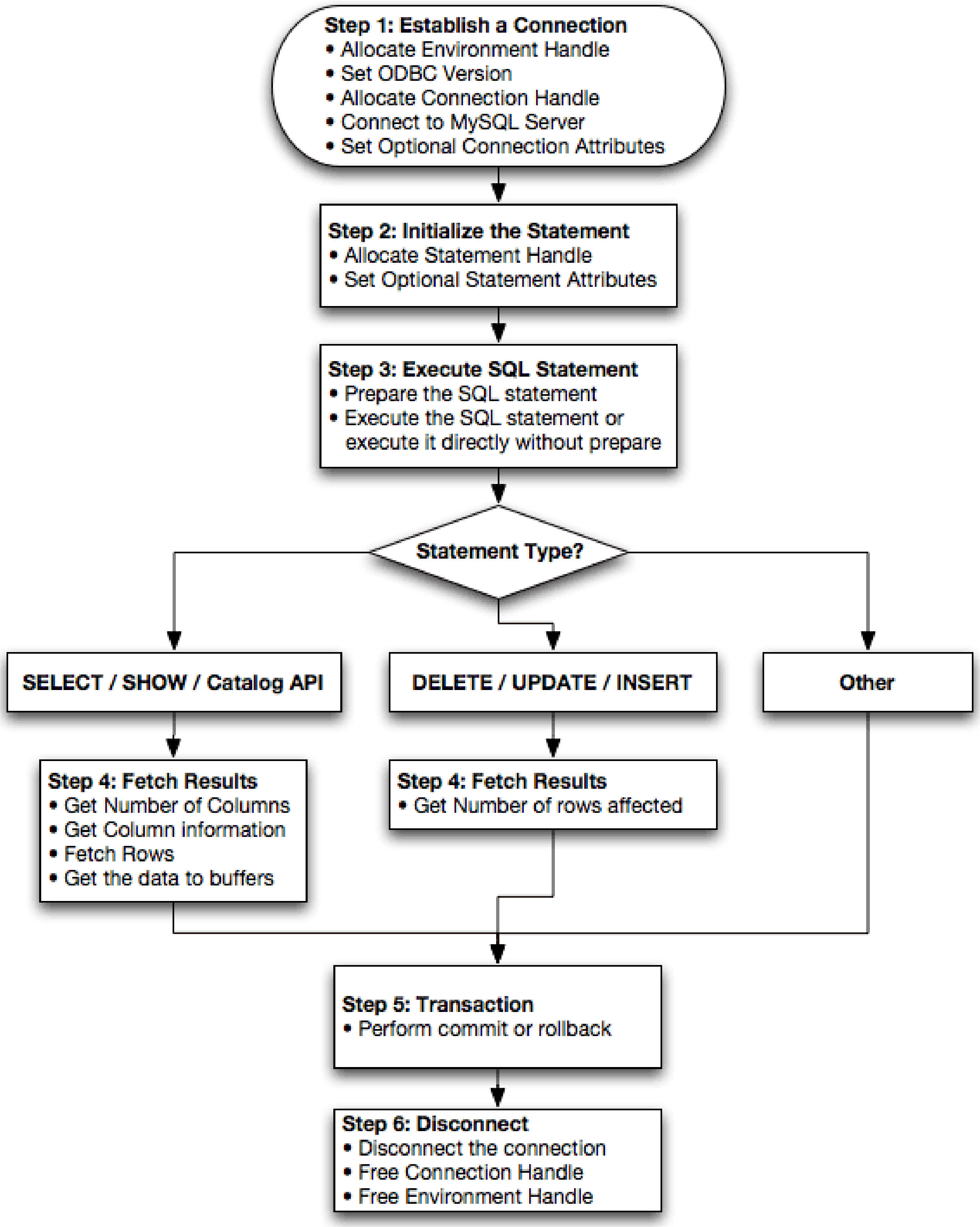
Once you have configured a DSN to provide access to a database, how you access and use that connection is dependent on the application or programming language. As ODBC is a standardized interface, any application or language that supports ODBC can use the DSN and connect to the configured database.

1.5.1. 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:


Step 1: Establish a Connection
• Allocate Environment Handle
• Set ODBC Version
• Allocate Connection Handle
• Connect to MySQL Server
• Set Optional Connection Attributes
Step 2: Initialize the Statement
• Allocate Statement Handle
• Set Optional Statement Attributes
Step 3: Execute SQL Statement
• Prepare the SQL statement
• Execute the SQL statement or execute it directly without prepare
Statement Type?
SELECT / SHOW / Catalog API
DELETE / UPDATE / INSERT
Other
Step 4: Fetch Results
• Get Number of Columns
• Get Column information
• Fetch Rows
• Get the data to buffers
Step 4: Fetch Results
• Get Number of rows affected
Step 5: Transaction
• Perform commit or rollback
Step 6: Disconnect
• Disconnect the connection
• Free Connection Handle
• Free Environment Handle

1.5.2. Step-by-step Guide to Connecting to a MySQL Database through Connector/ODBC

A typical installation situation where you would install Connector/ODBC is when you want 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 want to 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 `myuser`.

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 1.4.8, “Getting an ODBC Trace File”, for more information.

1.5.3. 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	
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.

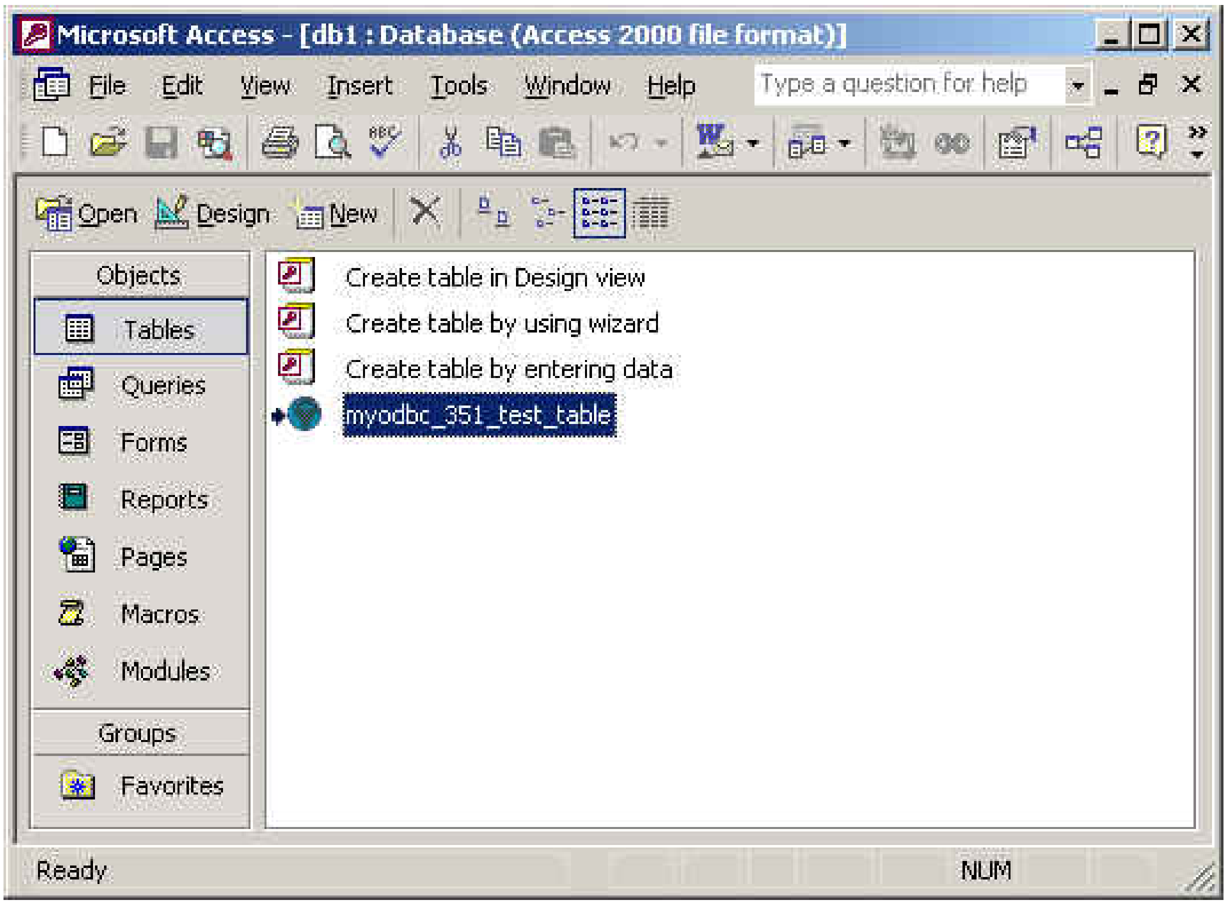
1.5.4. Using Connector/ODBC with Microsoft Access

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.

1.5.4.1. Exporting Access Data to MySQL

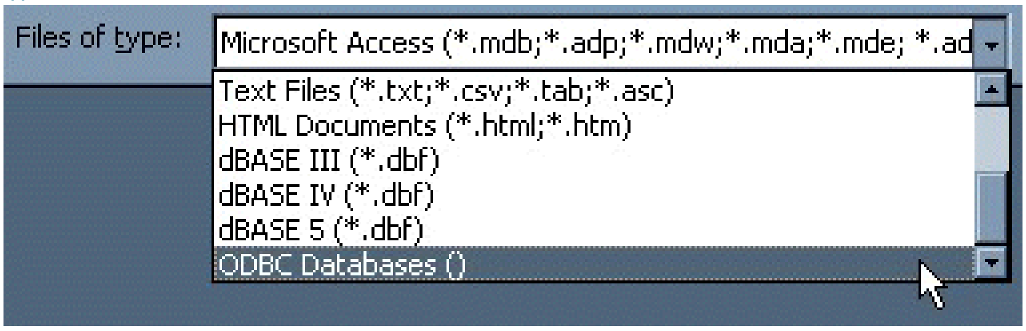
To export a table of data from an Access database to MySQL, follow these instructions:

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` you want 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:


Files of type:
Microsoft Access (*.mdb;*.adp;*.mdw;*.mda;*.mde; *.ad
Text Files (*.txt;*.csv;*.tab;*.asc)
HTML Documents (*.html;*.htm)
dBASE III (*.dbf)
dBASE IV (*.dbf)
dBASE 5 (*.dbf)
ODBC Databases ()

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 that you want to export to. To define a new data source for Connector/ODBC, please Section 1.4.3, "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.

1.5.4.2. Importing MySQL Data to Access

To import a table or tables from MySQL to Access, follow these instructions:

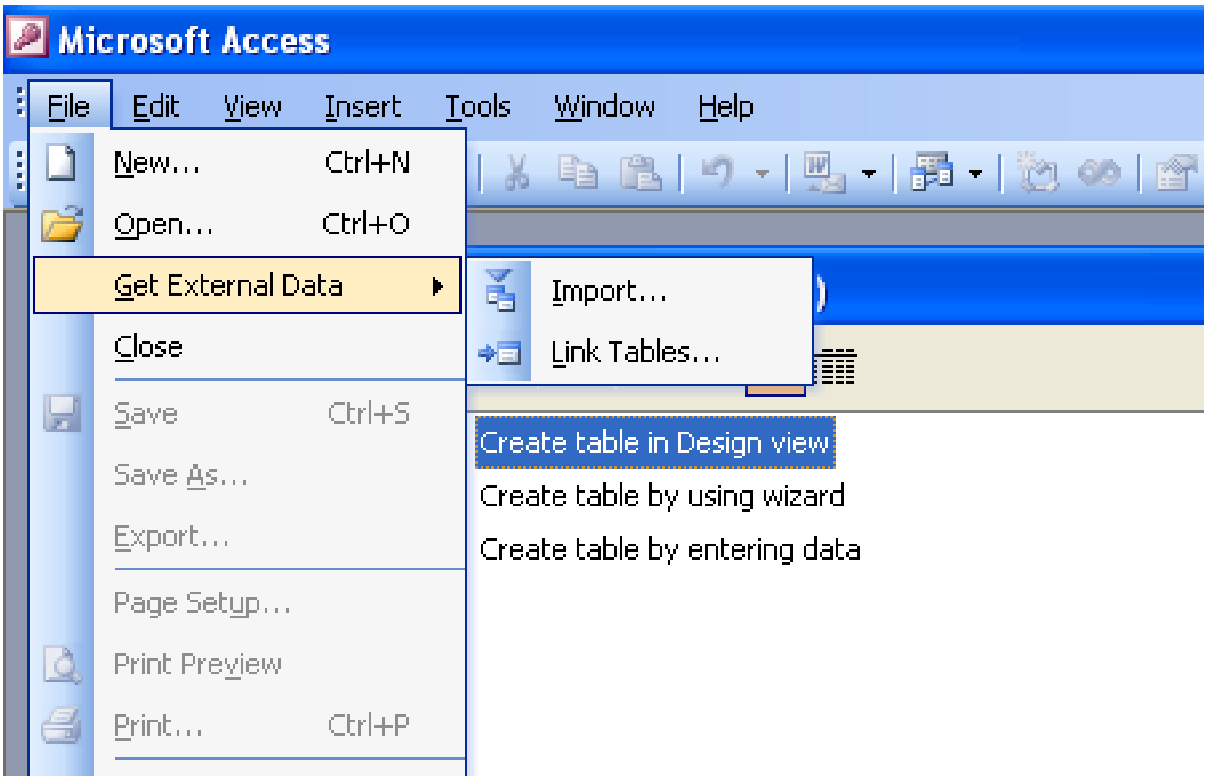
1. Open a database, or switch to the Database window for the open database.
2. To import tables, on the `File` menu, point to `Get External Data`, and then click `Import`.
3. In the `Import` dialog box, in the Files Of Type box, select **ODBC Databases ()**. The Select Data Source dialog box lists the 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 that you want to `import`, and then click `OK`.

1.5.4.3. Using Microsoft Access as a Front-end to MySQL

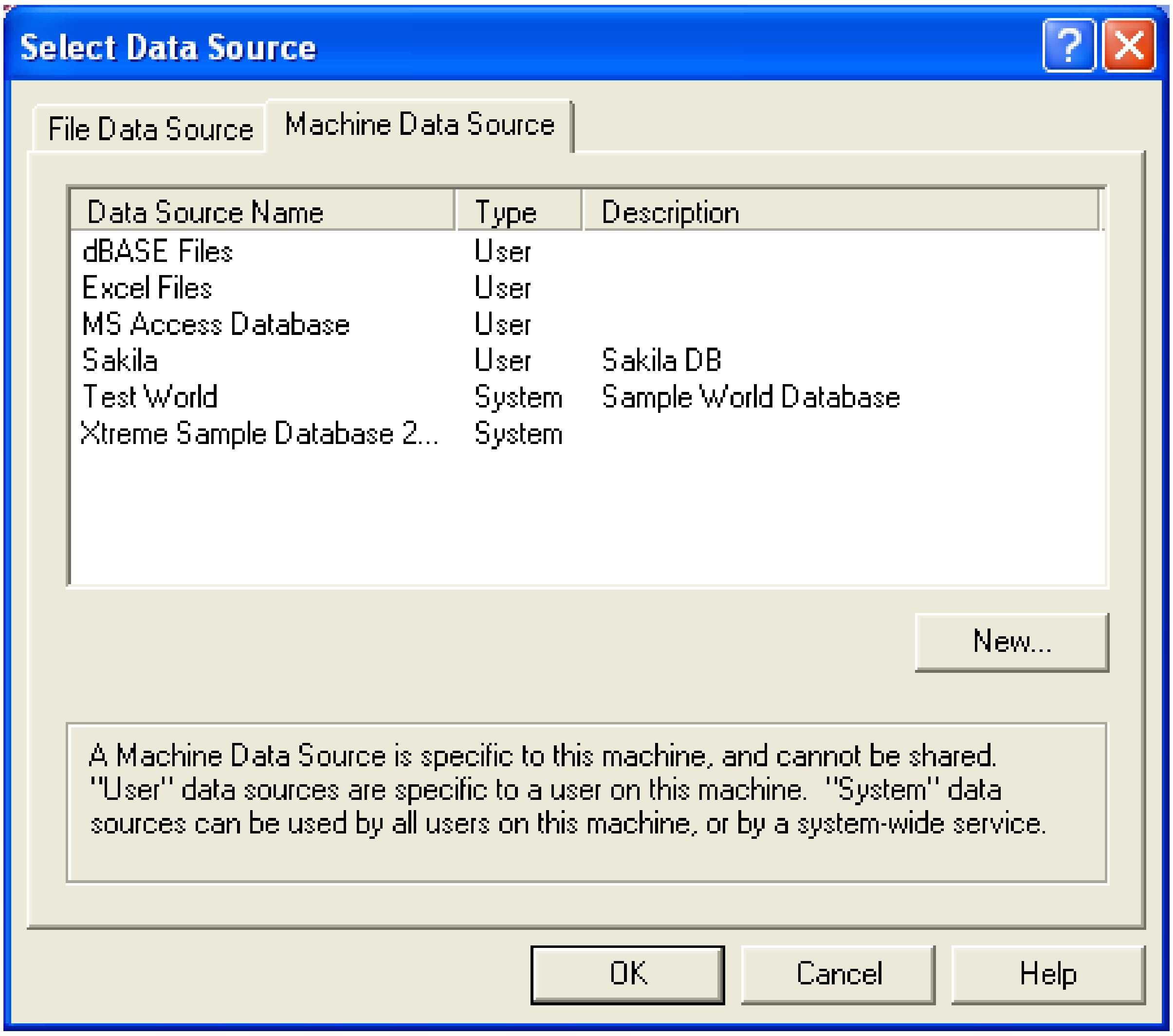
You can use Microsoft Access as a front end to a MySQL database by linking tables within your Microsoft Access database to tables that exist within your MySQL database. When a query is requested on a table within Access, ODBC is used to execute the queries on the MySQL database instead.

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.



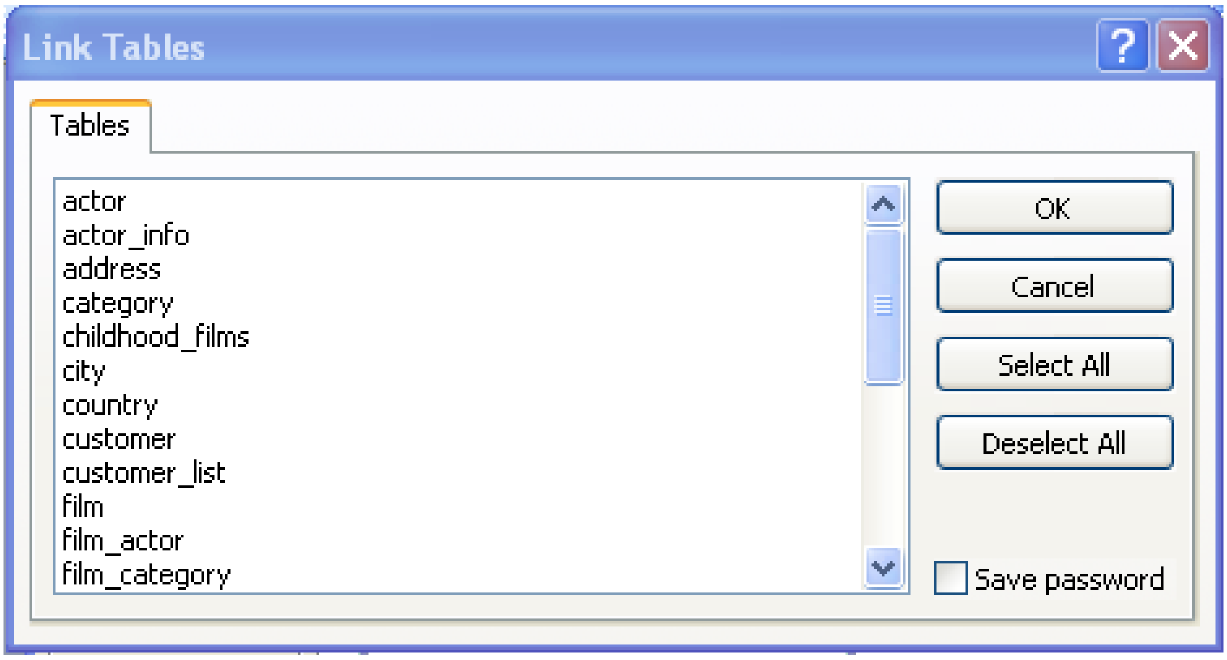
3. From the browser, choose **ODBC Databases ()** from the **Files of type** pop-up.
4. In the **Select Data Source** window, choose an existing DSN, either from a **File Data Source** or **Machine Data Source**.You can also create a new DSN using the New... button. For more information on creating a DSN see Section 1.4.3, “Configuring a Connector/ODBC DSN on Windows”.



Data Source Name	Type	Description
dBASE Files	User	
Excel Files	User	
MS Access Database	User	
Sakila	User	Sakila DB
Test World	System	Sample World Database
Xtreme Sample Database 2...	System	

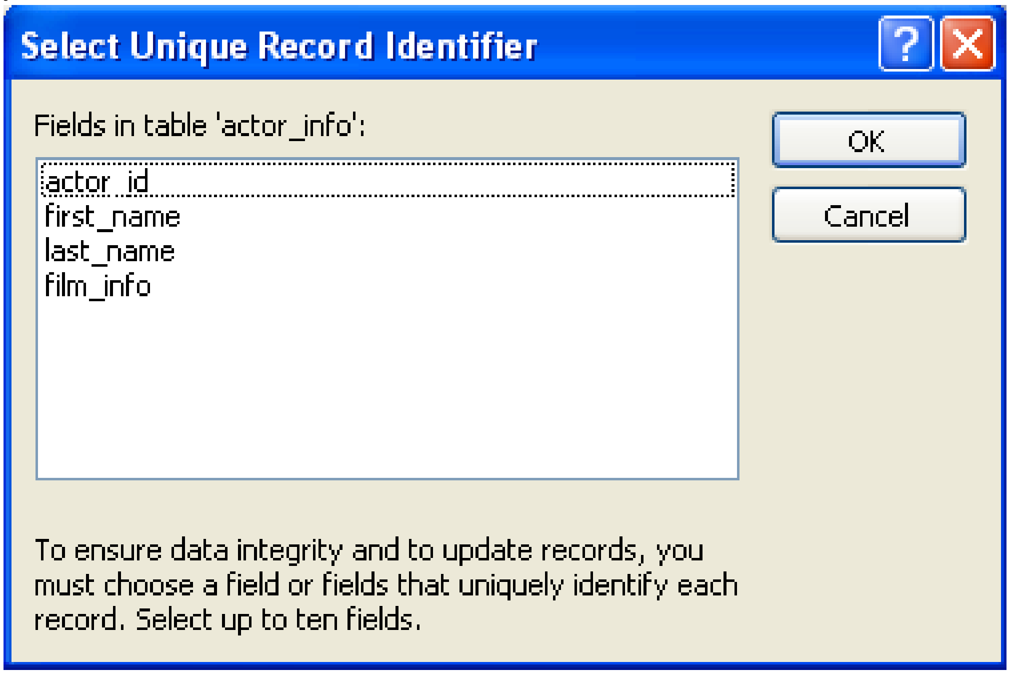
A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

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.



actor
actor_info
address
category
childhood_films
city
country
customer
customer_list
film
film_actor
film_category

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 you want to use and click OK.



Fields in table 'actor_info':

actor_id
first_name
last_name
film_info

To ensure data integrity and to update records, you must choose a field or fields that uniquely identify each record. Select up to ten fields.

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`.

1.5.5. Using Connector/ODBC with Microsoft Word or Excel

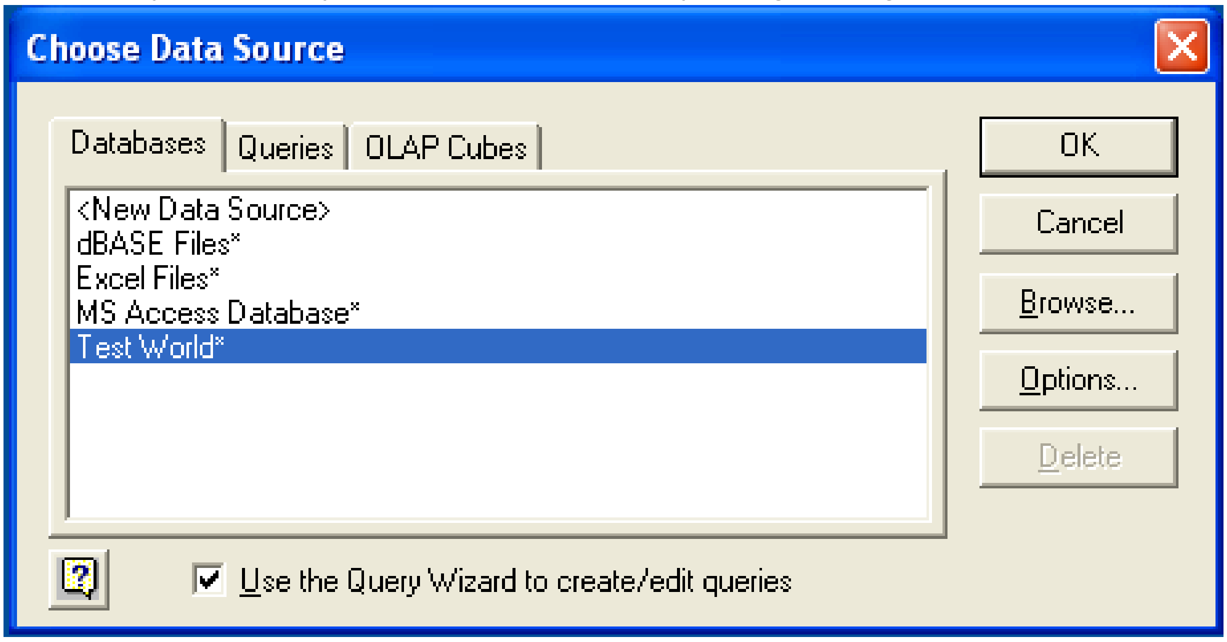
You can use Microsoft Word and Microsoft Excel to access information from a MySQL database using Connector/ODBC. Within Microsoft Word, this facility is most useful when importing data for mailmerge, or for tables and data to be included in reports. Within Microsoft Excel, you can execute queries on 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 enables you to execute a query though 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 1.4, "Connector/ODBC Configuration":

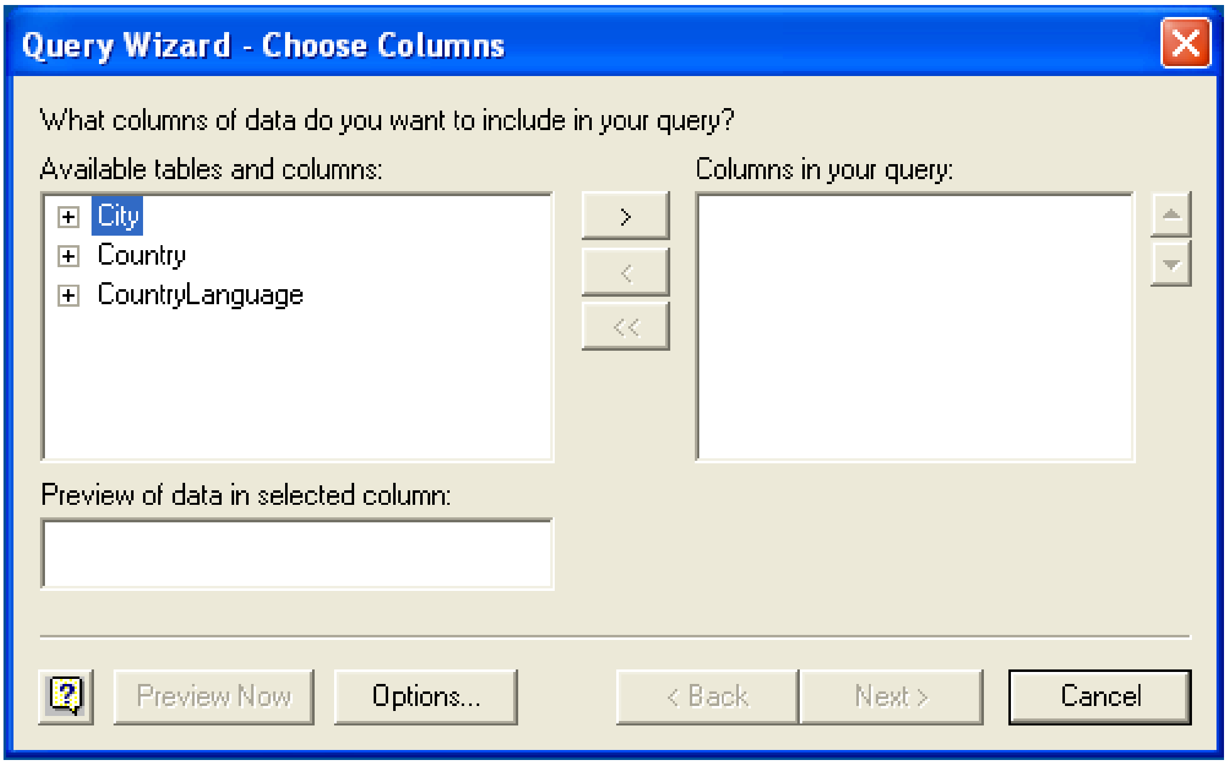
1. Create a new Worksheet.

2. From the `Data` menu, choose `Import External Data`, and then select `New Database Query`.

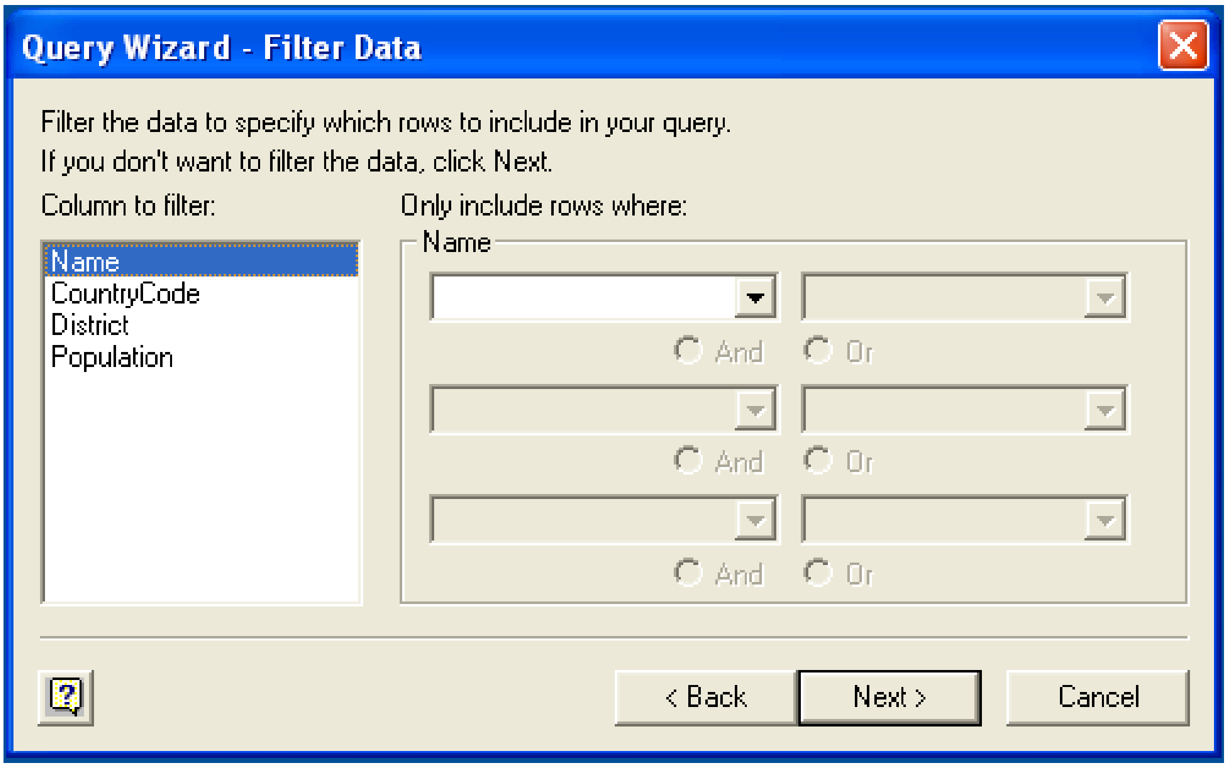
3. Microsoft Query will start. First, you need to choose the data source, by selecting an existing Data Source Name.



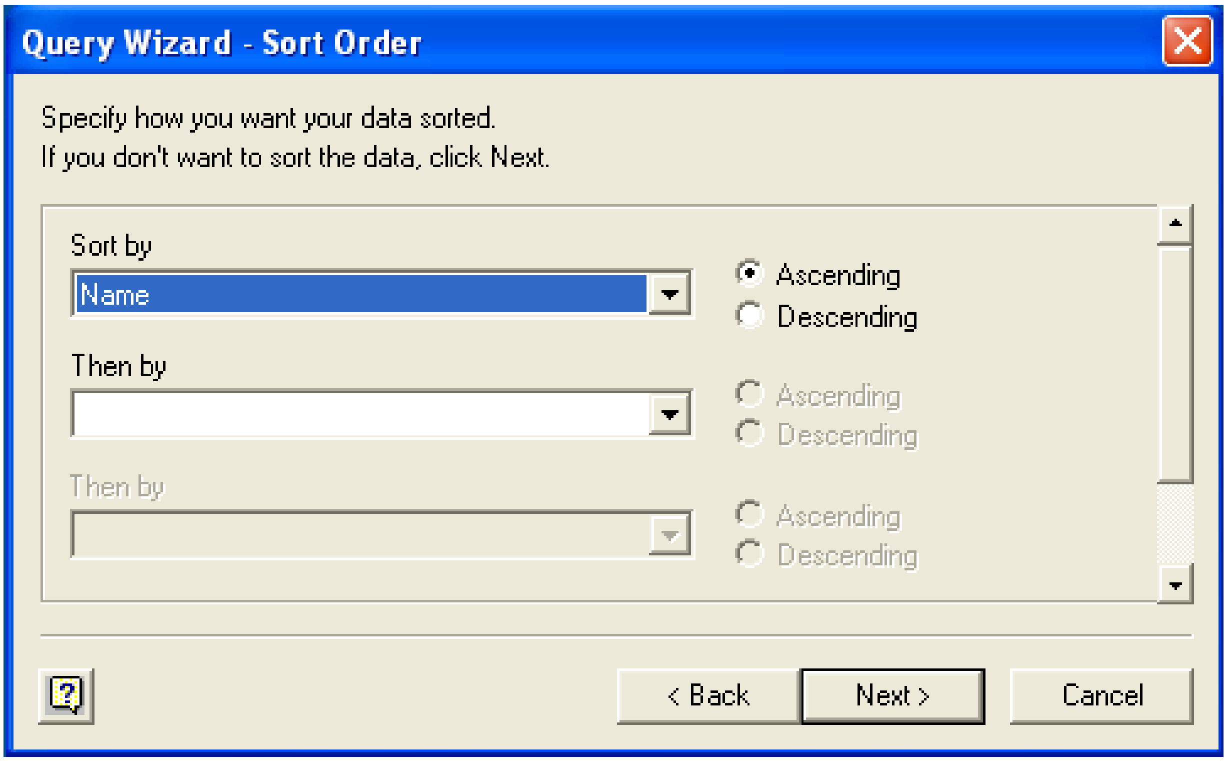
4. Within the `Query Wizard`, you must choose the columns that you want to import. The list of tables available to the user configured through the DSN is shown on the left, the columns that will be added to your query are shown on the right. The columns you choose are equivalent to those in the first 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:
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
< Back
Next >
Cancel

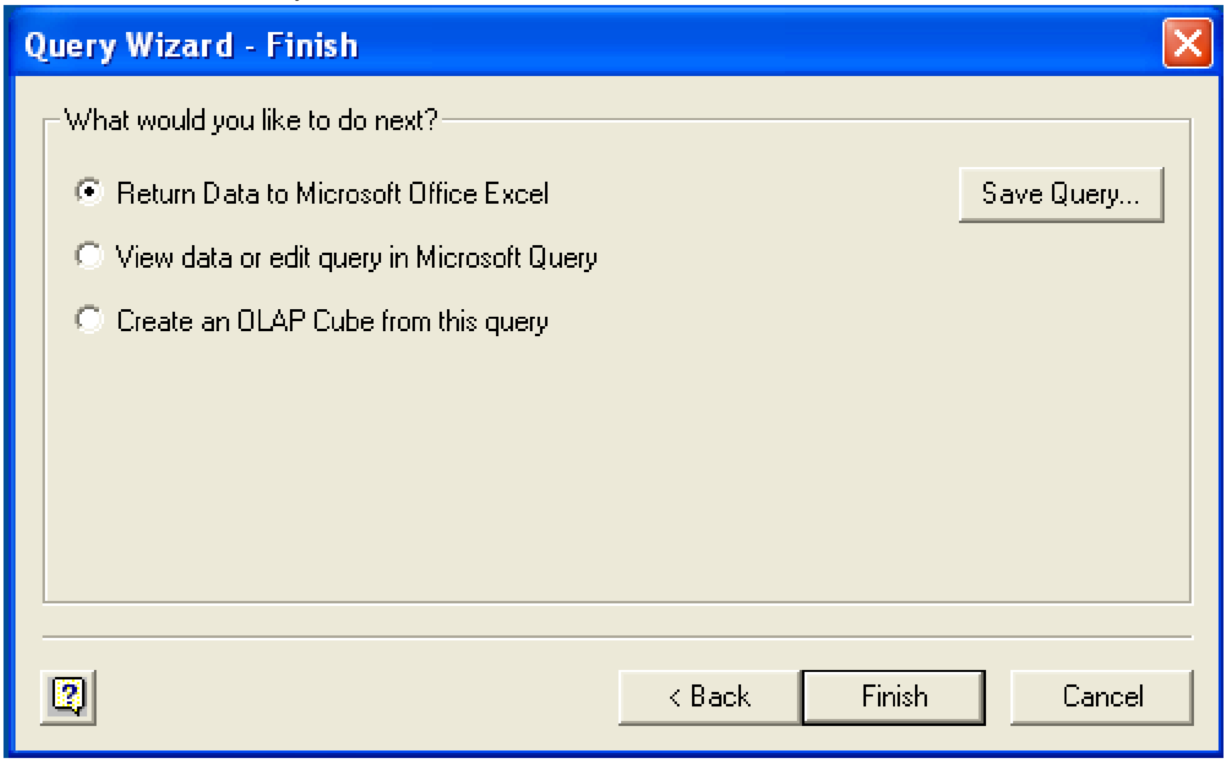
6. Select an (optional) sort order for the data. This is equivalent to using a `ORDER BY` clause in your SQL query. You can select up to three fields for sorting the information returned by the query. Click NEXT to continue.



Specify how you want your data sorted.
If you don't want to sort the data, click Next.

Sort by: Name [✓ Ascending] Descending

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.



What would you like to do next?

[✓ Return Data to Microsoft Office Excel]
View data or edit query in Microsoft Query
Create an OLAP Cube from this query

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.

1.5.6. 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, you should follow these steps:

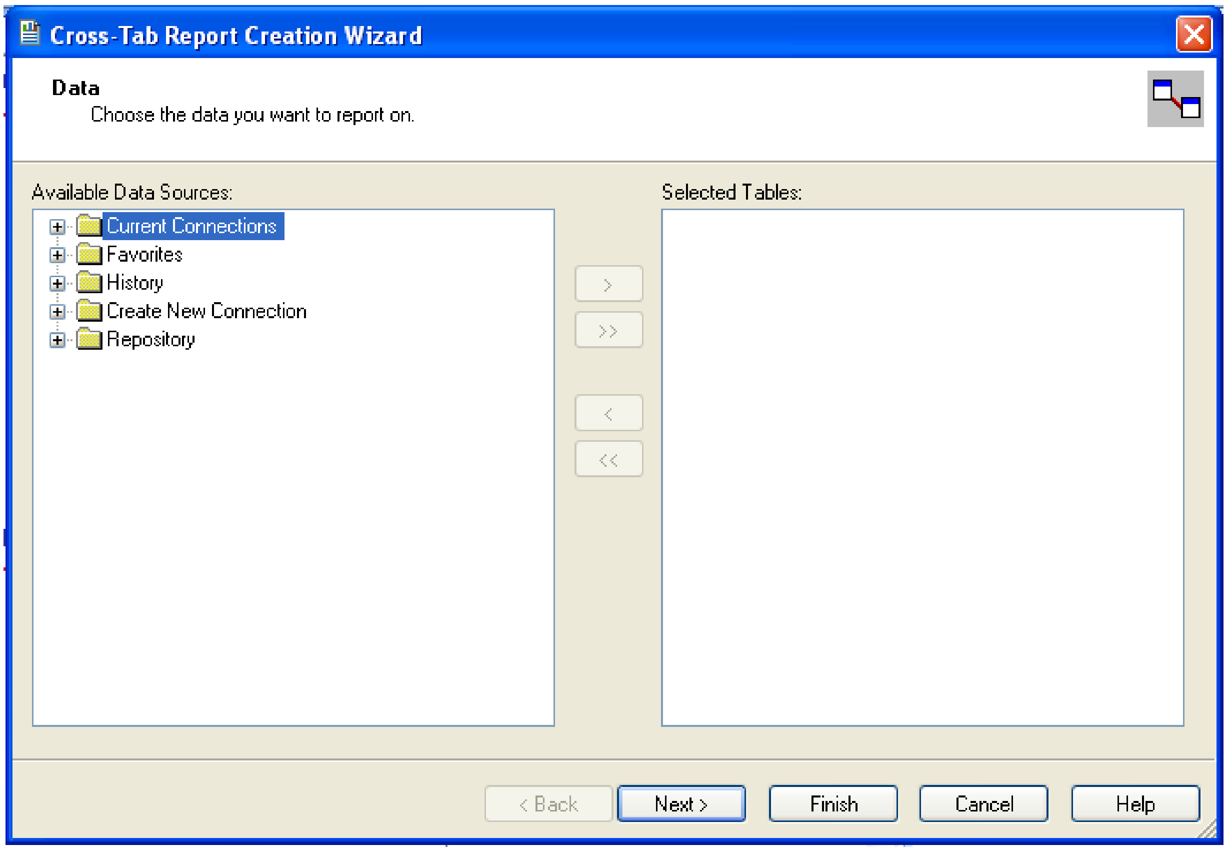
1. Create a DSN using the `Data Sources (ODBC)` tool. You can either specify a complete database, including user name and password, or you can build a basic DSN and use Crystal Reports to set the user name and password.

   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.

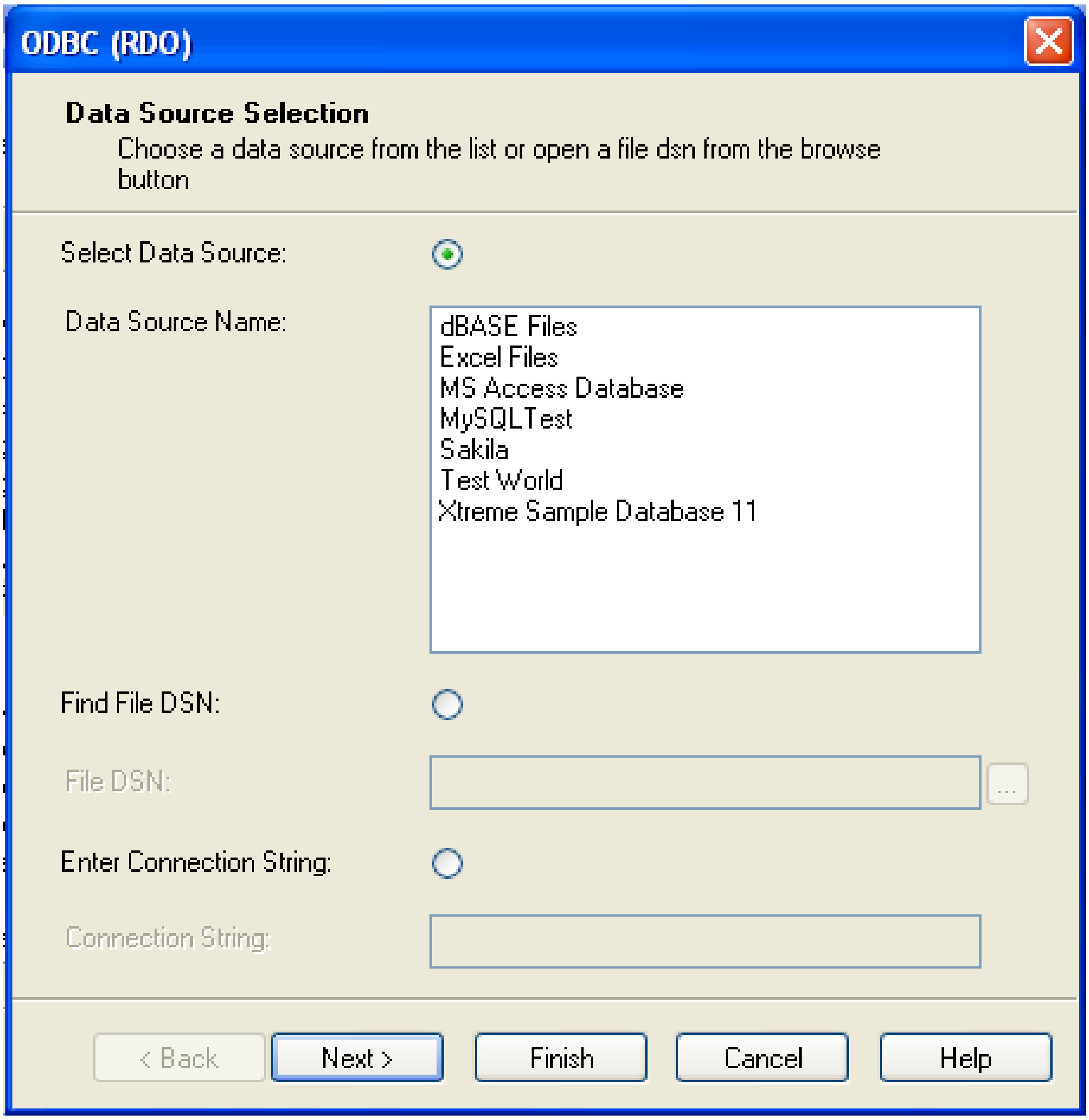
3. Start the Cross-Tab Report Wizard, either by clicking on the option on the Start Page. Expand the **CREATE NEW CONNECTION** folder, then expand the **ODBC (RDO)** folder to obtain a list of ODBC data sources.

   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.


ODBC (RDO)

Data Source Selection

Choose a data source from the list or open a file dsn from the browse button

Select Data Source: [✓]

Data Source Name:

dBASE Files
Excel Files
MS Access Database
MySQLTest
Sakila
Test World
Xtreme Sample Database 11

Find File DSN:

File DSN:

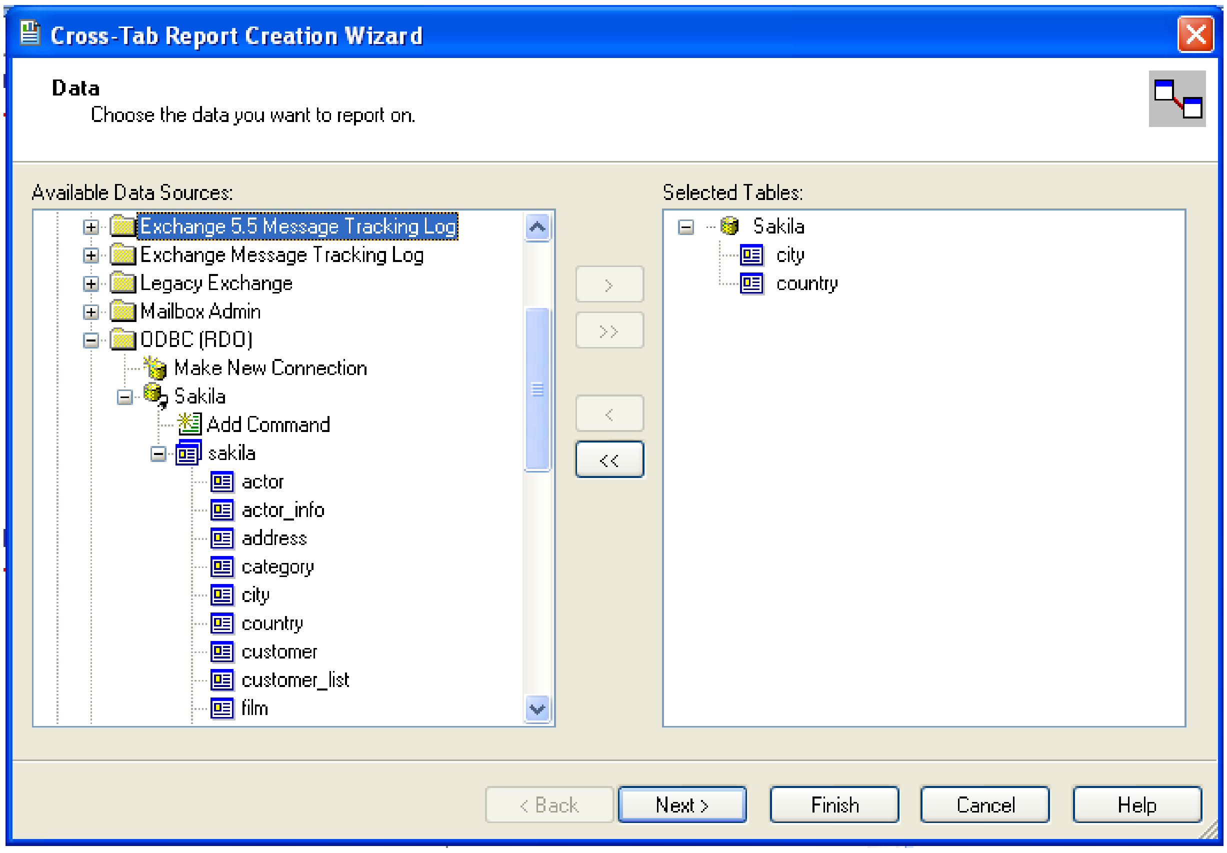
Enter Connection String:

Connection String:

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.

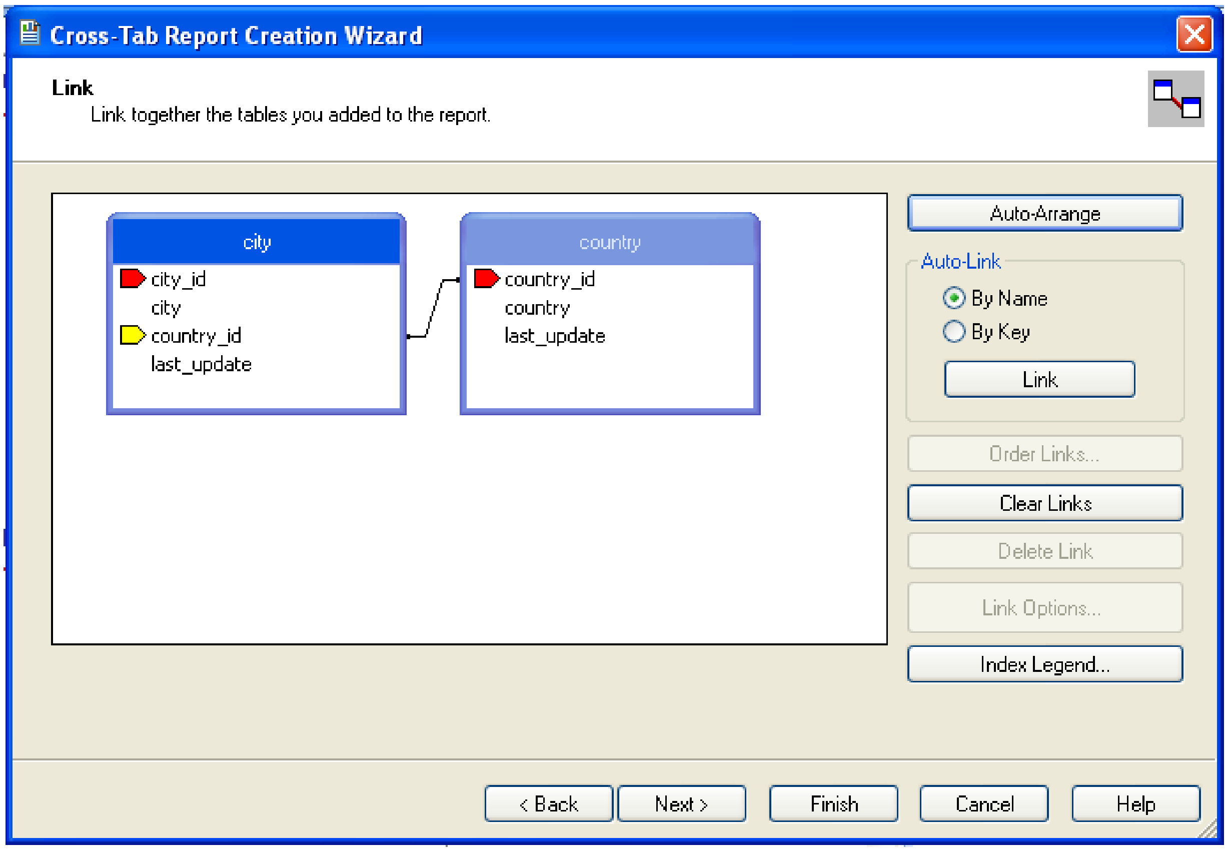
   Finally, you can select the parent **SAKILA** resource and add of the tables to the report.

   Once you have selected the tables you want to include, click NEXT to continue.



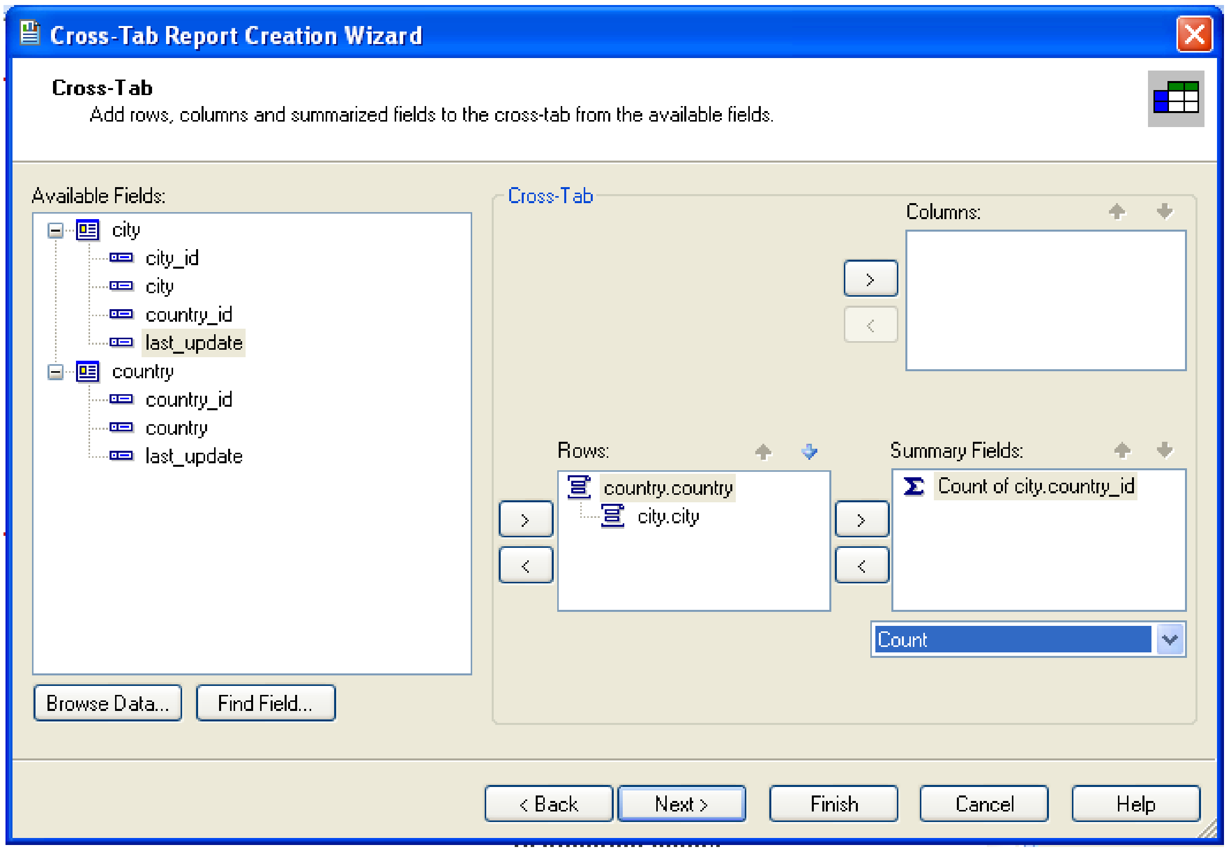
6. Crystal Reports will now read the table definitions and automatically identify the links between the tables. The identification of links between tables enables Crystal Reports to automatically lookup and summarize information based on all the tables in the database according to your query. If Crystal 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 you wish 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.


Cross-Tab Report Creation Wizard

Cross-Tab

Add rows, columns and summarized fields to the cross-tab from the available fields.

Available Fields:

- city
  - city_id
  - city
  - country_id
  - last_update
- country
  - country_id
  - country
  - last_update

Cross-Tab

Columns:

Rows:

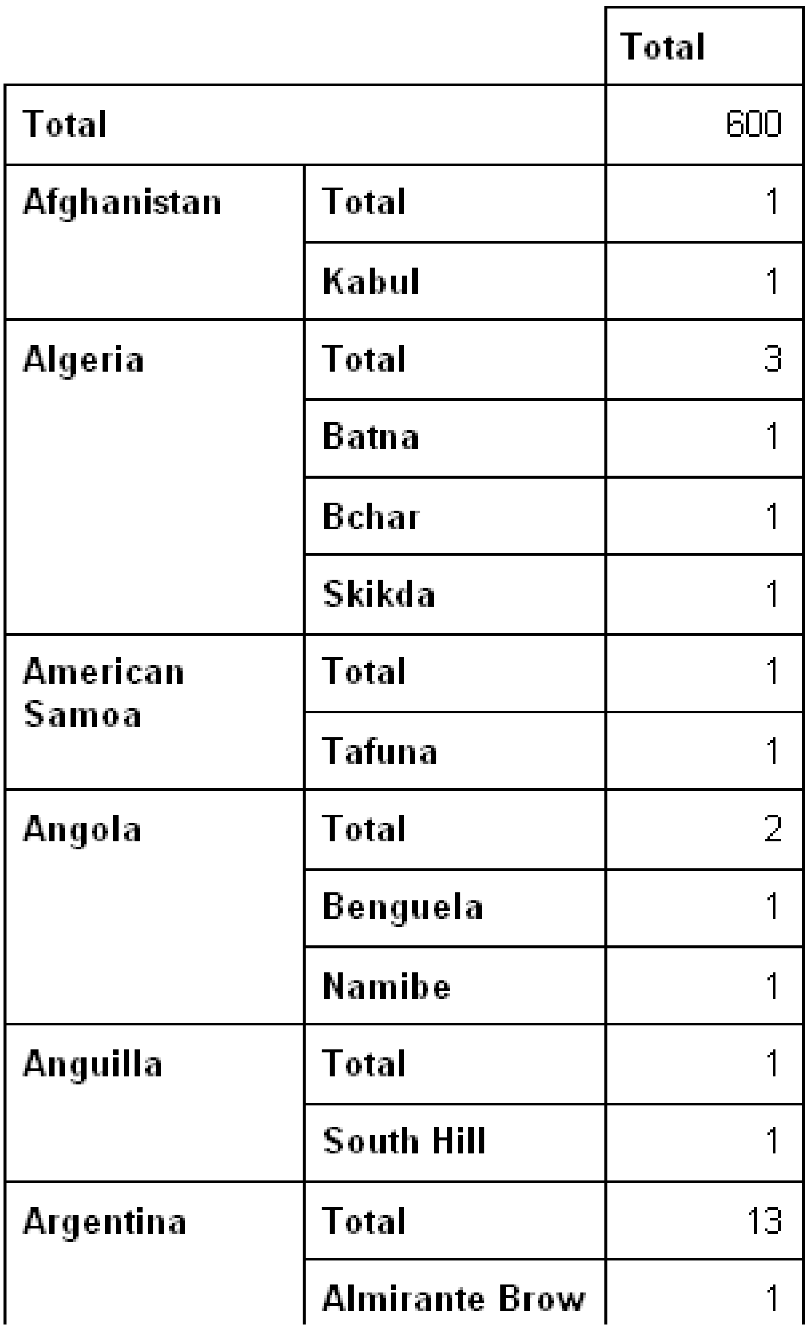
- country.country
  - city.city

Summary Fields:

- Count of city.country_id

Count

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.

1.5.7. Connector/ODBC Programming

With a suitable ODBC Manager and the Connector/ODBC driver installed, any programming language or environment that can support ODBC should be able to connect to a MySQL database through Connector/ODBC.

This includes, but is certainly 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).

1.5.7.1. Using Connector/ODBC with Visual Basic Using ADO, DAO and RDO

This section contains simple examples of the use of MySQL ODBC 3.51 Driver with ADO, DAO and RDO.

1.5.7.1.1. 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 MySQL ODBC 3.51 Driver
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
```

1.5.7.1.2. 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
```

1.5.7.1.3. 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
```

1.5.7.2. Using Connector/ODBC with .NET

This section contains simple examples that demonstrate the use of Connector/ODBC drivers with ODBC.NET.

1.5.7.2.1. 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);
          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.Write("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");
            }
        }
    }
  }
}
```

1.5.7.2.2. 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")
      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
```

1.6. 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.

1.6.1. 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 3.51 driver supports all deprecated functions.

The following tables list Connector/ODBC API calls grouped by task:

Connecting to a data source

Function name	C/ODBC 3.51	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.
`SQLAllocConnect`	Yes	Deprecated	Obtains a connection handle

Obtaining information about a driver and data source

Function name	C/ODBC 3.51	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.

Setting and retrieving driver attributes

Function name	C/ODBC 3.51	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

Preparing SQL requests

Function name	C/ODBC 3.51	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.

`SQLGetCursorName`	Yes	ISO 92	Returns the cursor name associated with a statement handle.
`SQLSetCursorName`	Yes	ISO 92	Specifies a cursor name.
`SQLSetScrollOptions`	Yes	ODBC	Sets options that control cursor behavior.

Submitting requests

Function name	**C/ODBC 3.51**	**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`	Yes	ODBC	Returns the description for a specific parameter in a statement.
`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.)

Retrieving results and information about results

Function name	**C/ODBC 3.51**	**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 allows 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.

Retrieving error or diagnostic information

Function name	**C/ODBC 3.51**	**Standard**	**Purpose**
`SQLError`	Yes	Deprecated	Returns additional error or status information
`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).

Obtaining information about the data source's system tables (catalog functions) item

Function name	**C/ODBC 3.51**	**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.

Performing transactions

Function name	C/ODBC 3.51	Standard	Purpose
SQLTransact	Yes	Deprecated	Commits or rolls back a transaction
SQLEndTran	Yes	ISO 92	Commits or rolls back a transaction.

Terminating a statement

Function name	C/ODBC 3.51	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.

Terminating a connection

Function name	C/ODBC 3.51	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

1.6.2. Connector/ODBC Data Types

The following table illustrates how driver maps the server data types to default 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`
`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`

1.6.3. Connector/ODBC Error Codes

The following tables lists the error codes returned by the driver apart from the server errors.

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
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

1.7. 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.

1.7.1. 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.

1.7.1.1. 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;
```

See How to Get the Unique ID for the Last Inserted Row.

1.7.1.2. 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` checkbox 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.

1.7.1.3. 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 you should 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 Mac OS X, check the TRACING panel of ODBC Administrator. See Section 1.4.8, "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 Mac OS X and Unix, ensure that the driver trace (option value 4) and query logging (option value 524288) are not enabled.

1.7.1.4. 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.

1.7.2. 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 you should 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.

1.7.2.1. 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.

1.7.2.1.1. Microsoft Access

To improve the integration between Microsoft Access and MySQL through Connector/ODBC:

- For all versions of Access, you should enable the Connector/ODBC `Return matching rows` option. For Access 2.0, you should additionally enable the `Simulate ODBC 1.0` option.

- You should have 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).
- You should have 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. You should set the `DefaultValue` property for the `TIMESTAMP` column to `NOW()`. It may be a good idea to hide 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, you should 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 converters section for list of available tools.

MySQL Enterprise
MySQL Enterprise subscribers will find more information about using ODBC with Access in Knowledge Base articles such as Use MySQL-Specific Syntax with Microsoft Access. To subscribe to MySQL Enterprise see http://www.mysql.com/products/enterprise/advisors.html.

1.7.2.1.2. Microsoft Excel and Column Types

If you have problems importing data into Microsoft Excel, particularly numerical, 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.

1.7.2.1.3. 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.

> **MySQL Enterprise**
> MySQL Enterprise subscribers can find a discussion about using VBA in the Knowledge Base article, MySQL-Specific Syntax with VBA. To subscribe to MySQL Enterprise see http://www.mysql.com/products/enterprise/advisors.html.

1.7.2.1.4. 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.

1.7.2.1.5. Visual Objects

You should select the `Don't optimize column widths` option.

1.7.2.1.6. 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/mdmthcnnexecute.asp.

For information, see ActiveX Data Objects(ADO) Frequently Asked Questions.

1.7.2.1.7. Using Connector/ODBC with Active Server Pages (ASP)

You should select the `Return matching rows` option in the DSN.

For more information about how to access MySQL via 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 ht-tp://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp.

1.7.2.1.8. 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>`).

1.7.2.2. 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.

1.7.2.2.1. Using Connector/ODBC with Borland Builder 4

When you start a query, you can use the `Active` property or the `Open` method. Note that `Active` starts by automatically issuing a `SELECT * FROM ...` query. That may not be a good thing if your tables are large.

1.7.2.2.2. 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);
```

1.7.2.2.3. 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.

1.7.2.3. 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 allows you to us 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, you must build and copy the Connector/ODBC driver to `/opt/coldfusion/lib/libmyodbc.so`.

The Contrib directory contains the program `mydsn-xxx.zip` which allows you to 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.

1.7.2.4. Using Connector/ODBC with OpenOffice.org

Open Office (http://www.openoffice.org) How-to: MySQL + OpenOffice. How-to: OpenOffice + MyODBC + unixODBC.

1.7.2.5. Using Connector/ODBC with Sambar Server

Sambar Server (http://www.sambarserver.info) How-to: MyODBC + SambarServer + MySQL.

1.7.2.6. 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.

1.7.2.7. Using Connector/ODBC with SunSystems Vision

You should select the `Return matching rows` option.

1.7.3. 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 1.8.1, “Connector/ODBC Community Support”.

Many problems can be resolved by upgrading your Connector/ODBC drivers to the latest available release. On Windows, you should also make sure that you have the latest versions of the Microsoft Data Access Components (MDAC) installed.

Questions

- 1.7.3.1: 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`.
- 1.7.3.2: When connecting or using the TEST button in `ODBC Data Source Administrator` I get error 10061 (Cannot connect to server)
- 1.7.3.3: The following error is reported when using transactions: `Transactions are not enabled`
- 1.7.3.4: Access reports records as `#DELETED#` when inserting or updating records in linked tables.
- 1.7.3.5: How do I handle Write Conflicts or Row Location errors?
- 1.7.3.6: Exporting data from Access 97 to MySQL reports a `Syntax Error`.
- 1.7.3.7: Exporting data from Microsoft DTS to MySQL reports a `Syntax Error`.
- 1.7.3.8: Using ODBC.NET with Connector/ODBC, while fetching empty string (0 length), it starts giving the SQL_NO_DATA exception.
- 1.7.3.9: Using `SELECT COUNT(*) FROM tbl_name` within Visual Basic and ASP returns an error.
- 1.7.3.10: Using the `AppendChunk()` or `GetChunk()` ADO methods, the `Multiple-step operation generated`

`errors. Check each status value` error is returned.

- 1.7.3.11: Access Returns `Another user had modified the record that you have modified` while editing records on a Linked Table.
- 1.7.3.12: When linking an application directly to the Connector/ODBC library under Unix/Linux, the application crashes.
- 1.7.3.13: Applications in the Microsoft Office suite are unable to update tables that have `DATE` or `TIMESTAMP` columns.
- 1.7.3.14: When connecting Connector/ODBC 5.x (Beta) to a MySQL 4.x server, the error `1044 Access denied for user 'xxx'@'%' to database 'information_schema'` is returned.
- 1.7.3.15: When calling `SQLTables`, the error `S1T00` is returned, but I cannot find this in the list of error numbers for Connector/ODBC.
- 1.7.3.16: 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.
- 1.7.3.17: When I try to use batched statements, the excution of the batched statements fails.
- 1.7.3.18: 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.
- 1.7.3.19: When using some applications to access a MySQL server using C/ODBC and outer joins, an error is reported regarding the Outer Join Escape Sequence.
- 1.7.3.20: I can correctly store extended characters in the database (Hebrew/CJK) using C/ODBC 5.1, but when I retrieve the data, the text is not formatted correctly and I get garbled characters.
- 1.7.3.21: I have a duplicate MySQL Connector/ODBC entry within my **INSTALLED PROGRAMS** list, but I cannot delete one of them.
- 1.7.3.22: When submitting queries with parameter binding using `UPDATE`, my field values are being truncated to 255 characters.
- 1.7.3.23: Is it possible to disable data-at-execution using a flag?

Questions and Answers

1.7.3.1: 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, you should 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.

1.7.3.2: 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`.

1.7.3.3: 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. In versions of MySQL before Mysql 5.1 you may also use the `BDB` engine.

You should 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 transaction database engine.
- Ensure that you have not enabled the `disable transactions` option in your DSN.

1.7.3.4: Access reports records as `#DELETED#` when inserting or updating records in linked tables.

If the inserted or updated records are shown as `#DELETED#` in the access, then:

- If you are using Access 2000, you should 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.

  You should 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, you should enable the Connector/ODBC `Return matching rows` option. For Access 2.0, you should additionally enable the `Simulate ODBC 1.0` option.
- You should have a timestamp in all tables that you want to be able to update.
- You should have 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.

1.7.3.5: 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.
```

1.7.3.6: 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.

1.7.3.7: 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.

1.7.3.8: 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.

1.7.3.9: 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).

1.7.3.10: 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 doesn't 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

1.7.3.11: 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, you should 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 1.4.8, “Getting an ODBC Trace File”.

1.7.3.12: When linking an application directly to the Connector/ODBC library under Unix/Linux, the application crashes.

Connector/ODBC 3.51 under Unix/Linux is not compatible with direct application linking. You must use a driver manager, such as iODBC or unixODBC to connect to an ODBC source.

1.7.3.13: Applications in the Microsoft Office suite are unable to update tables that have `DATE` or `TIMESTAMP` columns.

This is a known issue with Connector/ODBC. You must ensure that the field has a default value (rather than `NULL` and that the default value is non-zeo (i.e. the default value is not `0000-00-00 00:00:00`).

1.7.3.14: When connecting Connector/ODBC 5.x (Beta) 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.

1.7.3.15: 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.

1.7.3.16: 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.

1.7.3.17: When I try to use batched statements, the excution of the batched statements fails.

Batched statement support was added in 3.51.18. Support for batched statements is not enabled by default. You must enable option `FLAG_MULTI_STATEMENTS`, value 67108864, or select the ALLOW MULTIPLE STATEMENTS flag within a GUI configuration.

1.7.3.18: 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.

1.7.3.19: When using some applications to access a MySQL server using C/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 value > 0 when asked for `SQL_OJ_CAPABILITIES` even though no parsing takes place in the driver to handle the outer join escape sequence.

1.7.3.20: I can correctly store extended characters in the database (Hebrew/CJK) using C/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 you should add the following to your ASP files to ensure that the data returned is correctly encoded:

```
Response.CodePage = 65001
Response.CharSet = "utf-8"
```

1.7.3.21: 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 you should use any working uninstallers to remove existing installations and 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.

1.7.3.22: When submitting queries with parameter binding using `UPDATE`, my field values are being truncated to 255 characters.

You should ensure that the `FLAG_BIG_PACKETS` option is set for your connection. This removes the 255 character limitation on bound parameters.

1.7.3.23: Is it possible to disable data-at-execution using a flag?

If you do not wish to use data-at-execution, simply 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);
```

Note that in the call to `SQLBindCol()`, &ylen has been replaced by NULL.

For further information please refer to the MSDN documentation for `SQLBindCol()`.

1.8. Connector/ODBC Support

There are many different places where you can get support for using Connector/ODBC. You should always try the Connector/ODBC Mailing List or Connector/ODBC Forum. See Section 1.8.1, “Connector/ODBC Community Support”, for help before reporting a specific bug or issue to MySQL.

1.8.1. Connector/ODBC Community Support

Sun Microsystems, Inc. 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>` 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.

1.8.2. How to Report Connector/ODBC Problems or Bugs

If you encounter difficulties or problems with Connector/ODBC, you should 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 1.4.8, “Getting an ODBC Trace File”.

Check the Connector/ODBC trace file to find out what could be wrong. You should be able to determine what statements were issued by searching for the string `>mysql_real_query` in the `myodbc.log` file.

You should also try issuing the statements from the `mysql` client program or from `admndemo`. 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!

You should 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. You can send this to ftp://ftp.mysql.com/pub/mysql/upload/. 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, you should 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.

1.8.3. 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>`.

1.8.4. Connector/ODBC Change History

The Connector/ODBC Change History (Changelog) is located with the main Changelog for MySQL. See Appendix A, *MySQL Connector/ODBC (MyODBC) Change History*.

1.8.5. Credits

These are the developers that have worked on the Connector/ODBC and Connector/ODBC 3.51 Drivers from MySQL AB.

- Michael (Monty) Widenius
- Venu Anuganti
- Peter Harvey

Chapter 2. MySQL Connector/NET

Connector/NET enables developers to easily create .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#.

Connector/NET includes full support for:

- MySQL 6.0 features
- MySQL 5.1 features
- MySQL 5.0 features (such as stored procedures)
- MySQL 4.1 features (server-side prepared statements, Unicode, and shared memory access, and so forth)
- Large-packet support for sending and receiving rows and BLOBs up to 2 gigabytes in size.
- Protocol compression which allows for compressing the data stream between the client and server.
- Support for connecting using TCP/IP sockets, named pipes, or shared memory on Windows.
- Support for connecting using TCP/IP sockets or Unix sockets on Unix.
- Support for the Open Source Mono framework developed by Novell.
- Fully managed, does not utilize the MySQL client library.

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 `Documentation.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 may want also want to use the MySQL Visual Studio Plugin. The plugin acts as a DDEX (Data Designer Extensibility) provider, enabling you to use the data design tools within Visual Studio to manipulate the schema and objects within a MySQL database. For more information, see Section 2.3, “Visual Studio User Guide”.

Note

Connector/NET 5.1.2 and later include the Visual Studio Plugin by default.

Key topics:

- For connection string properties when using the `MySqlConnection` class, see Section 2.4.5, “Connector/NET Connection String Options Reference”.

2.1. Connector/NET Versions

There are several versions of Connector/NET available:

- Connector/NET 1.0 includes support for MySQL 4.0, and MySQL 5.0 features, and full compatibility with the ADO.NET driver interface.
- Connector/NET 5.0 includes support for MySQL 4.0, MySQL 4.1, MySQL 5.0 and MySQL 5.1 features. Connector/NET 5.0 also includes full support for the ADO.Net 2.0 interfaces and subclasses, includes support for the usage advisor and performance monitor (PerfMon) hooks.
- Connector/NET 5.1 includes support for MySQL 4.0, MySQL 5.0, MySQL 5.1 and MySQL 6.0 (Falcon Preview) features. Connector/NET 5.1 also 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.
- Connector/NET 5.2 includes support for MySQL 4.0, MySQL 5.0, MySQL 5.1 and MySQL 6.0 (Falcon Preview) features. Connector/NET 5.2 also 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.

- Connector/NET 6.0 includes support for MySQL 4.0, MySQL 5.0, MySQL 5.1 and MySQL 6.0. Connector/NET 6.0 is currently available as an Alpha release.

The latest source code for Connector/NET can be downloaded from the MySQL public Subversion server. For further details see Section 2.2.3, "Installing Connector/NET from the source code".

The following table shows the .NET Framework version required, and MySQL Server version supported by Connector/NET:

Connector/NET version	ADO.NET version supported	.NET Framework version required	MySQL Server version supported
1.0	1.x	1.x	4.0, 5.0
5.0	1.x+	1.x+	4.0, 5.0
5.1	1.x+	1.x+	4.0, 5.0, 5.1, 6.0
5.2	1.x+	1.x+	4.0, 5.0, 5.1, 6.0
6.0	2.x+	2.x+	4.0, 5.0, 5.1, 6.0

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 format is different.

2.2. Connector/NET Installation

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 framework (see http://www.mono-project.com).

Connector/NET is available for download from http://dev.mysql.com/downloads/connector/net/5.2.html.

2.2.1. Installing Connector/NET on Windows

On Windows, installation is supported either through a binary installation process or by downloading a Zip file with the Connector/NET components.

Before installing, you should ensure that your system is up to date, including installing the latest version of the .NET Framework.

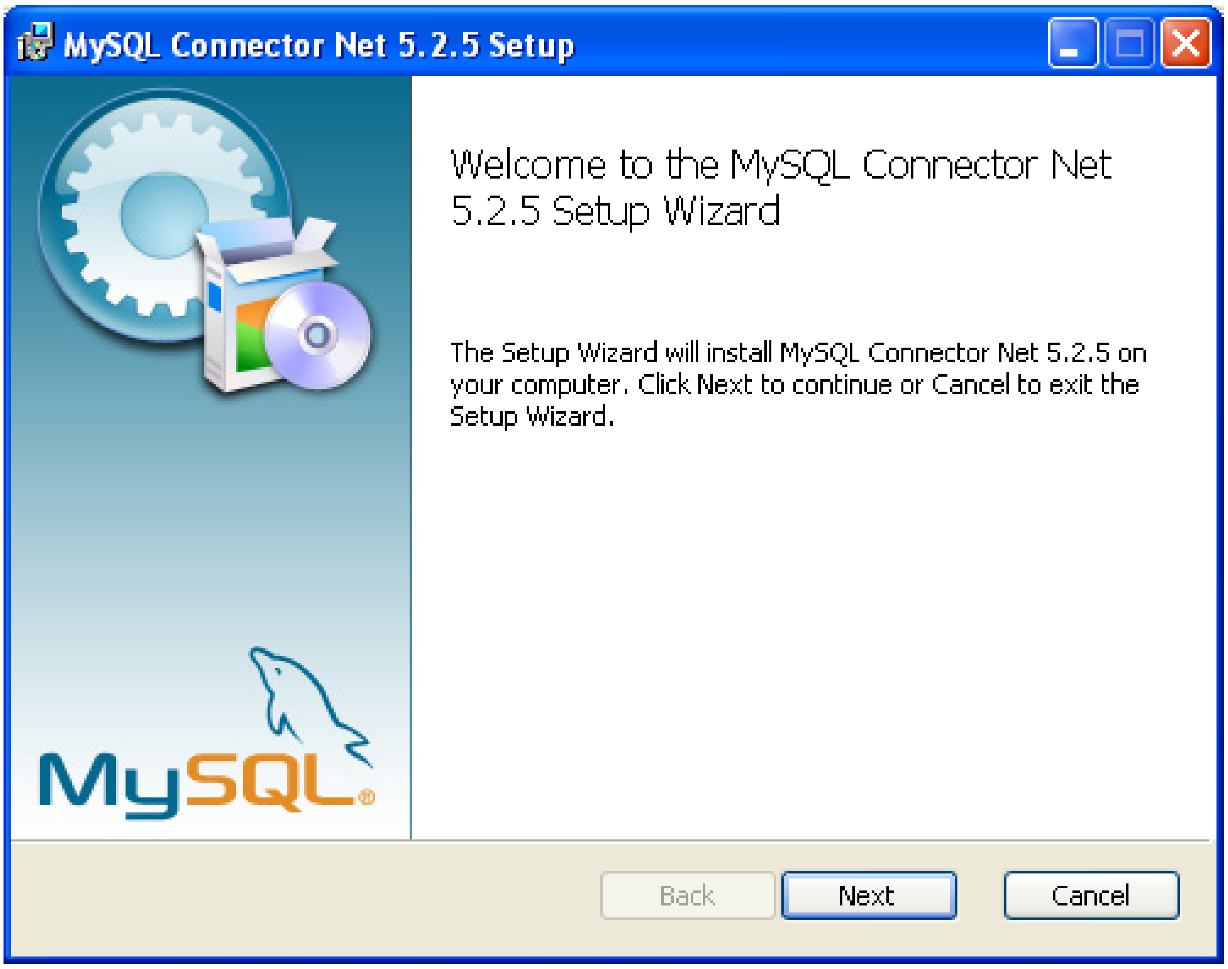
2.2.1.1. Installing Connector/NET using the Installer

Using the installer is the most straightforward method of installing Connector/NET on Windows and the installed components include the source code, test code and full reference documentation.

Connector/NET is installed through the use of a Windows Installer (`.msi`) installation package, which can be used to install Connector/NET on all Windows operating systems. The MSI package in contained within a ZIP archive named `mysql-connector-net-`*`version`*`.zip`, where *`version`* indicates the Connector/NET version.

To install Connector/NET:

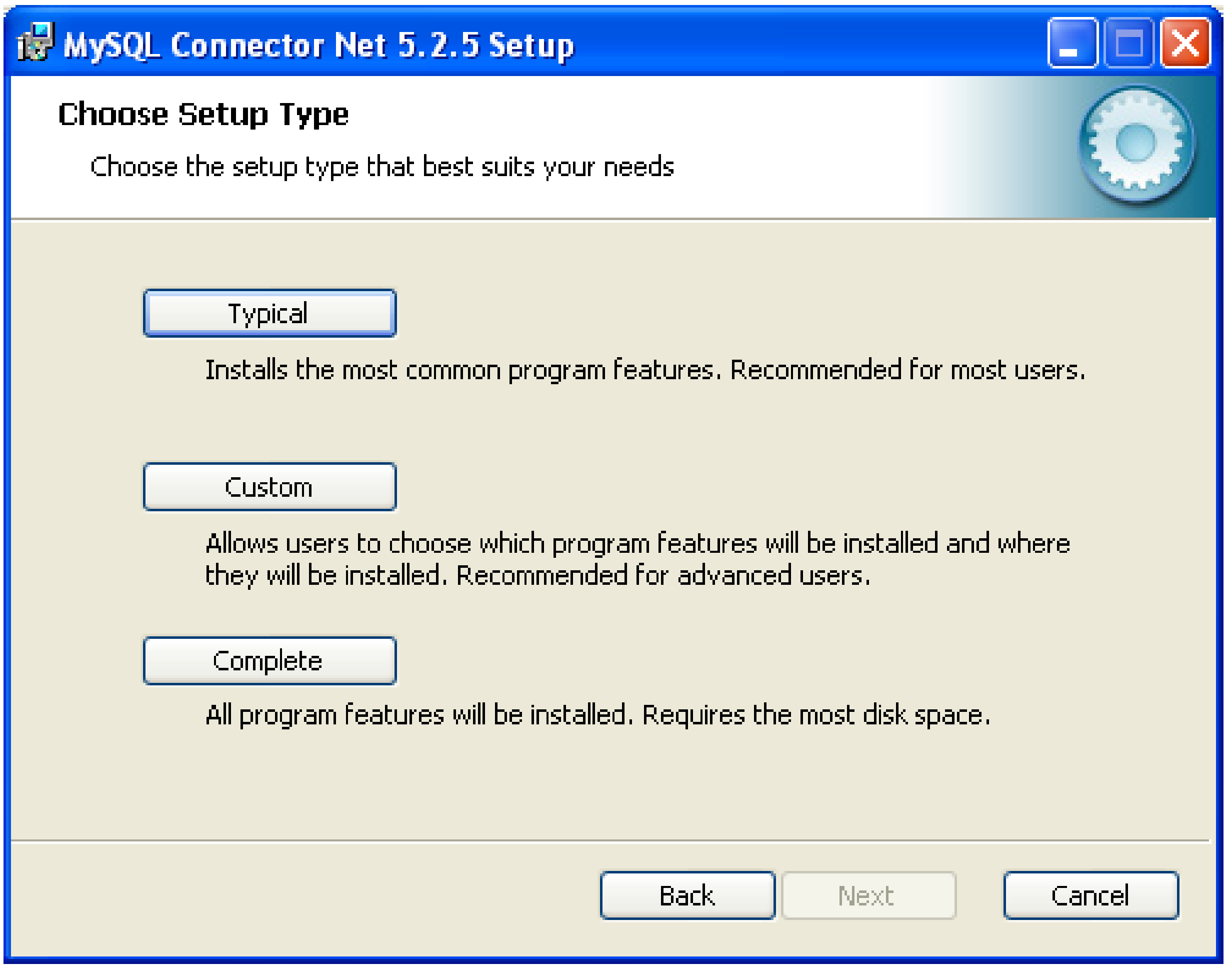
1. Double click on the MSI installer file extracted from the Zip you downloaded. Click NEXT to start the installation.


MySQL Connector Net 5.2.5 Setup

Welcome to the MySQL Connector Net 5.2.5 Setup Wizard

The Setup Wizard will install MySQL Connector Net 5.2.5 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

2. You must choose the type of installation that you want to perform.



Choose Setup Type

Choose the setup type that best suits your needs

Installs the most common program features. Recommended for most users.

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

All program features will be installed. Requires the most disk space.

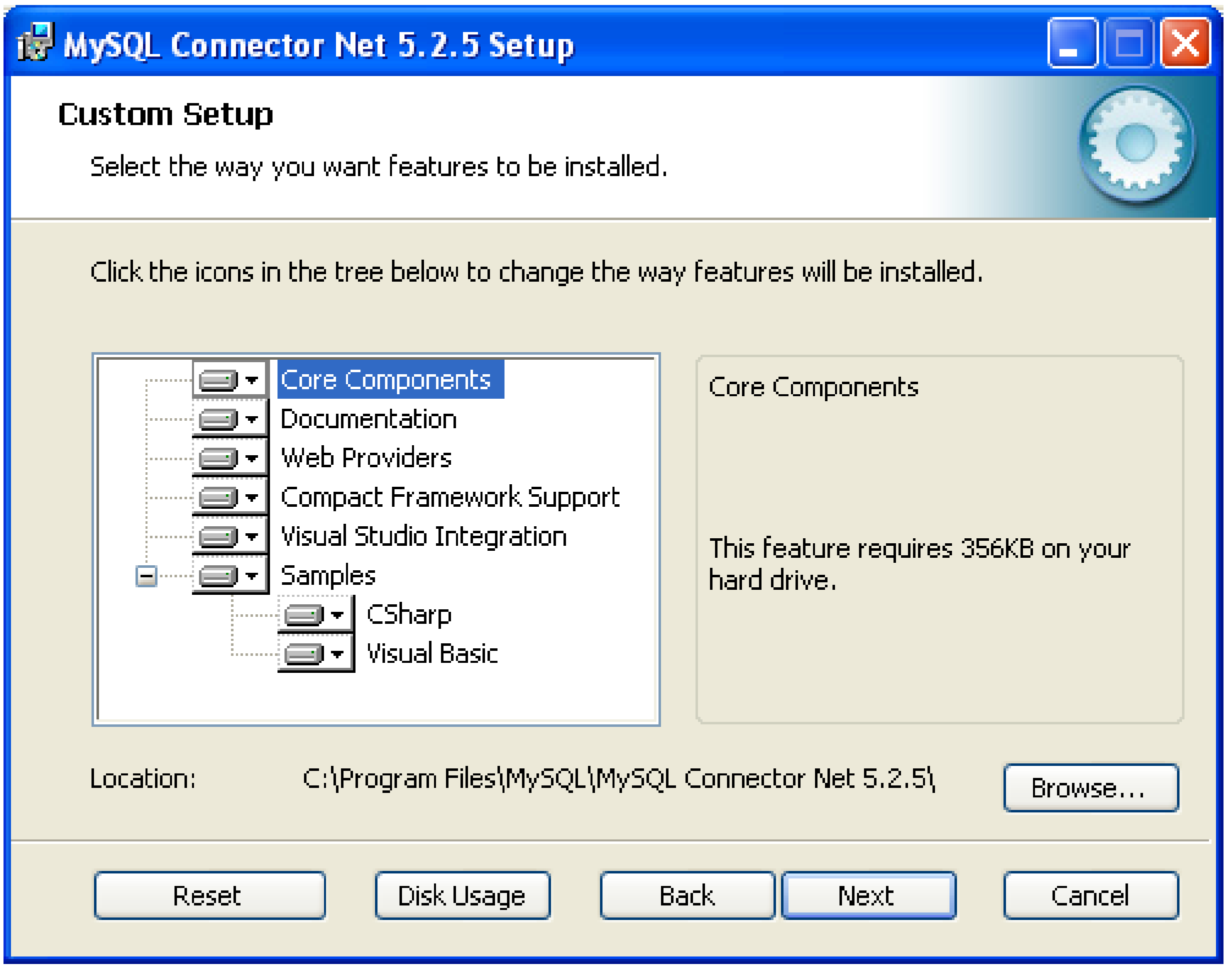
For most situations, the Typical installation will be suitable. Click the TYPICAL button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the COMPLETE button and proceed to step 5. If you want to customize your installation, including 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 that you want 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.

Note

For Connector/NET 1.0.8 or lower and Connector 5.0.4 and lower the installer will attempt to install binaries for both 1.x and 2.x of the .NET Framework. If you only have one version of the framework installed, the connector installation may fail. If this happens, you can choose the framework version to be installed through the custom installation step.


MySQL Connector Net 5.2.5 Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

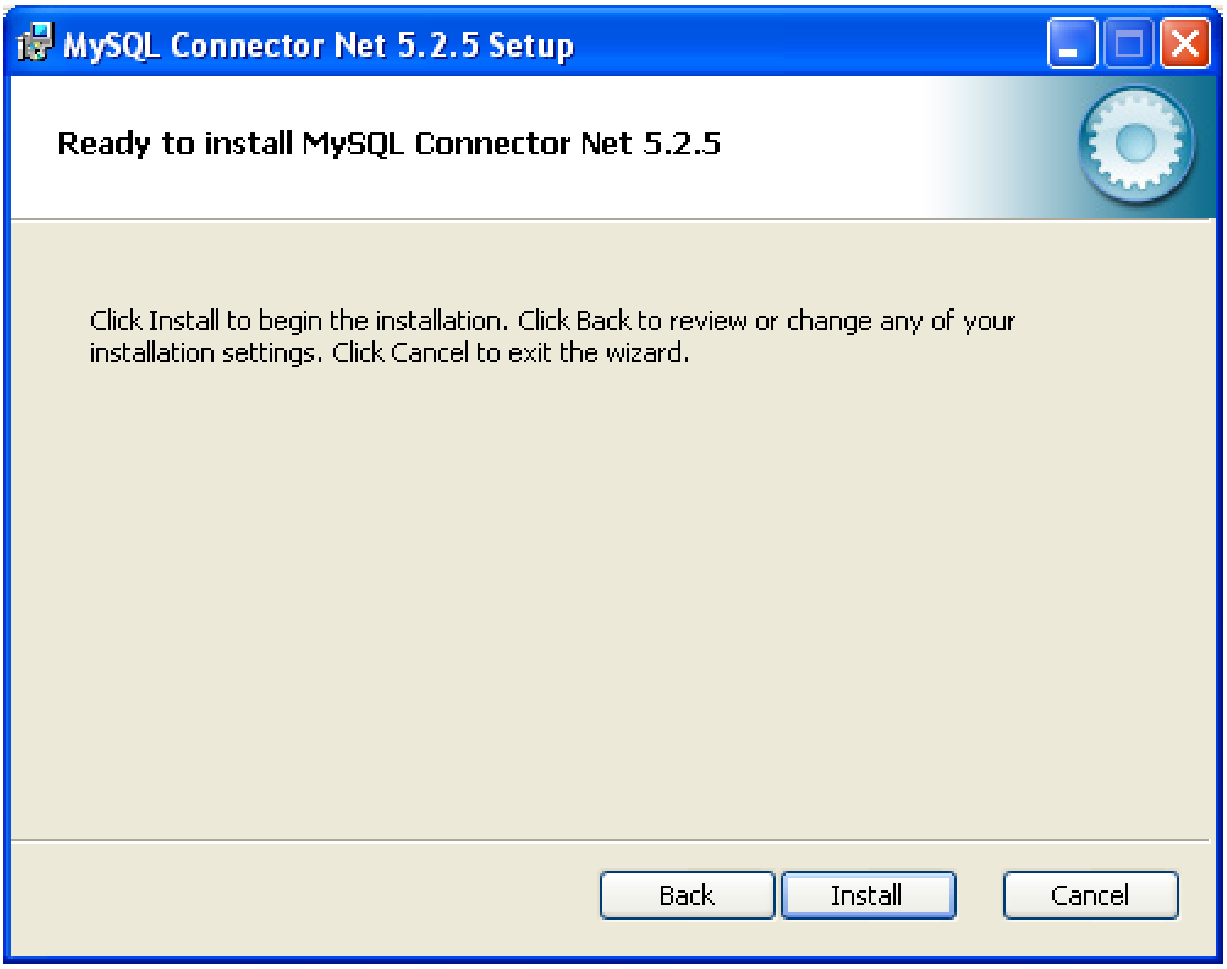
- Core Components
- Documentation
- Web Providers
- Compact Framework Support
- Visual Studio Integration
- Samples
  - CSharp
  - Visual Basic

Core Components

This feature requires 356KB on your hard drive.

Location: C:\Program Files\MySQL\MySQL Connector Net 5.2.5\

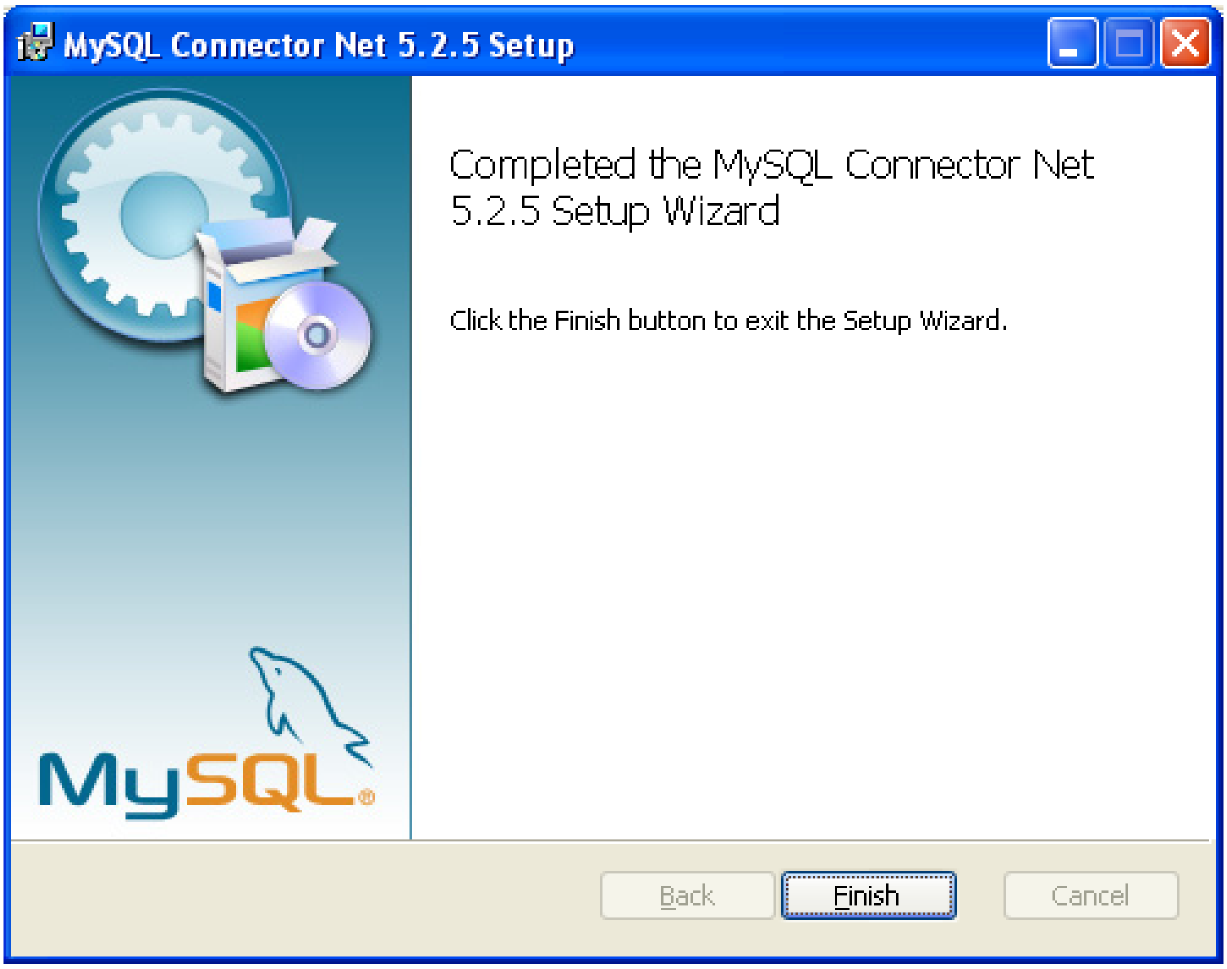
4. You will be given a final opportunity to confirm the installation. Click INSTALL to copy and install the files onto your machine.


MySQL Connector Net 5.2.5 Setup

Ready to install MySQL Connector Net 5.2.5

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

5. Once the installation has been completed, click FINISH to exit the installer.


MySQL Connector Net 5.2.5 Setup

Completed the MySQL Connector Net 5.2.5 Setup Wizard

Click the Finish button to exit the Setup Wizard.

Unless you choose otherwise, Connector/NET is installed in `C:\Program Files\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:

- `bin` - Connector/NET MySQL libraries for different versions of the .NET environment.
- `docs` - contains a CHM of the Connector/NET documentation.
- `samples` - sample code and applications that use the Connector/NET component.
- `src` - the source code for 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 conector-net.msi /quiet
```

To provide a progress bar to the user during automatic installation, use the `/passive` option.

2.2.1.2. Installing Connector/NET using the Zip packages

If you are having problems running the installer, you can download a Zip file without an installer as an alternative. That file is called `mysql-connector-net-`*`version`*`-noinstall.zip`. Once downloaded, you can extract the files to a location of your choice.

The file contains the following directories:

- `bin` - Connector/NET MySQL libraries for different versions of the .NET environment.
- `Docs` - contains a CHM of the Connector/NET documentation.
- `Samples` - sample code and applications that use the Connector/NET component.

Connector/NET 6.0.x has a different directory structure:

- `Assemblies` - contains a collection of DLLs that make up the connector functionality.
- `Documentation` - contains the Connector/NET documentation as a CHM file.
- `Samples` - sample code and applications that use the Connector/NET component.

There is also another Zip file available for download called `mysql-connector-net-`*`version`*`-src.zip`. This file contains the source code distribution.

The file contains the following directories:

- `Documentation` - This folder contains the source files to build the documentation into the compiled HTML (CHM) format.
- `Installer` - This folder contains the source files to build the Connector/NET installer program.
- `MySql.Data` - This folder contains the source files for the core data provider.
- `MySql.VisualStudio` - This folder contains the source files for the Microsoft Visual Studio extensions.
- `MySql.Web` - This folder contains the source files for the web providers. This includes code for the membership provider, role provider and profile provider. These are used in ASP.NET web sites.
- `Samples` - This folder contains the source files for several example applications.
- `Tests` - Ths folder contains a spreadsheet listing test cases.
- `VisualStudio` - Contains resources used by the Visual Studio plug in.

Finally, you need to ensure that `MySql.Data.dll` is accessible to your program at build time (and run time). If using Microsoft Visual Studio you will need to add `MySql.Data` as a Reference to your project.

2.2.2. Installing Connector/NET on Unix with Mono

There is no installer available for installing the Connector/NET component on your Unix installation. Before installing, please ensure that you have a working Mono project installation. You can test whether your system has Mono installed by typing:

```
shell> mono --version
```

The version of the Mono JIT compiler will be displayed.

To compile C# source code you will also need to make sure a Mono C# compiler, is installed. Note that there are two Mono C# compilers available, `mcs`, which accesses the 1.0-profile libraries, and `gmcs`, which acesses the 2.0-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` directory. Ensure the file `MySql.Data.dll` is present.

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
```

Note, the assemblies that need to be referenced will depend on the requirements of the application, but applications using Connector/NET will need to provide `-r:MySql.Data` as a minimum.

You can further check your installation by running the compiled program, for example:

```
shell> mono HelloWorld.exe
```

2.2.3. Installing Connector/NET from the source code

Caution

You should read this section only if you are interested in helping us test our new code. If you just want to get Connector/NET up and running on your system, you should use a standard release distribution.

Obtaining the source code

To be able to access the Connector/NET source tree, you must have Subversion installed. Subversion is freely available from http://subversion.tigris.org/.

The most recent development source tree is available from our public Subversion trees at http://dev.mysql.com/tech-resources/sources.html.

To checkout out the Connector/NET sources, change to the directory where you want the copy of the Connector/NET tree to be stored, then use the following command:

```
shell> svn co http://svn.mysql.com/svnpublic/connector-net
```

Source packages are also available on the downloads page.

Building the source code on Windows

The following procedure can be used to build the connector on Microsoft Windows.

- Obtain the source code, either from the Subversion server, or through one of the prepared source code packages.
- Navigate to the root of the source code tree.
- A Microsoft Visual Studio 2005 solution file is available to build the connector, this is called `MySQL-VS2005.sln`. Click on this file to load the solution into 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 currently not available.

2.3. Visual Studio User Guide

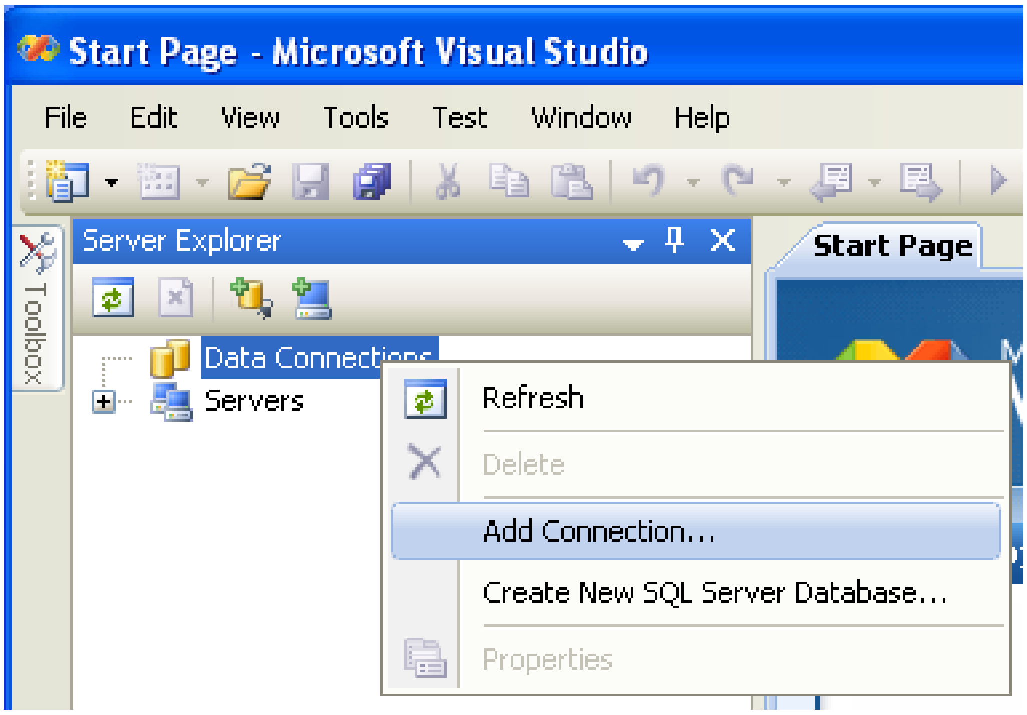
2.3.1. Making a connection

Once the connector is installed, you can use it to create, modify, and delete connections to MySQL databases. To create a connection with a MySQL database, perform the following steps:

- Start Visual Studio, and open the Server Explorer window (VIEW, SERVER EXPLORER option in the main Visual Studio menu, or **Ctrl**+**W**, **L** hot keys).
- Right-click on the Data Connections node, and choose the ADD CONNECTION... menu item.

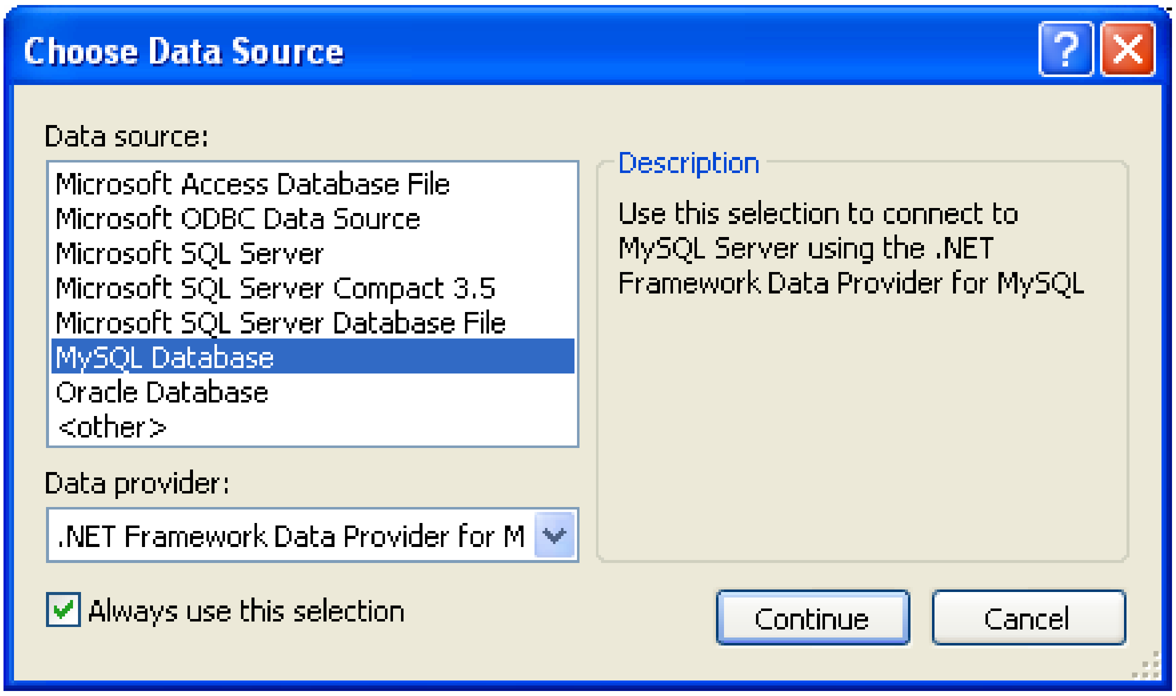
- Add Connection dialog opens. Press the CHANGE button to choose MySQL Database as a data source.

Figure 2.1. Add Connection Context Menu



- Change Data Source dialog opens. Choose MySQL Database in the list of data sources (or the `<other>` option, if MySQL Database is absent), and then choose .NET Framework Data Provider for MySQL in the combo box of data providers.

Figure 2.2. Choose Data Source



Choose Data Source

Data source:

Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Compact 3.5
Microsoft SQL Server Database File
MySQL Database
Oracle Database
<other>

Description

Use this selection to connect to MySQL Server using the .NET Framework Data Provider for MySQL

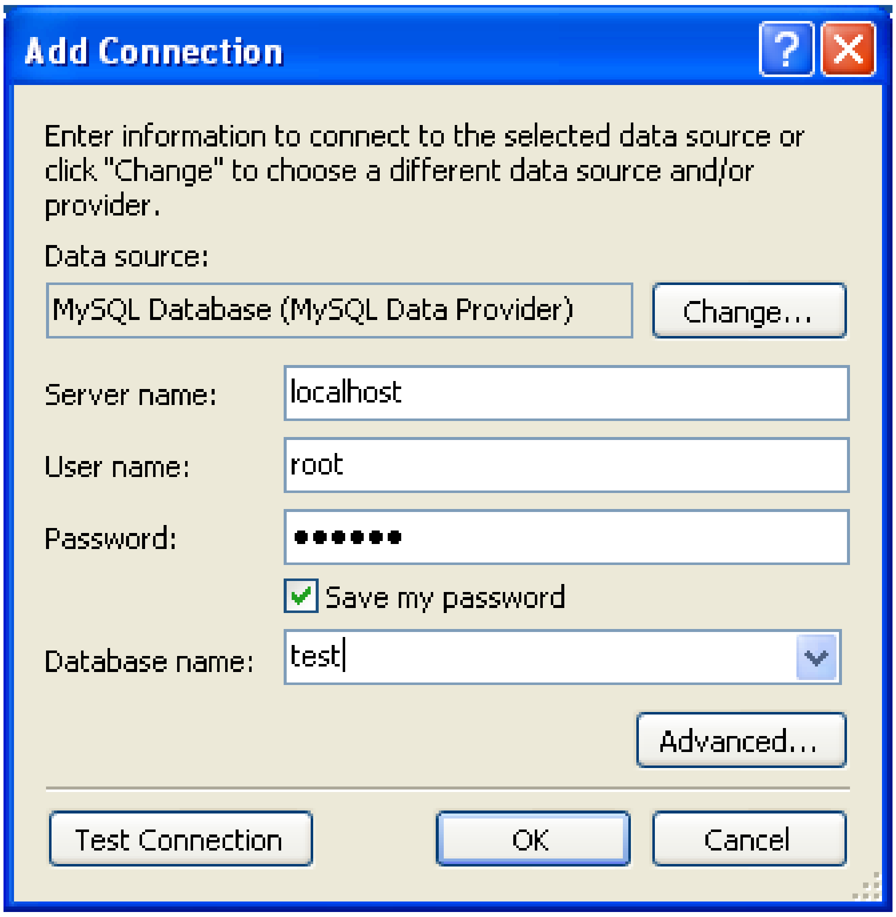
Data provider:

.NET Framework Data Provider for M

[✓] Always use this selection

- Input the connection settings: the server host name (for example, localhost if the MySQL server is installed on the local machine), the user name, the password, and the default schema name. Note that you must specify the default schema name to open the connection.

Figure 2.3. Add Connection Dialog


Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
MySQL Database (MySQL Data Provider)

Server name: localhost

User name: root

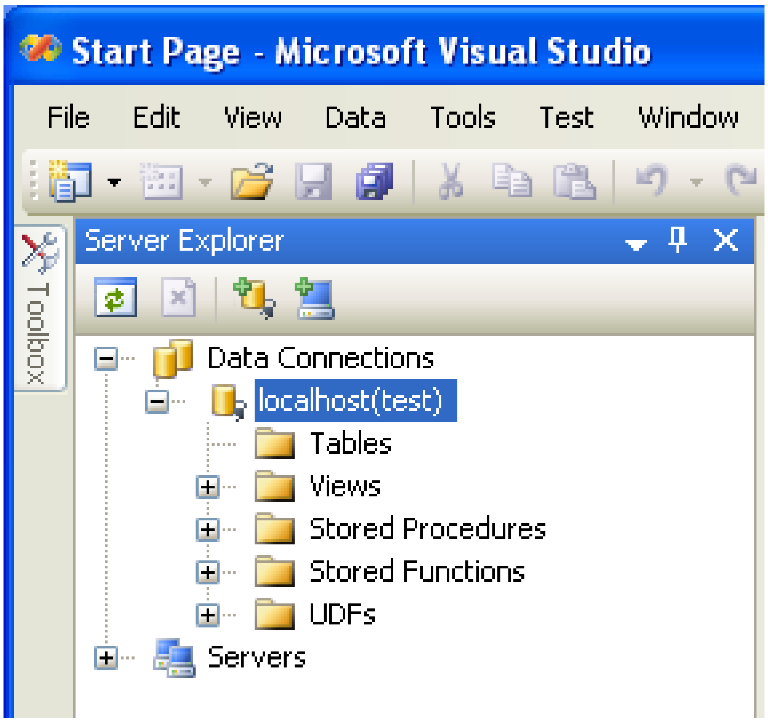
Password: ●●●●●●

✓ Save my password

Database name: test

- You can also set the port to connect with the MySQL server by pressing the ADVANCED button. To test connection with the MySQL server, set the server host name, the user name, and the password, and press the TEST CONNECTION button. If the test succeeds, the success confirmation dialog opens.
- After you set all settings and test the connection, press OK. The newly created connection is displayed in Server Explorer. Now you can work with the MySQL server through standard Server Explorer GUI.

Figure 2.4. New Data Connection



After the connection is successfully established, all settings are saved for future use. When you start Visual Studio for the next time, just open the connection node in Server Explorer to establish a connection to the MySQL server again.

To modify and delete a connection, use the Server Explorer context menu for the corresponding node. You can modify any of the settings just by overwriting the existing values with new ones. Note that the connection may be modified or deleted only if no active editor for its objects is opened: otherwise you may loose your data.

2.3.2. Editing Tables

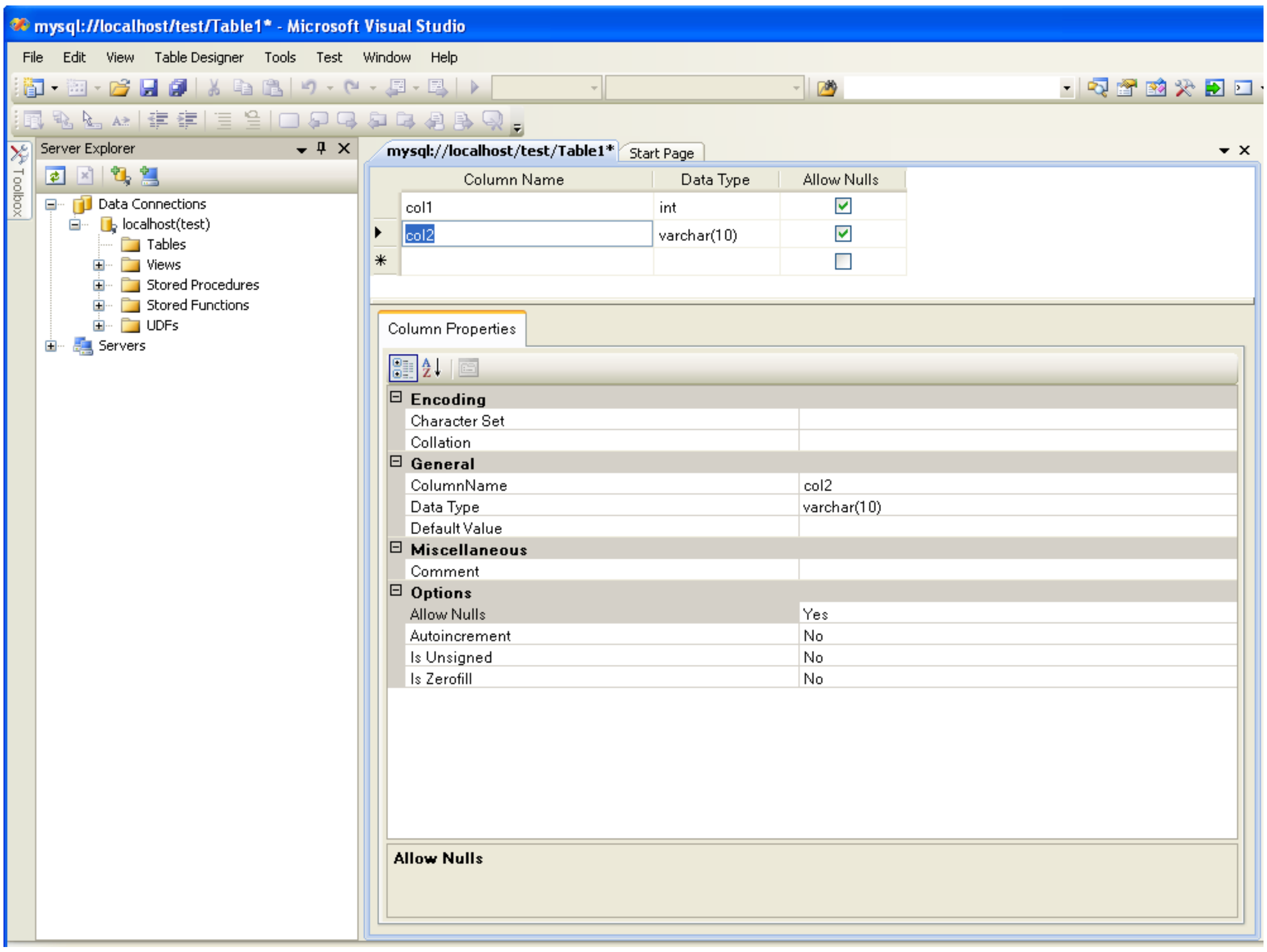
Connector/Net contains a table editor, which enables the visual creation and modification of tables.

The Table Designer can be accessed through a mouse action on table-type node of Server Explorer. To create a new table, right-click on the **Tables** node (under the connection node) and choose the Create Table command from the context menu.

To modify an existing table, double-click on the node of the table you wish to modify, or right-click on this node and choose the Design item from the context menu. Either of the commands opens the Table Designer.

The table editor is implemented in the manner of the well-known Query Browser Table Editor, but with minor differences.

Figure 2.5. Editing New Table


mysql://localhost/test/Table1* - Microsoft Visual Studio
File Edit View Table Designer Tools Test Window Help
Toolbox
Server Explorer
Data Connections
localhost(test)
Tables
Views
Stored Procedures
Stored Functions
UDFs
Servers
mysql://localhost/test/Table1*
Start Page
Column Properties

Column Name	Data Type	Allow Nulls
col1	int	✓
col2	varchar(10)	✓
		☐

Encoding	
Character Set	
Collation	
General	
ColumnName	col2
Data Type	varchar(10)
Default Value	
Miscellaneous	
Comment	
Options	
Allow Nulls	Yes
Autoincrement	No
Is Unsigned	No
Is Zerofill	No

Allow Nulls

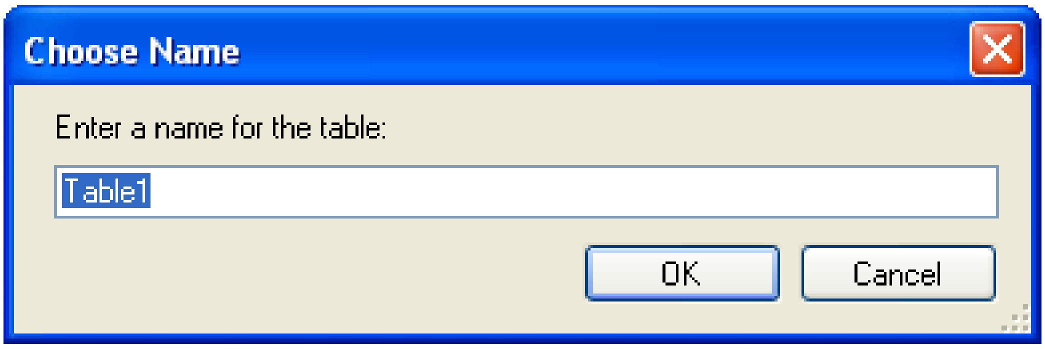
Table Designer consists of the following parts:

- Columns Editor - a data grid on top of the Table Designer. Use the Columns grid for column creation, modification, and deletion.
- Indexes tab - a tab on bottom of the Table Designer. Use the Indexes tab for indexes management.
- Foreign Keys tab - a tab on bottom of the Table Designer. Use the Foreign Keys tab for foreign keys management.
- Column Details tab - a tab on bottom of the Table Designer. Use the Column Details tab to set advanced column options.
- Properties window - a standard Visual Studio Properties window, where the properties of the edited table are displayed. Use the Properties window to set the table properties.

Each of these areas is discussed in more detail in subsequent sections.

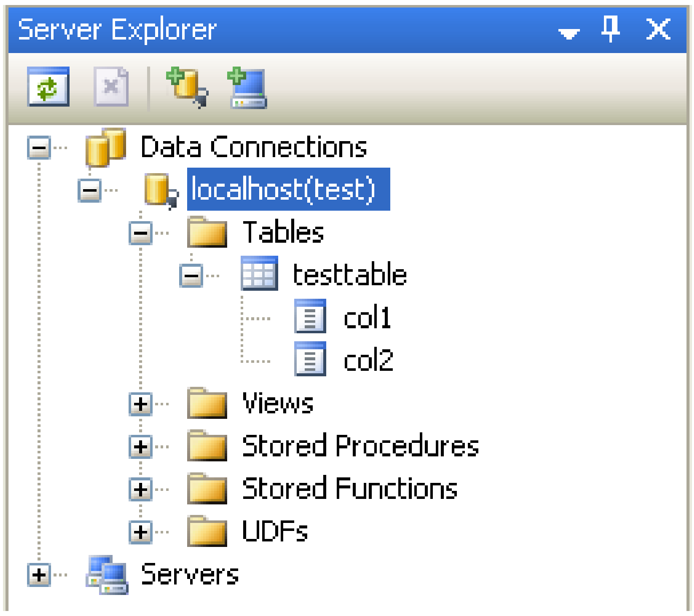
To save changes you have made in the Table Designer, use either SAVE or SAVE ALL button of the Visual Studio main toolbar, or just press **Ctrl+S**. If you have not already named the table you will be prompted to do so.

Figure 2.6. Choose Table Name



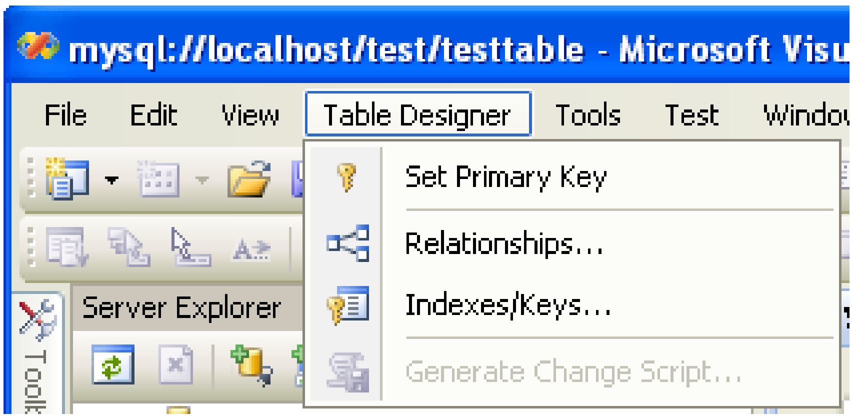
Once created you can view the table in the Server Explorer.

Figure 2.7. Newly Created Table



The Table Designer main menu allows you to set a Primary Key column, edit Relationships such as Foreign Keys, and create Indexes.

Figure 2.8. Table Designer Main Menu



2.3.2.1. Column Editor

You can use the Column Editor to set or change the name, data type, default value, and other properties of a table column. To set the focus to a needed cell of a grid, use the mouse click. Also you can move through the grid using **Tab** and **Shift**+**Tab** keys.

To set or change the name, data type, default value and comment of a column, activate the appropriate cell and type the desired value.

To set or unset flag-type column properties (`NOT NULL`, auto incremented, flags), check or uncheck the corresponding check boxes. Note that the set of column flags depends on its data type.

To reorder columns, index columns or foreign key columns in the Column Editor, select the whole column you wish to reorder by clicking on the selector column on the left of the column grid. Then move the column by using **Ctrl**+**Up** (to move the column up) or **Ctrl**+**Down** (to move the column down) keys.

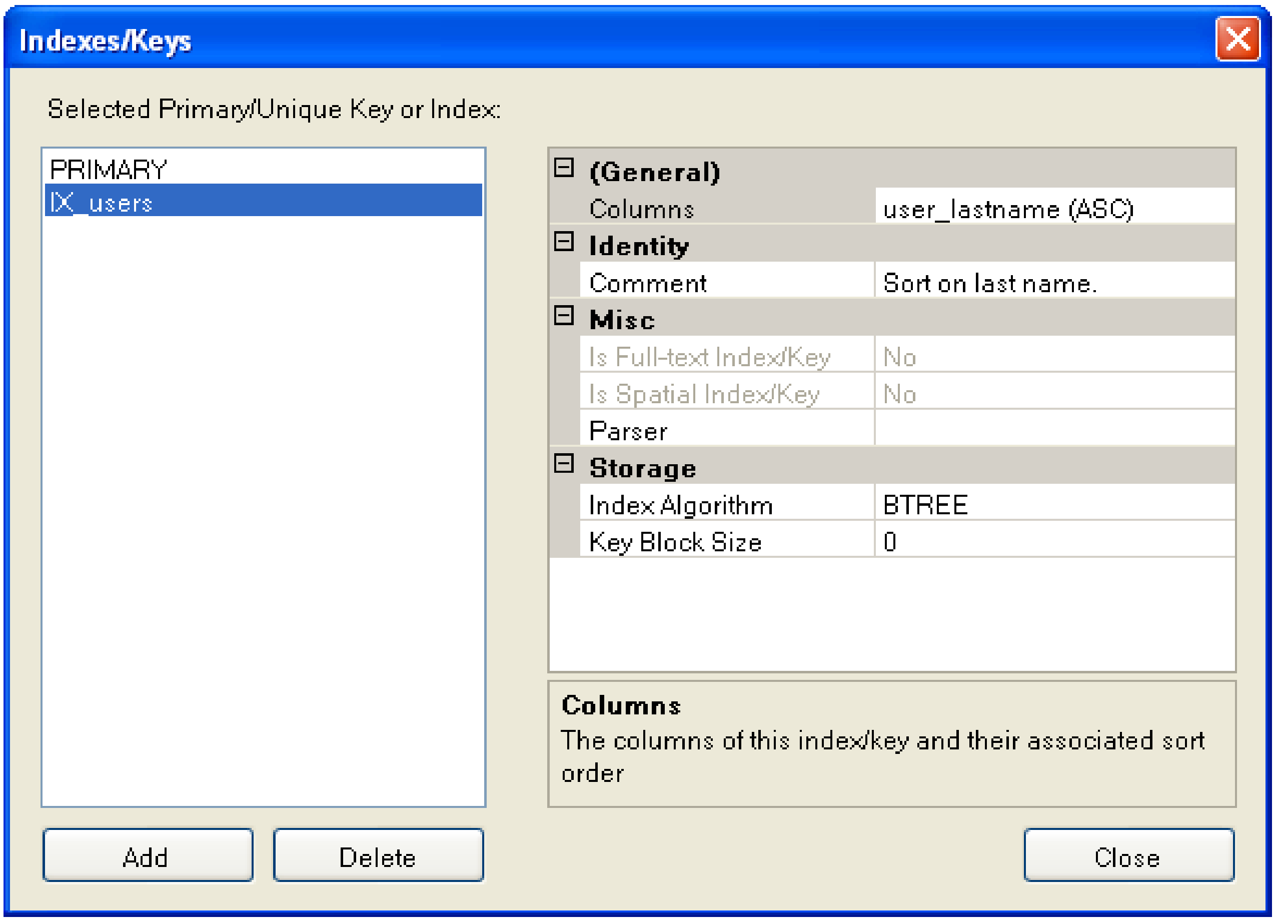
To delete a column, select it by clicking on the selector column on the left of the column grid, then press the **Delete** button on a keyboard.

2.3.2.2. Editing Indexes

Indexes management is performed via the **INDEXES/KEYS** dialog.

To add an index, select TABLE DESIGNER, INDEXES/KEYS... from the main menu, and click ADD to add a new index. You can then set the index name, index kind, index type, and a set of index columns.

Figure 2.9. Indexes Dialog



PRIMARY
IX_users

(General)	
Columns	user_lastname (ASC)
Identity	
Comment	Sort on last name.
Misc	
Is Full-text Index/Key	No
Is Spatial Index/Key	No
Parser	
Storage	
Index Algorithm	BTREE
Key Block Size	0

Columns
The columns of this index/key and their associated sort order

To remove an index, select it in the list box on the left, and click the DELETE button.

To change index settings, select the needed index in the list box on the left. The detailed information about the index is displayed in the panel on the right hand side. Change the desired values.

2.3.2.3. Editing Foreign Keys

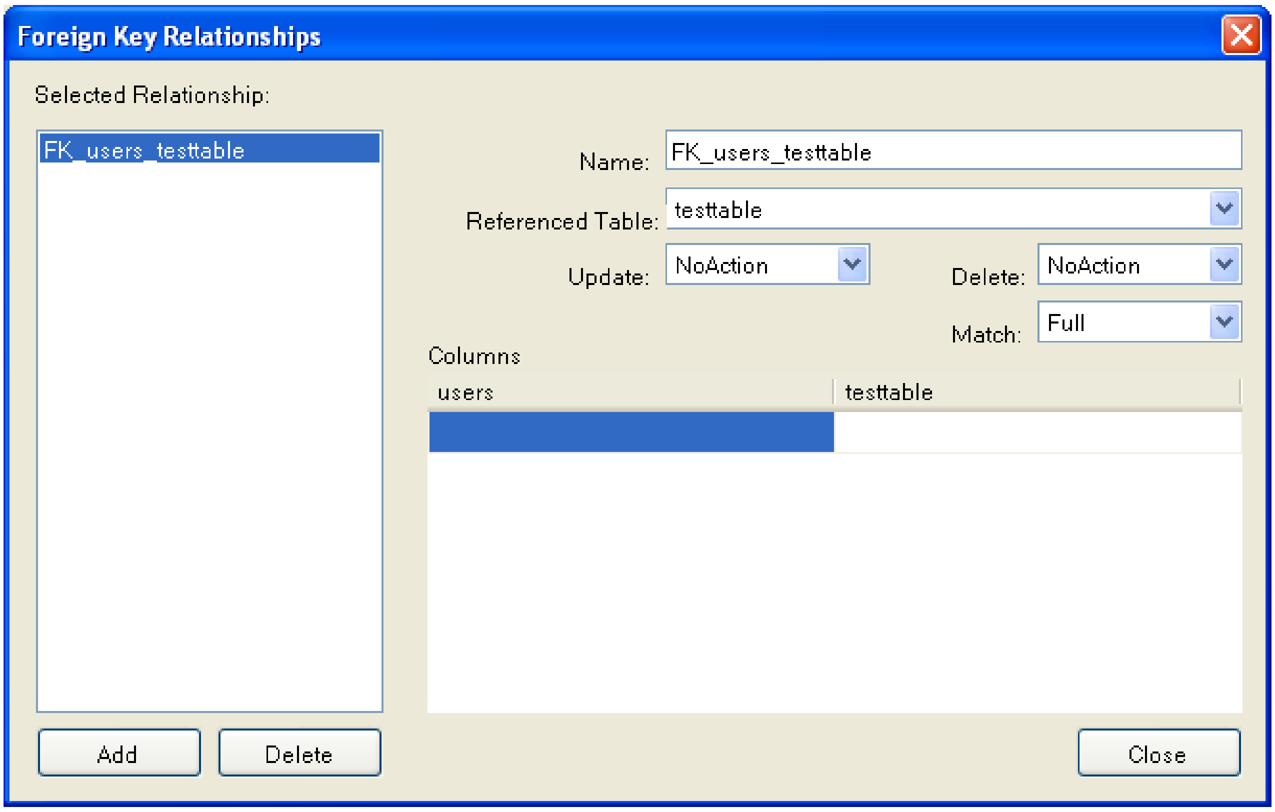
Foreign Keys management is performed via the **FOREIGN KEY RELATIONSHIPS** dialog.

To add a foreign key, select TABLE DESIGNER, RELATIONSHIPS... from the main menu. This displays the **FOREIGN KEY RELATIONSHIP** dialog. Click ADD. You can then set the foreign key name, referenced table name, foreign key columns, and actions upon update and delete.

To remove a foreign key, select it in the list box on the left, and click the DELETE button.

To change foreign key settings, select the required foreign key in the list box on the left. The detailed information about the foreign key is displayed in the right hand panel. Change the desired values.

Figure 2.10. Foreign Key Relationships Dialog



FK_users_testtable

Name: FK_users_testtable

Referenced Table: testtable

Update: NoAction

Delete: NoAction

Match: Full

users	testtable

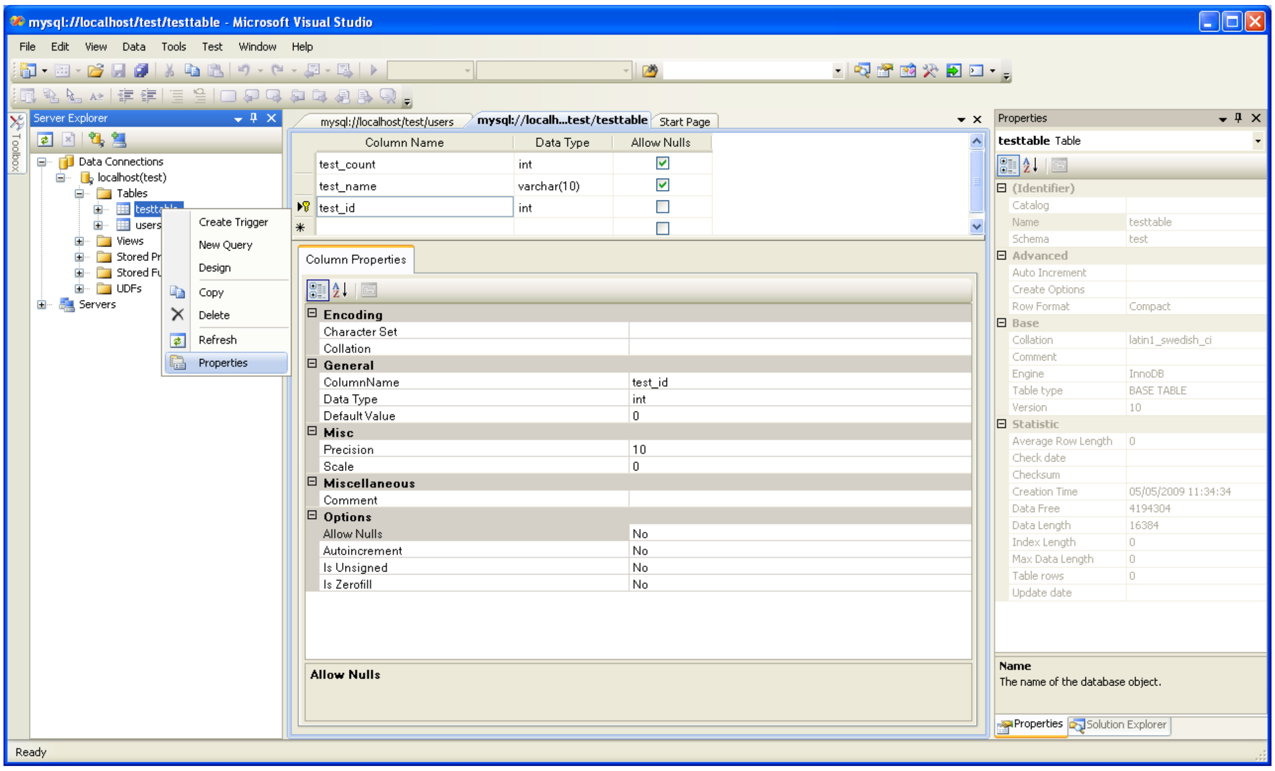
2.3.2.4. Column Properties

The **COLUMN PROPERTIES** tab can be used to set column options. In addition to the general column properties presented in the Column Editor, in the **COLUMN PROPERTIES** tab you can set additional properties such as Character Set, Collation and Precision.

2.3.2.5. Table Properties

To bring up Table Properties select the table and right click to activate the context menu. Select PROPERTIES. The **TABLE PROPERTIES** dockable window will be displayed.

Figure 2.11. Table Properties Menu Item



Column Name	Data Type	Allow Nulls
test_count	int	✓
test_name	varchar(10)	✓
test_id	int	☐

Encoding	
Character Set	
Collation	
General	
ColumnName	test_id
Data Type	int
Default Value	0
Misc	
Precision	10
Scale	0
Miscellaneous	
Comment	
Options	
Allow Nulls	No
Autoincrement	No
Is Unsigned	No
Is Zerofill	No

Allow Nulls

testtable Table

(Identifier)	
Catalog	
Name	testtable
Schema	test
Advanced	
Auto Increment	
Create Options	
Row Format	Compact
Base	
Collation	latin1_swedish_ci
Comment	
Engine	InnoDB
Table type	BASE TABLE
Version	10
Statistic	
Average Row Length	0
Check date	
Checksum	
Creation Time	05/05/2009 11:34:34
Data Free	4194304
Data Length	16384
Index Length	0
Max Data Length	0
Table rows	0
Update date	

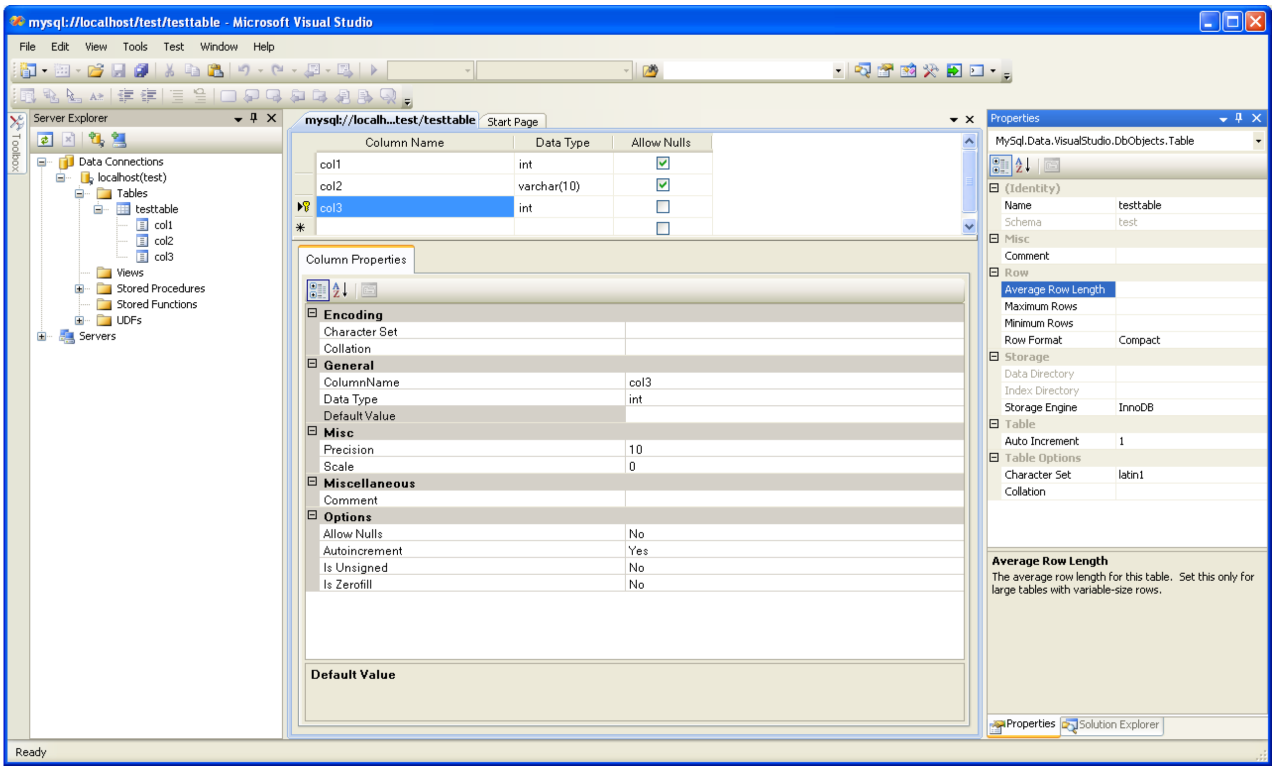
Name
The name of the database object.

The following table properties can be set:

- Auto Increment
- Average Row Length
- Character Set
- Collation
- Comment
- Data Directory
- Index Directory
- Maximum Rows
- Minimum Rows
- Name
- Row Format
- Schema
- Storage Engine

The property `Schema` is read only.

Figure 2.12. Table Properties



Column Name	Data Type	Allow Nulls
col1	int	✓
col2	varchar(10)	✓
col3	int	☐

Column Properties

Encoding	
Character Set	
Collation	
General	
ColumnName	col3
Data Type	int
Default Value	
Misc	
Precision	10
Scale	0
Miscellaneous	
Comment	
Options	
Allow Nulls	No
Autoincrement	Yes
Is Unsigned	No
Is Zerofill	No

Default Value

Properties: MySql.Data.VisualStudio.DbObjects.Table

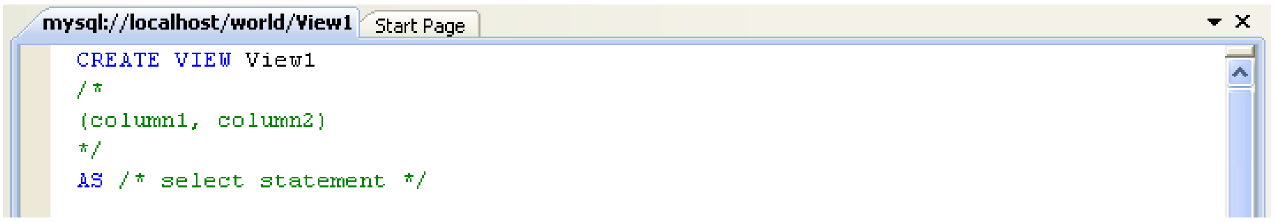
(Identity)	
Name	testtable
Schema	test
Misc	
Comment	
Row	
Average Row Length	
Maximum Rows	
Minimum Rows	
Row Format	Compact
Storage	
Data Directory	
Index Directory	
Storage Engine	InnoDB
Table	
Auto Increment	1
Table Options	
Character Set	latin1
Collation	

Average Row Length
The average row length for this table. Set this only for large tables with variable-size rows.

2.3.3. Editing Views

To create a new view, right click the Views node under the connection node in Server Explorer. From the node's context menu, choose the CREATE VIEW command. This command opens the SQL Editor.

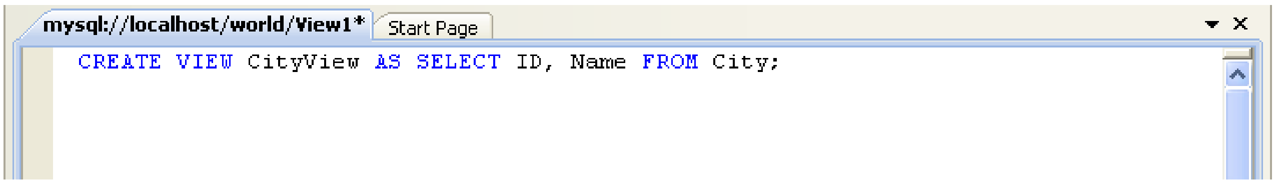
Figure 2.13. Editing View SQL



```
CREATE VIEW View1
/*
(column1, column2)
*/
AS /* select statement */
```

You can then enter the SQL for your view.

Figure 2.14. View SQL Added



```
CREATE VIEW CityView AS SELECT ID, Name FROM City;
```

To modify an existing view, double click on a node of the view you wish to modify, or right click on this node and choose the ALTER VIEW command from a context menu. Either of the commands opens the SQL Editor.

All other view properties can be set in the Properties window. These properties are:

- Catalog
- Check Option

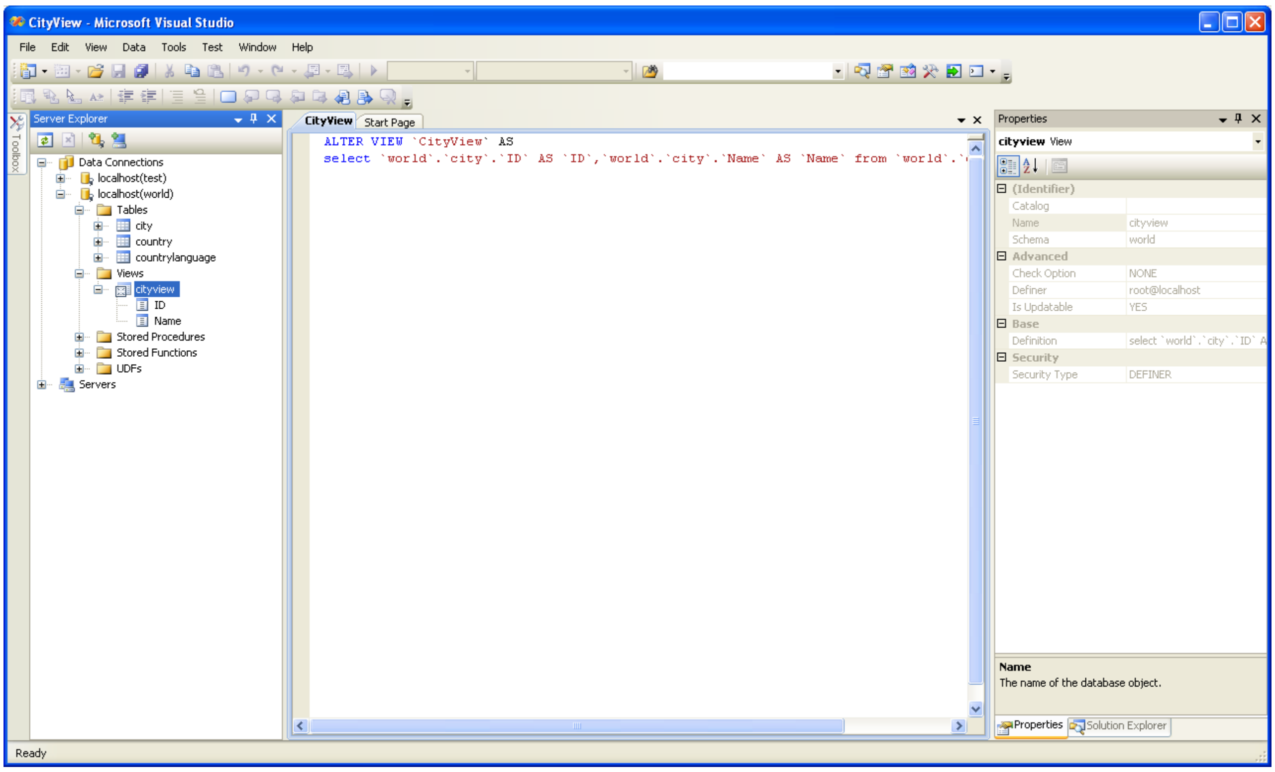
- Definer
- Definition
- Definer
- Is Updateable
- Name
- Schema
- Security Type

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case you set the desired value with an embedded combobox.

The properties `Is Updatable` and `Schema` are readonly.

To save changes you have made, use either SAVE or SAVE ALL buttons of the Visual Studio main toolbar, or just press **Ctrl+S**.

Figure 2.15. View SQL Saved



```
ALTER VIEW `CityView` AS
select `world`.`city`.`ID` AS `ID`,`world`.`city`.`Name` AS `Name` from `world`.`
```

cityview View

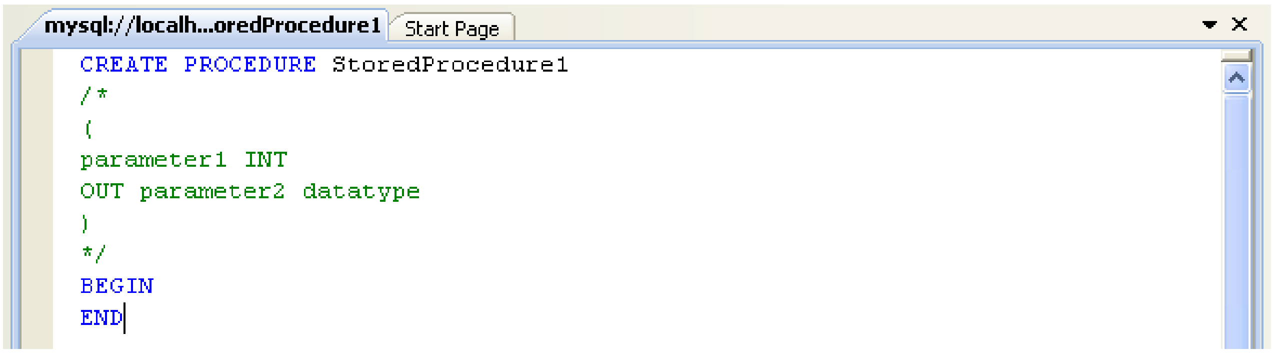
(Identifier)	
Catalog	
Name	cityview
Schema	world
Advanced	
Check Option	NONE
Definer	root@localhost
Is Updatable	YES
Base	
Definition	select `world`.`city`.`ID` A
Security	
Security Type	DEFINER

Name
The name of the database object.

2.3.4. Editing Stored Procedures and Functions

To create a new stored procedure, right-click on the **STORED PROCEDURES** node under the connection node in Server Explorer. From the node's context menu, choose the **CREATE ROUTINE** command. This command opens the SQL Editor.

Figure 2.16. Edit Stored Procedure SQL



```
CREATE PROCEDURE StoredProcedure1
/*
(
parameter1 INT
OUT parameter2 datatype
)
*/
BEGIN
END
```

To create a new stored function, right-click on the **Functions** node under the connection node in Server Explorer. From the node's context menu, choose the **Create Routine** command.

To modify an existing stored routine (procedure or function), double-click on the node of the routine you wish to modify, or right-click on this node and choose the **Alter Routine** command from the context menu. Either of the commands opens the SQL Editor.

To create or alter the routine definition using SQL Editor, type this definition in the SQL Editor using standard SQL. All other routine properties can be set in the Properties window. These properties are:

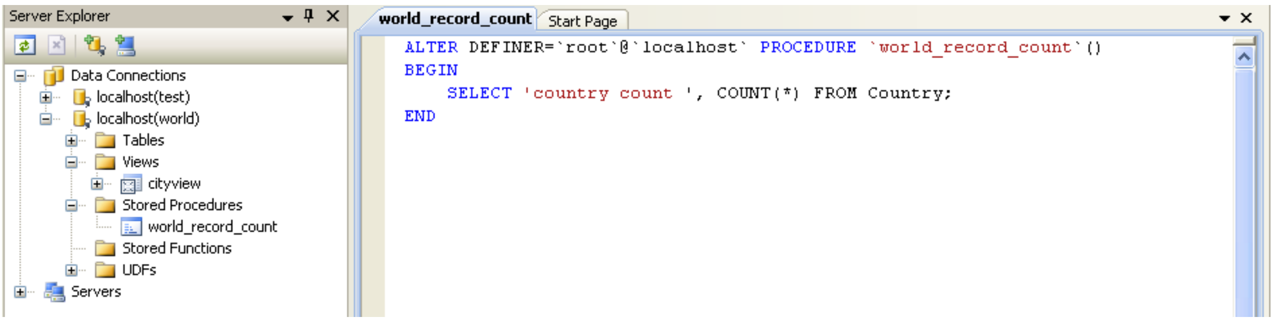
- Body
- Catalog
- Comment
- Creation Time
- Data Access
- Definer
- Definition
- External Name
- External Language
- Is Deterministic
- Last Modified
- Name
- Parameter Style
- Returns
- Schema
- Security Type
- Specific Name
- SQL Mode
- SQL Path
- Type

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case set the desired value using the embedded combo box.

You can also set all the options directly in the SQL Editor, using the standard `CREATE PROCEDURE` or `CREATE FUNCTION` statement. However, it is recommended to use the Properties window instead.

To save changes you have made, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or just press **Ctrl**+**S**.

Figure 2.17. Stored Procedure SQL Saved



```
ALTER DEFINER=`root`@`localhost` PROCEDURE `world_record_count`()
BEGIN
    SELECT 'country count ', COUNT(*) FROM Country;
END
```

2.3.5. Editing Triggers

To create a new trigger, right-click on the node of the table, for which you wish to add a trigger. From the node's context menu, choose the **Create Trigger** command. This command opens the SQL Editor.

To modify an existing trigger, double-click on the node of the trigger you wish to modify, or right-click on this node and choose the **Alter Trigger** command from the context menu. Either of the commands opens the SQL Editor.

To create or alter the trigger definition using SQL Editor, type the trigger statement in the SQL Editor using standard SQL.

Note

You should enter only the trigger statement, that is, the part of the `CREATE TRIGGER` query that is placed after the `FOR EACH ROW` clause.

All other trigger properties are set in the Properties window. These properties are:

- Definer
- Event Manipulation
- Name
- Timing

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case set the desired value using the embedded combo box.

The properties `Event Table`, `Schema`, and `Server` in the Properties window are read only.

To save changes you have made, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl**+**S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

2.3.6. Editing User Defined Functions (UDF)

To create a new User Defined Function (UDF), right-click on the **UDFs** node under the connection node in Server Explorer. From the node's context menu, choose the Create UDF command. This command opens the UDF Editor.

To modify an existing UDF, double-click on the node of the UDF you wish to modify, or right-click on this node and choose the Alter UDF command from the context menu. Either of the commands opens the UDF Editor.

The UDF editor allows you to set the following properties:

- Name
- So-name (DLL name)
- Return type

- Is Aggregate

There are text fields for both names, a combo box for the return type, and a check box to indicate if the UDF is aggregate. All these options are also accessible via the Properties window.

The property `Server` in the Properties window is read only.

To save changes you have made, use either SAVE or SAVE ALL buttons of the Visual Studio main toolbar, or just press **Ctrl**+**S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

2.3.7. Cloning Database Objects

Tables, views, stored procedures, and functions can be cloned using the appropriate Clone command from the context menu: CLONE TABLE, CLONE VIEW, CLONE ROUTINE. The clone commands open the corresponding editor for a new object: the **TABLE EDITOR** for cloning a table, and the **SQL EDITOR** for cloning a view or a routine.

The editor is filled with values of the original object. You can modify these values in a usual manner.

To save the cloned object, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl**+**S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

2.3.8. Dropping Database Objects

Tables, views, stored routines, triggers, and UDFs can be dropped with the appropriate Drop command selected from its context menu: DROP TABLE, DROP VIEW, DROP ROUTINE, DROP TRIGGER, DROP UDF.

You will be asked to confirm the execution of the corresponding drop query in a confirmation dialog.

Dropping of multiple objects is not supported.

2.3.9. Using the ADO.NET Entity Framework

Connector/NET 6.0 introduced support for the ADO.NET Entity Framework. ADO.NET Entity Framework was included with .NET Framework 3.5 Service Pack 1, and Visual Studio 2008 Service Pack 1. ADO.NET Entity Framework was released on 11th August 2008.

ADO.NET Entity Framework provides an Object Relational Mapping (ORM) service, mapping the relational database schema to objects. The ADO.NET Entity Framework defines several layers, these can be summarized as:

- **Logical** - this layer defines the relational data and is defined by the Store Schema Definition Language (SSDL).
- **Conceptual** - this layer defines the .NET classes and is defined by the Conceptual Schema Definition Language (CSDL)
- **Mapping** - this layer defines the mapping from .NET classes to relational tables and associations, and is defined by Mapping Specification Language (MSL).

Connector/NET integrates with Visual Studio 2008 to provide a range of helpful tools to assist the developer.

A full treatment of ADO.NET Entity Framework is beyond the scope of this manual. You are encouraged to review the Microsoft ADO.NET Entity Framework documentation.

2.3.9.1. 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 example database, 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 Section 2.3.1, “Making a connection”.

Creating a new Windows Forms application

The first step is to create a new Windows Forms application.

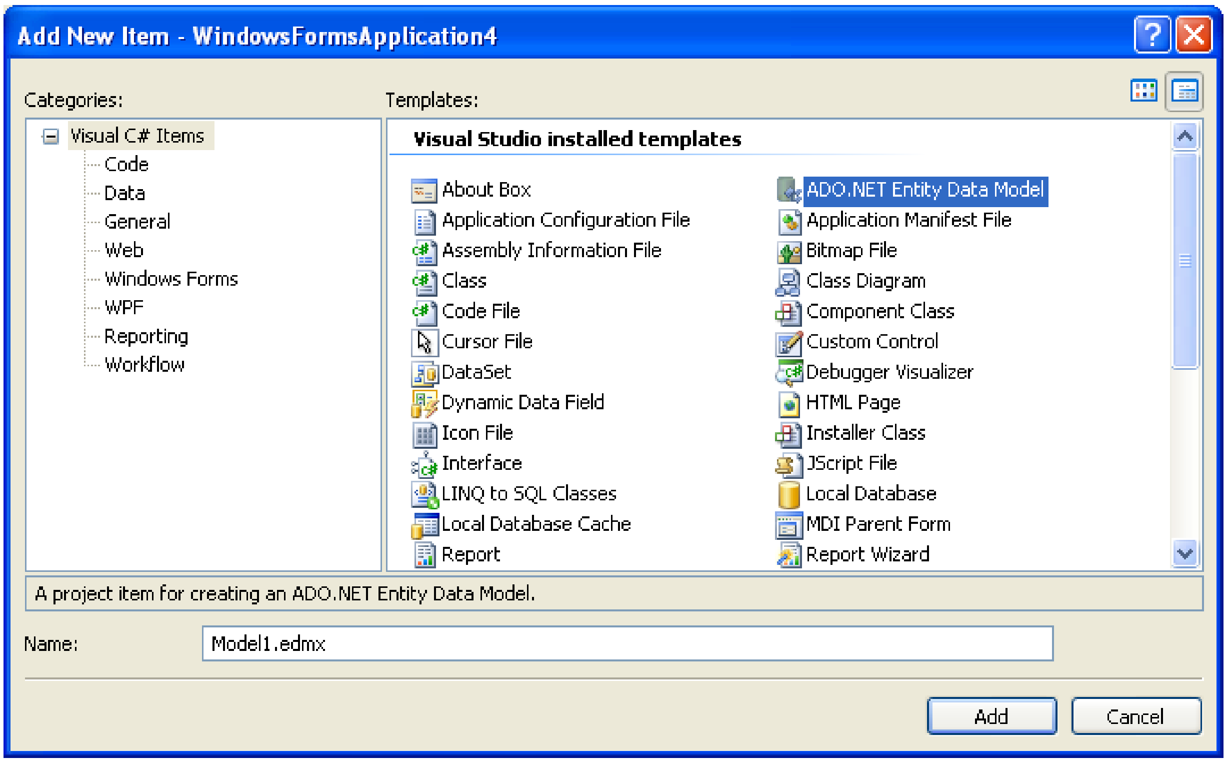
1. In Visual Studio, select FILE, NEW, PROJECT from the main menu.
2. Choose the **WINDOWS FORMS APPLICATION** installed template. Click OK. The solution is created.

Adding an Entity Data Model

You will now add an Entity Data Model to your solution.

1. In the Solution Explorer, right click on your application and select ADD, NEW ITEM.... From VISUAL STUDIO INSTALLED TEMPLATES select **ADO.NET ENTITY DATA MODEL**. Click ADD.

Figure 2.18. Add Entity Data Model

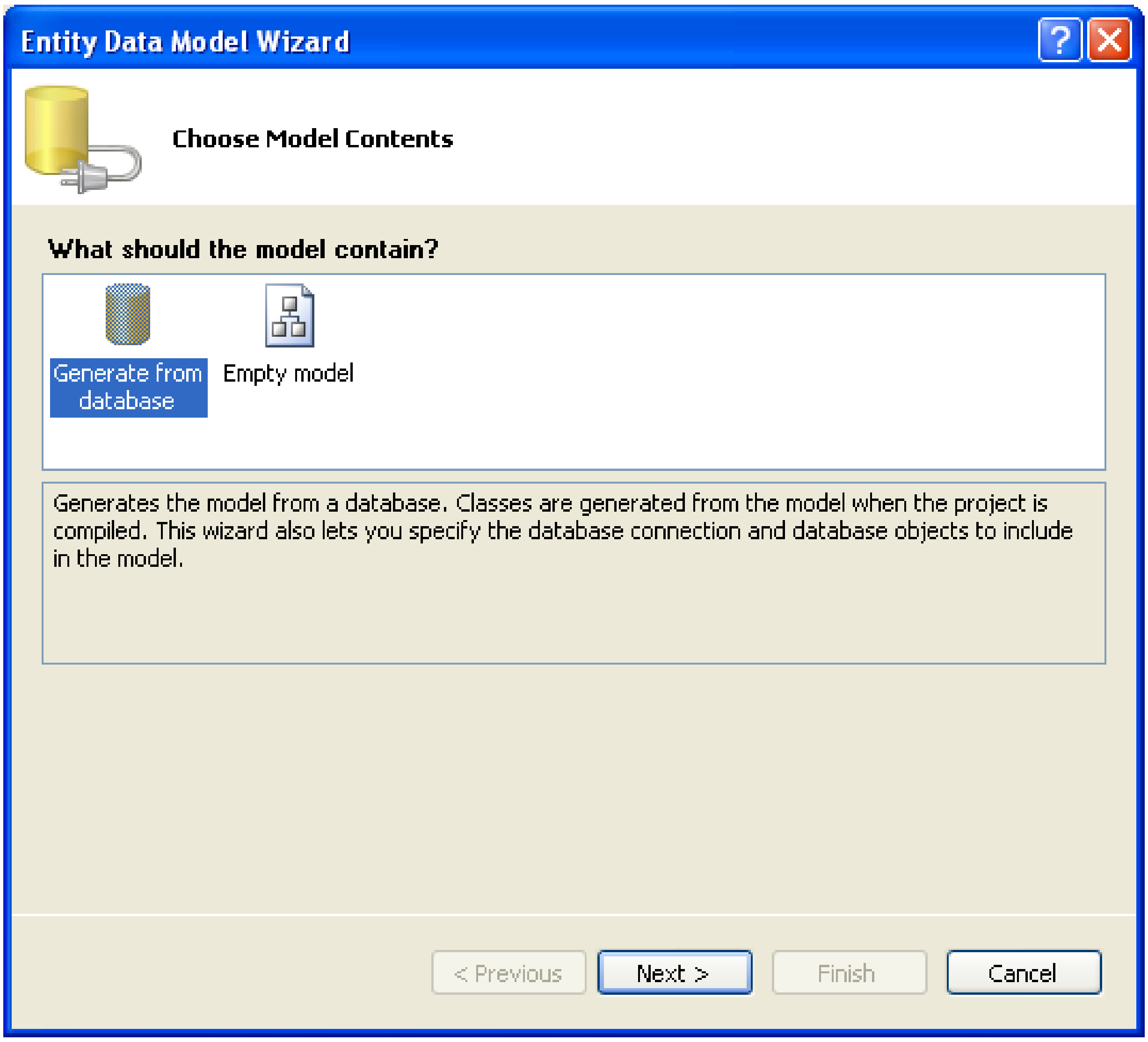


A project item for creating an ADO.NET Entity Data Model.

Name: Model1.edmx

2. You will now see the Entity Data Model Wizard. You will use the wizard to generate the Entity Data Model from the world example database. Select the icon **GENERATE FROM DATABASE**. Click NEXT.

Figure 2.19. Entity Data Model Wizard Screen 1

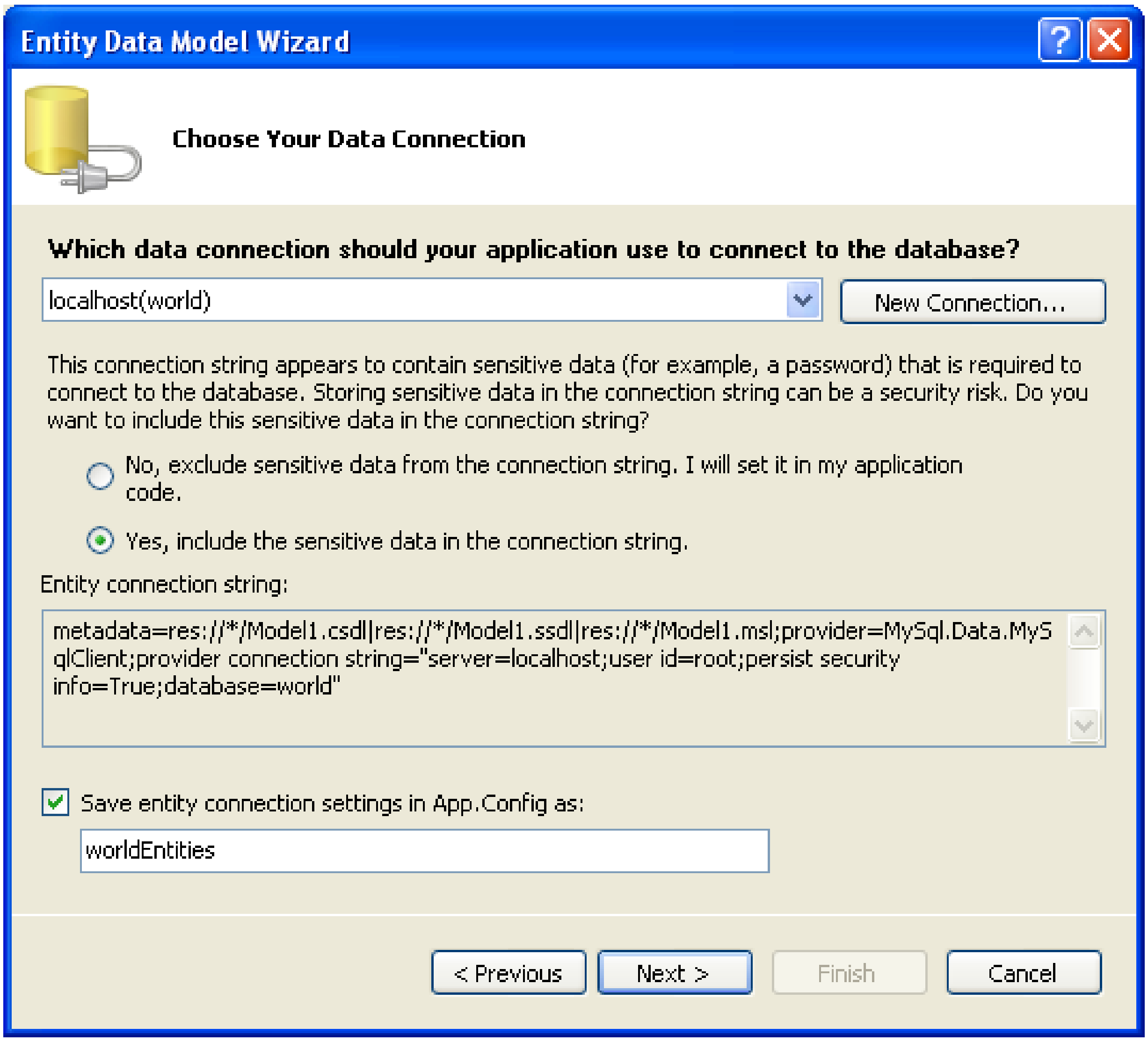


What should the model contain?

Generates the model from a database. Classes are generated from the model when the project is compiled. This wizard also lets you specify the database connection and database objects to include in the model.

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 on NEW CONNECTION.... For further instructions on creating a connection to a database see Section 2.3.1, “Making a connection”.

Figure 2.20. Entity Data Model Wizard Screen 2



Which data connection should your application use to connect to the database?

localhost(world)

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?

No, exclude sensitive data from the connection string. I will set it in my application code. [✓ Yes, include the sensitive data in the connection string.]

Entity connection 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"
```

✓ Save entity connection settings in App.Config as:

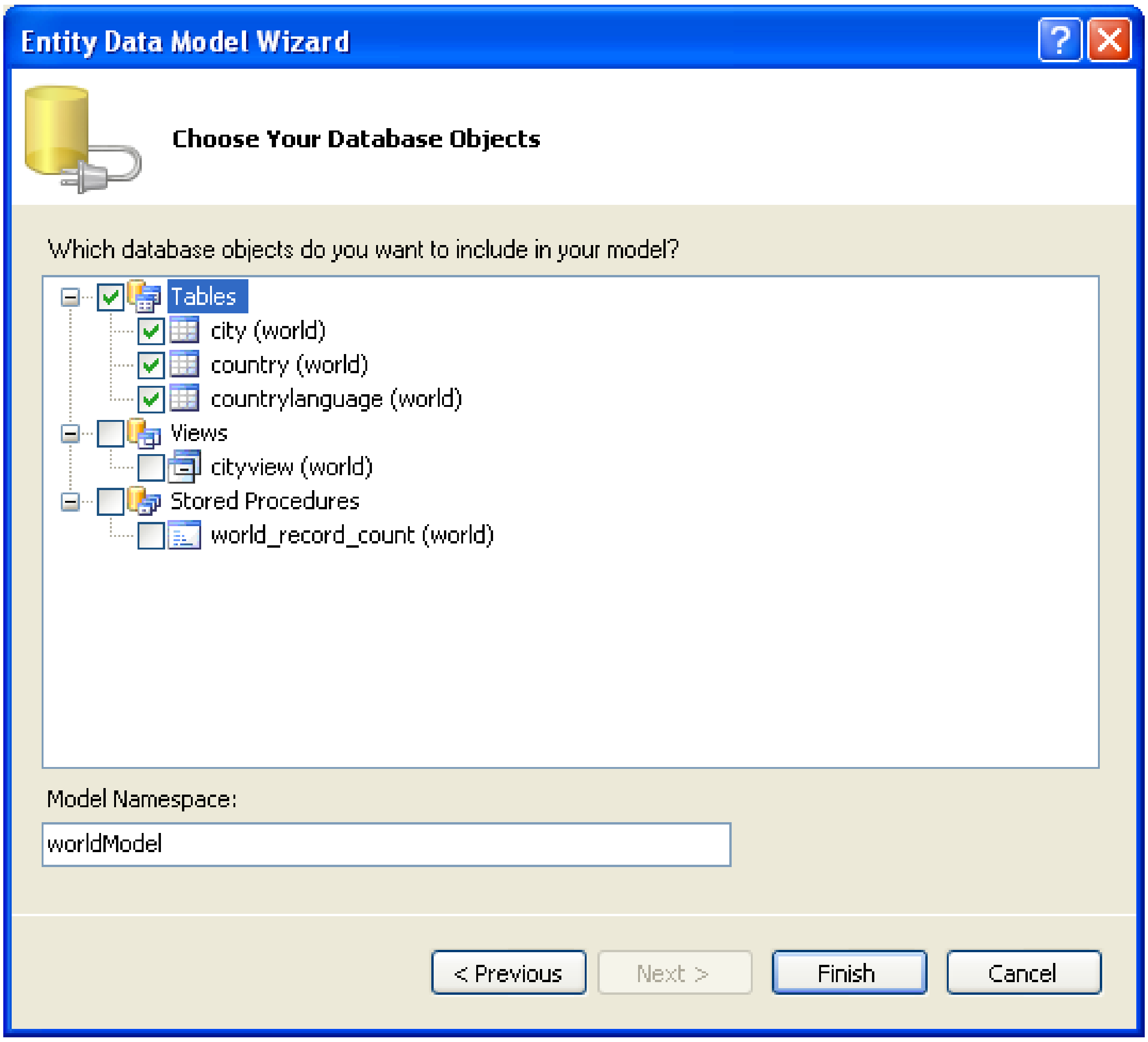
worldEntities

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 2.21. Entity Data Model Wizard Screen 3



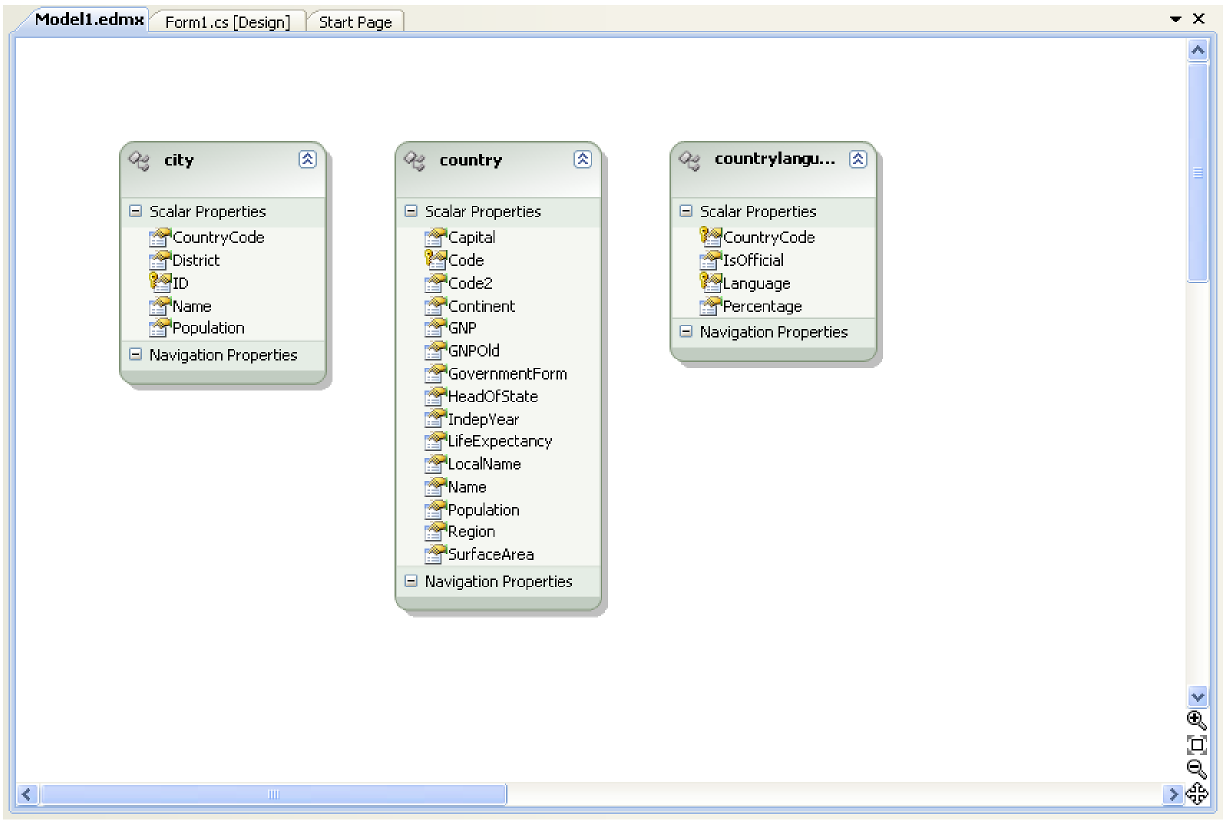
Which database objects do you want to include in your model?

- [✓] Tables
  - [✓] city (world)
  - [✓] country (world)
  - [✓] countrylanguage (world)
- [ ] Views
  - [ ] cityview (world)
- [ ] Stored Procedures
  - [ ] world_record_count (world)

Model Namespace: worldModel

7. Visual Studio will generate the model and then display it.

Figure 2.22. Entity Data Model Diagram



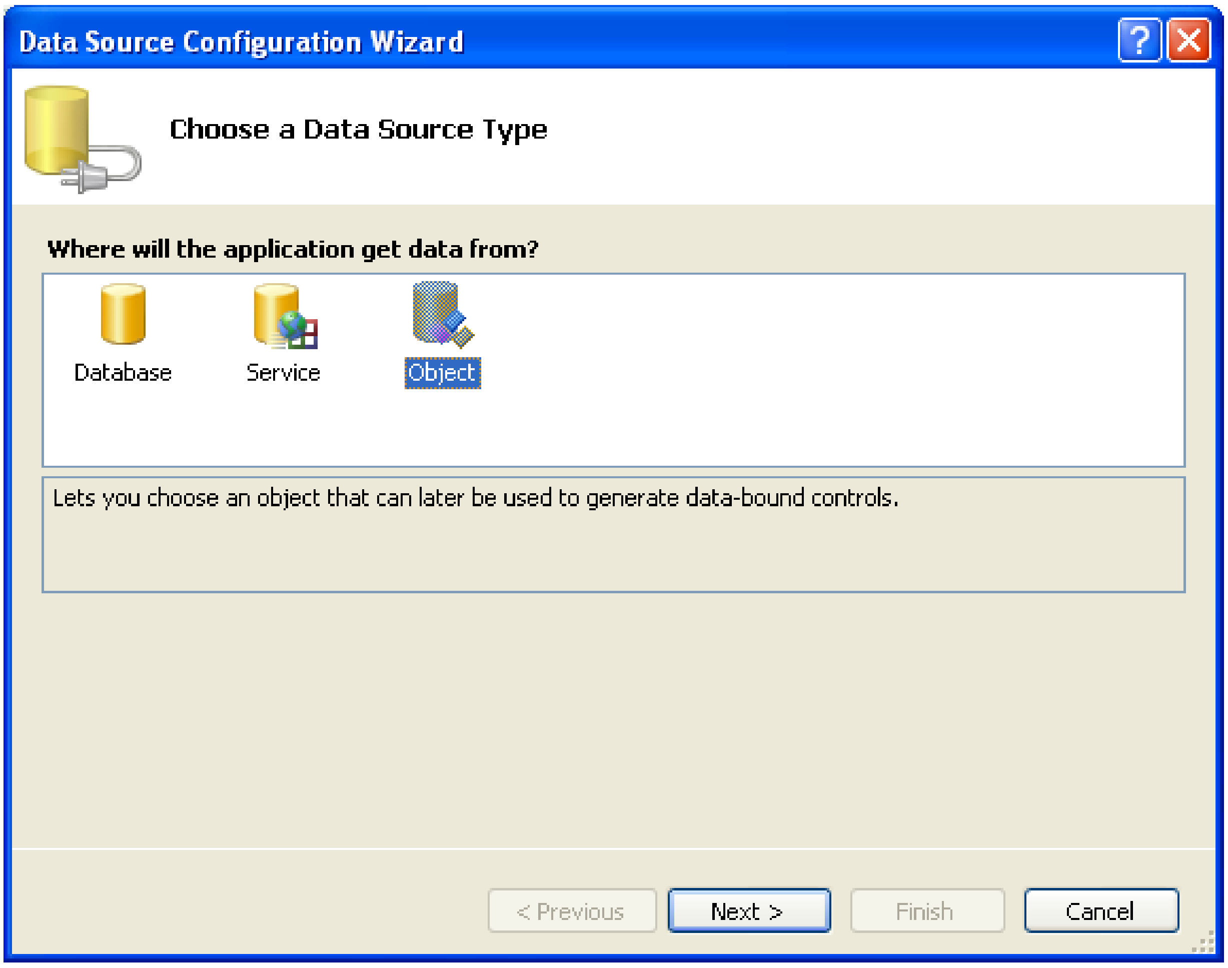
8. From the Visual Studio main menu select BUILD, BUILD SOLUTION, to ensure that everything compiles correctly so far.

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.

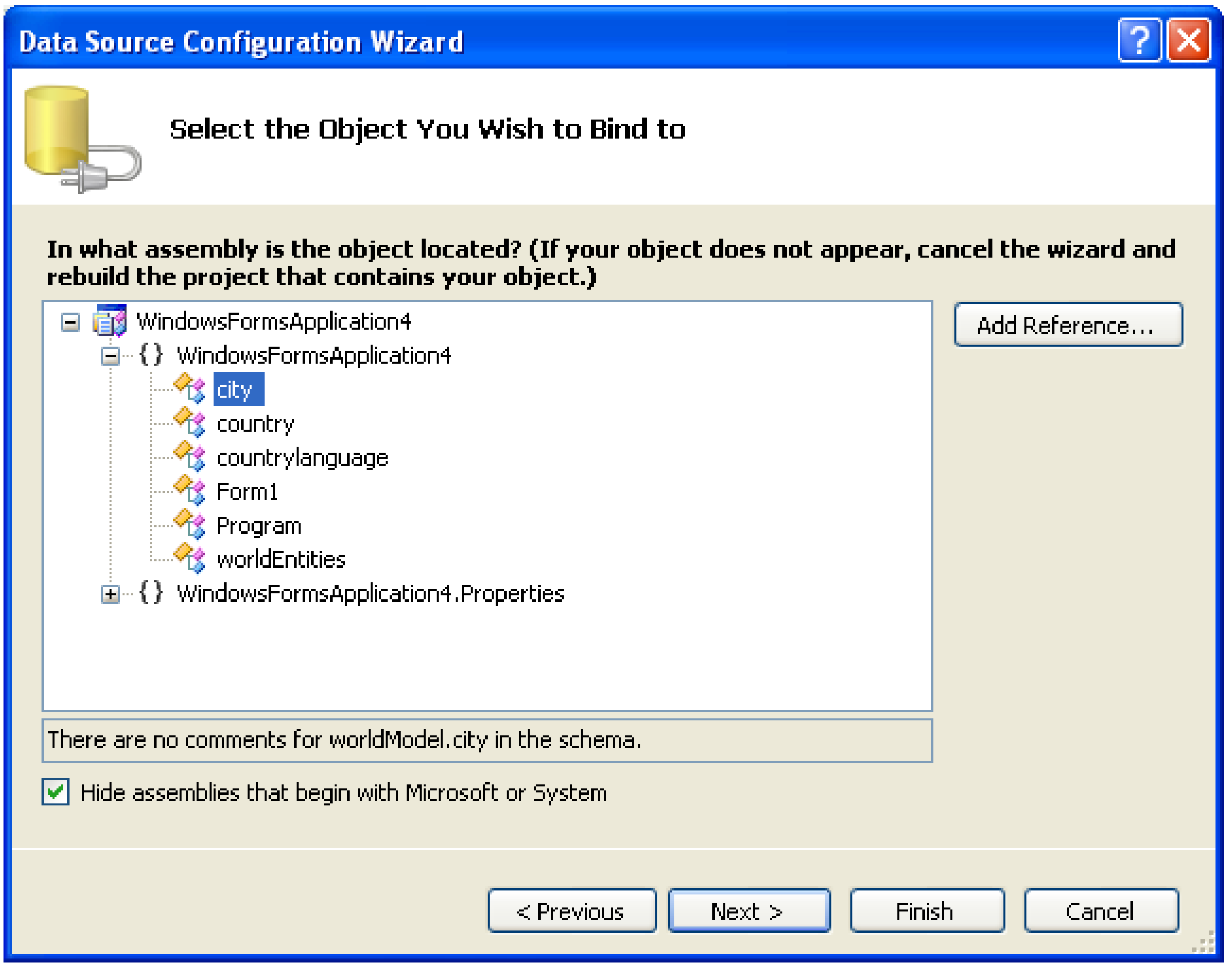
   Figure 2.23. Entity Data Source Configuration Wizard Screen 1


Data Source Configuration Wizard
Choose a Data Source Type
Where will the application get data from?
Database
Service
Object
< Previous
Next >
Finish
Cancel

Lets you choose an object that can later be used to generate data-bound controls.

2. Select the **OBJECT** icon. Click NEXT.
3. You will now select the Object you wish 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 2.24. Entity Data Source Configuration Wizard Screen 2


Data Source Configuration Wizard
Select the Object You Wish to Bind to
Add Reference...
< Previous
Next >
Finish
Cancel

In what assembly is the object located? (If your object does not appear, cancel the wizard and rebuild the project that contains your object.)

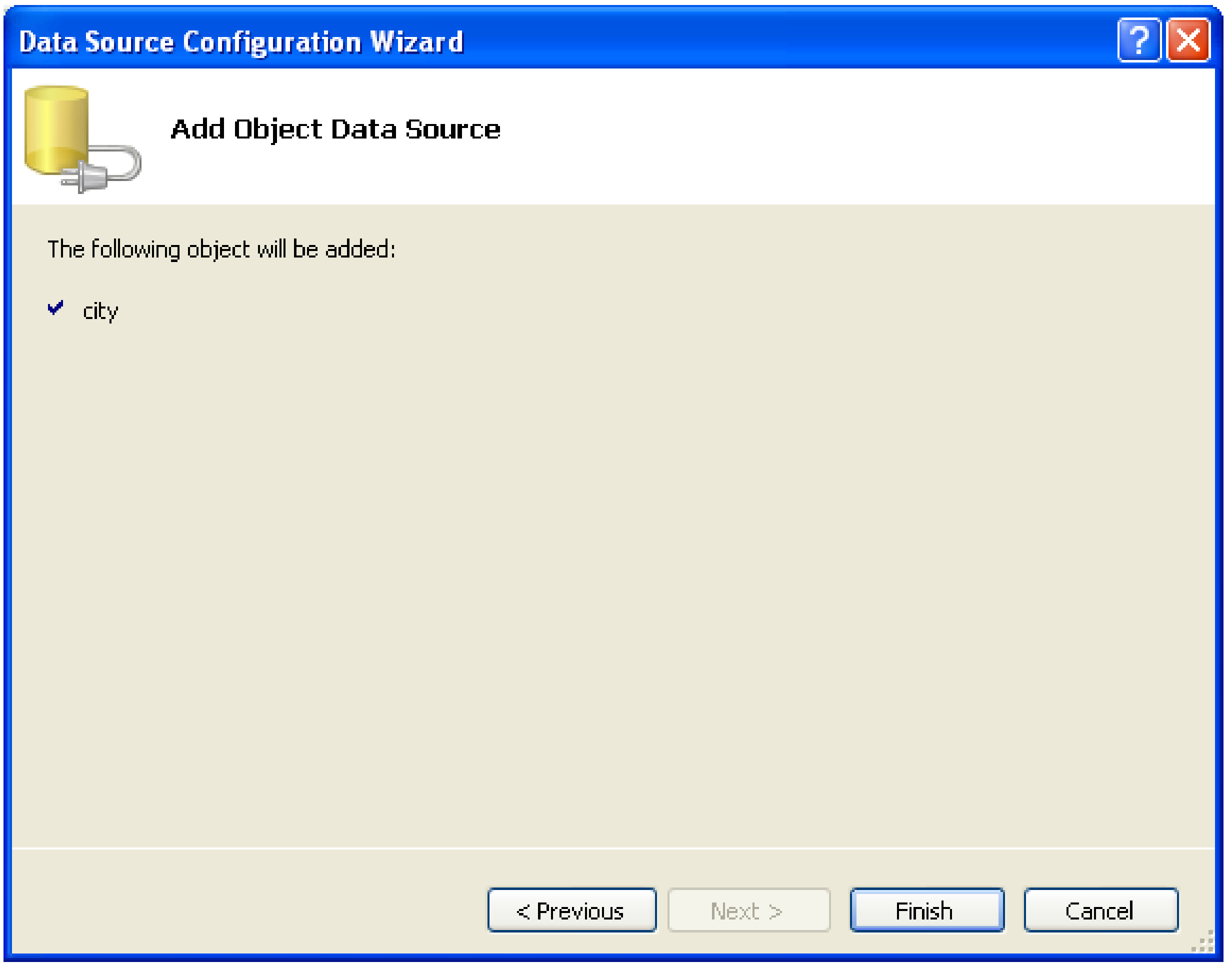
- WindowsFormsApplication4
  - { } WindowsFormsApplication4
    - city
    - country
    - countrylanguage
    - Form1
    - Program
    - worldEntities
  - { } WindowsFormsApplication4.Properties

There are no comments for worldModel.city in the schema.

✓ Hide assemblies that begin with Microsoft or System

4. The wizard will confirm that the city object is to be added. Click FINISH.

Figure 2.25. Entity Data Source Configuration Wizard Screen 3

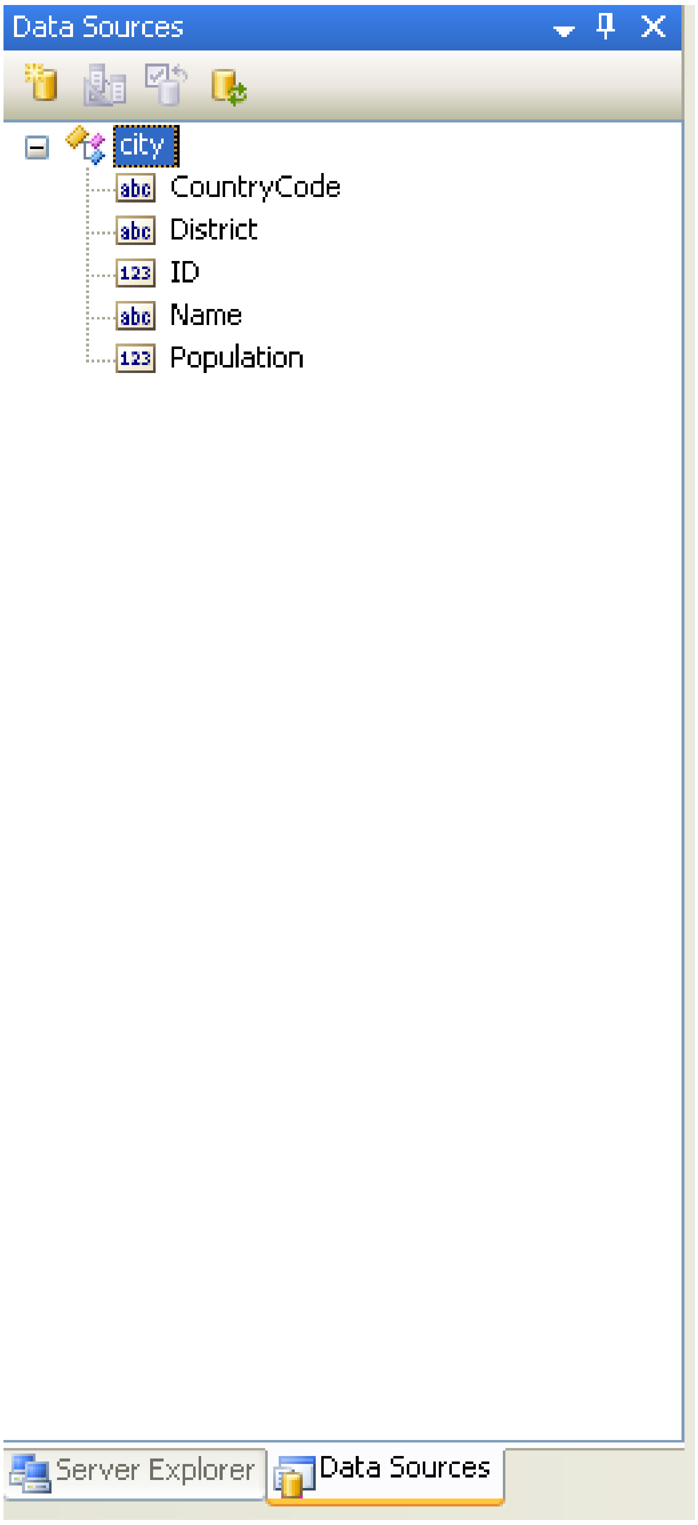


The following object will be added:

✓ city

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 2.26. Data Sources

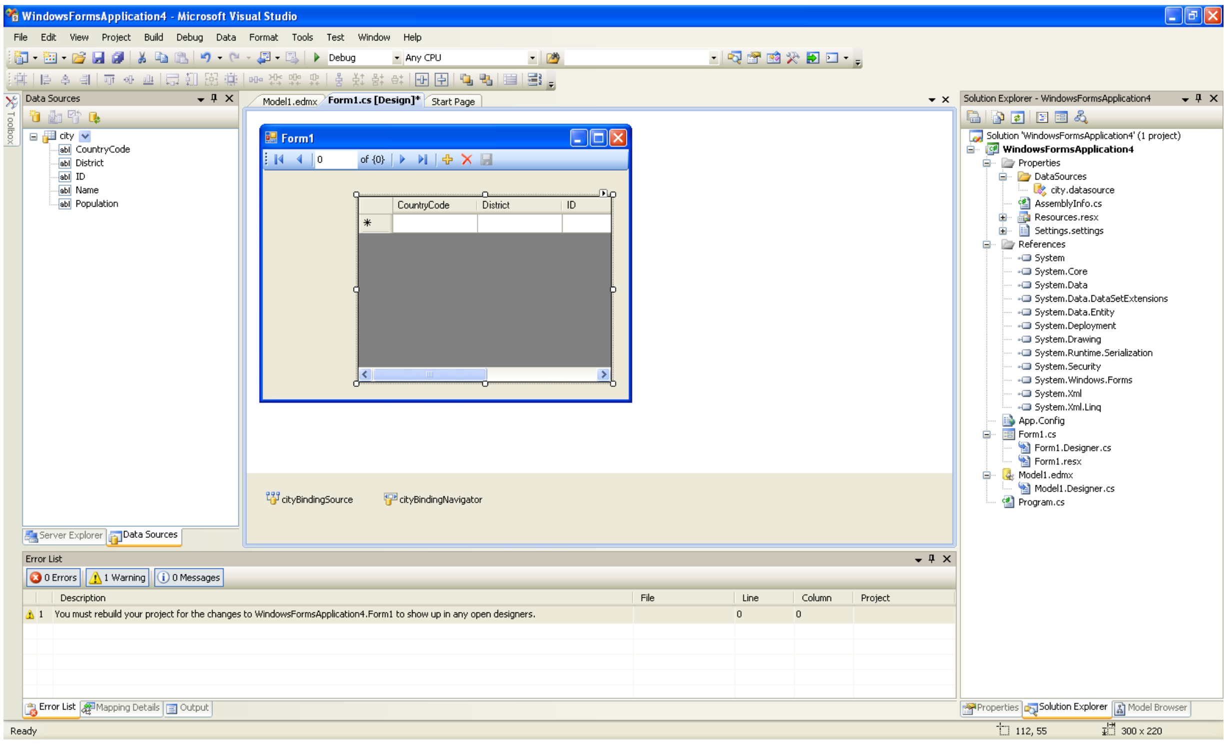


Using the Data Source in a Windows Form

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 that the Data Grid View control is bound to the `city-BindingSource` and the Navigator control is bound to `cityBindingNavigator`.

Figure 2.27. Data Form Designer



	Description	File	Line	Column	Project
1	You must rebuild your project for the changes to WindowsFormsApplication4.Form1 to show up in any open designers.		0	0	

2. Save and rebuild the solution before continuing.

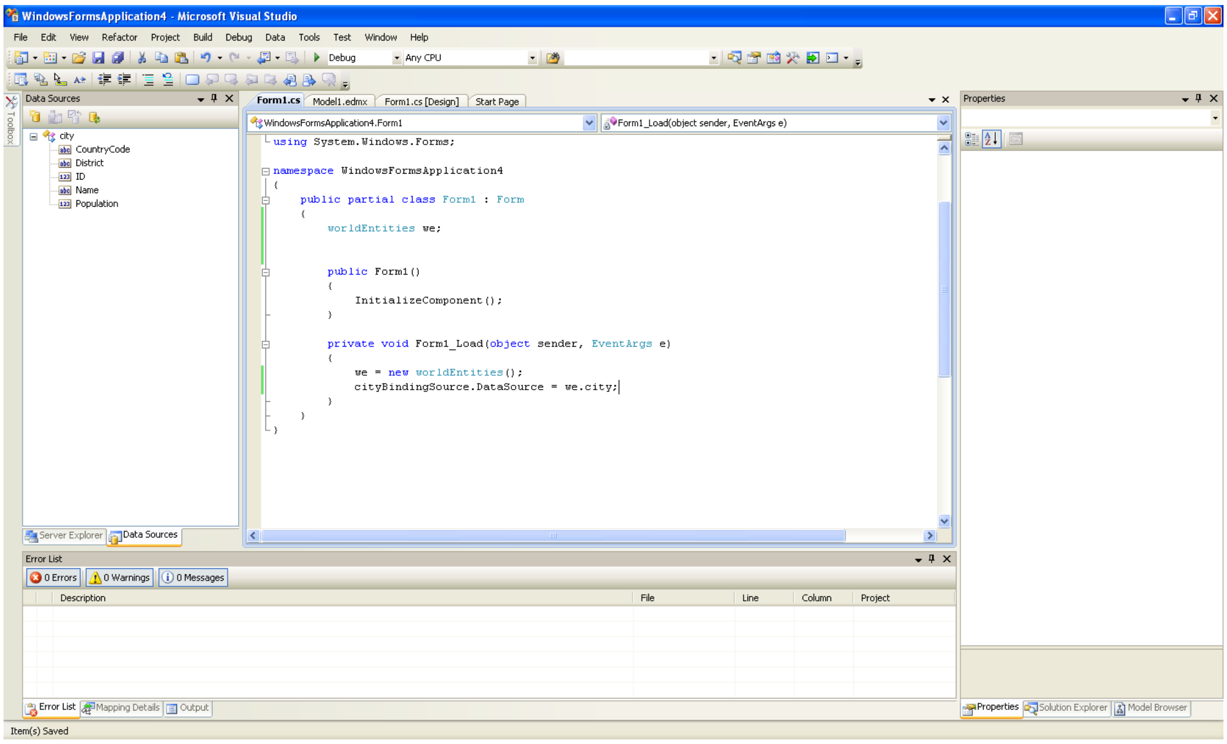
Adding Code to Populate the Data Grid View

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 instatiate the Entity Data Model's EntityContainer object and retrieve data from the database to populate the control.

Figure 2.28. Adding Code to the Form

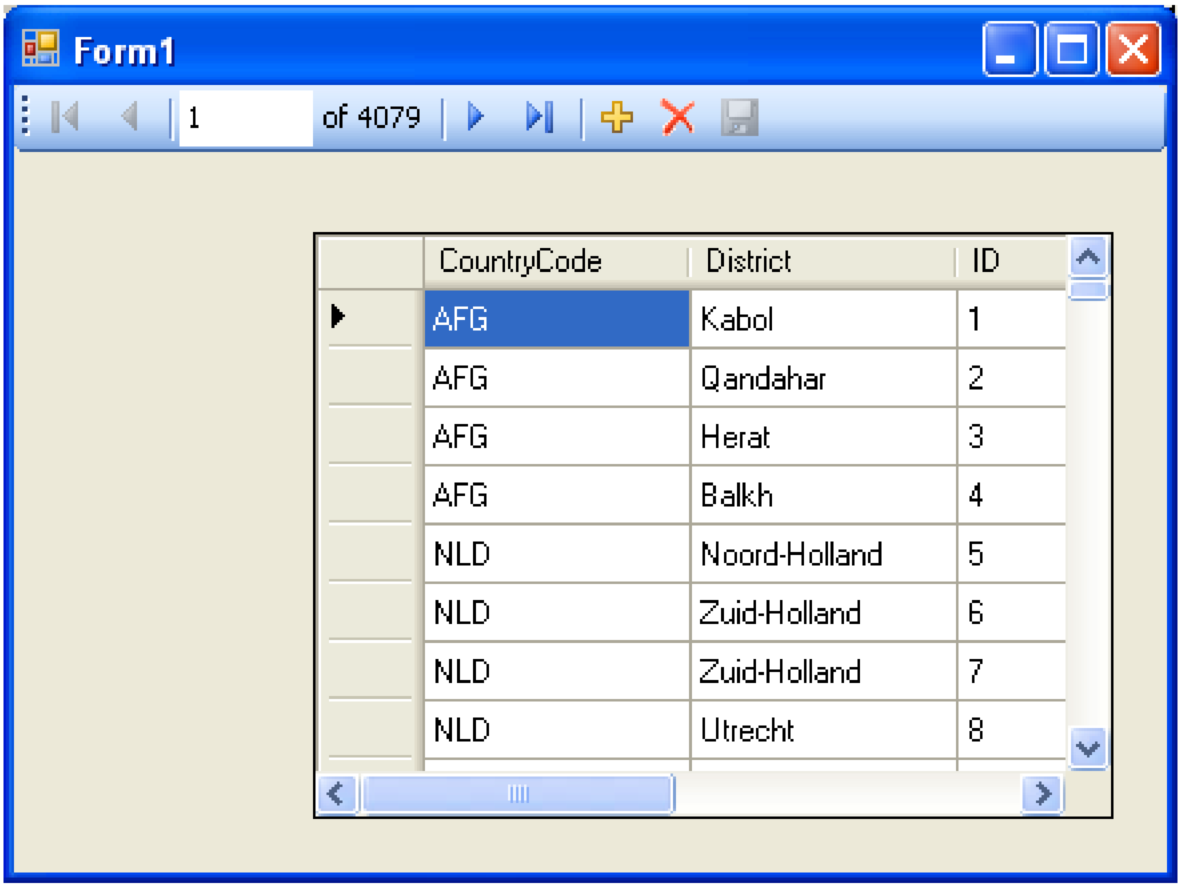


```
using System.Windows.Forms;

namespace WindowsFormsApplication4
{
    public partial class Form1 : Form
    {
        worldEntities we;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            we = new worldEntities();
            cityBindingSource.DataSource = we.city;
        }
    }
}
```

3. Save and rebuild the solution.

4. Run the solution. Ensure the grid is populated and you can navigate the database.

Figure 2.29. The Populated Grid Control



CountryCode	District	ID
AFG	Kabol	1
AFG	Qandahar	2
AFG	Herat	3
AFG	Balkh	4
NLD	Noord-Holland	5
NLD	Zuid-Holland	6
NLD	Zuid-Holland	7
NLD	Utrecht	8

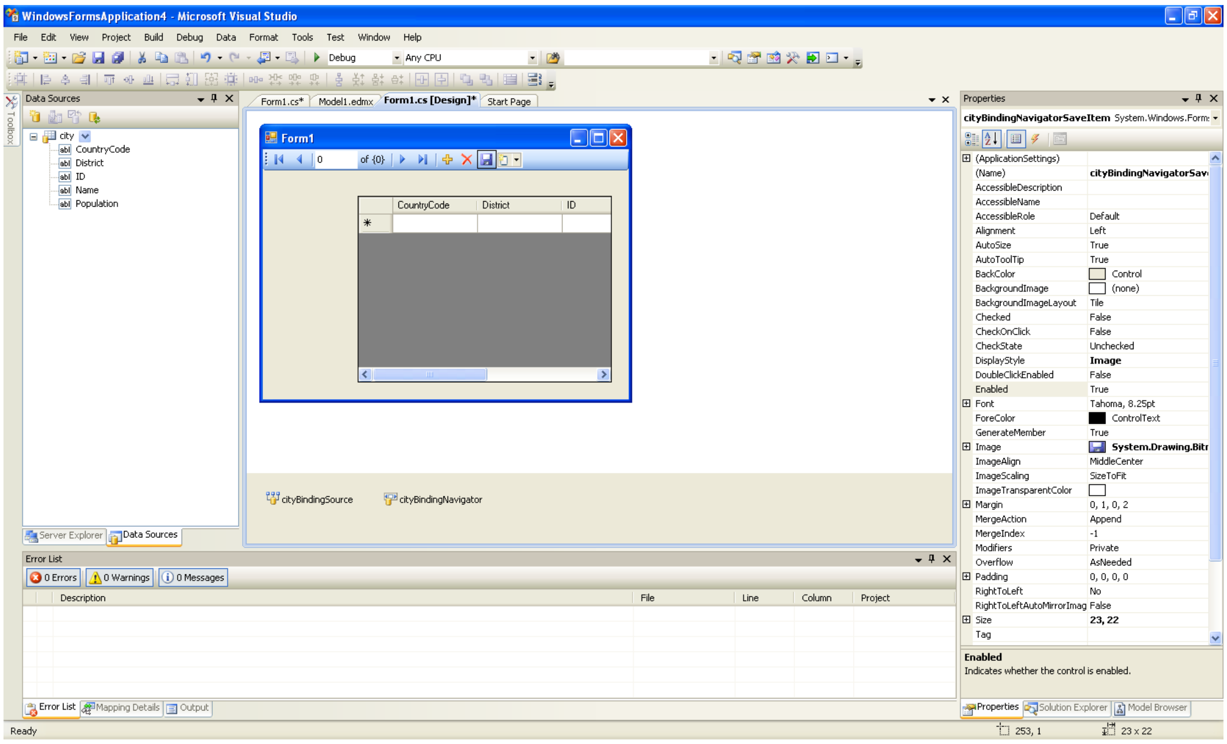
Adding Code to Save Changes to the Database

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 on the Save icon in the Form toolbar and ensure that its Enabled property is set to True.

Figure 2.30. Save Button Enabled


WindowsFormsApplication4 - Microsoft Visual Studio
File Edit View Project Build Debug Data Format Tools Test Window Help
Debug
Any CPU
Data Sources
city
CountryCode
District
ID
Name
Population
Form1.cs* Model1.edmx Form1.cs [Design]* Start Page
Form1
0 of {0}
CountryCode District ID
cityBindingSource
cityBindingNavigator
Server Explorer Data Sources
Error List
0 Errors 0 Warnings 0 Messages
Description File Line Column Project
Error List Mapping Details Output
Properties Solution Explorer Model Browser
Ready
253, 1
23 x 22

Properties

cityBindingNavigatorSaveItem System.Windows.Form:

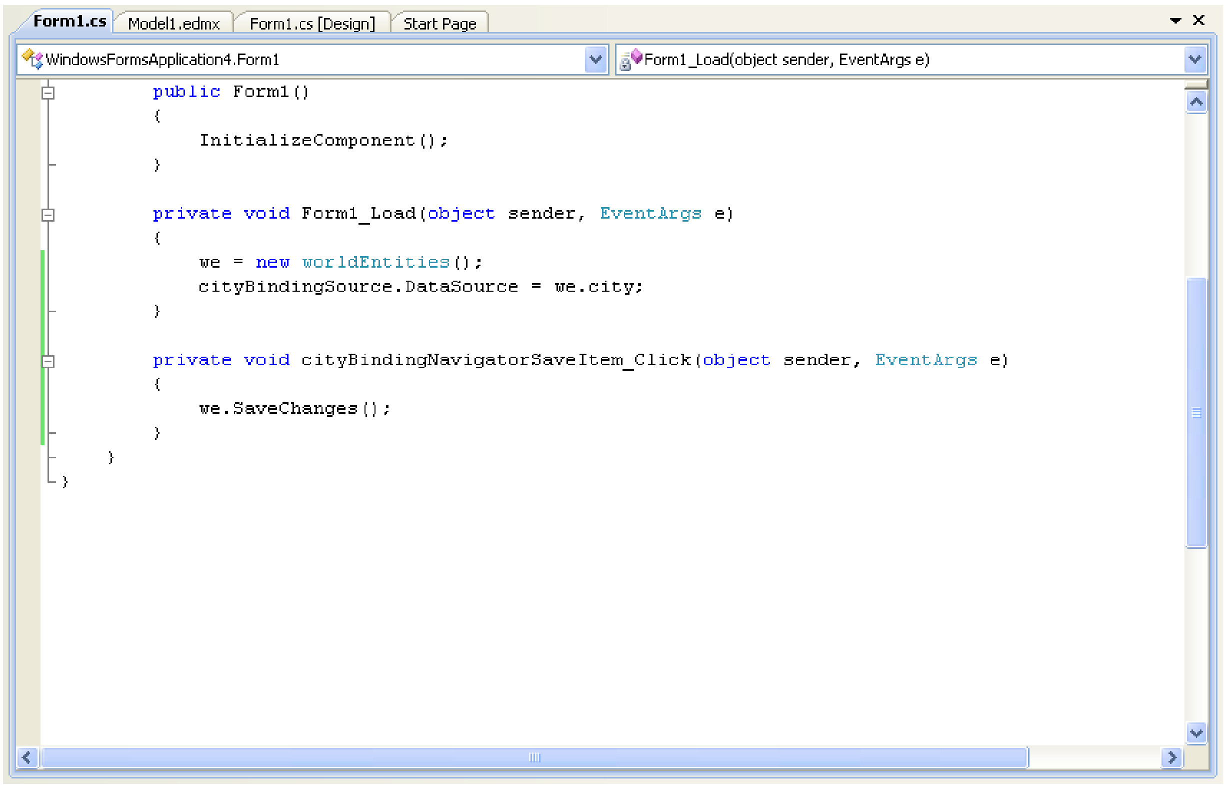
(ApplicationSettings)	
(Name)	cityBindingNavigatorSav
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Alignment	Left
AutoSize	True
AutoToolTip	True
BackColor	Control
BackgroundImage	(none)
BackgroundImageLayout	Tile
Checked	False
CheckOnClick	False
CheckState	Unchecked
DisplayStyle	Image
DoubleClickEnabled	False
Enabled	True
Font	Tahoma, 8.25pt
ForeColor	ControlText
GenerateMember	True
Image	System.Drawing.Bitr
ImageAlign	MiddleCenter
ImageScaling	SizeToFit
ImageTransparentColor	
Margin	0, 1, 0, 2
MergeAction	Append
MergeIndex	-1
Modifiers	Private
Overflow	AsNeeded
Padding	0, 0, 0, 0
RightToLeft	No
RightToLeftAutoMirrorImag	False
Size	23, 22
Tag	

Enabled
Indicates whether the control is 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 click in the application.

Figure 2.31. Adding Save Code to the Form


Form1.cs
Model1.edmx
Form1.cs [Design]
Start Page
WindowsFormsApplication4.Form1
Form1_Load(object sender, EventArgs e)

```
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            we = new worldEntities();
            cityBindingSource.DataSource = we.city;
        }

        private void cityBindingNavigatorSaveItem_Click(object sender, EventArgs e)
        {
            we.SaveChanges();
        }
    }
}
```

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.

2.3.9.2. 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.

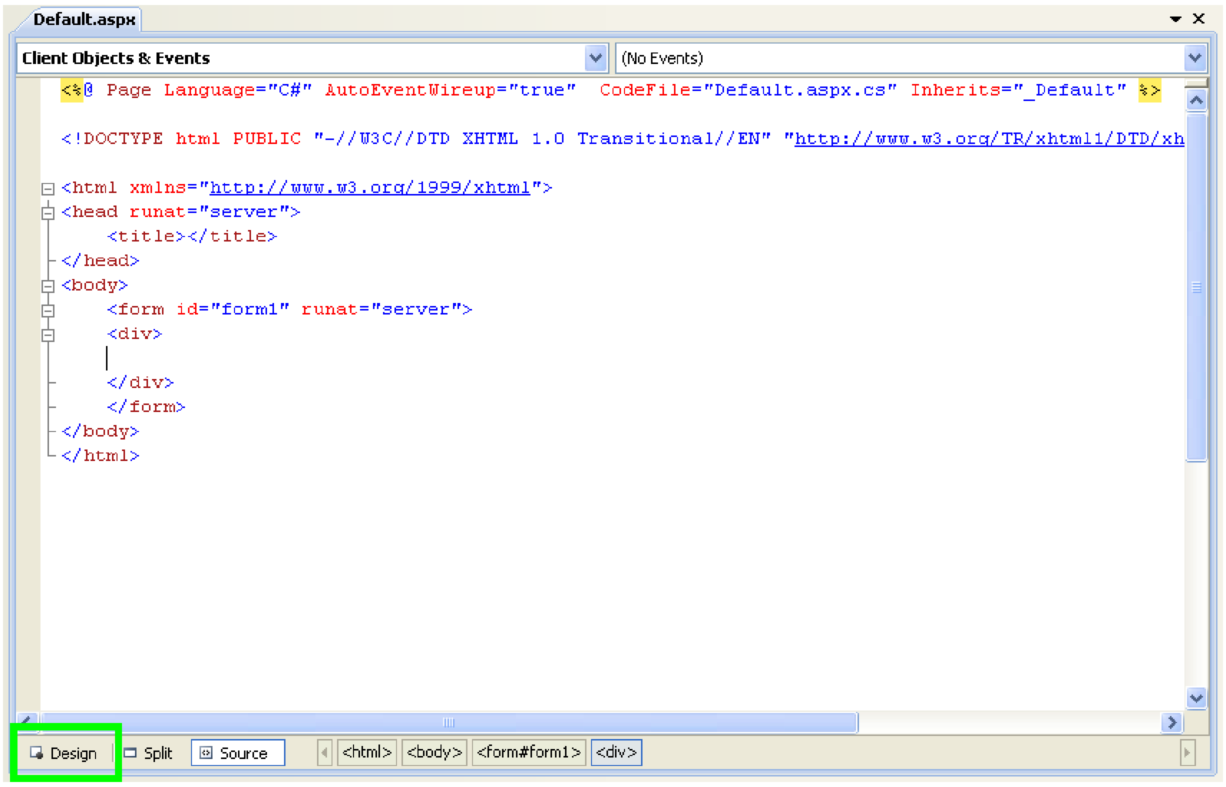
If you have not already done so, you should install the World example database prior to attempting this tutorial. Instructions on where to obtain the database and instructions on how to install it where given in the tutorial Section 2.3.9.1, “Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source”.

Creating an ASP.NET web site

In this part of the tutorial you will create an ASP.NET web site. The web site will use the World database. The main web page will feature a drop down list from which you can select a country, data about that country's cities will then be 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 2.32. The Design Tab

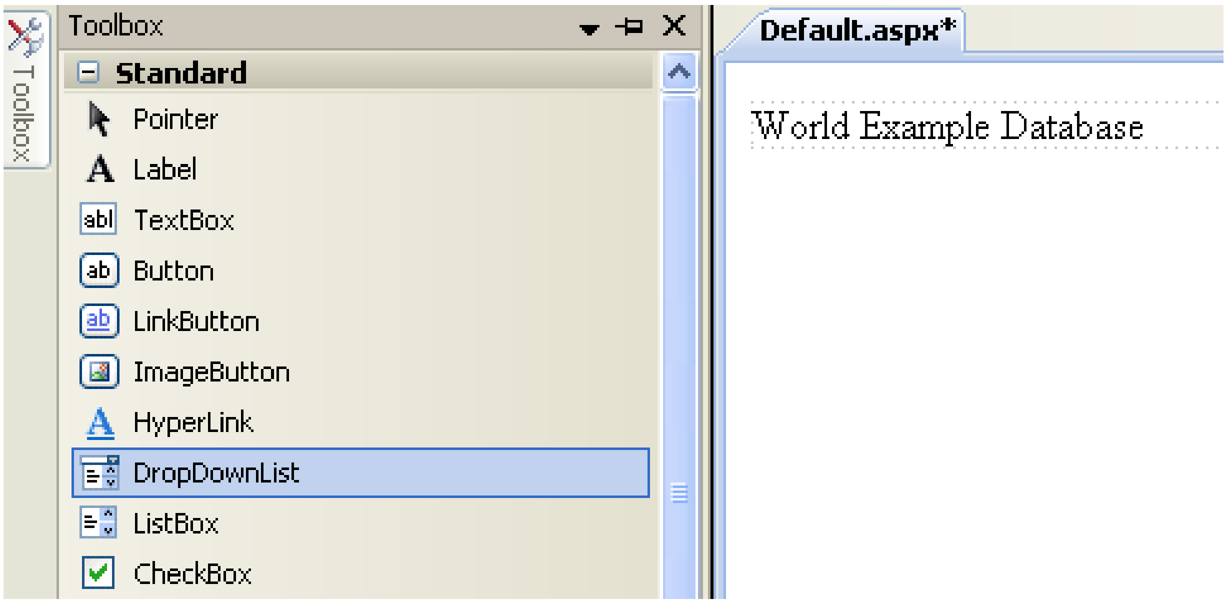


```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xh

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

4. In the Design view panel, enter some text to decorate the blank web page.

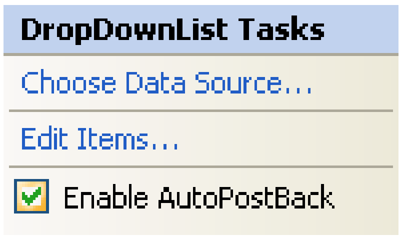
5. Click on Toolbox. From the list of controls select **DropDownList**. Drag and drop the control to a location beneath the text on your web page.

Figure 2.33. Drop Down List



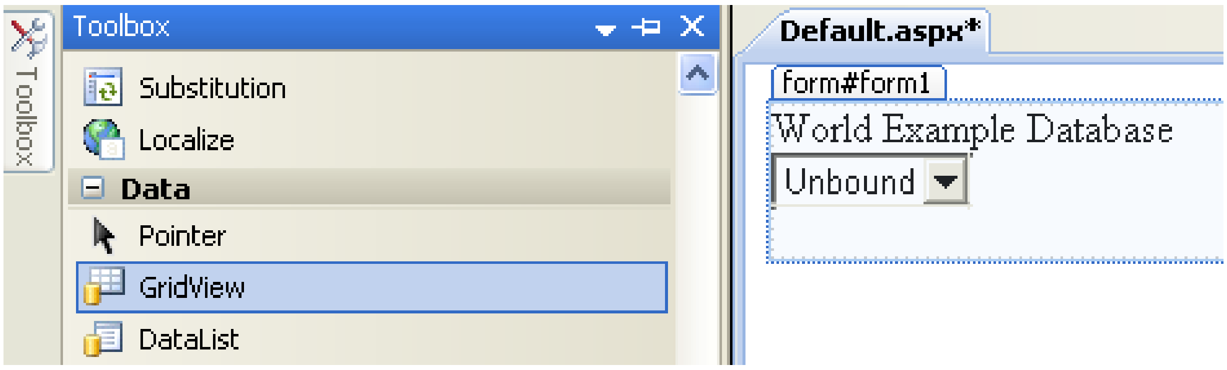
6. From the **DropDownList** control's context menu, ensure that the **Enable AutoPostBack** check box is enabled. This will ensure the control's event handler is called when an item is selected. The user's choice will in turn be used to populate the **GridView** control.

Figure 2.34. Enable AutoPostBack



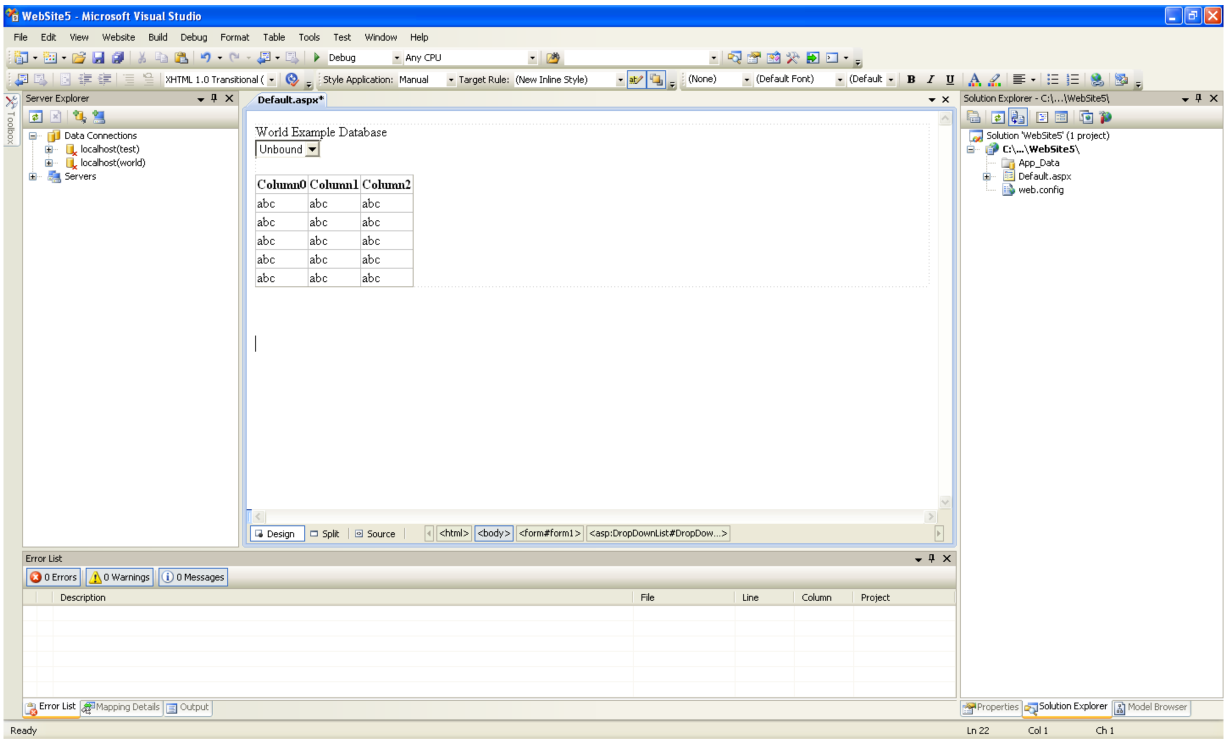
7. From the Toolbox select the **GRIDVIEW** control.

Figure 2.35. Grid View Control



Drag and drop the Grid Vew control to a location just below the Drop Down List you already placed.

Figure 2.36. Placed Grid Vew 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 2.3.9.1, "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 on 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();
        }
    }
...
```

   Note that 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 apear.

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.

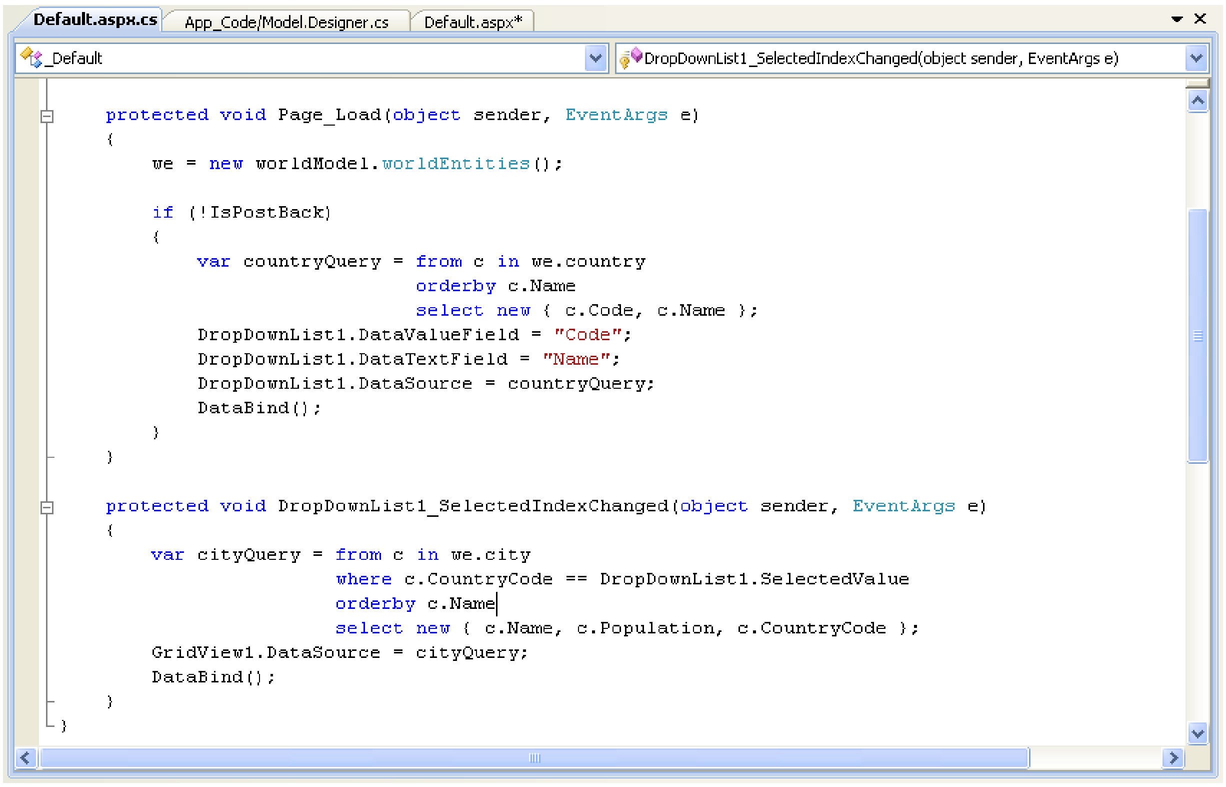
1. In the Design view double click on the **DropDownList** control. This causes its `SelectedIndexChanged` code to be displayed. This method is called when a user selects an item in the list control and thus fires an AutoPostBack event.
2. Modify the relevant section of code accordingly to the following 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:

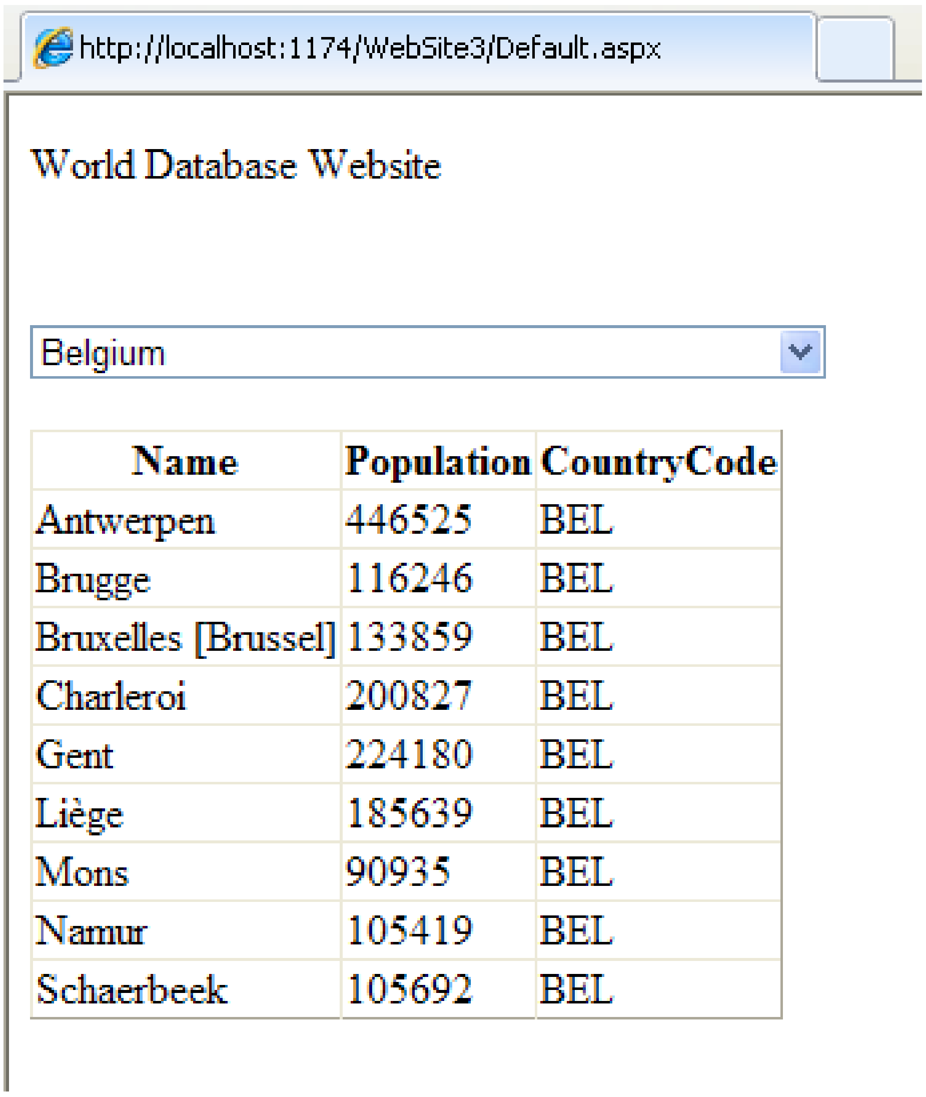
Figure 2.37. Source Code


Default.aspx.cs
App_Code/Model.Designer.cs
Default.aspx*
_Default
DropDownList1_SelectedIndexChanged(object sender, EventArgs e)

```
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();
    }
}

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();
}
}
```

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 2.38. The Working Web Site



http://localhost:1174/WebSite3/Default.aspx

World Database Website

Belgium

Name	Population	CountryCode
Antwerpen	446525	BEL
Brugge	116246	BEL
Bruxelles [Brussel]	133859	BEL
Charleroi	200827	BEL
Gent	224180	BEL
Liège	185639	BEL
Mons	90935	BEL
Namur	105419	BEL
Schaerbeek	105692	BEL

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 via LINQ queries on an entity data model.

2.4. 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.

2.4.1. 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 example database, 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`.

2.4.1.1. The MySqlConnection Object

For your Connector/NET application to connect to a MySQL database it needs to establish a connection. This is achieved through the use of 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 2.4.2, “Connecting to MySQL Using Connector/NET”. A reference containing a list of supported connection string options can also be found in Section 2.4.5, “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 databse operations
            conn.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        Console.WriteLine("Done.");
    }
}
```

When the `MySqlConnection` constructor is invoked it returns a connection object, which is used for subsequent database operations. The first operation in this example is to open the connection. This needs to be done before further operations take place. Before the application exits the connection to the database needs to be closed by calling `Close` on the connection object.

Sometimes an attempt to perform an `Open` on a connection object can fail, this will generate an exception that can be handled via 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.

2.4.1.2. The MySqlCommand Object

Once a connection has been established with the MySQL database, the next step is do 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();
            conn.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        Console.WriteLine("Done.");
    }
}
```

When a connection has been created and opened, the code then creates a `MySqlCommand` object. Note that 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 simply printed out as part of a `while` loop. Finally, the `MySqlReader` object is displosed 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 Mouse', 'Nor
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            cmd.ExecuteNonQuery();

            conn.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        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 Client program 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);
            }

            conn.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        Console.WriteLine("Done.");
    }
}
```

This example uses a simple query to count the rows in the Country table. The result is obtained by calling `ExecuteScaler` on the command object.

2.4.1.3. 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 in order 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 needs to write data back to the database. This decouples the working data set from the database.

This decouple 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 in order 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 should the database need to be updated. 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 you wish 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 ilustrates how this is done:

```
MySqlCommandBuilder cb = new MySqlCommandBuilder(daCountry);
```

Note that the MySqlDataAdapter object is passed as a parameter to the command builder.

Filling the Data Set

In order to do anything useful with the data from your datbase, 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");
```

Note the Fill method is a MySqlDataAdapter method, the Data Adapter knows how to establish a connec tion with the database and retrieve the required data, and then populates 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

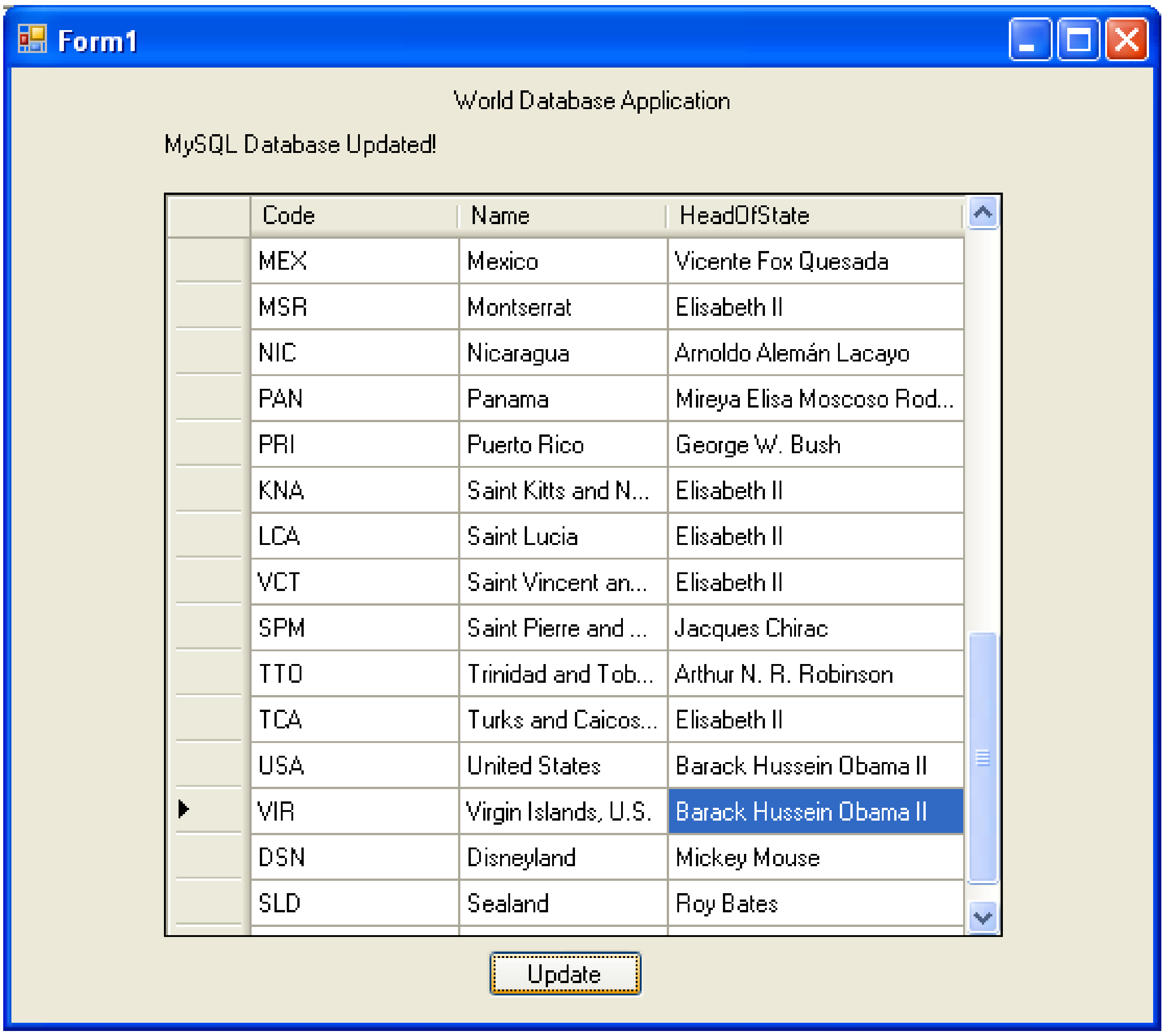
The interactions between the DataSet, MySqlDataAdapter and MySqlCommandBuilder classes can be a little confusing, so their operation can perhaps be best illustrated by working code.

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 in order to place and create the user interface controls, but the main code that uses the key classes descibed 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 2.39. World Database Application



World Database Application

MySQL Database Updated!

Code	Name	HeadOfState
MEX	Mexico	Vicente Fox Quesada
MSR	Montserrat	Elisabeth II
NIC	Nicaragua	Arnoldo Alemán Lacayo
PAN	Panama	Mireya Elisa Moscoso Rod...
PRI	Puerto Rico	George W. Bush
KNA	Saint Kitts and N...	Elisabeth II
LCA	Saint Lucia	Elisabeth II
VCT	Saint Vincent an...	Elisabeth II
SPM	Saint Pierre and ...	Jacques Chirac
TTO	Trinidad and Tob...	Arthur N. R. Robinson
TCA	Turks and Caicos...	Elisabeth II
USA	United States	Barack Hussein Obama II
VIR	Virgin Islands, U.S.	Barack Hussein Obama II
DSN	Disneyland	Mickey Mouse
SLD	Sealand	Roy Bates

2.4.1.4. 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 from 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 contructed 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 enetred by the user. The user could enter a string that generates a run time 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 user a parameter for user input would be:

```
string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent = @Continent";
```

Note that 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 create a parameter object that can be passed to the Command object. In Connector/NET the class `MySqlParameter` is used for this purpose. The use of `MySqlParameter` is best illustrated by a small code snippet:

```
MySqlParameter param = new MySqlParameter();
param.ParameterName = "@Continent";
param.Value = "North America";
cmd.Parameters.Add(param);
```

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. Once the parameter has its name and value set it needs to be added to the Command object using the `Add` method.

A further example illustrates this:

```
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();

            MySqlParameter param = new MySqlParameter();
            param.ParameterName = "@Continent";
            param.Value = user_input;
            cmd.Parameters.Add(param);

            MySqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine(rdr["Name"]+" --- "+rdr["HeadOfState"]);
            }

            conn.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        Console.WriteLine("Done.");
    }
}
```

In this part of the tutorial you have see how to use parameters to make your code more secure.

2.4.1.5. Working with Stored Procedures

In this section you will see how to work with Stored Procedures. This section assumes you have a basic understanding of what a Stored Procedure is, and how to create one.

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 the Stored Procedure wors as expec ted by typing the following into the MySQL Client program:

```
CALL country_hos('Europe');
```

Note that The Stored Routine takes a single parameter, which is the continent you wish to restrict your search to.

Having confirmed that the Stored Procedure is present and correct you can now move on to seeing how it can be accessed 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 a SQL query as a parameter it takes the name of the Stored Procedure to call. The `MySqlCommand` object also needs to be set to the type of Stored Procedure. This is illustrated by the following code snippet:

```
string rtn = "country_hos";
MySqlCommand cmd = new MySqlCommand(rtn, conn);
cmd.CommandType = CommandType.StoredProcedure;
```

In this case you also need to pass a parameter to the Stored Procedure. This can be achieved using the techniques seen in the previous section on parameters, Section 2.4.1.4, “Working with Parameters”. This is shown in the following code snippet:

```
MySqlParameter param = new MySqlParameter();
param.ParameterName = "@con";
param.Value = "Europe";
cmd.Parameters.Add(param);
```

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 all that now needs to be done is to call the routine. This can be achieved using techniques also learned in earlier sections, but 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;
            MySqlParameter param = new MySqlParameter();
            param.ParameterName = "@con";
            param.Value = "Europe";
            cmd.Parameters.Add(param);

            MySqlDataReader rdr = cmd.ExecuteReader();
            while (rdr.Read())
            {
                Console.WriteLine(rdr[0] + " --- " + rdr[1]);
            }
            conn.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        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.

2.4.2. Connecting to MySQL Using Connector/NET

Introduction

All interaction between a .NET application and the MySQL server is routed through a `MySqlConnection` object. Before your application can interact with the server, a `MySqlConnection` object must be instanced, configured, and opened.

Even when using the `MySqlHelper` class, a `MySqlConnection` object is created by the helper class.

In this section, we will describe how to connect to MySQL using the `MySqlConnection` object.

2.4.3. Creating a Connection String

The `MySqlConnection` object is configured using a connection string. A connection string contains sever key/value pairs, separated by semicolons. Each key/value pair is joined with an equals sign.

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.

The following options are available:

> **Note**
>
> Using the '@' symbol for parameters is now the preferred approach although the old pattern of using '?' is still supported.
>
> Please be aware however that using '@' can cause conflicts when user variables are also used. To help with this situation please see the documentation on the `Allow User Variables` connection string option, which can be found here: Section 2.4.3, "Creating a Connection String". The `Old Syntax` connection string option has now been deprecated.

2.4.3.1. 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.

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.

2.4.3.2. 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");
        case 1045:
            MessageBox.Show("Invalid username/password, please try again");
    }
}
```

Important

Note that if you are using multilanguage databases 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;");
```

2.4.4. Using Connector/NET with Connection Pooling

The Connector/NET supports connection pooling. This is enabled by default, but can be turned off via connection string options. See Section 2.4.3, "Creating a 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.

To work as designed, it is best to let the connection pooling system manage all connections. You should 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.

2.4.5. Connector/NET Connection String Options Reference

Name	Default	Description
`Allow Batch`	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. Commands sent to earlier versions of MySQL should be separated with ';'.
`Allow User Variables`	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`	false	True to have MySqlDataReader.GetValue() return a MySqlDateTime for date or datetime columns that have illegal values. False will cause a `System.DateTime` object to be returned for legal values and an exception will be thrown for illegal values.
`AutoEnlist`	true	
`BlobAsUTF8ExcludePattern`	null	
`BlobAsUTF8IncludePattern`	null	
`CharSet`, `Character Set`		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 data returned.
`Connect Timeout`, `Connection Timeout`	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.

`Connection Reset`	false	
`Convert Zero Datetime`	false	True to have `MySqlDataReader.GetValue()` and `MySqlDataReader.GetDateTime()` return DateTime.MinValue for date or datetime columns that have illegal values.
`Default Command Timeout`		Sets the default value of the command timeout to be used. This does not supercede 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
`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. Recognized values are `true`, `false`, `yes`, and `no`. In versions before 5.0.3, this option had no effect.
`FunctionsReturnString`	false	
`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 &. 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 care should be taken when using this option. In Unix environment with Mono, this can be a fully qualified path to MySQL socket file name. With this configuration, the Unix socket will be used instead of 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`	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 use with server side prepared statements. If you want to 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.
`Initial Catalog`, `Database`	mysql	The name of the database to use intially
`InteractiveSession`	false	
`Logging`	false	When true, various pieces of information is output to any configured TraceListeners.
`Old Syntax`, `OldSyntax`	false	Allows use of '@' symbol as a parameter marker. See `MySqlCommand` for more info. This option was deprecated in Connector/NET 5.2.2. All future code should be written using the '@' symbol.
`Password`, `pwd`		The password for the MySQL account being used.
`Persist Security Info`	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`	mysql	When set to the name of a named pipe, the `MySqlConnection` will attempt to connect to MySQL on that named pipe.This settings 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`	25	Sets the size of the stored procedure cache. By default, Connector/NET will store the metadata (input/output datatypes) 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`	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.
`Respect Binary Flags`	true	Setting this option to `false` means that Connector/NET will ignore a column's binary flags as set by the server. This option was added in Connector/NET version 5.1.3.
`Shared Memory Name`	MYSQL	The name of the shared memory object to use for communication if the

		connection protocol is set to memory.
TreatBlobsAsUTF8	false	
Treat Tiny As Boolean	true	Setting this value to false indicates that TINYINT(1) will be treated as an INT. See also Overview of Numeric Types for a further explanation of the TINYINT and BOOL data types.
Use Affected Rows	false	When true the connection will report changed rows instead of found rows. This option was added in Connector/NET version 5.2.6.
Use Procedure Bodies	true	Setting this option to false indicates that the user connecting to the database does not have the SELECT privileges for the mysql.proc (stored procedures) table. When to set to false, Connector/NET will not rely on this information being available when the procedure is called. Because Connector/NET will be unable to determine this information, you should explicitly set the types of the all the parameters before the call and the parameters should be added to the command in the exact same order as they appear in the procedure definition. This option was added in Connector/NET 5.0.4 and Connector/NET 1.0.10.
User Id, Username, Uid, User name		The MySQL login account being used.
Use Compression	false	Setting this option to true 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 then 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	false	
Use Performance Monitor	false	

The following table lists the valid names for connection pooling values within the ConnectionString. For more information about connection pooling, see Connection Pooling for the MySQL Data Provider.

Name	Default	Description
Cache Server Configuration, CacheServerConfiguration, CacheServerConfig	false	Specifies whether server variables should be updated when a pooled connection is returned. Turning this one will yeild faster opens but will also not catch any server changes made by other connections.
Connection Lifetime	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 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.
Max Pool Size	100	The maximum number of connections allowed in the pool.
Min Pool Size	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.
Reset Pooled Connections, ResetConnections, ResetPooledConnections	true	Specifies whether a ping and a reset should be sent to the server before a pooled connection is returned. Not resetting will yield faster connection opens but also will not clear out session items such as temp tables.

2.4.6. Using the Connector/NET with Prepared Statements

Introduction

As of MySQL 4.1, it is possible to use prepared statements with Connector/NET. Use of prepared statements can provide significant performance improvements on queries that are executed more than once.

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 it uses a binary protocol that makes data transfer between client and server more efficient.

2.4.6.1. 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.Add("@number", 1)
   cmd.Parameters.Add("@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, MessageBoxIc
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.Add("@number", 1);
    cmd.Parameters.Add("@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);
}
```

2.4.7. Accessing Stored Procedures with Connector/NET

Introduction

With the release of MySQL version 5 the MySQL server now supports stored procedures with the SQL 2003 stored procedure syntax.

A stored procedure is a set of SQL statements that can be stored in the server. Once this has been done, clients do not need to keep reissuing the individual statements but can refer to the stored procedure instead.

Stored procedures can be particularly useful in situations such as the following:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. 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 our 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.

2.4.7.1. Creating Stored Procedures 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 the `MySQL Query Browser` GUI client. Finally, stored procedures can be created using the `.ExecuteNonQuery` method of the `MySqlCommand` object:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()
    cmd.Connection = conn

    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 ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIc
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 = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn.Open();
    cmd.Connection = conn;

    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 (MySql.Data.MySqlClient.MySqlException ex)
{
MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
    "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

It should be noted that, unlike the command-line and GUI clients, you are not required to specify a special delimiter when creating stored procedures in Connector/NET.

2.4.7.2. Calling a Stored Procedure from Connector/NET

To call a stored procedure using Connector/NET, create a `MySqlCommand` object and pass the stored procedure name as the `.CommandText` property. Set the `.CommandType` property to `CommandType.StoredProcedure`.

After the stored procedure is named, 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 datatype that is expected to be returned. All parameters need the parameter direction defined.

After defining parameters, call the stored procedure by using the `MySqlCommand.ExecuteNonQuery()` method:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()
    cmd.Connection = conn

    cmd.CommandText = "add_emp"
    cmd.CommandType = CommandType.StoredProcedure

    cmd.Parameters.Add("@lname", 'Jones')
    cmd.Parameters["@lname"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("@fname", 'Tom')
    cmd.Parameters["@fname"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("@bday", #12/13/1977 2:17:36 PM#)
    cmd.Parameters["@bday"].Direction = ParameterDirection.Input

    cmd.Parameters.Add("@empno", MySqlDbType.Int32)
    cmd.Parameters["@empno"].Direction = ParameterDirection.Output

    cmd.ExecuteNonQuery()

    MessageBox.Show(cmd.Parameters["@empno"].Value)
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIc
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 = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";
```

```
try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "add_emp";
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.Add("@lname", "Jones");
    cmd.Parameters["@lname"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("@fname", "Tom");
    cmd.Parameters["@fname"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("@bday", DateTime.Parse("12/13/1977 2:17:36 PM"));
    cmd.Parameters["@bday"].Direction = ParameterDirection.Input;

    cmd.Parameters.Add("@empno", MySqlDbType.Int32);
    cmd.Parameters["@empno"].Direction = ParameterDirection.Output;

    cmd.ExecuteNonQuery();

    MessageBox.Show(cmd.Parameters["@empno"].Value);
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
      "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Once the stored procedure is called, the values of output parameters can be retrieved by using the `.Value` property of the `MySql-Connector.Parameters` collection.

2.4.8. Handling BLOB Data With Connector/NET

Introduction

One common use for MySQL is the storage of binary data in `BLOB` columns. MySQL supports four different BLOB datatypes: `TINYBLOB`, `BLOB`, `MEDIUMBLOB`, and `LONGBLOB`.

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.

2.4.8.1. 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 may need to modify the max_allowed_packet system variable. This variable determines how large of a packet (i.e. a single row) can be sent to the MySQL server. By default, the server will only accept a maximum size of 1 meg from our client application. If you do not intend to exceed 1 meg, this should be fine. If you do intend to exceed 1 meg in your file transfers, this number has to be increased.

The max_allowed_packet option can be modified using MySQL Administrator's Startup Variables screen. Adjust the Maximum allowed option in the Memory section of the Networking tab to an appropriate setting. After adjusting the value, click the APPLY CHANGES button and restart the server using the `Service Control` screen of MySQL Administrator. 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.

2.4.8.2. 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.Add("@FileName", strFileName)
    cmd.Parameters.Add("@FileSize", FileSize)
    cmd.Parameters.Add("@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.Add("@FileName", strFileName);
    cmd.Parameters.Add("@FileSize", FileSize);
    cmd.Parameters.Add("@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.

2.4.8.3. 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.Asterisk)

    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, 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.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.

2.4.9. Using Connector/NET with Crystal Reports

Introduction

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.

2.4.9.1. 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.

2.4.9.2. Creating the Report

For most purposes the Standard Report wizard should help 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 will first prompt you for a data source. If you are using 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.

2.4.9.3. 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:

- CrytalDecisions.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 2.4.9.1, “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 it 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 the 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 allows 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 alphabetical 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.

2.4.10. Handling Date and Time Information in Connector/NET

Introduction

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.

In this section we will demonstrate how to properly handle date and time information when using Connector/NET.

2.4.10.1. 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.

2.4.10.2. 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.

2.4.10.3. 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` datatype.

The `MySqlDateTime` datatype 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
```

Please note that the use of 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.

2.4.10.4. Handling NULL Dates

The .NET `DateTime` datatype 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.

2.4.11. ASP.NET Provider Model

MySQL Connector/Net provides support for the ASP.NET 2.0 provider model. This model allows application developers to focus on the business logic of there application instead of having to recreate such boilerplate items as membership and roles support. Currently, only membership and role providers are supplied although session state and profile providers will be provided in upcoming releases.

Installing The Providers

The installation of Connector/Net 5.1 or later will install the providers and register them in your machines .NET configuration file. The providers are implemented in the file `mysql.web.dll` and this file can be found in your Connector/Net installation folder. There is no need to run any type of SQL script to setup the database 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 will be able to select the MySQL membership and roles provider by indicating that you want to pick a custom provider for each area.

When the provider is installed, it creates a dummy connection string named `LocalMySqlServer`. This has to be done so that the provider will work in the ASP.NET configuration tool. However, you will want to 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. Here is an example:

```
                <connectionStrings>
                    <remove name="LocalMySqlServer"/>
                    <add name="LocalMySqlServer" connectionString="server=xxx;uid=xxx;pwd=xxx"/>
                </connectionStrings>
```

Distribution

To use the providers on a production server you will need to 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 applications bin folder.

2.4.12. 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 metadat.

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 temp 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 sever. 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 will respect the binary flags returned by the server. This will mean that you may need to make small changes to your application to accomodate this change.

In the event that the changes required to your application would be too large, you can add 'respect binary flags=false' to your connection string. This will cause the connector to use the prior behavior. In a nutshell, that behavior was that 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`.

2.4.13. Character Sets

Treating Binary Blobs As UTF8

MySQL doesn't currently support 4 byte UTF8 sequences. This makes it difficult to represent some multi-byte languages such as Japanese. To try and alleviate this, Connector/Net now 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. If you wish to convert only some of your blob columns, then you can make use of the 'BlobAsUTF8IncludePattern' and 'BlobAsUTF8ExcludePattern' keywords. These should be set to the regular expression pattern that matches the column names you wish to include or exlude respectively.

One thing to note is that the regular expression patterns can both match a single column. When this happens, the include pattern is applied before the exclude pattern. The result, in this case, would be that the column would be excluded. You should 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.

Currently this mode only applies to reading strings out of MySQL. To insert 4-byte UTF8 strings into blob columns you will need to 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.

2.4.14. Working with 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 medium trust level. Some hosting providers host shared applications in their own app pools and allow the application to run under full trust, but this seems to be the exception rather than the rule.

Connector/Net versions prior to 5.0.8 and 5.1.3 were not compatible with medium trust hosting. Starting with these versions, Connector/Net can be used under medium trust hosting that has been modified to allow the use of sockets for communication. By default, medium trust does not include `SocketPermission`. Connector/Net uses sockets to talk with the MySQL server so it is required that a new trust level be created that is an exact clone of medium trust but that has `SocketPermission` added.

2.5. 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.

2.5.1. 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 Sun Microsystems, Inc. Additional information is available at http://www.mysql.com/support/.

2.5.2. How to report Connector/NET Problems or Bugs

If you encounter difficulties or problems with Connector/NET, contact the Connector/NET community Section 2.5.1, “Connector/NET Community Support”.

You should first try to execute the same SQL statements and commands from the `mysql` client program or from `admndemo`. This helps you determine whether the error is in Connector/NET or MySQL.

If reporting a problem, you should 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/.

2.5.3. Connector/NET Change History

The Connector/NET Change History (Changelog) is located with the main Changelog for MySQL. See Appendix B, *MySQL Connector/NET Change History*.

2.6. Connector/NET FAQ

Questions

- 2.6.1: How do I obtain the value of an auto-incremented column?

Questions and Answers

2.6.1: How do I obtain the value of an auto-incremented column?

When using the `commandBuilder` you should make sure that you set the `ReturnGeneratedIdentifiers` property to `true`.

Then, you can use an active view on a table to access the updated ID. For example:

```
conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
da = new MySql.Data.MySqlClient.MySqlDataAdapter();
cmdBuilder = new MySql.Data.MySqlClient.MySqlCommandBuilder();
SystemDataDataSet = new System.Data.DataSet();
SystemDataDataView = new System.Data.DataView();
...
cmd.Connection = conn;
cmd.CommandText = "SELECT * FROM contacts";
da.SelectCommand = cmd;
da.Fill(SystemDataDataSet, "contacts");
cmdBuilder.DataAdapter = da;
cmdBuilder.ReturnGeneratedIdentifiers = true;
cmdBuilder.DataAdapter.SelectCommand.CommandText = "SELECT * FROM contacts";
cmdBuilder.RefreshSchema();

SystemDataDataView = SystemDataDataSet.Tables["contacts"].DefaultView;

SystemDataDataRow = SystemDataDataView.Table.NewRow();
SystemDataDataRow["status"] = 1;

SystemDataDataView.Table.Rows.Add(SystemDataDataRow);
da.Update(SystemDataDataSet, "contacts");
System.Console.WriteLine("ID after update: "  + SystemDataDataRow["id"]);
```

The `SystemDataDataRow` object in this instance provides the interface to the updated auto-increment value in the `id` column.

Chapter 3. MySQL Visual Studio Plugin

The MySQL Visual Studio Plugin is a DDEX provider; a plug-in for Visual Studio 2005 that allows developers to maintain database structures, and supports built-in data-driven application development tools.

The current version of the MySQL Visual Studio Plugin includes only database maintenance tools. Data-driven application development tools are not supported.

The MySQL DDEX Provider operates as a standard extension to the Visual Studio Data Designer functionality available through the Server Explorer menu of Visual Studio 2005, and enables developers to create database objects and data within a MySQL database.

The MySQL Visual Studio Plugin is designed to work with MySQL version 5.0, but is also compatible with MySQL 4.1.1 and provides limited compatibility with MySQL 5.1.

3.1. Installing the MySQL Visual Studio Plugin

The MySQL Visual Studio Plugin requires one of Visual Studio 2005 Standard, Professional or Team Developer Edition to be installed. Other editions of Visual Studio 2005 are not supported.

> **Note**
>
> Starting with Connector/NET 5.1.2, the Visual Studio Plugin is included in the installation. If you have installed Connector/NET 5.1.2, then you do not need to separately install the Visual Studio Plugin.

Here is the list of components that should already be installed before starting the installation of the MySQL Visual Studio Plugin:

- Visual Studio 2005 Standard, Professional or Team Developer Edition.
- MySQL Server 4.1.1 or later (either installed on the same machine, or a separate server).
- MySQL Connector/NET 5.0.

> **Note**
>
> When installing Connector/NET you must ensure that the connector is installed into the Global Assembly Cache (GAC). The Connector/NET installer handles this for you automatically, but in a custom installation the option may have been disabled.

The user used to connect to the MySQL server must have the following privileges to use the functionality provided by the MySQL Visual Studio Plugin:

- The `SELECT` privilege for the `INFORMATION_SCHEMA` database.
- The `EXECUTE` privilege for the `SHOW CREATE TABLE` statement.
- The `SELECT` privilege for the `mysql.proc` table (required for operations with stored procedures and functions).
- The `SELECT` privilege for the `mysql.func` table (required for operations with User Defined Functions (UDF)).
- The `EXECUTE` privilege for the `SHOW ENGINE STATUS` statement (required for retrieving extended error information).
- Appropriate privileges for performed operations (e.g. the `SELECT` privilege is required to browse data from a table etc.).

The MySQL Visual Studio Plugin is delivered as a MSI package that can be used to install, uninstall or reinstall the Provider. If you are not using Windows XP or Windows Server 2003 you upgrade the Windows Installer system to the latest version (see http://support.microsoft.com/default.aspx?scid=kb;EN-US;292539 for details).

The MSI-package is named `MySQL.VisualStudio.msi`. To install the MySQL Visual Studio Plugin, right click on the MSI file and select INSTALL. The installation process is as follow:

1. The standard Welcome dialog is opened. Click Next to continue installation.
2. The License agreement (GNU GPL) window is opened. Accept the agreement and click NEXT to continue.

3. The destination folder choice dialog is opened. Here you can point out the folder where the MySQL Visual Studio Plugin will be installed. The default destination folder is `%ProgramFilesDir%\MySQL\MySQL DDEX Data Provider`, where `%ProgramFilesDir%` is the Program Files folder of the installation machine. After choosing the destination folder, click Next to continue.
4. The installer will ask to confirm that installation. Click Install to start installation process.
5. The installation will now take place. At the end of this step the Visual Studio command table is rebuilt (this process may take several minutes).
6. Once installation is complete, click Finish to end the installation process.

To uninstall the MySQL Visual Studio Plugin, you can use either Add/Remove Programs component of the Control Panel or the same MSI-package. Choose the **Remove** option, and the Provider will be uninstalled automatically.

To repair the Provider, right click the MSI-package and choose the Repair option. The MySQL Visual Studio Plugin will be repaired automatically.

The installation package includes the following files:

- `MySQL.VisualStudio.dll` — the MySQL DDEX Provider assembly.
- `MySQL.Data.dll` — the assembly containing the MySQL Connector .NET which is used by the Provider.
- `MySql.VisualStudio.dll.config` — the configuration file for the MySQL Visual Studio Plugin. This file contains default values for the provider GUI layout.

  Note

  Do not remove this file before the first use of the Provider.
- `Register.reg` — the file with registry entries that can be used to register the MySQL DDEX Provider in the case of the manual installation.
- `Install.js` — the script used to register the Connector .NET as an ADO.NET data provider in the machine.config file.
- `Release notes.doc` — the document with release notes.

To install the Provider manually, copy all files of the installation package in a desired folder, then set the full path to the Provider assembly as a value of the CodeBase entry. For example:

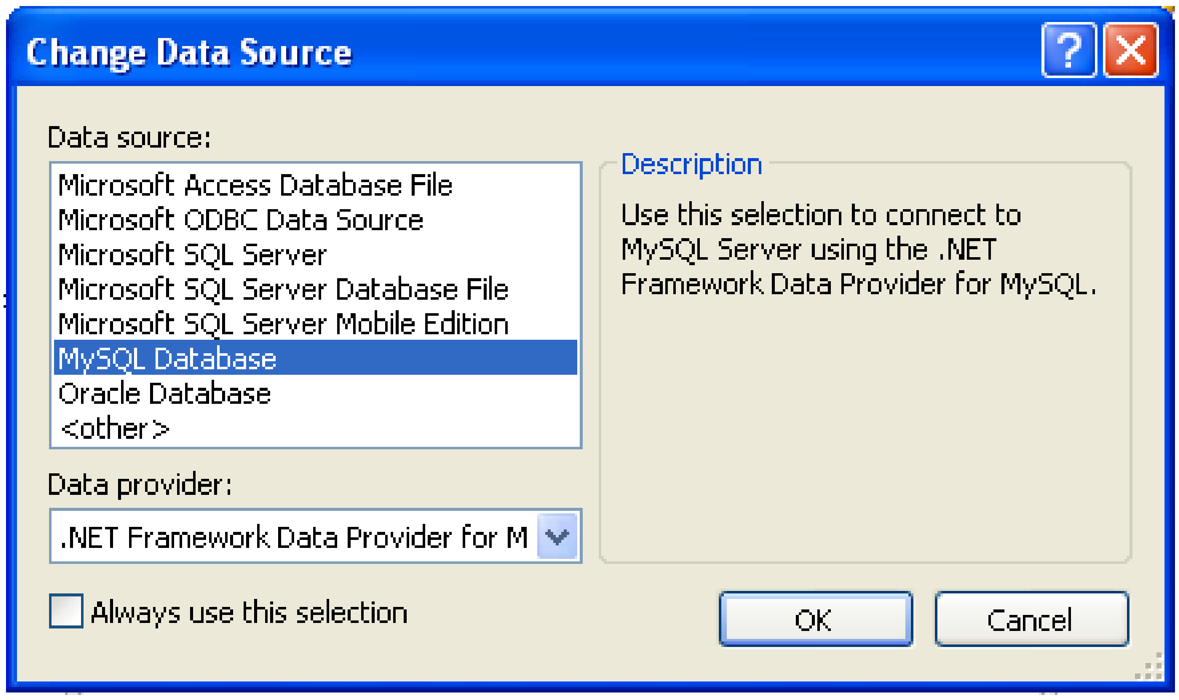
```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\Packages\{79A115C9-B133-4891-9E7B-242509DAD272}]@="MySql.Data.
"InprocServer32"="C:\\WINNT\\system32\\mscoree.dll"
"Class"="MySql.Data.VisualStudio.MySqlDataProviderPackage"
"CodeBase"="C:\\MySqlDdexProvider\\MySql.VisualStudio.dll"
```

Then import information from the Register.reg file to the registry by clicking of the file. At the confirmation dialog choose Yes. Next you must run the command devenv.exe /setup within a Command Prompt to rebuild the Visual Studio command table.

3.2. Creating a connection to the MySQL server

Once the MySQL Visual Studio Plugin is installed, you can use it to create, modify and delete connections to MySQL databases. To create a connection with a MySQL database, perform the following steps:

1. Start Visual Studio 2005 and open Server Explorer window by choosing the Server Explorer option from the View menu.
2. Right click on the **Data Connections** node and choose the Add Connection button.
3. The Add Connection dialog is opened. Press the Change button to choose MySQL Database as a data source.
4. Change Data Source dialog is opened. Choose MySQL Database in the list of data sources (or the `other` option, if MySQL Database is absent), and then choose **.NET Framework Data Provider for MySQL** in the combo box of data providers.



Change Data Source

Data source:

- Microsoft Access Database File
- Microsoft ODBC Data Source
- Microsoft SQL Server
- Microsoft SQL Server Database File
- Microsoft SQL Server Mobile Edition
- MySQL Database
- Oracle Database
- <other>

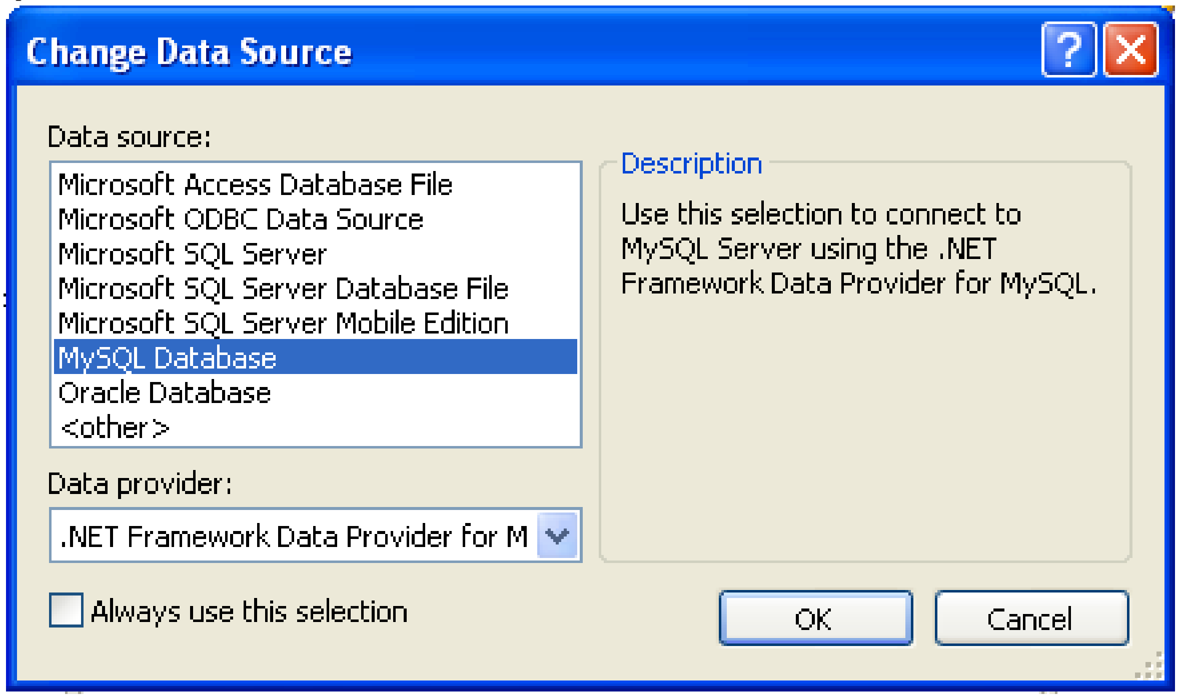
Description

Use this selection to connect to MySQL Server using the .NET Framework Data Provider for MySQL.

Data provider: .NET Framework Data Provider for M

Press OK to confirm your choice.

5. Enter the connection settings: the server host name (for example, localhost if the MySQL server is installed on the local machine), the user name, the password, and the default database schema. Note that you must specify the default schema name to open the connection.



Change Data Source

Data source:

- Microsoft Access Database File
- Microsoft ODBC Data Source
- Microsoft SQL Server
- Microsoft SQL Server Database File
- Microsoft SQL Server Mobile Edition
- MySQL Database
- Oracle Database
- <other>

Description

Use this selection to connect to MySQL Server using the .NET Framework Data Provider for MySQL.

Data provider: .NET Framework Data Provider for M

6. You can also set the port to connect with the MySQL server by pressing the ADVANCED button. To test a connection with the MySQL server, ser the server host name, the user name, and the password, and press the TEST CONNECTION button. If the test fails, check the connection values that you have supplied are correct and that the corresponding user and privileges have been configured on the MySQL server.

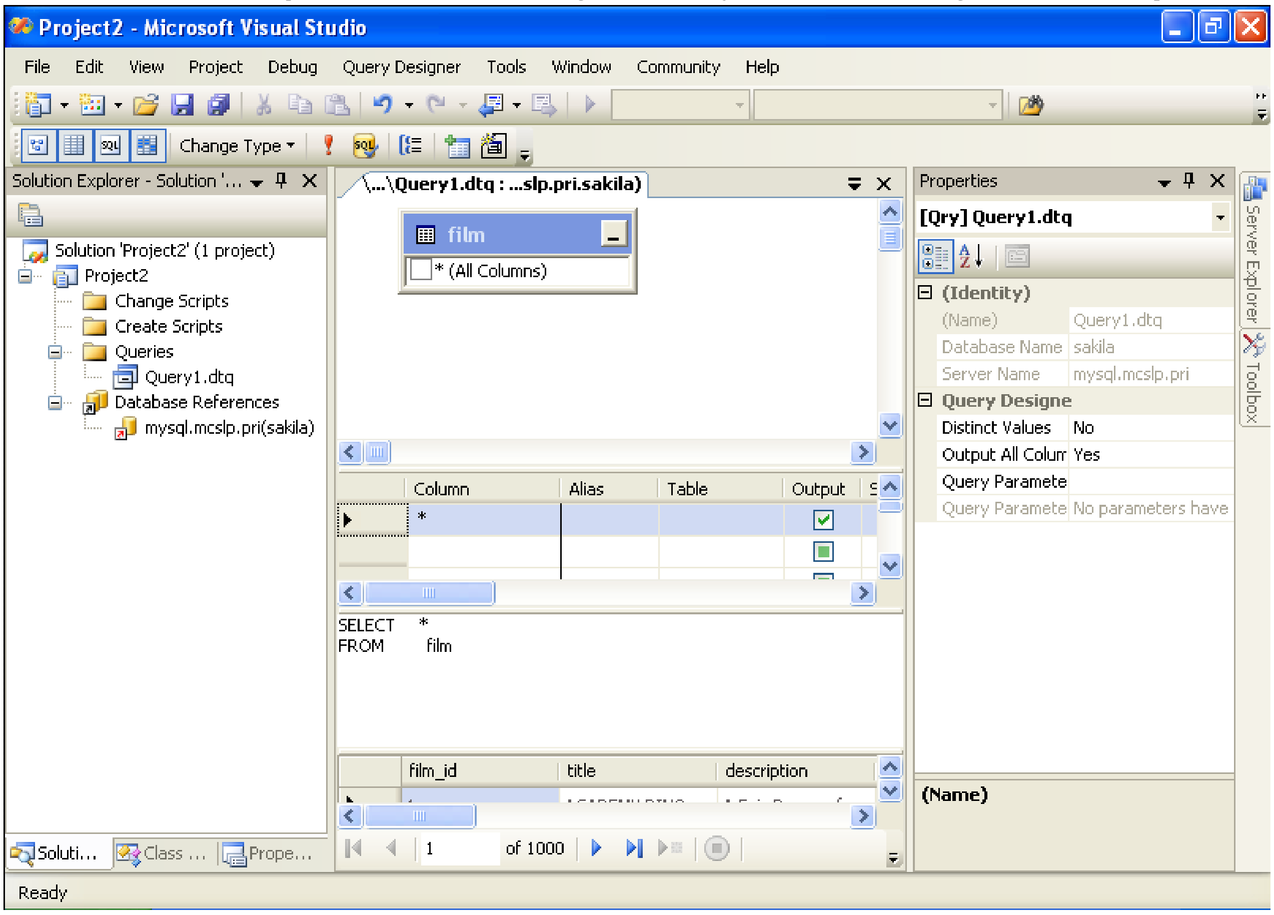
7. After you set all settings and test the connection, press OK. The newly created connection is displayed in Server Explorer. Now you can work with the MySQL server through standard Server Explorer interface.

After a connection is successfully established, all the connection settings are saved. When you next open Visual Studio, the connection to the MySQL server will appear within Server Explorer so that you can re-establish a connection to the MySQL server.

To modify and delete a connection, use the SERVER EXPLORER context menu for the corresponding node. You can modify any of the settings just by overwriting the existing values with new ones. Note that a connection should be modified or deleted only if no active editor for its objects is opened. Otherwise your data could be lost.

3.3. Using the MySQL Visual Studio Plugin

To work with a MySQL server using the MySQL Visual Studio Plugin, open the Visual Studio 2005, open the **SERVER EXPLORER**, and select the required connection. The working area of the MySQL Visual Studio Plugin consists of three parts.


Project2 - Microsoft Visual Studio
File Edit View Project Debug Query Designer Tools Window Community Help
Change Type
Solution Explorer - Solution '...
Solution 'Project2' (1 project)
Project2
Change Scripts
Create Scripts
Queries
Query1.dtq
Database References
mysql.mcslp.pri(sakila)
...\Query1.dtq : ...slp.pri.sakila)
film
* (All Columns)
Column Alias Table Output
Soluti... Class ... Prope...
Ready
Properties
[Qry] Query1.dtq
Server Explorer
Toolbox

```
SELECT    *
FROM      film
```

film_id	title	description

1 of 1000

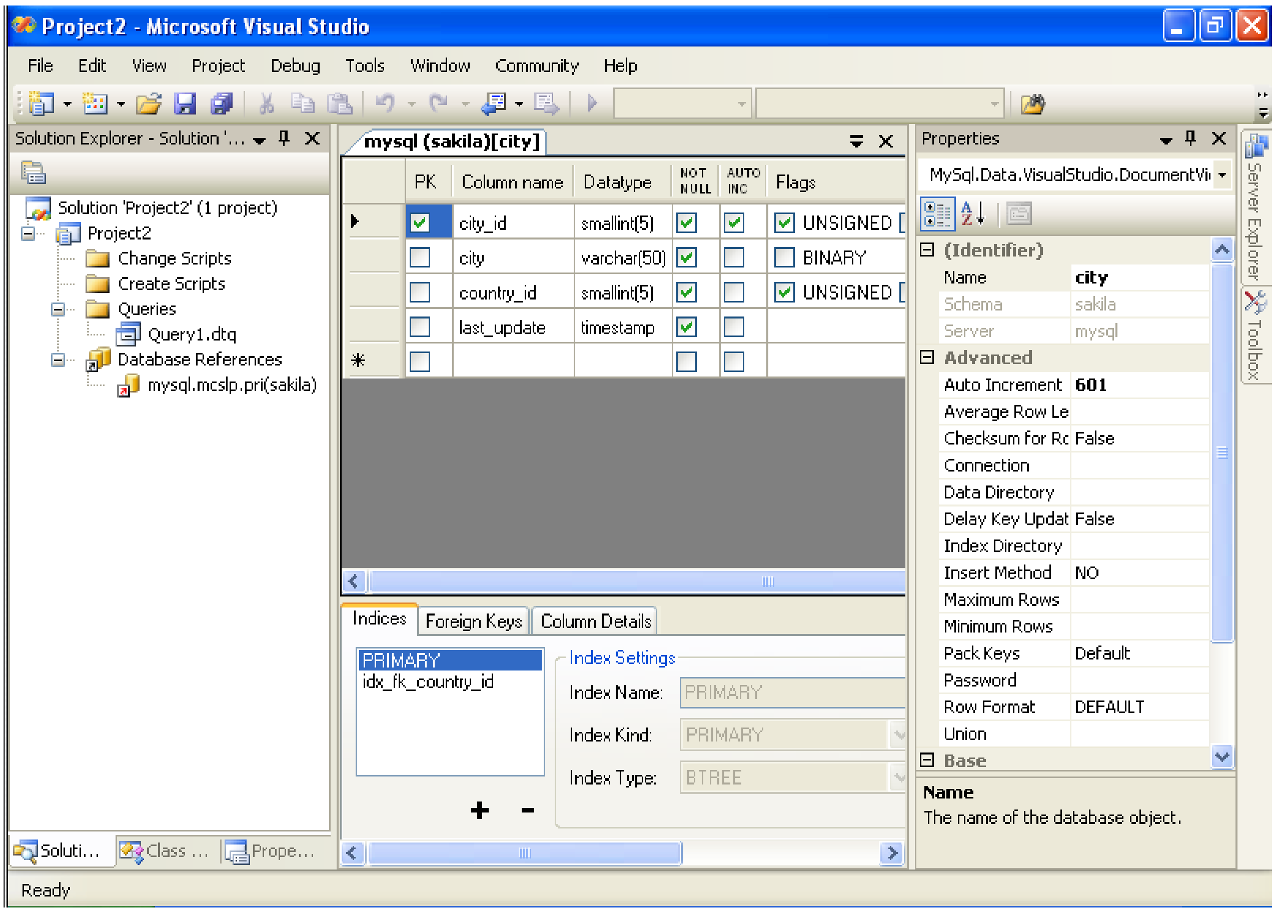
(Identity)	
(Name)	Query1.dtq
Database Name	sakila
Server Name	mysql.mcslp.pri
Query Designe	
Distinct Values	No
Output All Colum	Yes
Query Paramete	
Query Paramete	No parameters have

(Name)

- Database objects (tables, views, stored routines, triggers, and user defined functions) are displayed in the Server Explorer tree. Here you can choose an object and edit its properties and definition.
- Properties of a selected database object are displayed in the **PROPERTIES** panel. Certain properties can be edited directly within this window.
- The editor panel provides direct access to the SQL statement and definition of specific objects. Fore example, the SQL statements within a stored procedure definition are shown and edited within this panel.

3.3.1. Editing Tables

The Table Editor can be accessed through a mouse action on table-type node of Server Explorer. To create a new table, right click on the **TABLES** node (under the connection node) and choose the CREATE TABLE command from a context menu. To modify an existing table, double click on a node of the table you wish to modify, or right click on this node and choose the ALTER TABLE command from a context menu. Either of the commands opens the Table Editor.


Project2 - Microsoft Visual Studio
File Edit View Project Debug Tools Window Community Help
Solution Explorer - Solution '...
Solution 'Project2' (1 project)
Project2
Change Scripts
Create Scripts
Queries
Query1.dtq
Database References
mysql.mcslp.pri(sakila)
Soluti... Class ... Prope...
mysql (sakila)[city]
Indices Foreign Keys Column Details
Properties
MySql.Data.VisualStudio.DocumentVi
Server Explorer
Toolbox
Ready

PK	Column name	Datatype	NOT NULL	AUTO INC	Flags
✓	city_id	smallint(5)	✓	✓	✓ UNSIGNED
☐	city	varchar(50)	✓	☐	☐ BINARY
☐	country_id	smallint(5)	✓	☐	✓ UNSIGNED
☐	last_update	timestamp	✓	☐	
☐			☐	☐	

Indices: PRIMARY, idx_fk_country_id

Index Settings

Index Name: PRIMARY

Index Kind: PRIMARY

Index Type: BTREE

(Identifier)	
Name	city
Schema	sakila
Server	mysql
Advanced	
Auto Increment	601
Average Row Le	
Checksum for Rc	False
Connection	
Data Directory	
Delay Key Updat	False
Index Directory	
Insert Method	NO
Maximum Rows	
Minimum Rows	
Pack Keys	Default
Password	
Row Format	DEFAULT
Union	
Base	

Name
The name of the database object.

The MySQL Visual Studio Plugin Table Editor is implemented in a similar fashion to the standard Query Browser Table Editor, but with minor differences.

The Table Editor consists of the following parts:

- Columns Editor — for column creation, modification and deletion.
- Indexes tab — for table/column index management.
- Foreign Keys tab — for configuration of foreign keys.
- Column Details tab — used to set advanced column options.
- Properties window — used to set table properties.

To save changes you have made in the Table Editor, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. Before changes are saved, a confirmation dialog will be displayed to confirm that you want to update the corresponding object within the MySQL database.

3.3.1.1. Column Editor

You can use the Column Editor to set or change the name, data type, default value and other properties of a table column. To set the properties of an individual column, select the column using the mouse. Alternatively, you can move through the grid using **Tab** and **Shift+Tab** keys.

- To set or change the name, data type, default value and comment of a column, select the appropriate cell and edit the desired value.
- To set or unset flag-type column properties (i.e., primary key, `NOT NULL`, auto-incremented, flags), check or uncheck the corresponding checkboxes. Note that the available column flags will depend on the columns data type.
- To reorder columns, index columns or foreign key columns in the Column Editor, select the whole column you wish to reorder by clicking on the selector column at the left of the column grid. Then move the column by using **Ctrl+Up** (to move the

column up) and **Ctrl+Down** (to move the column down) keys.

- To delete a column, select it by clicking on the selector column at the left of the column grid, then press the **Delete** button on a keyboard.

3.3.1.2. Indexes tab

Index management is performed via the Indexes tab.

- To add an index, press the + button and set the properties in the **INDEX SETTINGS** groupbox at the right. You can set the index name, index kind, index type and a set of index columns.
- To remove an index, select the index from the list and press the - button.
- To change index settings, select the index from the list; detailed information about the index is displayed in the **INDEX SETTINGS** panel.

You cannot change a table column to an index column using drag and drop. Instead, you can add new index columns to a table and set their table columns by using the embedded editor within the Indexes tab

3.3.1.3. Foreign Keys tab

Foreign Key management is performed via the Foreign Keys tab.

- To add a foreign key, press the + button and set properties in the **FOREIGN KEYS SETTINGS** panel. You can set the foreign key name, referenced table name, foreign key columns and actions on update and delete.
- To remove a foreign key, select the foreign key and press the - button.
- To change foreign key settings, select the foreign key and use the **FOREIGN KEYS SETTINGS** panel to edit the properties.
- When a foreign key is changed, the MySQL Visual Studio Plugin generates two queries: the first query drops the changed keys and the second one recreates the new values. The reason for such a behavior is to avoid the Bug#8377 and Bug#8919.

> **Note**
>
> If changed values are for some reason inconsistent and cause the second query to fail, all affected foreign keys will be dropped. If this is the case, the MySQL Visual Studio Plugin will mark them as new in the Table Editor, and you will have to recreate them later. But if you close the Table Editor without saving, these foreign keys will be lost.

3.3.1.4. Column Details tab

The Column Details tab can be used to set column options. Besides the main column properties that are presented in the Column Editor, in the Column Details tab you can set two additional properties options: the character set and the collation sequence.

3.3.1.5. Table Properties window

There is no separate tab for table options and advanced options. All table options can be browsed and changed using the **PROPERTIES** window of Visual Studio 2005.

The following table properties can be set:

- **AUTO INCREMENT**
- **AVERAGE ROW LENGTH**
- **CHARACTER SET**
- **CHECKSUM FOR ROWS**
- **COLLATION**
- **COMMENT**

- **Connection**
- **Data Directory**
- **Delay Key Updates**
- **Engine**
- **Index Directory**
- **Insert Method**
- **Maximum Rows**
- **Minimum Rows**
- **Name**
- **Pack Keys**
- **Password**
- **Row Format**
- **Union**

Some of these properties can have arbitrary text values, others accept values from a predefined set.

The properties **Schema** and **Server** are read only.

3.3.2. Editing Table Data

The Table Data Editor, allows a user to browse, create and edit data of tables. The Table Data Editor is implemented as a simple data grid with auto generated columns.

To access the Table Data Editor, right click on a node representing the table or view in Server Explorer. From the nodes context menu, choose the Browse or Edit Data command. For tables and updatable views, this command opens the Table Data Editor in edit mode. For non-updatable views, this command opens the Table Data Editor in read-only mode.

When in the edit mode, you can modify table data by modifying the displayed table contents directly. To add a row, set desired values in the last row of the grid. To modify values, set new values in appropriate cells. To delete a row, select it by clicking on the selector column at the left of the grid, then press the Delete button.

To save changes you have made in the Table Data Editor, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

3.3.3. Editing Views

To create a new view, right click the Views node under the connection node in Server Explorer. From the nodes context menu, choose the Create View command. This command opens the SQL Editor.

To modify an existing view, double click on a node of the view you wish to modify, or right click on this node and choose the Alter View command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the view definition using SQL Editor, type the appropriate SQL statement in the SQL Editor.

Note

You should enter only the defining statement itself, without the `CREATE VIEW AS` preface.

All other view properties can be set in the **Properties** window. These properties are:

- **Algorithm**
- **Check Option**
- **Definer**
- **Name**

- **Security Type**

Some of these properties can have arbitrary text values, others accept values from a predefined set.

The properties **Is Updatable**, **Schema** and **Server** are readonly.

To save changes you have made, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

3.3.4. Editing Stored Procedures and Functions

To create a new stored procedure, right click the Stored Procedures node under the connection node in Server Explorer. From the nodes context menu, choose the Create Routine command. This command opens the SQL Editor.

To create a new stored function, right click the **Functions** node under the connection node in Server Explorer. From the node's context menu, choose the Create Routine command.

To modify an existing stored routine (procedure or function), double click on a node of the routine you wish to modify, or right click on this node and choose the Alter Routine command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the routine definition using SQL Editor, type this definition in the SQL Editor using standard SQL.

All other routine properties can be set in the **Properties** window. These properties are:

- Comment
- Data Access
- Definer
- Is Deterministic
- Security Type

Some of these properties can have arbitrary text values, others accept values only from a predefined set.

Also you can set all the options directly in the SQL Editor, using the standard `CREATE PROCEDURE` or `CREATE FUNCTION` statement. However, it is recommended to use the **Properties** window instead.

> **Note**
>
> You should never add the `CREATE` preface to the routine definition.

The properties **Name**, **Schema** and **Server** in the **Properties** window are read-only. Set or change the procedure name in the SQL editor.

To save changes you have made, use either Save or Save All buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database..

3.3.5. Editing Triggers

To create a new trigger, right click on a node of a table for which you wish to add a trigger. From the node's context menu, choose the Create Trigger command. This command opens the SQL Editor.

To modify an existing trigger, double click on a node of the trigger you wish to modify, or right click on this node and choose the Alter Trigger command from a context menu. Either of the commands opens the SQL Editor.

To create or alter the trigger definition using SQL Editor, type the trigger statement in the SQL Editor using standard SQL.

> **Note**
>
> You should enter only the trigger statement, that is the part of the `CREATE TRIGGER` query that is placed after the `FOR EACH ROW` clause.

All other trigger properties are set in the **Properties** window. These properties are:

- **DEFINER**
- **EVENT MANIPULATION**
- **NAME**
- **TIMING**

Some of these properties can have arbitrary text values, others accept values only from a predefined set.

The properties **EVENT TABLE**, **SCHEMA** and **SERVER** in the **PROPERTIES** window are read-only.

To save changes you have made, use either SAVE or SAVE ALL buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

3.3.6. Editing User Defined Functions (UDF)

To create a new User Defined Function (UDF), right click the UDFs node under the connection node in Server Explorer. From the node's context menu, choose the CREATE UDF command. This command opens the UDF Editor.

To modify an existing UDF, double click on a node of the UDF you wish to modify, or right click on this node and choose the Alter UDF command from a context menu. Either of the commands opens the UDF Editor.

The UDF editor allows you to set the following properties through the properties panel:

- **NAME**
- **SO-NAME (DLL NAME)**
- **RETURN TYPE**
- **IS AGGREGATE**

The property Server in the **PROPERTIES** window is read-only.

To save changes you have made, use either SAVE or SAVE ALL buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

3.3.7. Dropping database objects

Tables, views, stored routines, triggers, an UDFs can be dropped with the appropriate DROP command from its context menu: DROP TABLE, DROP VIEW, DROP ROUTINE, DROP TRIGGER, DROP UDF.

You will be asked to confirm the execution of the corresponding drop query in a confirmation dialog.

Dropping of multiple objects is not supported.

3.3.8. Cloning database objects

Tables, views, stored procedures and functions can be cloned with the appropriate CLONE command from its context menu: CLONE TABLE, CLONE VIEW, CLONE ROUTINE. The clone commands open the corresponding editor for a new object: the **TABLE EDITOR** for cloning a table and the SQL Editor for cloning a view or a routine.

To save the cloned object, use either SAVE or SAVE ALL buttons of the Visual Studio main toolbar, or just press **Ctrl+S**. A confirmation dialog will confirm whether you want the changes saved to the database.

3.4. Visual Studio Plugin Support

If you have a comment, or if you discover a bug, please, use our MySQL bug tracking system (http://bugs.mysql.com) to report problem or add your suggestion.

3.4.1. Visual Studio Plugin FAQ

Questions

- 3.4.1.1: When creating a connection, typing the connection details causes the connection window to immediately close.

Questions and Answers

3.4.1.1: When creating a connection, typing the connection details causes the connection window to immediately close.

There are known issues with versions of Connector/NET earlier than 5.0.2. Connector/NET 1.0.x is known not to work. If you have any of these versions installed, or have previously upgraded from an earlier version, uninstall Connector/NET completely and then install Connector/NET 5.0.2.

Chapter 4. MySQL Connector/J

MySQL provides connectivity for client applications developed in the Java programming language via a JDBC driver, which is called MySQL Connector/J.

MySQL Connector/J is a JDBC Type 4 driver. Different versions are available that are compatible with the JDBC 3.0 and JDBC 4.0 specifications. The Type 4 designation means that the driver is pure-Java implementation of the MySQL protocol and does not rely on the MySQL client libraries.

Although JDBC is useful by itself, we would hope that if you are not familiar with JDBC that after reading the first few sections of this manual, that you would avoid using naked JDBC for all but the most trivial problems and consider using one of the popular persistence frameworks such as Hibernate, Spring's JDBC templates or Ibatis SQL Maps to do the majority of repetitive work and heavier lifting that is sometimes required with JDBC.

This section is not designed to be a complete JDBC tutorial. If you need more information about using JDBC you might be interested in the following online tutorials that are more in-depth than the information presented here:

- JDBC Basics — A tutorial from Sun covering beginner topics in JDBC
- JDBC Short Course — A more in-depth tutorial from Sun and JGuru

Key topics:

- For help with connection strings, connection options setting up your connection through JDBC, see Section 4.4.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”.
- For tips on using Connector/J and JDBC with generic J2EE toolkits, see Section 4.5.2, “Using Connector/J with J2EE and Other Java Frameworks”.
- Developers using the Tomcat server platform, see Section 4.5.2.2, “Using Connector/J with Tomcat”.
- Developers using JBoss, see Section 4.5.2.3, “Using Connector/J with JBoss”.
- Developers using Spring, see Section 4.5.2.4, “Using Connector/J with Spring”.

MySQL Enterprise
MySQL Enterprise subscribers will find more information about using JDBC with MySQL in the Knowledge Base articles about JDBC. Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information, see http://www.mysql.com/products/enterprise/advisors.html.

4.1. Connector/J Versions

There are currently four versions of MySQL Connector/J available:

- Connector/J 5.1 is the Type 4 pure Java JDBC driver, which conforms to the JDBC 3.0 and JDBC 4.0 specifications. It provides compatibility with all the functionality of MySQL, including 4.1, 5.0, 5.1 and the 6.0 alpha release featuring the new Falcon storage engine. Connector/J 5.1 provides ease of development features, including auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for the JDBC-4.0 XML processing, per connection client information, `NCHAR`, `NVARCHAR` and `NCLOB` 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 with connectivity to MySQL 3.x or MySQL 4.1 servers, although it will provide 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:

Connector/J version	Driver Type	JDBC version	MySQL Server version	Status
5.1	4	3.0, 4.0	4.1, 5.0, 5.1, 6.0	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 covers all four connector versions, with specific notes given where a setting applies to a specific option.

4.1.1. Java Versions Supported

The following table summarizes Connector/J Java dependencies:

Connector/J version	Java RTE required	JDK required (to build source code)
5.1	1.4.x, 1.5.x, 1.6.x	1.6.x and 1.5.x (or older)
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

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

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.

If you are building Connector/J from source code using the source distribution (see Section 4.2.4, “Installing from the Development Source Tree”) then you must use JDK 1.4.2 or newer to compile the Connector package. For Connector/J 5.1 you must have both JDK-1.6.x. and JDK-1.5.x installed in order to be able to build the source code.

4.2. Connector/J Installation

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 enables you to customize your installation further. With either solution, you must manually add the Connector/J location to your Java `CLASSPATH`.

If you are upgrading from a previous version, read the upgrade information before continuing. See Section 4.2.3, “Upgrading from an Older Version”.

4.2.1. Installing Connector/J from a Binary Distribution

The easiest method of installation is to use the binary distribution of the Connector/J package. The binary distribution is available either as a Tar/Gzip or Zip file which you must extract to a suitable location and then optionally make the information about the package available by changing your `CLASSPATH` (see Section 4.2.2, “Installing the Driver and Configuring the `CLASSPATH`”).

MySQL Connector/J is distributed as a .zip or .tar.gz archive containing the sources, the class files, and the JAR archive named `mysql-connector-java-[version]-bin.jar`, and starting with Connector/J 3.1.8 a debug build of the driver in a file named `mysql-connector-java-[version]-bin-g.jar`.

Starting with Connector/J 3.1.9, the `.class` files that constitute the JAR files are only included as part of the driver JAR file.

You should not use the debug build of the driver unless instructed to do so when reporting a problem or a bug to MySQL AB, 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.

You will need to 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. You will need to use GNU tar (or an application that understands the GNU tar archive format) to unpack the .tar.gz variant of the distribution.

4.2.2. 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 your JVM.

If you are going to use the driver with the JDBC DriverManager, you would use `com.mysql.jdbc.Driver` as the class that implements `java.sql.Driver`.

You can set the `CLASSPATH` environment variable under UNIX, Linux or Mac 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, under a C shell (csh, tcsh) you would add the Connector/J driver to your `CLASSPATH` using the following:

```
shell> setenv CLASSPATH /path/mysql-connector-java-[ver]-bin.jar:$CLASSPATH
```

Or with a Bourne-compatible shell (sh, ksh, bash):

```
shell> export set CLASSPATH=/path/mysql-connector-java-[ver]-bin.jar:$CLASSPATH
```

Within Windows 2000, Windows XP, Windows Server 2003 and Windows Vista, you must set the environment variable through the System Control Panel.

If you want to use MySQL Connector/J with an application server such as GlassFish, Tomcat or JBoss, you will have to 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 4.5.2, “Using Connector/J with J2EE and Other Java Frameworks”. 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 via the `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` class, which supports XA distributed transactions when used in combination with MySQL server version 5.0.

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

- user
- password
- serverName (see the previous section about fail-over hosts)
- databaseName
- port

4.2.3. Upgrading from an Older Version

We try to keep the upgrade process as easy as possible, however as is the case with any software, sometimes changes need to be made in new versions to support new features, improve existing functionality, or comply with new standards.

This section has information about what users who are upgrading from one version of Connector/J to another (or to a new version of the MySQL server, with respect to JDBC functionality) should be aware of.

4.2.3.1. 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, SQL-State codes returned in error messages by the server and various performance enhancements that can be enabled or disabled via configuration properties.

- **Unicode Character Sets** — See the next section, as well as Character Set Support, for information on this new feature of MySQL. 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 via 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 can not 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 allowable 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 via noDatetimeStringSync=true (the default value is `false`) so that you can retrieve the unaltered all-zero value as a String. It should be noted 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.String`s 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.

  You should 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.

4.2.3.2. 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 was 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()`. If you want to use the old non-compliant 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.

4.2.3.3. 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, you should 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`

4.2.4. Installing from the Development Source Tree

Caution

You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL Connector/J up and running on your system, you should use a standard binary release distribution.

To install MySQL Connector/J from the development source tree, make sure that you have the following prerequisites:

- A Bazaar client, to check out the sources from our Launchpad repository (available from http://bazaar-vcs.org/).
- Apache Ant version 1.7 or newer (available from http://ant.apache.org/).
- JDK 1.4.2 or later. Although MySQL Connector/J can be be used with older JDKs, to compile it from source you must have at least JDK 1.4.2. If you are building Connector/J 5.1 you will need JDK 1.6.x and an older JDK such as JDK 1.5.x. You will then need to point your JAVA_HOME environment variable at the older installation.

The source code repository for MySQL Connector/J is located on Launchpad at https://code.launchpad.net/connectorj.

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

1. Check out the latest code from the branch that you want with one of the following commands.

   To check out the latest development branch use:

   ```
   shell> bzr branch lp:connectorj
   ```

   This creates a `connectorj` subdirectory in the current directory that contains the latest sources for the requested branch.

   To check out the latest 5.1 code use:

   ```
   shell> bzr branch lp:connectorj/5.1
   ```

   This will create a `5.1` subdirectory in the current directory containing the latest 5.1 code.

2. If you are building Connector/J 5.1 make sure that you have both JDK 1.6.x installed and an older JDK such as JDK 1.5.x. This is because Connector/J supports both JDBC 3.0 (which was prior to JDK 1.6.x) and JDBC 4.0. Set your JAVA_HOME environment variable to the path of the older JDK installation.

3. Change location to either the `connectorj` or `5.1` directory, depending on which branch you want to build, to make it your current working directory. For example:

   ```
   shell> cd connectorj
   ```

4. If you are building Connector/J 5.1 you need to edit the `build.xml` to reflect the location of your JDK 1.6.x installation. The lines that you need to change are:

   ```
               <property name="com.mysql.jdbc.java6.javac" value="C:\jvms\jdk1.6.0\bin\javac.exe" />
               <property name="com.mysql.jdbc.java6.rtjar" value="C:\jvms\jdk1.6.0\jre\lib\rt.jar" />
   ```

   Alternatively, you can set the value of these property names through the Ant `-D` option.

5. Issue the following command to compile the driver and create a `.jar` file suitable for installation:

   ```
   shell> ant dist
   ```

   This creates a `build` directory in the current directory, where all build output will go. A directory is created in the `build` directory that includes the version number of the sources you are building from. This directory contains the sources, compiled `.class` files, and a `.jar` file suitable for deployment. For other possible targets, including ones that will create a fully packaged distribution, issue the following command:

   ```
   shell> ant -projecthelp
   ```

6. A newly created `.jar` file containing the JDBC driver will be placed in the directory `build/mysql-connector-java-[version]`.

   Install the newly created JDBC driver as you would a binary `.jar` file that you download from MySQL by following the instructions in Section 4.2.2, “Installing the Driver and Configuring the `CLASSPATH`”.

A package containing both the binary and source code for Connector/J 5.1 can also be found at the following location: Connector/J 5.1 Download

4.3. Connector/J Examples

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

- Example 4.1, “Connector/J: Obtaining a connection from the `DriverManager`”
- Example 4.2, “Connector/J: Using java.sql.Statement to execute a `SELECT` query”
- Example 4.3, “Connector/J: Calling Stored Procedures”
- Example 4.4, “Connector/J: Using `Connection.prepareCall()`”
- Example 4.5, “Connector/J: Registering output parameters”
- Example 4.6, “Connector/J: Setting `CallableStatement` input parameters”
- Example 4.7, “Connector/J: Retrieving results and output parameter values”
- Example 4.8, “Connector/J: Retrieving `AUTO_INCREMENT` column values using `Statement.getGeneratedKeys()`”
- Example 4.9, “Connector/J: Retrieving `AUTO_INCREMENT` column values using `SELECT LAST_INSERT_ID()`”
- Example 4.10, “Connector/J: Retrieving `AUTO_INCREMENT` column values in `Updatable ResultSets`”
- Example 4.11, “Connector/J: Using a connection pool with a J2EE application server”
- Example 4.12, “Connector/J: Example of transaction with retry logic”

4.4. Connector/J (JDBC) Reference

This section of the manual contains reference material for MySQL Connector/J, some of which is automatically generated during the Connector/J build process.

4.4.1. 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 to remain backward-compatible with MM.MySQL. You should use this class name when registering the driver, or when otherwise configuring software to use MySQL Connector/J.

The JDBC URL format for MySQL Connector/J is as follows, with items in square brackets ([,]) being optional:

```
jdbc:mysql://[host][,failoverhost...][:port]/[database] »
[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

If the host name is not specified, it defaults to 127.0.0.1. If the port is not specified, it defaults to 3306, the default port number for MySQL servers.

```
jdbc:mysql://[host:port],[host:port].../[database] »
[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...
```

If the database is not specified, the connection will be made with no default database. In this case, you will need to either call the `setCatalog()` method on the Connection instance or fully-specify table names using the database name (i.e. `SELECT dbname.tablename.colname FROM dbname.tablename...`) in your SQL. Not specifying the database to use upon connection is generally only useful when building tools that work with multiple databases, such as GUI database managers.

MySQL Connector/J has fail-over support. This allows the driver to fail-over to any number of slave hosts and still perform read-only queries. Fail-over only happens when the connection is in an `autoCommit(true)` state, because fail-over can not happen reliably when a transaction is in progress. Most application servers and connection pools set `autoCommit` to `true` at the end of every transaction/connection use.

The fail-over functionality has the following behavior:

- If the URL property autoReconnect is false: Failover only happens at connection initialization, and failback occurs when the driver determines that the first host has become available again.
- If the URL property autoReconnect is true: Failover happens when the driver determines that the connection has failed (before *every* query), and falls back to the first host when it determines that the host has become available again (after `queriesBeforeRetryMaster` queries have been issued).

In either case, whenever you are connected to a "failed-over" server, the connection will be set to read-only state, so queries that would modify data will have exceptions thrown (the query will **never** be processed by the MySQL server).

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.

  > **Note**
  >
  > If the mechanism you use to configure a JDBC URL is XML-based, you will need to use the XML character literal &amp; to separate configuration parameters, as the ampersand is a reserved character for XML.

The properties are listed in the following tables.

Connection/Authentication.

Property Name	Definition	Default Value	Since Version
user	The user to connect as		all versions
password	The password to use when connecting		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.	com.mysql.jdbc.StandardSocketFactory	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'.	0	3.0.1
socketTimeout	Timeout on network socket operations (0, the default means no timeout).	0	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.		5.1.4
useConfigs	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.		3.1.5
interactiveClient	Set the CLIENT_INTERACTIVE flag, which tells MySQL to timeout connections based on INTERACTIVE_TIMEOUT instead of WAIT_TIMEOUT	false	3.1.0
localSocketAddress	Host name 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.		5.0.5
mysqlIOFactory	The name of the class which implements "com.mysql.jdbc.MysqlIO" for communicating with mysqld. (default is "com.mysql.jdbc.MysqlIOprotocol")	com.mysql.jdbc.MysqlIOprotocol	6.0.0
propertiesTransform	An implementation of com.mysql.jdbc.ConnectionPropertiesTransform that the driver will use to modify URL properties passed to the driver before attempting a connection		3.1.4
useCompression	Use zlib compression when communicating with the server (true/false)? Defaults to 'false'.	false	3.0.17

Networking.

Property Name	Definition	Default Value	Since Version
tcpKeepAlive	If connecting using TCP/IP, should the driver set SO_KEEPALIVE?	true	5.0.7
tcpNoDelay	If connecting using TCP/IP, should the driver set SO_TCP_NODELAY (disabling the Nagle Algorithm)?	true	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)	0	5.0.7
tcpSndBuf	If connecting using TCP/IP, shuold the driver set SO_SND_BUF to the given value? The default value of '0', means use the platform default value for this property)	0	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.	0	5.0.7

High Availability and Clustering.

Property Name	Definition	Default Value	Since Version
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 do not 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, investigate setting the MySQL server variable "wait_timeout" to some high value rather than the default of 8 hours.	false	1.1
autoReconnectForPools	Use a reconnection strategy appropriate for connection pools (defaults to 'false')	false	3.1.3
failOverReadOnly	When failing over in autoReconnect mode, should the connection be set to 'read-only'?	true	3.0.12
maxReconnects	Maximum number of reconnects to attempt if autoReconnect is true, default is '3'.	3	1.1
reconnectAtTxEnd	If autoReconnect is set to true, should the driver attempt reconnections at the end of every transaction?	false	3.0.10
initialTimeout	If autoReconnect is enabled, the initial time to wait between reconnect attempts (in seconds, defaults to '2').	2	1.1
roundRobinLoadBalance	When autoReconnect is enabled, and failoverReadonly is false, should we pick hosts to connect to on a round-robin basis?	false	3.1.2
queriesBeforeRetryMaster	Number of queries to issue before falling back to master when failed over (when using multi-host failover). Whichever condition is met first, 'queriesBeforeRetryMaster' or 'secondsBeforeRetryMaster' will cause an attempt to be made to reconnect to the master. Defaults to 50.	50	3.0.2
secondsBeforeRetryMaster	How long should the driver wait, when failed over, before attempting	30	3.0.2
selfDestructOnPingMaxOperations	=If set to a nonzero value, the driver will report close the connection and report failure when Connection.ping() or Connection.isValid(int) is called if the connnection's count of commands sent to the server exceeds this value.	0	5.1.6
selfDestructOnPingSecondsLifetime	If set to a nonzero value, the driver will report close the connection and report failure when Connection.ping() or Connection.isValid(int) is called if the connnection's lifetime exceeds this value.	0	5.1.6
resourceId	A globally unique name that identifies the resource that this datasource or connection is connected to, used for XAResource.isSameRM() when the driver cannot determine this value based on host names used in the URL		5.0.1

Security.

Property Name	Definition	Default Value	Since Version
allowMultiQueries	Allow the use of ';' to delimit multiple queries during one statement (true/false), defaults to 'false'	false	3.1.1
useSSL	Use SSL when communicating with the server (true/false), defaults to 'false'	false	3.0.2
requireSSL	Require SSL connection if useSSL=true? (defaults to 'false').	false	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*" proper-	true	5.1.6

	ties, rather than system properties.		
clientCertificateKeyStoreUrl	URL to the client certificate KeyStore (if not specified, use defaults)		5.1.0
clientCertificateKeyStoreType	KeyStore type for client certificates (NULL or empty means use default, 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.		5.1.0
clientCertificateKeyStorePassword	Password for the client certificates KeyStore		5.1.0
trustCertificateKeyStoreUrl	URL to the trusted root certificate KeyStore (if not specified, use defaults)		5.1.0
trustCertificateKeyStoreType	KeyStore type for trusted root certificates (NULL or empty means use default, 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.		5.1.0
trustCertificateKeyStorePassword	Password for the trusted root certificates KeyStore		5.1.0
allowLoadLocalInfile	Should the driver allow use of 'LOAD DATA LOCAL INFILE...' (defaults to 'true').	true	3.0.3
allowUrlInLocalInfile	Should the driver allow URLs in 'LOAD DATA LOCAL INFILE' statements?	false	3.1.4
paranoid	Take measures to prevent exposure sensitive information in error messages and clear data structures holding sensitive data when possible? (defaults to 'false')	false	3.0.1

Performance Extensions.

Property Name	**Definition**	**Default Value**	**Since Version**
callableStmtCacheSize	If 'cacheCallableStmts' is enabled, how many callable statements should be cached?	100	3.1.2
metadataCacheSize	The number of queries to cache ResultSetMetadata for if cacheResultSetMetaData is set to 'true' (default 50)	50	3.1.1
prepStmtCacheSize	If prepared statement caching is enabled, how many prepared statements should be cached?	25	3.0.10
prepStmtCacheSqlLimit	If prepared statement caching is enabled, what is the largest SQL the driver will cache the parsing for?	256	3.0.10
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.	true	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.getCurrentTimeMillis() per query.	true	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?	false	5.0.0
blobSendChunkSize	Chunk to use when sending BLOB/CLOBs via ServerPreparedStatements	1048576	3.1.9
cacheCallableStmts	Should the driver cache the parsing stage of CallableStatements	false	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?	false	3.0.10

cacheResultSetMetadata	Should the driver cache ResultSetMetaData for Statements and PreparedStatements? (Req. JDK-1.4+, true/false, default 'false')	false	3.1.1
cacheServerConfiguration	Should the driver cache the results of 'SHOW VARIABLES' and 'SHOW COLLATION' on a per-URL basis?	false	3.1.5
defaultFetchSize	The driver will call setFetchSize(n) with this value on all newly-created Statements	0	3.1.9
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.	false	3.1.7
dynamicCalendars	Should the driver retrieve the default calendar when required, or cache it per connection/session?	false	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)?	false	3.1.3
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 will not 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.	true	5.0.6
holdResultsOpenOverStatementClose	Should the driver close result sets on Statement.close() as required by the JDBC specification?	false	3.1.7
largeRowSizeThreshold	What size result set row should the JDBC driver consider "large", and thus use a more memory-efficient way of representing the row internally?	2048	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.	random	5.0.6
locatorFetchBufferSize	If 'emulateLocators' is configured to 'true', what size buffer should be used when fetching BLOB data for getBinaryInputStream?	1048576	3.2.1
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 do not specify stream lengths when using PreparedStatement.set*Stream(), the driver will not 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.	false	3.1.13
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.	true	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	true	5.0.6

	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?		
useFastDateParsing	Use internal String->Date/Time/Timestamp conversion routines to avoid excessive object creation?	true	5.0.5
useFastIntParsing	Use internal String->Integer conversion routines to avoid excessive object creation?	true	3.1.4
useJvmCharsetConverters	Always use the character encoding routines built into the JVM, rather than using lookup tables for single-byte character sets?	false	5.0.1
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?	false	3.1.7
useReadAheadInput	Use newer, optimized non-blocking, buffered input stream when reading from the server?	true	3.1.5

Debugging/Profiling.

Property Name	Definition	Default Value	Since Version
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)	com.mysql.jdbc.log.StandardLogger	3.1.1
gatherPerfMetrics	Should the driver gather performance metrics, and report them via the configured logger every 'reportMetricsIntervalMillis' milliseconds?	false	3.1.2
profileSQL	Trace queries and their execution/fetch times to the configured logger (true/false) defaults to 'false'	false	3.1.0
profileSql	Deprecated, use 'profileSQL' instead. Trace queries and their execution/fetch times on STDERR (true/false) defaults to 'false'		2.0.14
reportMetricsIntervalMillis	If 'gatherPerfMetrics' is enabled, how often should they be logged (in ms)?	30000	3.1.2
maxQuerySizeToLog	Controls the maximum length/size of a query that will get logged when profiling or tracing	2048	3.1.3
packetDebugBufferSize	The maximum number of packets to retain when 'enablePacketDebug' is true	20	3.1.3
slowQueryThresholdMillis	If 'logSlowQueries' is enabled, how long should a query (in ms) before it is logged as 'slow'?	2000	3.1.2
slowQueryThresholdNanos	If 'useNanosForElapsedTime' is set to true, and this property is set to a nonzero value, the driver will use this threshold (in nanosecond units) to determine if a query was slow.	0	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')?	false	3.1.1
autoGenerateTestcaseScript	Should the driver dump the SQL it is executing, including server-side prepared statements to STDERR?	false	3.1.9
autoSlowLog	Instead of using slowQueryThreshold* 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?	true	5.1.4
clientInfoProvider	The name of a class that implements the com.mysql.jdbc.JDBC4ClientInfoProvider interface in order to support JDBC-4.0's Connection.get/setClientInfo() methods	com.mysql.jdbc.JDBC4CommentClientInfoProvider	5.1.0
dumpMetadataOnColumnNotFound	Should the driver dump the field-level metadata of a result set	false	3.1.13

	into the exception message when ResultSet.findColumn() fails?		
dumpQueriesOnException	Should the driver dump the contents of the query sent to the server in the message for SQLExceptions?	false	3.1.3
enablePacketDebug	When enabled, a ring-buffer of 'packetDebugBufferSize' packets will be kept, and dumped when exceptions are thrown in key areas in the driver's code	false	3.1.3
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?	false	3.1.2
includeInnodbStatusInDeadlockExceptions	Include the output of "SHOW ENGINE INNODB STATUS" in exception messages when deadlock exceptions are detected?	false	5.0.7
logSlowQueries	Should queries that take longer than 'slowQueryThresholdMillis' be logged?	false	3.1.2
logXaCommands	Should the driver log XA commands sent by MysqlXaConnection to the server, at the DEBUG level of logging?	false	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.	com.mysql.jdbc.profiler.LoggingProfilerEventHandler	5.1.6
resultSetSizeThreshold	If the usage advisor is enabled, how many rows should a result set contain before the driver warns that it is suspiciously large?	100	5.0.5
traceProtocol	Should trace-level network protocol be logged?	false	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)?	false	5.0.7

Miscellaneous.

Property Name	Definition	Default Value	Since Version
useUnicode	Should the driver use Unicode character encodings when handling strings? Should only be used when the driver cannot 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'	true	1.1g
characterEncoding	If 'useUnicode' is set to true, what character encoding should the driver use when dealing with strings? (defaults is to 'autodetect')		1.1g
characterSetResults	Character set to tell the server to return results as.		3.0.13
connectionCollation	If set, tells the server to use this collation via 'set collation_connection'		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 cannot handle natively.	false	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.		5.1.3
utf8OutsideBmpIncludedColumnNamePattern	Used to specify exclusion rules to "utf8OutsideBmpExcludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package.		5.1.3
sessionVariables	A comma-separated list of name/value pairs to be sent as SET SESSION ... to the server when the driver connects.		3.1.8
allowNanAndInf	Should the driver allow NaN or +/- INF values in PreparedStatement.setDouble()?	false	3.1.5

autoClosePStmtStreams	Should the driver automatically call .close() on streams/readers passed as arguments via set*() methods?	false	3.1.12
autoDeserialize	Should the driver automatically detect and de-serialize objects stored in BLOB fields?	false	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?	false	5.0.8
capitalizeTypeNames	Capitalize type names in DatabaseMetaData? (usually only useful when using WebObjects, true/false, defaults to 'false')	true	2.0.7
clobCharacterEncoding	The character encoding to use for sending and retrieving TEXT, MEDIUMTEXT and LONGTEXT values instead of the configured connection characterEncoding		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.	false	3.0.9
continueBatchOnError	Should the driver continue processing batch commands if one statement fails. The JDBC spec allows either way (defaults to 'true').	true	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.	false	3.1.9
emptyStringsConvertToZero	Should the driver allow conversions from empty string fields to numeric values of '0'?	true	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(), 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.	false	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?	true	3.1.7
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?	false	5.0.8
generateSimpleParameterMetadata	Should the driver generate simplified parameter metadata for PreparedStatements when no metadata is available either because the server couldn't support preparing the statement, or server-side prepared statements are disabled?	false	5.0.5
ignoreNonTxTables	Ignore non-transactional table warning for rollback? (defaults to 'false').	false	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.	true	3.1.2
maxRows	The maximum number of rows to return (0, the default means return all rows).	-1	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)	600	5.1.0
noAccessToProcedureBodies	When determining procedure parameter types for CallableStatements, and the connected user cannot access procedure bodies through "SHOW CREATE PROCEDURE" or select on mysql.proc should the driver instead create basic metadata (all	false	5.0.3

	parameters reported as IN VARCHARs, but allowing registerOutParameter() to be called on them anyway) instead of throwing an exception?		
noDatetimeStringSync	Do not ensure that ResultSet.getDatetimeType().toString().equals(ResultSet.getString())	false	3.1.7
noTimezoneConversionForTimeType	Do not convert TIME values using the server timezone if 'useTimezone'='true'	false	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)	true	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)	true	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)?	false	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)?	false	5.0.6
pedantic	Follow the JDBC spec to the letter.	false	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	false	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.	false	5.0.5
processEscapeCodesForPrepStmts	Should the driver process escape codes in queries that are prepared?	true	3.1.12
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')?	false	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.	false	3.1.11
rollbackOnPooledClose	Should the driver issue a rollback() when the logical connection in a pool is closed?	true	3.0.15
runningCTS13	Enables workarounds for bugs in Sun's JDBC compliance testsuite version 1.3	false	3.1.7
serverTimezone	Override detection/mapping of timezone. Used when timezone from server doesn't map to Java timezone		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 in the chain, from left-to-right order, as specified in this property.		5.1.1
strictFloatingPoint	Used only in older versions of compliance test	false	3.0.0

strictUpdates	Should the driver do strict checking (all primary keys selected) of updatable result sets (true, false, defaults to 'true')?	true	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)?	true	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?	false	3.1.9
treatUtilDateAsTimestamp	Should the driver treat java.util.Date as a TIMESTAMP for the purposes of PreparedStatement.setObject()?	true	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')	false	2.0.3
useGmtMillisForDatetimes	Convert between session timezone and GMT before creating Date and Timestamp instances (value of "false" is legacy behavior, "true" leads to more JDBC-compliant behavior.	false	3.1.12
useHostsInPrivileges	Add '@hostname' to users in DatabaseMetaData.getColumn/TablePrivileges() (true/false), defaults to 'true'.	true	3.0.2
useInformationSchema	When connected to MySQL-5.0.7 or newer, should the driver use the INFORMATION_SCHEMA to derive information used by DatabaseMetaData?	false	5.0.0
useJDBCCompliantTimezoneShift	Should the driver use JDBC-compliant rules when converting TIME/TIMESTAMP/DATETIME values' timezone information for those JDBC arguments which take a java.util.Calendar argument? (Notice that this option is exclusive of the "useTimezone=true" configuration option.)	false	5.0.0
useLegacyDatetimeCode	Use code for DATE/TIME/DATETIME/TIMESTAMP handling in result sets and statements that consistently handles timezone 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?	true	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.getColumnName() or ResultSetMetaData.getTableName() rather than the original column/table name? In 5.0.x, the default value was true.	false	5.0.4
useOldUTF8Behavior	Use the UTF-8 behavior the driver did when communicating with 4.0 and older servers	false	3.1.6
useOnlyServerErrorMessages	Do not prepend 'standard' SQLState error messages to error messages returned by the server.	true	3.0.15
useSSPSCompatibleTimezoneShift	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.	false	5.0.5
useServerPrepStmts	Use server-side prepared statements if the server supports them?	false	3.1.0
useSqlStateCodes	Use SQL Standard state codes instead of 'legacy' X/Open/SQL state codes (true/false), default is 'true'	true	3.1.3
useStreamLengthsInPrepStmts	Honor stream length parameter in PreparedStatement/ResultSet.setXXXStream() method calls (true/false, defaults to 'true')?	true	3.0.2
useTimezone	Convert time/date types between client and server timezones (true/false, defaults to 'false')?	false	3.0.2
useUnbufferedInput	Do not use BufferedInputStream for reading data from the server	true	3.0.11
yearIsDateType	Should the JDBC driver treat the MySQL type "YEAR" as a java.sql.Date, or as a SHORT?	true	3.1.9
zeroDateTimeBehavior	What should happen when the driver encounters DATETIME values that are composed entirely of zeroes (used by MySQL to represent invalid dates)? Valid values are "exception", "round"	exception	3.1.4

	and "convertToNull".		

Connector/J also supports access to MySQL via named pipes on Windows NT/2000/XP using the NamedPipeSocketFactory as a plugin-socket factory via the socketFactory property. If you do not use a namedPipePath property, the default of '\\.\pipe\MySQL' will be used. If you use the `NamedPipeSocketFactory`, the host name and port number values in the JDBC url will be ignored. You can enable this feature using:

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

Named pipes only work when connecting to a MySQL server on the same physical machine as the one the JDBC driver is being used on. In simple performance tests, it appears that 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.

You can create your own socket factories by following the example code in `com.mysql.jdbc.NamedPipeSocketFactory`, or `com.mysql.jdbc.StandardSocketFactory`.

4.4.2. JDBC API Implementation Notes

MySQL Connector/J passes all of the tests in the publicly-available version of Sun's JDBC compliance test suite. However, in many places the JDBC specification is vague about how certain functionality should be implemented, or the specification allows leeway in implementation.

This section gives details on a interface-by-interface level about how certain implementation decisions may affect how you use 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.

  For this to work, 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
  ```

  For this to work, you must also follow these rules:

  - The `SELECT` must also 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, you should use the corresponding `PreparedStatement.setBlob()` or `ResultSet.updateBlob()` (in the case of updatable result sets) methods to save changes back to the database.

  MySQL Enterprise
  MySQL Enterprise subscribers will find more information about type conversion in the Knowledge Base article, Type Conversions Supported by MySQL Connector/J. To subscribe to MySQL Enterprise see http://www.mysql.com/products/enterprise/advisors.html.

- **CallableStatement**

  Starting with Connector/J 3.1.1, stored procedures are supported when connecting to MySQL version 5.0 or newer via 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, you should use the `PreparedStatement.setClob()` method to save changes back to the database. The JDBC API does not have a `ResultSet.updateClob()` method.

- **Connection**

  Unlike older versions of MM.MySQL the `isClosed()` method does not ping the server to determine if it is alive. 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, you should 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. However, the driver uses `SHOW CREATE TABLE` to retrieve this information, so when other storage engines support foreign keys, the driver will 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 via `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()`, or `setClob()`. If you want to re-execute the statement with any large parameter changed to a non-large parameter, it is necessary to 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 non-large, 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, via the `setBinaryStream()`, `setAsciiStream()`, `setUnicodeStream()`, `setBlob()` or `setClob()` methods.

  Consequently, if you want to change the type of a parameter to a non-large 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 can not 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, you need to 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 will have to 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, you should 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.

  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 via `setLocalInfileInputStream()`.

4.4.3. 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 numerical type can be converted to any of the Java numerical types, although round-off, overflow, or loss of precision may occur.

Starting with Connector/J 3.1.0, the JDBC driver will issue warnings or throw 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:

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 `java.sql.Types` classes see Java 2 Platform Types.

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

MySQL Type Name	Return value of `GetColumnClassName`	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 `tinyInt1isBit` is set to `true` (the default) and the storage size is 1, or `java.lang.Integer` 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 if UNSIGNED or not)
MEDIUMINT[(M)] [UNSIGNED]	MEDIUMINT [UNSIGNED]	`java.lang.Integer,` if UNSIGNED `java.lang.Long` (C/J 3.1 and earlier), or `java.lang.Integer` for C/J 5.0 and later
INT,INTEGER[(M)] [UNSIGNED]	INTEGER [UNSIGNED]	`java.lang.Integer`, if UNSIGNED `java.lang.Long`
BIGINT[(M)] [UNSIGNED]	BIGINT [UNSIGNED]	`java.lang.Long`, if UNSIGNED `java.math.BigInteger`
FLOAT[(M,D)]	FLOAT	`java.lang.Float`
DOUBLE[(M,B)]	DOUBLE	`java.lang.Double`
DECIMAL[(M[,D])]	DECIMAL	`java.math.BigDecimal`
DATE	DATE	`java.sql.Date`
DATETIME	DATETIME	`java.sql.Timestamp`
TIMESTAMP[(M)]	TIMESTAMP	`java.sql.Timestamp`
TIME	TIME	`java.sql.Time`
YEAR[(2|4)]	YEAR	If `yearIsDateType` configuration property is set to false, then the returned object type is `java.sql.Short`. If set to true (the default) then an object of type `java.sql.Date` (with the date set to January 1st, at midnight).
CHAR(M)	CHAR	`java.lang.String` (unless the character set for the column is BINARY, then `byte[]` is returned.
VARCHAR(M) [BINARY]	VARCHAR	`java.lang.String` (unless the character set for the column is BINARY, then `byte[]` is returned.
BINARY(M)	BINARY	`byte[]`
VARBINARY(M)	VARBINARY	`byte[]`
TINYBLOB	TINYBLOB	`byte[]`
TINYTEXT	VARCHAR	`java.lang.String`
BLOB	BLOB	`byte[]`
TEXT	VARCHAR	`java.lang.String`
MEDIUMBLOB	MEDIUMBLOB	`byte[]`
MEDIUMTEXT	VARCHAR	`java.lang.String`
LONGBLOB	LONGBLOB	`byte[]`
LONGTEXT	VARCHAR	`java.lang.String`
ENUM('value1','value2',...)	CHAR	`java.lang.String`
SET('value1','value2',...)	CHAR	`java.lang.String`

4.4.4. 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 via `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()`.

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.

Starting with MySQL Server 4.1, 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 `ResultSets`.

The character encoding between client and server is automatically detected upon connection. The encoding used by the driver is specified on the server via the `character_set` system variable for server versions older than 4.1.0 and `character_set_server` for server versions 4.1.0 and newer. For more information, see Server Character Set and Collation.

To override the automatically-detected encoding on the client side, use the *`characterEncoding`* property in the URL used to connect to the server.

When specifying character encodings on the client side, Java-style names should be used. The following table lists Java-style names for MySQL character sets:

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	ISO8859_1
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.

To allow multiple character sets to be sent from the client, the UTF-8 encoding should be used, either by configuring `utf8` as the default server character set, or by configuring the JDBC driver to use UTF-8 through the *`characterEncoding`* property.

4.4.5. Connecting Securely Using SSL

SSL in MySQL Connector/J encrypts all data (other than the initial handshake) between the JDBC driver and the server. The performance penalty for enabling SSL is an increase in query processing time between 35% and 50%, depending on the size of the query, and the amount of data it returns.

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://developer.java.sun.com/developer/bugParade/bugs/4273544.html
- A MySQL server that supports SSL and has been compiled and configured to do so, which is MySQL-4.0.4 or later, see Using SSL for Secure Connections, for more information.
- 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` should be located in the `bin` subdirectory of your JDK or JRE):

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

You will need to enter the password when prompted for the keystore file. Interaction with `keytool` will look like this:

```
Enter keystore password:  *********
Owner: EMAILADDRESS=walrus@example.com, CN=Walrus,
       O=MySQL AB, L=Orenburg, ST=Some-State, C=RU
Issuer: EMAILADDRESS=walrus@example.com, CN=Walrus,
       O=MySQL AB, 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, you can either import the client certificate that matches the CA certificate you just imported, or you can create a new client certificate.

To import an existing certificate, the certificate should 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
```

You now need to 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.

You should 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]:  MySQL AB
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=MySQL AB,
 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  :.O..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.

4.4.6. Using Master/Slave Replication with ReplicationConnection

Starting with Connector/J 3.1.7, we've made available 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)`, this replication-aware connection will use one of the slave connections, which are load-balanced per-vm using a round-robin scheme (a given connection is sticky to a slave unless that slave is removed from service). 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, simple 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://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");
     .......
  }
}
```

You may also want to investigate the Load Balancing JDBC Pool (lbpol) tool, which provides a wrapper around the standard JDBC driver and allows you to use DB connection pools that includes checks for system failures and uneven load distribution. For more information, see Load Balancing JDBC Pool (lbpool).

4.4.7. Mapping MySQL Error Numbers to SQLStates

The table below provides a mapping of the MySQL Error Numbers to `SQL States`

Table 4.1. Mapping of MySQL Error Numbers to SQLStates

MySQL Error Number	**MySQL Error Name**	**Legacy (X/Open) SQLState**	**SQL Standard SQLState**
102	ER_DUP	S10	230

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
2	_KEY	00	00
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	S1000	42000
1045	ER_ACCESS_DENIED_ERROR	28000	28000
1047	ER_UNKNOWN_COM_ERROR	08S01	HY000
1050	ER_TABLE_EXISTS_ERROR	S1000	42S01
1051	ER_BAD_TABLE_ERROR	42S02	42S02
1052	ER_NON_UNIQ_ERROR	S1000	23000
1053	ER_SERVER_SHUTDOWN	S1000	08S01
1054	ER_BAD_FIELD_ERROR	S0022	42S22
1055	ER_WRONG_FIELD_WITH_GROUP	S1009	42000
1056	ER_WRONG_G	S1009	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
	ROUP_FIELD		
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_INVAL-ID_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_DO	S1009	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
	ES_NOT_EXITS		
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	S1000	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	S1000	42000
1091	ER_CANT_DROP_FIELD_OR_KEY	S1000	42000
1101	ER_BLOB_CANT_HAVE_DEFAULT	S1000	42000
1102	ER_WRONG_DB_NAME	S1000	42000
1103	ER_WRONG_TABLE_NAME	S1000	42000
110	ER_TOO	S10	420

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
4	_BIG_SELECT	00	00
1106	ER_UNKNOWN_PROCEDURE	S1000	42000
1107	ER_WRONG_PARAMCOUNT_TO_PROCEDURE	S1000	42000
1109	ER_UNKNOWN_TABLE	S1000	42S02
1110	ER_FIELD_SPECIFIED_TWICE	S1000	42000
1112	ER_UNSUPPORTED_EXTENSION	S1000	42000
1113	ER_TABLE_MUST_HAVE_COLUMNS	S1000	42000
1115	ER_UNKNOWN_CHARACTER_SET	S1000	42000
1118	ER_TOO_BIG_ROWSIZE	S1000	42000
1120	ER_WRONG_OUTER_JOIN	S1000	42000
1121	ER_NULL_COLUMN_IN_INDEX	S1000	42000
1129	ER_HOST_IS_BLOCKED	08004	HY000
1130	ER_HOST_NOT_PRIVILEGED	08004	HY000
113	ER_PAS	S10	420

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1	SWORD_ANONYMOUS_USER	00	00
1132	ER_PASSWORD_NOT_ALLOWED	S1000	42000
1133	ER_PASSWORD_NO_MATCH	S1000	42000
1136	ER_WRONG_VALUE_COUNT_ON_ROW	S1000	21S01
1138	ER_INVALID_USE_OF_NULL	S1000	42000
1139	ER_REGEXP_ERROR	S1000	42000
1140	ER_MIX_OF_GROUP_FUNC_AND_FIELDS	S1000	42000
1141	ER_NONEXISTING_GRANT	S1000	42000
1142	ER_TABLEACCESS_DENIED_ERROR	S1000	42000
1143	ER_COLUMNACCESS_DENIED_ERROR	S1000	42000
1144	ER_ILLEGAL_GRANT_FOR_TABLE	S1000	42000
1145	ER_GRANT_WRONG_H	S1000	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
	OST_OR_USER		
1146	ER_NO_SUCH_TABLE	S1000	42S02
1147	ER_NONEXISTING_TABLE_GRANT	S1000	42000
1148	ER_NOT_ALLOWED_COMMAND	S1000	42000
1149	ER_SYNTAX_ERROR	S1000	42000
1152	ER_ABORTING_CONNECTION	S1000	08S01
1153	ER_NET_PACKET_TOO_LARGE	S1000	08S01
1154	ER_NET_READ_ERROR_FROM_PIPE	S1000	08S01
1155	ER_NET_FCNTL_ERROR	S1000	08S01
1156	ER_NET_PACKETS_OUT_OF_ORDER	S1000	08S01
1157	ER_NET_UNCOMPRESS_ERROR	S1000	08S01
1158	ER_NET_READ_ERROR	S1000	08S01
1159	ER_NET_READ_INTERRUPTED	S1000	08S01
1160	ER_NET_ERROR_ON_WRITE	S1000	08S01

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1161	ER_NET_WRITE_INTERRUPTED	S1000	08S01
1162	ER_TOO_LONG_STRING	S1000	42000
1163	ER_TABLE_CANT_HANDLE_BLOB	S1000	42000
1164	ER_TABLE_CANT_HANDLE_AUTO_INCREMENT	S1000	42000
1166	ER_WRONG_COLUMN_NAME	S1000	42000
1167	ER_WRONG_KEY_COLUMN	S1000	42000
1169	ER_DUP_UNIQUE	S1000	23000
1170	ER_BLOB_KEY_WITHOUT_LENGTH	S1000	42000
1171	ER_PRIMARY_CANT_HAVE_NULL	S1000	42000
1172	ER_TOO_MANY_ROWS	S1000	42000
1173	ER_REQUIRES_PRIMARY_KEY	S1000	42000
1177	ER_CHECK_NO_SUCH_TABLE	S1000	42000
1178	ER_CHECK_NOT_IMPLEMENTED	S1000	42000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1179	ER_CANT_DO_THIS_DURING_AN_TRANSACTION	S1000	25000
1184	ER_NEW_ABORTING_CONNECTION	S1000	08S01
1189	ER_MASTER_NET_READ	S1000	08S01
1190	ER_MASTER_NET_WRITE	S1000	08S01
1203	ER_TOO_MANY_USER_CONNECTIONS	S1000	42000
1205	ER_LOCK_WAIT_TIMEOUT	41000	41000
1207	ER_READ_ONLY_TRANSACTION	S1000	25000
1211	ER_NO_PERMISSION_TO_CREATE_USER	S1000	42000
1213	ER_LOCK_DEADLOCK	41000	40001
1216	ER_NO_REFERENCED_ROW	S1000	23000
1217	ER_ROW_IS_REFERENCED	S1000	23000
1218	ER_CONNECT_TO_MASTER	S1000	08S01

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1222	ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT	S1000	21000
1226	ER_USER_LIMIT_REACHED	S1000	42000
1230	ER_NO_DEFAULT	S1000	42000
1231	ER_WRONG_VALUE_FOR_VAR	S1000	42000
1232	ER_WRONG_TYPE_FOR_VAR	S1000	42000
1234	ER_CANT_USE_OPTION_HERE	S1000	42000
1235	ER_NOT_SUPPORTED_YET	S1000	42000
1239	ER_WRONG_FK_DEF	S1000	42000
1241	ER_OPERAND_COLUMNS	S1000	21000
1242	ER_SUBQUERY_NO_1_ROW	S1000	21000
1247	ER_ILLEGAL_REFERENCE	S1000	42S22
1248	ER_DERIVED_MUST_HAVE_ALIAS	S1000	42000
1249	ER_SELECT_REDUCED	S1000	01000

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1250	ER_TABLENAME_NOT_ALLOWED_HERE	S1000	42000
1251	ER_NOT_SUPPORTED_AUTH_MODE	S1000	08004
1252	ER_SPATIAL_CANT_HAVE_NULL	S1000	42000
1253	ER_COLLATION_CHARSET_MISMATCH	S1000	42000
1261	ER_WARN_TOO_FEW_RECORDS	S1000	01000
1262	ER_WARN_TOO_MANY_RECORDS	S1000	01000
1263	ER_WARN_NULL_TO_NOTNULL	S1000	01000
1264	ER_WARN_DATA_OUT_OF_RANGE	S1000	01000
1265	ER_WARN_DATA_TRUNCATED	S1000	01000
1280	ER_WRONG_NAME_FOR_INDEX	S1000	42000
1281	ER_WRONG_NAME_FOR_CATALOG	S1000	42000
128	ER_UNK	S10	420

MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
6	NOWN_STORAGE_ENGINE	00	00

4.5. Connector/J Notes and Tips

4.5.1. Basic JDBC Concepts

This section provides some general JDBC background.

4.5.1.1. Connecting to MySQL Using the `DriverManager` Interface

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

The `DriverManager` needs to be told which JDBC drivers it should 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:

```
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 4.1. Connector/J: Obtaining a connection from the `DriverManager`

This example shows how you can obtain a `Connection` instance from the `DriverManager`. There are a few different signatures for the `getConnection()` method. You should see 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=monty&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.

4.5.1.2. Using Statements 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 via 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 affected by the update statement.

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 4.2. 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;
    }
}
```

4.5.1.3. Using `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 http://dev.mysql.com/doc/mysql/en/stored-routines.html.

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 via `inputParam` as a `ResultSet`:

Example 4.3. 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 4.4. 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, you should try to 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 4.5. 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 4.6. 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 4.7. 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
...
```

4.5.1.4. Retrieving `AUTO_INCREMENT` Column Values

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 4.8. 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(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))");
    //
    // 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
    }
    rs.close();
    rs = null;
    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 4.9. 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
    }
    rs.close();
    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 4.10. 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");
    rs.close();
    rs = null;
    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
        }
    }
}
```

When you run the preceding example code, you should get the following output: Key returned from `getGeneratedKeys()`: 1 Key returned from `SELECT LAST_INSERT_ID()`: 1 Key returned for inserted row: 2 You should be aware, that 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 will be 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.

4.5.2. Using Connector/J with J2EE and Other Java Frameworks

This section describes how to use Connector/J in several contexts.

4.5.2.1. General J2EE Concepts

This section provides general background on J2EE concepts that pertain to use of Connector/J.

4.5.2.1.1. Understanding Connection Pooling

Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them.

This technique of pooling connections is based on the fact that most applications only need a thread to have access to a JDBC connection when they are actively processing a transaction, which usually take only milliseconds to complete. When not processing a transaction, the connection would otherwise sit idle. Instead, connection pooling allows 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 may be used by any other threads that want to use it.

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, however with connection pooling, your thread may end up using either a new, or already-existing connection.

Connection pooling can greatly increase the performance of your Java application, while reducing overall resource usage. 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 straight-forward JDBC programming techniques.

- Controlled resource usage

  If you do not use connection pooling, and instead create a new connection every time a thread needs one, your application's resource usage can be quite wasteful and lead to unpredictable behavior under load.

Remember that 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.

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

Generally, you configure a connection pool in your application server configuration files, and access it via 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 4.11. 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 allows 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 should look familiar to anyone who has done JDBC programming in the past.

The most important thing to remember when using connection pooling is to make sure that no matter what happens in your code (exceptions, flow-of-control, and so forth), connections, and anything created by them (such as statements or result sets) are closed, so that they may be re-used, otherwise they will be stranded, which in the best case means that the MySQL server resources they represent (such as buffers, locks, or sockets) may be tied up for some time, or worst case, may be tied up forever.

What Is the Best Size for my Connection Pool?

As with all other configuration rules-of-thumb, the answer is: it depends. Although the optimal size depends on anticipated load and average database transaction time, the optimum connection pool size is smaller than you might expect. If you take Sun'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 response times that are acceptable.

To correctly size a connection pool for your application, you should 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.

4.5.2.2. 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"/>
  <!-- The name you used above, must match _exactly_ here!
       The connection pool will be bound into JNDI with the name
       "java:/comp/env/jdbc/MySQLDB"
  -->
  <ResourceParams name="jdbc/MySQLDB">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>
    <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->
    <parameter>
      <name>maxActive</name>
      <value>10</value>
    </parameter>
    <!-- You don't want to many idle connections hanging around
         if you can avoid it, only enough to soak up a spike in
         the load -->
    <parameter>
      <name>maxIdle</name>
      <value>5</value>
    </parameter>
    <!-- Don't use autoReconnect=true, it's going away eventually
         and it's a crutch for older connection pools that couldn't
         test connections. You need to decide whether your application
         is supposed to deal with SQLExceptions (hint, it should), and
         how much of a performance penalty you're willing to pay
         to ensure 'freshness' of the connection -->
    <parameter>
      <name>validationQuery</name>
      <value>SELECT 1</value>
    </parameter>
   <!-- The most conservative approach is to test connections
        before they're given to your application. For most applications
        this is okay, the query used above is very small and takes
        no real server resources to process, other than the time used
        to traverse the network.
        If you have a high-load application you'll need to rely on
        something else. -->
    <parameter>
      <name>testOnBorrow</name>
      <value>true</value>
    </parameter>
   <!-- Otherwise, or in addition to testOnBorrow, you can test
        while connections are sitting idle -->
    <parameter>
      <name>testWhileIdle</name>
      <value>true</value>
    </parameter>
    <!-- You have to set this value, otherwise even though
         you've asked connections to be tested while idle,
         the idle evicter thread will never run -->
    <parameter>
      <name>timeBetweenEvictionRunsMillis</name>
      <value>10000</value>
    </parameter>
    <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes or even fraction of a minute
         is sometimes okay here, it depends on your application
         and how much spikey load it will see -->
    <parameter>
      <name>minEvictableIdleTimeMillis</name>
      <value>60000</value>
    </parameter>
    <!-- Username and password used when connecting to MySQL -->
    <parameter>
     <name>username</name>
     <value>someuser</value>
    </parameter>
    <parameter>
     <name>password</name>
     <value>somepass</value>
    </parameter>
    <!-- Class name for the Connector/J driver -->
    <parameter>
       <name>driverClassName</name>
       <value>com.mysql.jdbc.Driver</value>
    </parameter>
    <!-- The JDBC connection url for connecting to MySQL, notice
         that if you want to pass any other MySQL-specific parameters
         you should pass them here in the URL, setting them using the
         parameter tags above will have no effect, you will also
         need to use &amp; to separate parameter values as the
         ampersand is a reserved character in XML -->
    <parameter>
      <name>url</name>
      <value>jdbc:mysql://localhost:3306/test</value>
    </parameter>
  </ResourceParams>
</Context>
```

In general, you should 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 unfortunately 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
```

4.5.2.3. 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>
        <!-- This connection pool will be bound into JNDI with the name
             "java:/MySQLDB" -->
        <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>
        <!-- Don't set this any higher than max_connections on your
         MySQL server, usually this should be a 10 or a few 10's
         of connections, not hundreds or thousands -->
        <max-pool-size>20</max-pool-size>
        <!-- Don't allow connections to hang out idle too long,
         never longer than what wait_timeout is set to on the
         server...A few minutes is usually okay here,
         it depends on your application
         and how much spikey load it will see -->
        <idle-timeout-minutes>5</idle-timeout-minutes>
        <!-- If you're using Connector/J 3.1.8 or newer, you can use
             our implementation of these to increase the robustness
             of the connection pool. -->
        <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>
```

4.5.2.4. 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.

4.5.2.4.1. 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 towards 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 allows 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`.

4.5.2.4.2. 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 allows declarative transaction management (configuration instead of code).

In order 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.

4.5.2.4.3. Connection Pooling

In many sitations, 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"/>
```

4.5.2.5. Using Connector/J with GlassFish

4.5.3. Common Problems and Solutions

There are a few issues that seem to be commonly encountered often by users of MySQL Connector/J. This section deals with their symptoms, and their resolutions.

Questions

- 4.5.3.1: 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.

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

- 4.5.3.3: 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
  ```

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

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

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

- 4.5.3.7: I am trying to connect to my MySQL server within my application, but I get the following error and stack trace:

  ```
  java.net.SocketException
  MESSAGE: Software caused connection abort: recv failed
  STACKTRACE:
  java.net.SocketException: Software caused connection abort: recv failed
  ```

```
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(Unknown Source)
at com.mysql.jdbc.MysqlIO.readFully(MysqlIO.java:1392)
at com.mysql.jdbc.MysqlIO.readPacket(MysqlIO.java:1414)
at com.mysql.jdbc.MysqlIO.doHandshake(MysqlIO.java:625)
at com.mysql.jdbc.Connection.createNewIO(Connection.java:1926)
at com.mysql.jdbc.Connection.<init>(Connection.java:452)
at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:411)
```

- 4.5.3.8: 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 a error and stack trace, but these only occur after a fixed period of heavy activity.
- 4.5.3.9: When using `gcj` an `java.io.CharConversionException` is raised when working with certain character sequences.
- 4.5.3.10: 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.
- 4.5.3.11: You get an `ER_NET_PACKET_TOO_LARGE` exception, even though the binary blob size you want to insert via JDBC is safely below the `max_allowed_packet` size.

Questions and Answers

4.5.3.1: 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 should be allowed.

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.

4.5.3.2: 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 4.2, “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.

4.5.3.3: 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).

4.5.3.4: 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 4.4.1, “Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J”).

Also, you should be catching SQLExceptions in your application and dealing 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 APIDOCS) to "08S01" when it encounters network-connectivity issues during the processing of a query. Your application code should then attempt to re-connect to MySQL at this point.

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

Example 4.12. 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 transacational 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--;
            } else {
                retryCount = 0;
            }
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException sqlEx) {
                    // You'd probably want to log this . . .
                }
            }
            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, you should 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.

4.5.3.5: 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 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.

4.5.3.6: I cannot connect to the MySQL server using Connector/J, and I'm sure the connection paramters 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`.

4.5.3.7: I am trying to connect to my MySQL server within my application, but I get the following error and stack trace:

```
java.net.SocketException
MESSAGE: Software caused connection abort: recv failed
STACKTRACE:
java.net.SocketException: Software caused connection abort: recv failed
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(Unknown Source)
at com.mysql.jdbc.MysqlIO.readFully(MysqlIO.java:1392)
at com.mysql.jdbc.MysqlIO.readPacket(MysqlIO.java:1414)
at com.mysql.jdbc.MysqlIO.doHandshake(MysqlIO.java:625)
at com.mysql.jdbc.Connection.createNewIO(Connection.java:1926)
at com.mysql.jdbc.Connection.<init>(Connection.java:452)
at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:411)
```

The error probably indicates that you are using a older version of the Connector/J JDBC driver (2.0.14 or 3.0.x) and you are trying to connect to a MySQL server with version 4.1x or newer. The older drivers are not compatible with 4.1 or newer of MySQL as they do not support the newer authentication mechanisms.

It is likely that the older version of the Connector/J driver exists within your application directory or your `CLASSPATH` includes the older Connector/J package.

4.5.3.8: 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 a 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 TransactionTimeoute within the JBoss wiki for more information.

4.5.3.9: When using `gcj` an `java.io.CharConversionException` 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. You should add `useJvmCharsetConverters=true` to your connection string to force character conversion outside of the `gcj` libraries, or try a different JDK.

4.5.3.10: 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`.

4.5.3.11: You get an `ER_NET_PACKET_TOO_LARGE` exception, even though the binary blob size you want to insert via 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.

4.6. Connector/J Support

4.6.1. Connector/J Community Support

Sun Microsystems, Inc. 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.

4.6.2. 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 have found a sensitive security bug in MySQL, you can send email to `<security@mysql.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, you should create a repeatable, stanalone 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, you should 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/.

4.6.3. Connector/J Change History

The Connector/J Change History (Changelog) is located with the main Changelog for MySQL. See Appendix D, *MySQL Connector/J Change History*.

Chapter 5. MySQL Connector/MXJ

MySQL Connector/MXJ is a Java Utility package for deploying and managing a MySQL database. Deploying and using MySQL can be as easy as adding an additional parameter to the JDBC connection url, which will result in the database being started when the first connection is made. This makes it easy for Java developers to deploy applications which require a database by reducing installation barriers for their end-users.

MySQL Connector/MXJ makes the MySQL database appear to be a java-based component. It does this by determining what platform the system is running on, selecting the appropriate binary, and launching the executable. It will also optionally deploy an initial database, with any specified parameters.

Included are instructions for use with a JDBC driver and deploying as a JMX MBean to JBoss.

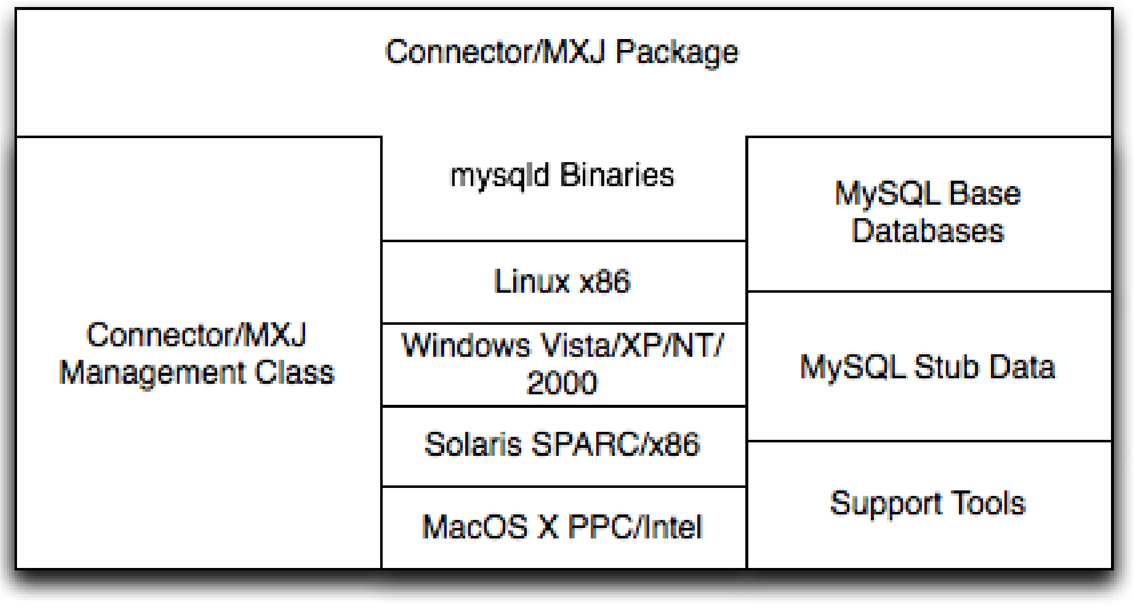
You can download sources and binaries from: http://dev.mysql.com/downloads/connector/mxj/

This a beta release and feedback is welcome and encouraged.

Please send questions or comments to the MySQL and Java mailing list.

5.1. Connector/MXJ Overview

Connector/MXJ consists of a Java class, a copy of the `mysqld` binary for a specific list of platforms, and associated files and support utilities. The Java class controls the initialization of an instance of the embedded `mysqld` binary, and the ongoing management of the `mysqld` process. The entire sequence and management can be controlled entirely from within Java using the Connector/MXJ Java classes. You can see an overview of the contents of the Connector/MXJ package in the figure below.



It is important to note that Connector/MXJ is not an embedded version of MySQL, or a version of MySQL written as part of a Java class. Connector/MXJ works through the use of an embedded, compiled binary of `mysqld` as would normally be used when deploying a standard MySQL installation.

It is the Connector/MXJ wrapper, support classes and tools, that enable Connector/MXJ to appear as a MySQL instance.

When Connector/MXJ is initialized, the corresponding `mysqld` binary for the current platform is extracted, along with a preconfigured data directed. Both are contained within the Connector/MXJ JAR file. The `mysqld` instance is then started, with any additional options as specified during the initialization, and the MySQL database becomes accessible.

Because Connector/MXJ works in combination with Connector/J, you can access and integrate with the MySQL instance through a JDBC connection. When you have finished with the server, the instance is terminated, and, by default, any data created during the session is retained within the temporary directory created when the instance was started.

Connector/MXJ and the embedded `mysqld` instance can be deployed in a number of environments where relying on an existing database, or installing a MySQL instance would be impossible, including CD-ROM embedded database applications and temporary database requirements within a Java-based application environment.

5.2. Connector/MXJ Versions

- Connector/MXJ 5.x, currently in beta status, includes `mysqld` version 5.x and includes binaries for Linux x86, Mac OS X PPC, Windows XP/NT/2000 x86 and Solaris SPARC. Connector/MXJ 5.x requires the Connector/J 5.x package.

  The exact version of `mysqld` included depends on the version of Connector/MXJ

  1. Connector/MXJ v5.0.3 included MySQL v5.0.22
  2. Connector/MXJ v5.0.4 includes MySQL v5.0.27 (Community) or MySQL v5.0.32 (Enterprise)
  3. Connector/MXJ v5.0.6 includes MySQL 5.0.37 (Community)
  4. Connector/MXJ v5.0.7 includes MySQL 5.0.41 (Community) or MySQL 5.0.42 (Enterprise)
  5. Connector/MXJ v5.0.8 includes MySQL 5.0.45 (Community) or MySQL 5.0.46 (Enterprise)
  6. Connector/MXJ v5.0.9 includes MySQL 5.0.51a (Community) or MySQL 5.0.54 (Enterprise)

- Connector/MXJ 1.x includes `mysqld` version 4.1.13 and includes binaries for Linux x86, Windows XP/NT/2000 x86 and Solaris SPARC. Connector/MXJ 1.x requires the Connector/J 3.x package.

A summary of the different MySQL versions supplied with each Connector/MXJ release are shown in the table.

Connector/MXJ Version	**MySQL Version(s)**
5.0.8	5.0.45 (CS), 5.0.46 (ES)
5.0.7	5.0.41 (CS), 5.0.42 (ES)
5.0.6	5.0.37 (CS), 5.0.40 (ES)
5.0.5	5.0.37 (CS), 5.0.36 (ES)
5.0.4	5.0.27 (CS), 5.0.32 (ES)
5.0.3	5.0.22
5.0.2	5.0.19

This guide provides information on the Connector/MXJ 5.x release. For information on using the older releases, please see the documentation included with the appropriate distribution.

5.3. Connector/MXJ Installation

Connector/MXJ does not have a installation application or process, but there are some steps you can follow to make the installation and deployment of Connector/MXJ easier.

Before you start, there are some baseline requirements for

- Java Runtime Environment (v1.4.0 or newer) if you are only going to deploy the package.
- Java Development Kit (v1.4.0 or newer) if you want to build Connector/MXJ from source.
- Connector/J 5.0 or newer.

Depending on your target installation/deployment environment you may also require:

- JBoss - 4.0rc1 or newer
- Apache Tomcat - 5.0 or newer
- Sun's JMX reference implementation version 1.2.1 (from http://java.sun.com/products/JavaManagement/)

5.3.1. Supported Platforms

Connector/MXJ is compatible with any platform supporting Java and MySQL. By default, Connector/MXJ incorporates the `mysqld` binary for a select number of platforms which differs by version. The following platforms have been tested and working as deployment platforms. Support for all the platforms listed below is not included by default.

- Linux (i386)
- FreeBSD (i386)
- Windows NT (x86), Windows 2000 (x86), Windows XP (x86), Windows Vista (x86)
- Solaris 8, SPARC 32-bit (compatible with Solaris 8, Solaris 9 and Solaris 10 on SPARC 32-bit and 64-bit platforms)
- Mac OS X (PowerPC and Intel)

The Connector/MXJ 5.0.8 release includes `mysqld` binaries for the following platforms by as standard:

- Linux (i386)
- Windows (x86), compatible with Windows NT, Windows 2000, Windows XP , Windows Vista
- Solaris 8, SPARC 32-bit (compatible with Solaris 8, Solaris 9 and Solaris 10 on SPARC 32-bit and 64-bit platforms)
- Mac OS X (PowerPC and Intel)

For more information on packaging your own Connector/MXJ with the platforms you require, see Section 5.6.1, “Creating your own Connector/MXJ Package”

5.3.2. Connector/MXJ Base Installation

Because there is no formal installation process, the method, installation directory, and access methods you use for Connector/MXJ are entirely up to your individual requirements.

To perform a basic installation, choose a target directory for the files included in the Connector/MXJ package. On Unix/Linux systems you may opt to use a directory such as `/usr/local/connector-mxj`; On Windows, you may want to install the files in the base directory, `C:\Connector-MXJ`, or within the `Program Files` directory.

To install the files, for a Connector/MXJ 5.0.4 installation:

1. Download the Connector/MXJ package, either in Tar/Gzip format (ideal for Unix/Linux systems) or Zip format (Windows).
2. Extract the files from the package. This will create a directory `mysql-connector-mxj-gpl-[ver]`. Copy and optionally rename this directory to your desired location.
3. For best results, you should update your global `CLASSPATH` variable with the location of the required `jar` files.

   Within Unix/Linux you can do this globally by editing the global shell profile, or on a user by user basis by editing their individual shell profile.

   On Windows 2000, Windows NT and Windows XP, you can edit the global `CLASSPATH` by editing the `Environment Variables` configured through the `System` control panel.

For Connector/MXJ 5.0.6 and later you need the following JAR files in your `CLASSPATH`:

1. `mysql-connector-mxj-gpl-[ver].jar` — contains the main Connector/MXJ classes.
2. `mysql-connector-mxj-gpl-[ver]-db-files.jar` — contains the embedded `mysqld` and database files.
3. `aspectjrt.jar` — the AspectJ runtime library, located in `lib/aspectjrt.jar` in the Connector/MXJ package.
4. `mysql-connector-java-[ver]-bin.jar` — Connector/J, see Chapter 4, *MySQL Connector/J*.

For Connector/MXJ 5.0.4 and later you need the following JAR files in your `CLASSPATH`:

1. `connector-mxj.jar` — contains the main Connector/MXJ classes.

2. `connector-mxj-db-files.jar` — contains the embedded `mysqld` and database files.
3. `aspectjrt.jar` — the AspectJ runtime library, located in `lib/aspectjrt.jar` in the Connector/MXJ package.
4. `mysql-connector-mxj-gpl-[ver].jar` — Connector/J, see Chapter 4, *MySQL Connector/J*.

For Connector/MXJ 5.0.3 and earlier, you need the following JAR files:

1. `connector-mxj.jar`
2. `aspectjrt.jar` — the AspectJ runtime library, located in `lib/aspectjrt.jar` in the Connector/MXJ package.
3. `mysql-connector-mxj-gpl-[ver].jar` — Connector/J, see Chapter 4, *MySQL Connector/J*.

5.3.3. Connector/MXJ Quick Start Guide

Once you have extracted the Connector/MXJ and Connector/J components you can run one of the sample applications that initiates a MySQL instance. You can test the installation by running the `ConnectorMXJUrlTestExample`:

```
shell> java ConnectorMXJUrlTestExample
jdbc:mysql:mxj://localhost:3336/our_test_app?server.basedir»
    =/var/tmp/test-mxj&createDatabaseIfNotExist=true&server.initialize-user=true
[/var/tmp/test-mxj/bin/mysqld][--no-defaults][--port=3336][--socket=mysql.sock]»
    [--basedir=/var/tmp/test-mxj][--datadir=/var/tmp/test-mxj/data]»
    [--pid-file=/var/tmp/test-mxj/data/MysqldResource.pid]
[MysqldResource] launching mysqld (driver_launched_mysqld_1)
InnoDB: The first specified data file ./ibdata1 did not exist:
InnoDB: a new database to be created!
080220  9:40:20  InnoDB: Setting file ./ibdata1 size to 10 MB
InnoDB: Database physically writes the file full: wait...
080220  9:40:20  InnoDB: Log file ./ib_logfile0 did not exist: new to be created
InnoDB: Setting log file ./ib_logfile0 size to 5 MB
InnoDB: Database physically writes the file full: wait...
080220  9:40:20  InnoDB: Log file ./ib_logfile1 did not exist: new to be created
InnoDB: Setting log file ./ib_logfile1 size to 5 MB
InnoDB: Database physically writes the file full: wait...
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
080220  9:40:21  InnoDB: Started; log sequence number 0 0
080220  9:40:21 [Note] /var/tmp/test-mxj/bin/mysqld: ready for connections.
Version: '5.0.51a'  socket: 'mysql.sock'  port: 3336  MySQL Community Server (GPL)
[MysqldResource] mysqld running as process: 2238
------------------------
SELECT VERSION()
------------------------
5.0.51a
------------------------
[MysqldResource] stopping mysqld (process: 2238)
080220  9:40:27 [Note] /var/tmp/test-mxj/bin/mysqld: Normal shutdown
080220  9:40:27  InnoDB: Starting shutdown...
080220  9:40:29  InnoDB: Shutdown completed; log sequence number 0 43655
080220  9:40:29 [Note] /var/tmp/test-mxj/bin/mysqld: Shutdown complete
[MysqldResource] shutdown complete
```

The above output shows an instance of MySQL starting, the necessary files being created (log files, InnoDB data files) and the MySQL database entering the running state. The instance is then shutdown by Connector/MXJ before the example terminates.

> **Warning**
>
> You should avoid running your Connector/MXJ application as the `root` user, because this will cause the `mysqld` to also be executed with root privileges. For more information, see How to Run MySQL as a Normal User.

5.3.4. Deploying Connector/MXJ using Driver Launch

Connector/MXJ and Connector/J work together to enable you to launch an instance of the `mysqld` server through the use of a keyword in the JDBC connection string. Deploying Connector/MXJ within a Java application can be automated through this method, making the deployment of Connector/MXJ a simple process:

1. Download and unzip Connector/MXJ, add `mysql-connector-mxj-gpl-[ver].jar` to the `CLASSPATH`.

   If you are using Connector/MXJ v5.0.4 or later you will also need to add the `mysql-connector-mxj-gpl-[ver]-db-files.jar` file to your `CLASSPATH`.

2. To the JDBC connection string, embed the `mxj` keyword, for example: `jdbc:mysql:mxj://localhost:`*`PORT`*`/`*`DB-NAME`*.

For more details, see Section 5.4, “Connector/MXJ Configuration”.

5.3.5. Deploying Connector/MXJ within JBoss

For deployment within a JBoss environment, you must configure the JBoss environment to use the Connector/MXJ component within the JDBC parameters:

1. Download Connector/MXJ and copy the `mysql-connector-mxj-gpl-[ver].jar` file to the `$JBOSS_HOME/server/default/lib` directory.

   If you are using Connector/MXJ v5.0.4 or later you will also need to copy the `mysql-connector-mxj-gpl-[ver]-db-files.jar` file to `$JBOSS_HOME/server/default/lib`.

2. Download Connector/J and copy the `mysql-connector-java-`*`5.1.5`*`-bin.jar` file to the `$JBOSS_HOME/server/default/lib` directory.

3. Create an MBean service xml file in the `$JBOSS_HOME/server/default/deploy` directory with any attributes set, for instance the `datadir` and `autostart`.

4. Set the JDBC parameters of your web application to use:

```
String driver = "com.mysql.jdbc.Driver";
String url = "jdbc:mysql:///test?propertiesTransform="+
             "com.mysql.management.jmx.ConnectorMXJPropertiesTransform";
String user = "root";
String password = "";
Class.forName(driver);
Connection conn = DriverManager.getConnection(url, user, password);
```

You may wish to create a separate users and database table spaces for each application, rather than using "root and test".

We highly suggest having a routine backup procedure for backing up the database files in the `datadir`.

5.3.6. Verifying Installation using JUnit

The best way to ensure that your platform is supported is to run the JUnit tests. These will test the Connector/MXJ classes and the associated components.

5.3.6.1. JUnit Test Requirements

The first thing to do is make sure that the components will work on the platform. The `MysqldResource` class is really a wrapper for a native version of MySQL, so not all platforms are supported. At the time of this writing, Linux on the i386 architecture has been tested and seems to work quite well, as does OS X v10.3. There has been limited testing on Windows and Solaris.

Requirements:

1. JDK-1.4 or newer (or the JRE if you aren't going to be compiling the source or JSPs).

2. MySQL Connector/J version 5.0 or newer (from http://dev.mysql.com/downloads/connector/j/) installed and available via your CLASSPATH.

3. The `javax.management` classes for JMX version 1.2.1, these are present in the following application servers:

   - JBoss - 4.0rc1 or newer.
   - Apache Tomcat - 5.0 or newer.
   - Sun's JMX reference implementation version 1.2.1 (from http://java.sun.com/products/JavaManagement/).

4. JUnit 3.8.1 (from http://www.junit.org/).

If building from source, All of the requirements from above, plus:

1. Ant version 1.5 or newer (download from http://ant.apache.org/).

5.3.6.2. Running the JUnit Tests

1. The tests attempt to launch MySQL on the port 3336. If you have a MySQL running, it may conflict, but this isn't very likely because the default port for MySQL is 3306. However, You may set the "c-mxj_test_port" Java property to a port of your choosing. Alternatively, you may wish to start by shutting down any instances of MySQL you have running on the target machine.

   The tests suppress output to the console by default. For verbose output, you may set the "c-mxj_test_silent" Java property to "false".

2. To run the JUnit test suite, the $CLASSPATH must include the following:

   - JUnit
   - JMX
   - Connector/J
   - MySQL Connector/MXJ

3. If `connector-mxj.jar` is not present in your download, unzip MySQL Connector/MXJ source archive.

   ```
   cd mysqldjmx
   ant dist
   ```

   Then add `$TEMP/cmxj/stage/connector-mxj/connector-mxj.jar` to the CLASSPATH.

4. if you have `junit`, execute the unit tests. From the command line, type:

   ```
   java com.mysql.management.AllTestsSuite
   ```

   The output should look something like this:

   ```
   .........................................
   .........................................
   ..........
   Time: 259.438
   OK (101 tests)
   ```

   Note that the tests are a bit slow near the end, so please be patient.

5.4. Connector/MXJ Configuration

5.4.1. Running as part of the JDBC Driver

A feature of the MySQL Connector/J JDBC driver is the ability to specify a connection to an embedded Connector/MXJ instance through the use of the mxj keyword in the JDBC connection string.

In the following example, we have a program which creates a connection, executes a query, and prints the result to the System.out. The MySQL database will be deployed and started as part of the connection process, and shutdown as part of the finally block.

You can find this file in the Connector/MXJ package as `src/ConnectorMXJUrlTestExample.java`.

```
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import com.mysql.management.driverlaunched.ServerLauncherSocketFactory;
import com.mysql.management.util.QueryUtil;
public class ConnectorMXJUrlTestExample {
  public static String DRIVER = "com.mysql.jdbc.Driver";
  public static String JAVA_IO_TMPDIR = "java.io.tmpdir";
  public static void main(String[] args) throws Exception {
    File ourAppDir = new File(System.getProperty(JAVA_IO_TMPDIR));
    File databaseDir = new File(ourAppDir, "test-mxj");
    int port = Integer.parseInt(System.getProperty("c-mxj_test_port", "3336"));
    String dbName = "our_test_app";
```

```
    String url = "jdbc:mysql:mxj://localhost:" + port + "/" + dbName //
      + "?" + "server.basedir=" + databaseDir //
      + "&" + "createDatabaseIfNotExist=true"//
      + "&" + "server.initialize-user=true" //
    ;
    System.out.println(url);
    String userName = "alice";
    String password = "q93uti0opwhkd";
    Class.forName(DRIVER);
    Connection conn = null;
    try {
      conn = DriverManager.getConnection(url, userName, password);
      String sql = "SELECT VERSION()";
      String queryForString = new QueryUtil(conn).queryForString(sql);
      System.out.println("------------------------");
      System.out.println(sql);
      System.out.println("------------------------");
      System.out.println(queryForString);
      System.out.println("------------------------");
      System.out.flush();
      Thread.sleep(100); // wait for System.out to finish flush
    } finally {
      try {
        if (conn != null)
           conn.close();
      } catch (Exception e) {
        e.printStackTrace();
      }
      ServerLauncherSocketFactory.shutdown(databaseDir, null);
    }
  }
}
```

To run the above program, be sure to have connector-mxj.jar and Connector/J in the CLASSPATH. Then type:

```
java ConnectorMXJTestExample
```

5.4.2. Running within a Java Object

If you have a java application and wish to “embed” a MySQL database, make use of the `com.mysql.management.MysqldResource` class directly. This class may be instantiated with the default (no argument) constructor, or by passing in a java.io.File object representing the directory you wish the server to be "unzipped" into. It may also be instantiated with printstreams for "stdout" and "stderr" for logging.

Once instantiated, a `java.util.Map`, the object will be able to provide a `java.util.Map` of server options appropriate for the platform and version of MySQL which you will be using.

The `MysqldResource` enables you to "start" MySQL with a `java.util.Map` of server options which you provide, as well as "shutdown" the database. The following example shows a simplistic way to embed MySQL in an application using plain java objects.

You can find this file in the Connector/MXJ package as `src/ConnectorMXJObjectTestExample.java`.

```
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.HashMap;
import java.util.Map;
import com.mysql.management.MysqldResource;
import com.mysql.management.MysqldResourceI;
import com.mysql.management.util.QueryUtil;
public class ConnectorMXJObjectTestExample {
    public static final String DRIVER = "com.mysql.jdbc.Driver";
    public static final String JAVA_IO_TMPDIR = "java.io.tmpdir";
    public static void main(String[] args) throws Exception {
        File ourAppDir = new File(System.getProperty(JAVA_IO_TMPDIR));
        File databaseDir = new File(ourAppDir, "mysql-mxj");
        int port = Integer.parseInt(System.getProperty("c-mxj_test_port",
                "3336"));
        String userName = "alice";
        String password = "q93uti0opwhkd";
        MysqldResource mysqldResource = startDatabase(databaseDir, port,
                userName, password);
        Class.forName(DRIVER);
        Connection conn = null;
        try {
            String dbName = "our_test_app";
            String url = "jdbc:mysql://localhost:" + port + "/" + dbName //
                    + "?" + "createDatabaseIfNotExist=true"//
            ;
            conn = DriverManager.getConnection(url, userName, password);
            String sql = "SELECT VERSION()";
            String queryForString = new QueryUtil(conn).queryForString(sql);
            System.out.println("------------------------");
            System.out.println(sql);
            System.out.println("------------------------");
            System.out.println(queryForString);
            System.out.println("-------------------------");
            System.out.flush();
            Thread.sleep(100); // wait for System.out to finish flush
        } finally {
            try {
                if (conn != null) {
                    conn.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            try {
                mysqldResource.shutdown();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    public static MysqldResource startDatabase(File databaseDir, int port,
            String userName, String password) {
        MysqldResource mysqldResource = new MysqldResource(databaseDir);
        Map database_options = new HashMap();
        database_options.put(MysqldResourceI.PORT, Integer.toString(port));
        database_options.put(MysqldResourceI.INITIALIZE_USER, "true");
        database_options.put(MysqldResourceI.INITIALIZE_USER_NAME, userName);
        database_options.put(MysqldResourceI.INITIALIZE_PASSWORD, password);
        mysqldResource.start("test-mysqld-thread", database_options);
        if (!mysqldResource.isRunning()) {
            throw new RuntimeException("MySQL did not start.");
        }
        System.out.println("MySQL is running.");
        return mysqldResource;
    }
}
```

5.4.3. Setting server options

Of course there are many options we may wish to set for a MySQL database. These options may be specified as part of the JDBC connection string simply by prefixing each server option with `server.`. In the following example we set two driver parameters and two server parameters:

```
        String url = "jdbc:mysql://" + hostColonPort + "/"
                + "?"
                + "cacheServerConfiguration=true"
                + "&"
                + "useLocalSessionState=true"
                + "&"
                + "server.basedir=/opt/myapp/db"
                + "&"
                + "server.datadir=/mnt/bigdisk/myapp/data";
```

Starting with Connector/MXJ 5.0.6 you can use the `initializer-user` property to a connection string. If set to true, the default anonymous and root users will be removed and the user/password combination from the connection URL will be used to create a new user. For example:

```
        String url = "jdbc:mysql:mxj://localhost:" + port
        + "/alice_db"
        + "?server.datadir=" + dataDir.getPath()
        + "&server.initialize-user=true"
        + "&createDatabaseIfNotExist=true"
        ;
```

5.5. Connector/MXJ Reference

The following sections include detailed information on the different API interfaces to Connector/MXJ.

5.5.1. MysqldResource Constructors

The `MysqldResource` class supports three different constructor forms:

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString, PrintStream out, PrintStream err)`

  Enables you to set the base directory, data directory, select a server by its version string, standard out and standard error.

- `public MysqldResource(File baseDir, File dataDir, String mysqlVersionString)`

Enables you to set the base directory, data directory and select a server by its version string. Output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource(File baseDir, File dataDir)`

  Enables you to set the base directory and data directory. The default MySQL version is selected, and output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource(File baseDir);`

  Allows the setting of the "basedir" to deploy the MySQL files to. Output for standard out and standard err are directed to System.out and System.err.

- `public MysqldResource();`

  The basedir is defaulted to a subdirectory of the java.io.tempdir. Output for standard out and standard err are directed to System.out and System.err;

5.5.2. MysqldResource Methods

MysqldResource API includes the following methods:

- `void start(String threadName, Map mysqldArgs);`

  Deploys and starts MySQL. The "threadName" string is used to name the thread which actually performs the execution of the MySQL command line. The map is the set of arguments and their values to be passed to the command line.

- `void shutdown();`

  Shuts down the MySQL instance managed by the MysqldResource object.

- `Map getServerOptions();`

  Returns a map of all the options and their current (or default, if not running) options available for the MySQL database.

- `boolean isRunning();`

  Returns true if the MySQL database is running.

- `boolean isReadyForConnections();`

  Returns true once the database reports that is ready for connections.

- `void setKillDelay(int millis);`

  The default “Kill Delay” is 30 seconds. This represents the amount of time to wait between the initial request to shutdown and issuing a “force kill” if the database has not shutdown by itself.

- `void addCompletionListenser(Runnable listener);`

  Allows for applications to be notified when the server process completes. Each "listener" will be fired off in its own thread.

- `String getVersion();`

  Returns the version of MySQL.

- `void setVersion(int MajorVersion, int minorVersion, int patchLevel);`

  The standard distribution comes with only one version of MySQL packaged. However, it is possible to package multiple versions, and specify which version to use.

5.6. Connector/MXJ Notes and Tips

This section contains notes and tips on using the Connector/MXJ component within your applications.

5.6.1. Creating your own Connector/MXJ Package

If you want to create a custom Connector/MXJ package that includes a specific `mysqld` version or platform then you must extract and rebuild the `mysql-connector-mxj.jar` (Connector/MXJ v5.0.3 or earlier) or `mysql-connector-mxj-gpl-[ver]-db-files.jar` (Connector/MXJ v5.0.4 or later) file.

First, you should create a new directory into which you can extract the current `connector-mxj.jar`:

```
shell> mkdir custom-mxj
shell> cd custom-mxj
shell> jar -xf connector-mxj.jar
shell> ls
5-0-22/
ConnectorMXJObjectTestExample.class
ConnectorMXJUrlTestExample.class
META-INF/
TestDb.class
com/
kill.exe
```

If you are using Connector/MXJ v5.0.4 or later, you should unpack the `connector-mxj-db-files.jar`:

```
shell> mkdir custom-mxj
shell> cd custom-mxj
shell> jar -xf connector-mxj-db-files.jar
shell> ls
5-0-51a/
META-INF/
connector-mxj.properties
```

The MySQL version directory, `5-0-22` or `5-0-51a` in the preceding examples, contains all of the files used to create an instance of MySQL when Connector/MXJ is executed. All of the files in this directory are required for each version of MySQL that you want to embed. Note as well the format of the version number, which uses hyphens instead of periods to separate the version number components.

Within the version specific directory are the platform specific directories, and archives of the `data` and `share` directory required by MySQL for the various platforms. For example, here is the listing for the default Connector/MXJ package:

```
shell>> ls
Linux-i386/
META-INF/
Mac_OS_X-ppc/
SunOS-sparc/
Win-x86/
com/
data_dir.jar
share_dir.jar
win_share_dir.jar
```

Platform specific directories are listed by their OS and platform - for example the `mysqld` for Mac OS X PowerPC is located within the `Mac_OS_X-ppc` directory. You can delete directories from this location that you do not require, and add new directories for additional platforms that you want to support.

To add a platform specific `mysqld`, create a new directory with the corresponding name for your operating system/platform. For example, you could add a directory for Mac OS X/Intel using the directory `Mac_OS_X-i386`.

On Unix systems, you can determine the platform using `uname`:

```
shell> uname -p
i386
```

In Connector/MXJ v5.0.9 and later, an additional `platform-map.properties` file is used to associate a specific platform and operating system combination with the directory in which the `mysqld` for that combination is located. The determined operating system and platform are on the left, and the directory name where the appropriate mysqld is located is on the right. You can see a sample of the file below:

```
Linux-i386=Linux-i386
Linux-x86=Linux-i386
Linux-i686=Linux-i386
Linux-x86_64=Linux-i386
Linux-ia64=Linux-i386
#Linux-ppc=Linux-ppc
#Linux-ppc64=Linux-ppc
Mac_OS_X-i386=Mac_OS_X-i386
Mac_OS_X-ppc=Mac_OS_X-ppc
Rhapsody-PowerPC=Mac_OS_X-ppc
#Mac_OS-PowerPC=
#macos-PowerPC=
#MacOS-PowerPC=
SunOS-sparc=SunOS-sparc
Solaris-sparc=SunOS-sparc
SunOS-x86=SunOS-x86
Solaris-x86=SunOS-x86
```

```
FreeBSD-x86=FreeBSD-x86
Windows_Vista-x86=Win-x86
Windows_2003-x86=Win-x86
Windows_XP-x86=Win-x86
Windows_2000-x86=Win-x86
Windows_NT-x86=Win-x86
Windows_NT_(unknown)-x86=Win-x86
```

Now you need to download or compile `mysqld` for the MySQL version and platform you want to include in your custom `connector-mxj.jar` package into the new directory.

Create a file called `version.txt` in the OS/platform directory you have just created that contains the version string/path of the mysqld binary. For example:

```
mysql-5.0.22-osx10.3-i386/bin/mysqld
```

You can now recreate the `connector-mxj.jar` file with the added `mysqld`:

```
shell> cd custom-mxj
shell> jar -cf ../connector-mxj.jar *
```

For Connector/MXJ v5.0.4 and later, you should repackage to the `connector-mxj-db-files.jar`:

```
shell> cd custom-mxj
shell> jar -cf ../mysql-connector-mxj-gpl-[ver]-db-files.jar *
```

You should test this package using the steps outlined in Section 5.3.3, “Connector/MXJ Quick Start Guide”.

Note

Because the `mysql-connector-mxj-gpl-[ver]-db-files.jar` file is separate from the main Connector/MXJ classes you can distribute different `mysql-connector-mxj-gpl-[ver]-db-files.jar` files to different hotsts or for different projects without having to create a completely new main `mysql-connector-mxj-gpl-[ver].jar` file for each one.

5.6.2. Deploying Connector/MXJ with a pre-configured database

To include a pre-configured/populated database within your Connector/MXJ JAR file you must create a custom `data_dir.jar` file, as included within the main `connector-mxj.jar` (Connector/MXJ 5.0.3 or earlier) or `mysql-connector-mxj-gpl-[ver]-db-files.jar` (Connector/MXJ 5.0.4 or later) file:

1. First extract the `connector-mxj.jar` or `mysql-connector-gpl-[ver]-db-files.jar` file, as outlined in the previous section (see Section 5.6.1, “Creating your own Connector/MXJ Package”).

2. First, create your database and populate the database with the information you require in an existing instance of MySQL - including Connector/MXJ instances. Data file formats are compatible across platforms.

3. Shutdown the instance of MySQL.

4. Create a JAR file of the data directory and databases that you want to include your Connector/MXJ package. You should include the `mysql` database, which includes user authentication information, in addition to the specific databases you want to include. For example, to create a JAR of the `mysql` and `mxjtest` databases:

   ```
   shell> jar -cf ../data_dir.jar mysql mxjtest
   ```

5. For Connector/MXJ 5.0.3 or earlier, copy the `data_dir.jar` file into the extracted `connector-mxj.jar` directory, and then create an archive for `connector-mxj.jar`.

   For Connector/MXJ 5.0.4 or later, copy the `data_dir.jar` file into the extracted `mysql-connector-mxj-gpl-[ver]-db-files.jar` directory, and then create an archive for `mysql-connector-mxj-db-gpl-[ver]--files.jar`.

Note that if you are create databases using the InnoDB engine, you must include the `ibdata.*` and `ib_logfile*` files within the `data_dir.jar` archive.

5.6.3. Running within a JMX Agent (custom)

As a JMX MBean, MySQL Connector/MXJ requires a JMX v1.2 compliant MBean container, such as JBoss version 4. The MBean will uses the standard JMX management APIs to present (and allow the setting of) parameters which are appropriate for that platform.

If you are not using the SUN Reference implementation of the JMX libraries, you should skip this section. Or, if you are deploying to JBoss, you also may wish to skip to the next section.

We want to see the MysqldDynamicMBean in action inside of a JMX agent. In the `com.mysql.management.jmx.sunri` package is a custom JMX agent with two MBeans:

1. the MysqldDynamicMBean, and
2. a com.sun.jdmk.comm.HtmlAdaptorServer, which provides a web interface for manipulating the beans inside of a JMX agent.

When this very simple agent is started, it will allow a MySQL database to be started and stopped with a web browser.

1. Complete the testing of the platform as above.
   - current JDK, JUnit, Connector/J, MySQL Connector/MXJ
   - this section *requires* the SUN reference implementation of JMX
   - `PATH`, `JAVA_HOME`, `ANT_HOME`, `CLASSPATH`
2. If not building from source, skip to next step

   rebuild with the "sunri.present"

   ```
   ant -Dsunri.present=true dist
   re-run tests:
   java junit.textui.TestRunner com.mysql.management.AllTestsSuite
   ```
3. launch the test agent from the command line:

   ```
   java com.mysql.management.jmx.sunri.MysqldTestAgentSunHtmlAdaptor &
   ```
4. from a browser:

   ```
   http://localhost:9092/
   ```
5. under MysqldAgent,

   ```
   select "name=mysqld"
   ```
6. Observe the MBean View
7. scroll to the bottom of the screen press the STARTMYSQLD button
8. click `Back to MBean View`
9. scroll to the bottom of the screen press STOPMYSQLD button
10. kill the java process running the Test Agent (jmx server)

5.6.4. Deployment in a standard JMX Agent environment (JBoss)

Once there is confidence that the MBean will function on the platform, deploying the MBean inside of a standard JMX Agent is the next step. Included are instructions for deploying to JBoss.

1. Ensure a current version of java development kit (v1.4.x), see above.
   - Ensure `JAVA_HOME` is set (JBoss requires `JAVA_HOME`)

- Ensure JAVA_HOME/bin is in the PATH (You will NOT need to set your CLASSPATH, nor will you need any of the jars used in the previous tests).

2. Ensure a current version of JBoss (v4.0RC1 or better)

```
http://www.jboss.org/index.html
select "Downloads"
select "jboss-4.0.zip"
pick a mirror
unzip ~/dload/jboss-4.0.zip
create a JBOSS_HOME environment variable set to the unzipped directory
unix only:
cd $JBOSS_HOME/bin
chmod +x *.sh
```

3. Deploy (copy) the connector-mxj.jar to $JBOSS_HOME/server/default/lib.

4. Deploy (copy) mysql-connector-java-3.1.4-beta-bin.jar to $JBOSS_HOME/server/default/lib.

5. Create a mxjtest.war directory in $JBOSS_HOME/server/default/deploy.

6. Deploy (copy) index.jsp to $JBOSS_HOME/server/default/deploy/mxjtest.war.

7. Create a mysqld-service.xml file in $JBOSS_HOME/server/default/deploy.

```
<?xml version="1.0" encoding="UTF-8"?>
 <server>
  <mbean code="com.mysql.management.jmx.jboss.JBossMysqldDynamicMBean"
     name="mysql:type=service,name=mysqld">
  <attribute name="datadir">/tmp/xxx_data_xxx</attribute>
  <attribute name="autostart">true</attribute>
  </mbean>
 </server>
```

8. Start jboss:

   - on unix: $JBOSS_HOME/bin/run.sh

   - on windows: %JBOSS_HOME%\bin\run.bat

   Be ready: JBoss sends a lot of output to the screen.

9. When JBoss seems to have stopped sending output to the screen, open a web browser to: ht-tp://localhost:8080/jmx-console

10. Scroll down to the bottom of the page in the mysql section, select the bulleted mysqld link.

11. Observe the JMX MBean View page. MySQL should already be running.

12. (If "autostart=true" was set, you may skip this step.) Scroll to the bottom of the screen. You may press the INVOKE button to stop (or start) MySQL observe Operation completed successfully without a return value. Click Back to MBean View

13. To confirm MySQL is running, open a web browser to http://localhost:8080/mxjtest/ and you should see that

```
SELECT 1
```

    returned with a result of

```
1
```

14. Guided by the $JBOSS_HOME/server/default/deploy/mxjtest.war/index.jsp you will be able to use MySQL in your Web Application. There is a test database and a root user (no password) ready to experiment with. Try creating a table, inserting some rows, and doing some selects.

15. Shut down MySQL. MySQL will be stopped automatically when JBoss is stopped, or: from the browser, scroll down to the bottom of the MBean View press the stop service INVOKE button to halt the service. Observe Operation completed successfully without a return value. Using ps or task manager see that MySQL is no longer running

As of 1.0.6-beta version is the ability to have the MBean start the MySQL database upon start up. Also, we've taken advantage of the JBoss life-cycle extension methods so that the database will gracefully shut down when JBoss is shutdown.

5.7. Connector/MXJ Support

There are a wide variety of options available for obtaining support for using Connector/MXJ. You should contact the Connector/MXJ community for help before reporting a potential bug or problem. See Section 5.7.1, "Connector/MXJ Community Support".

5.7.1. Connector/MXJ Community Support

Sun Microsystems, Inc. provides assistance to the user community by means of a number of mailing lists and web based forums.

You can find help and support through the MySQL and Java mailing list.

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 MyODBC 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.

5.7.2. How to Report Connector/MXJ Problems

If you encounter difficulties or problems with Connector/MXJ, contact the Connector/MXJ community Section 5.7.1, "Connector/MXJ Community Support".

If reporting a problem, you should ideally include the following information with the email:

- Operating system and version
- Connector/MXJ 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/.

5.7.3. Connector/MXJ Change History

The Connector/MXJ Change History (Changelog) is located with the main Changelog for MySQL. See Appendix E, *MySQL Connector/MXJ Change History*.

Chapter 6. MySQL Connector/C++

Caution

Please note, official support is not available for the beta version of MySQL Connector/C++.

MySQL Connector/C++ is a MySQL database connector for C++.

The MySQL Connector/C++ is licensed under the terms of the GPL, like most MySQL Connectors. There are special exceptions to the terms and conditions of the GPL as it is applied to this software, see FLOSS License Exception. If you need a non-GPL license for commercial distribution please contact us.

The MySQL Connector/C++ is compatible with the JDBC 4.0 API. However, MySQL Connector/C++ does not implement all of the JDBC 4.0 API. The MySQL Connector/C++ current version features the following classes:

- `Connection`
- `DatabaseMetaData`
- `Driver`
- `PreparedStatement`
- `ResultSet`
- `ResultSetMetaData`
- `Savepoint`
- `Statement`

The JDBC 4.0 API defines approximately 450 methods for the above mentioned classes. MySQL Connector/C++ implements around 80% of these and makes them available in the preview release.

The Beta release has been successfully compiled and tested on the following platforms:

AIX

- 5.2 (PPC32, PPC64)
- 5.3 (PPC32, PPC64)

FreeBSD

- 6.0 (x86, x86_64)

HPUX

- 11.11 (PA-RISC 32bit, PA-RISC 64bit)

Linux

- Debian 3.1 (PPC32, x86)
- FC4 (x86)
- RHEL 3 (ia64, x86, x86_64)
- RHEL 4 (ia64, x86, x86_64)
- RHEL 5 (ia64, x86, x86_64)

- SLES 9 (ia64, x86, x86_64)
- SLES 10 (ia64, x86_64)
- SuSE 10.3, (x86_64)
- Ubuntu 8.04 (x86)
- Ubuntu 8.10 (x86_64)

Mac

- MacOSX 10.3 (PPC32, PPC64)
- MacOSX 10.4 (PPC32, PPC64, x86)
- MacOSX 10.5 (PPC32, PPC64, x86, x86_64)

SunOS

- Solaris 8 (SPARC32, SPARC64, x86)
- Solaris 9 (SPARC32, SPARC64, x86)
- Solaris 10 (SPARC32, SPARC64, x86, x86_64)

Windows

- XP Professional (32bit)
- 2003 (64bit)

Future versions will run on all platforms supported by the MySQL Server.

> **Note**
>
> MySQL Connector/C++ supports MySQL 5.1 and later.

> **Note**
>
> MySQL Connector/C++ supports only Microsoft Visual Studio 2003 and above on Windows.

MySQL Connector/C++ Download

You can download the source code for the MySQL Connector/C++ preview release at the download site.

MySQL Connector/C++ Source repository

The latest development version is also available through Launchpad.

Bazaar is used for the MySQL Connector/C++ code repository. You can check out the source code using the `bzr` command line tool:

```
shell> bzr branch lp:~mysql/mysql-connector-cpp/trunk .
```

The source of the 1.0.3alpha release is available from the following branch:

```
shell> bzr branch lp:~andrey-mysql/mysql-connector-cpp/v1_0_3
```

The source of the 1.0.2alpha release is available from the following branch:

```
shell> bzr branch lp:~andrey-mysql/mysql-connector-cpp/v1_0_2
```

The source of the 1.0.1alpha release is available from the following branch:

```
shell> bzr branch lp:~andrey-mysql/mysql-connector-cpp/v1_0_1
```

Binary distributions

Starting with 1.0.4 Beta, binary distributions will be made available in addition to source code releases. The releases available are shown below.

Microsoft Windows platform:

- Without installer, a Zip file
- MSI installer package

Other platforms:

- Compressed GNU TAR archive (tar.gz)

> **Note**
>
> Note that source packages are available for all platforms in the Compressed GNU TAR archive (tar.gz) format.

MySQL Connector/C++ Advantages

Using MySQL Connector/C++ instead of the MySQL C API (MySQL Client Library) offers the following advantages for C++ users:

- Convenience of pure C++, no C function calls required
- Supports an industry standard API, JDBC 4.0
- Supports the object-oriented programming paradigm
- Reduces development time
- MySQL Connector/C++ is licensed under the GPL with the FLOSS License Exception
- MySQL Connector/C++ is available under a commercial license upon request

MySQL Connector/C++ Status

MySQL Connector/C++ is available as a development preview version. We kindly ask users and developers to try it out and provide us with feedback. We do not encourage you to use it in production environments.

Note that MySQL Workbench is successfully using MySQL Connector/C++.

> **Note**
>
> Sun Microsystems does not provide formal support for MySQL Connector/C++.

If you have any queries please contact us.

6.1. MySQL Connector/C++ Binary Installation

> **Caution**
>
> Please note, official support is not available for the beta version of MySQL Connector/C++.

> **Caution**
>
> One problem that can occur is when the tools you use to build your application are not compatible with the tools used to build the binary versions of MySQL Connector/C++. Ideally you need to build your application with the same tools that were used to build the MySQL Connector/C++ binaries. To help with this the following resources are provided.
>
> All distributions contain a `README` file, which contains platform-specific notes. At the end of the `README` file con-

tained in the binary distribution you will find the settings used to build the binaries. If you experience build-related issues on a platform, it may help to check the settings used on the platform to build the binary.

For your convenience the same information, but more frequently updated, can be found on the MySQL Forge site.

A better solution is to build your MySQL Connector/C++ libraries from the source code, using the same tools that you use for building your application. This ensures compatibility.

Archive Package

Unpack the archive into an appropriate directory. If you plan to use a dynamically linked version of MySQL Connector/C++, make sure that your system can reference the MySQL Client Library. Consult your operating system documentation on how do modify and expand the search path for libraries. In case you cannot modify the library search path it may help to copy your application, the MySQL Connector/C++ library and the MySQL Client Library into the same directory. Most systems search for libraries in the current directory.

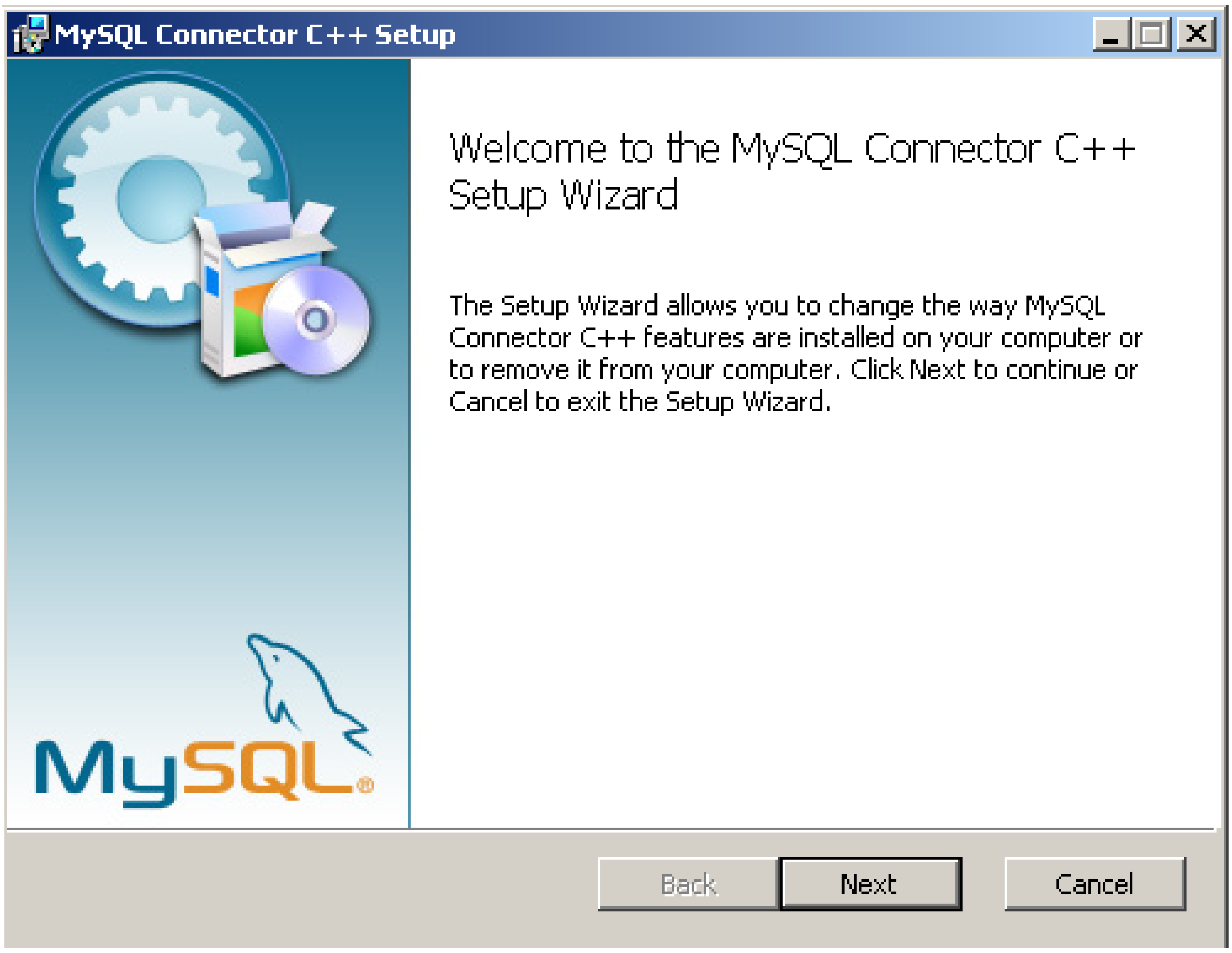
Windows MSI Installer

Windows users can choose between two binary packages:

1. Without installer (unzip in C:\)
2. Windows MSI Installer (x86)

Using the MSI Installer may be the easiest solution. Running the MSI Installer does not require any administrative permissions as it simply copies files.

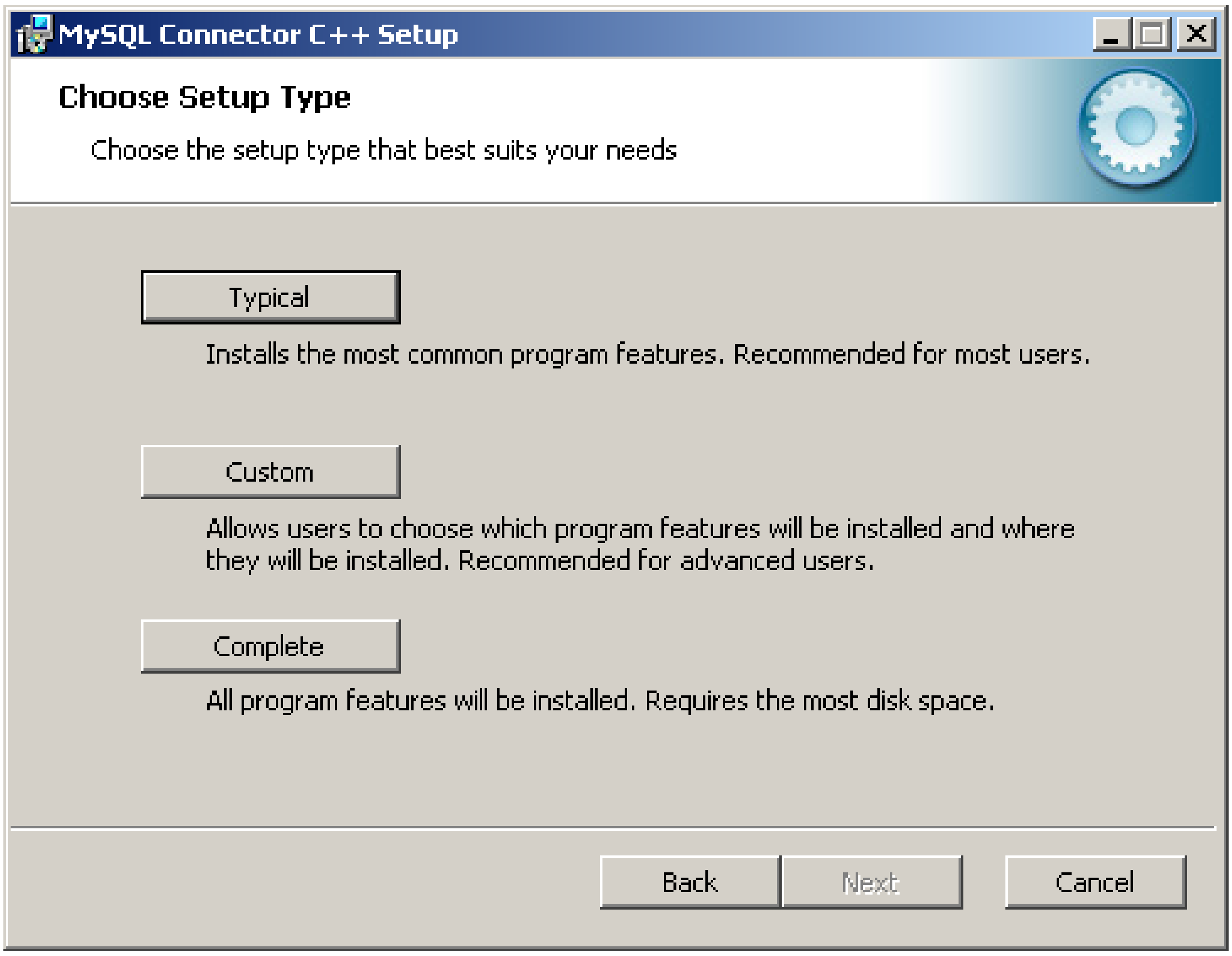
Figure 6.1. Windows Installer Welcome Screen


MySQL Connector C++ Setup

Welcome to the MySQL Connector C++ Setup Wizard

The Setup Wizard allows you to change the way MySQL Connector C++ features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 6.2. Windows Installer Overview Screen



MySQL Connector C++ Setup

Choose Setup Type

Choose the setup type that best suits your needs

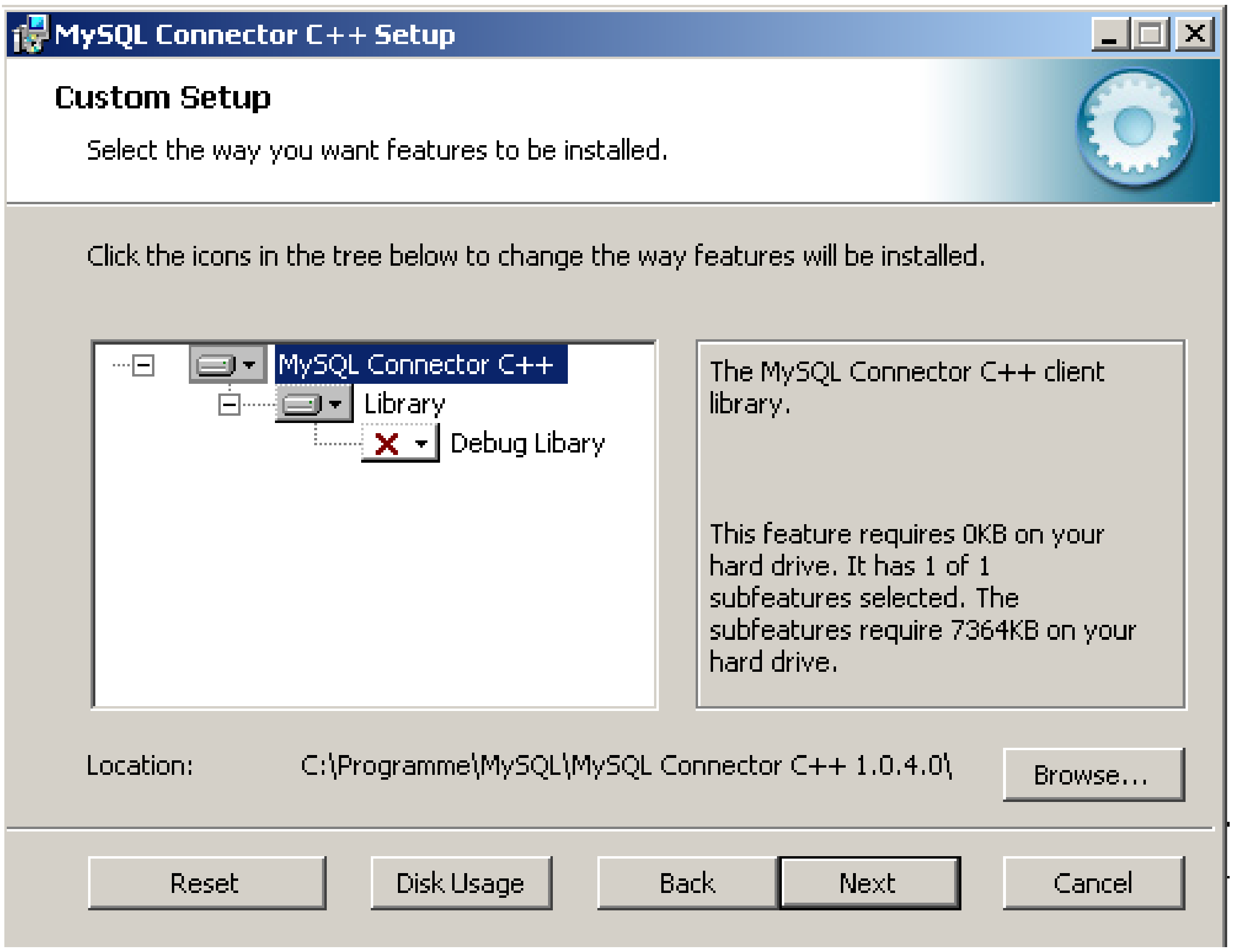
Installs the most common program features. Recommended for most users.

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

All program features will be installed. Requires the most disk space.

The “Typical” installation consists of all required header files and the Release libraries. The only available “Custom” installation option allows you to install additional Debug versions of the connector libraries.

Figure 6.3. Windows Installer Custom Setup Screen


MySQL Connector C++ Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
MySQL Connector C++
Library
Debug Libary
Reset
Disk Usage
Back
Next
Cancel
Browse...

The MySQL Connector C++ client library.

This feature requires 0KB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 7364KB on your hard drive.

Location: C:\Programme\MySQL\MySQL Connector C++ 1.0.4.0\

6.2. MySQL Connector/C++ Source Installation

Caution

Please note, official support is not available for the beta version of MySQL Connector/C++.

The MySQL Connector/C++ is based on the MySQL Client Library (MySQL C API). MySQL Connector/C++ is linked against the MySQL Client Library. You need to have the MySQL Client Library installed in order to compile MySQL Connector/C++.

You also need to have the cross-platform build tool CMake 2.4, or newer, and GLib 2.2.3 or newer installed. Check the README file included with the distribution for platform specific notes on building for Windows and SunOS.

Typically the MySQL Client Library is installed when the MySQL Server is installed. However, check your operating system documentation for other installation options.

6.2.1. Building source on Unix, Solaris and Mac OS X

1. Run CMake to build a Makefile:

```
shell> me@host:/path/to/mysql-connector-cpp> 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/
```

On non-Windows systems, CMake first checks to see if the CMake variable `MYSQL_CONFIG_EXECUTABLE` is set. If it is not found CMake will try to locate `mysql_config` in the default locations.

If you have any problems with the configure process please check the troubleshooting instructions below.

2. Use make to build the libraries. First make sure you have a clean build:

   ```
   shell> me@host:/path/to/mysql-connector-cpp> make clean
   ```

   Then build the connector:

   ```
   me@host:/path/to/mysql-connector-cpp> 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 MySQL Connector/C++ library in `/path/to/cppconn/libmysqlcppconn.so`.

3. Finally make sure the header and library files are installed to their correct locations:

   ```
   make install
   ```

   Unless you have changed this in the configuration step, the header files will be copied to the directory `/usr/local/include`. The header files copied are `mysql_connection.h` and `mysql_driver.h`.

   Again, unless you have specified otherwise, the library files will be copied to `/usr/local/lib`. The files copied are `libmysqlcppcon.so`, the dynamic library, and `libmysqlcppconn-static.a`, the static library.

If you encounter any errors, please first carry out the checks shown below:

1. CMake options: MySQL installation path, debug version and more

   In case of configure and/or compile problems check the list of CMake options:

   ```
   shell> me@host:/path/to/mysql-connector-cpp> 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/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 MySQL Connector/C++ use:

   ```
   shell> me@host:/path/to/mysql-connector-cpp> cmake »
   -D CMAKE_BUILD_TYPE:STRING=Debug »
   -D MYSQL_CONFIG_EXECUTABLE=/path/to/my/mysql/server/bin/mysql_config .
   ```

2. Verify your settings with `cmake -L`:

   ```
   shell> me@host:/path/to/mysql-connector-cpp> 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=/path/to/my/mysql/server/bin/mysql_config
   ```

   Proceed by carrying out a `make clean` command followed by a `make` command, as described above.

6.2.2. Building on Windows

Note

Please note the only compiler formally supported for Windows is Microsoft Visual Studio 2003 and above.

The basic steps for building the connector on Windows are the same as for Unix. It is important to use CMake 2.6.2 or newer to generate build files for your compiler and to invoke the compiler.

> **Note**
>
> On Windows, `mysql_config` is not present, so CMake will attempt to retrieve the location of MySQL from the environment variable `$ENV{MYSQL_DIR}`. If `MYSQL_DIR` is not set, CMake will then proceed to check for MySQL in the following locations: `$ENV{ProgramFiles}/MySQL/*/include`, and `$ENV{SystemDrive}/MySQL/*/include`.

CMake makes it easy for you to try out other compilers. However, you may experience compile warnings, compile errors or linking issues not detected by Visual Studio. Patches are gratefully accepted to fix issues with other compilers.

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 will support more compilers, known by CMake as *generators*, than supported by MySQL Connector/C++. We have built the connector using the following generators:

- Microsoft Visual Studio 8 (Visual Studio 2005)
- Microsoft Visual Studio 9 (Visual Studio 2008, Visual Studio 2008 Express)
- NMake

Please see the building instructions for Unix, Solaris and Mac OS X for troubleshooting and configuration hints.

The steps to build the connector are given below:

1. Run CMake to generate build files for your *generator*:

   Visual Studio

   ```
    C:\path_to_mysql_cpp>cmake -G "Visual Studio 9 2008"
   -- 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:/Programme/MySQL/MySQL Server 5.1/include
   -- MySQL Library    : C:/Progams/MySQL/MySQL Server 5.1/lib/opt/mysqlclient.lib
   -- MySQL Library dir: C:/Progams/MySQL/MySQL Server 5.1/lib/opt
   -- MySQL CFLAGS:
   -- MySQL Link flags:
   -- MySQL Include dir: C:/Progams/MySQL/MySQL Server 5.1/include
   -- MySQL Library dir: C:/Progams/MySQL/MySQL Server 5.1/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:\path_to_mysql_cpp>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:\path_to_mysql_cpp>cmake -G "NMake Makefiles"
-- The C compiler identification is MSVC
-- The CXX compiler identification is MSVC
[...]
-- Build files have been written to: C:\path_to_mysql_cpp
```

2. Use your compiler to build MySQL Connector/C++

   Visual Studio - GUI

   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. Among them debug and non-debug versions.

   Visual Studio - NMake

```
C:\path_to_mysql_cpp>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
```

6.3. MySQL Connector/C++ Building Windows applications with Microsoft Visual Studio

MySQL Connector/C++ is available as a static or dynamic library to use with your application. This section looks at how to link the library to your application.

> **Note**
>
> To avoid potential crashes the build configuration of MySQL Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of MySQL Connector/C++ with a debug build of the client application.

Static library

The MySQL Connector/C++ static library file is `mysqlcppconn-static.lib`. This needs to be statically linked with your application. You also need to link against the files `libmysql.dll` and `libmysql.lib`. Once linking has been successfully completed, the application will require access to `libmysql.dll` at run time.

Dynamic library

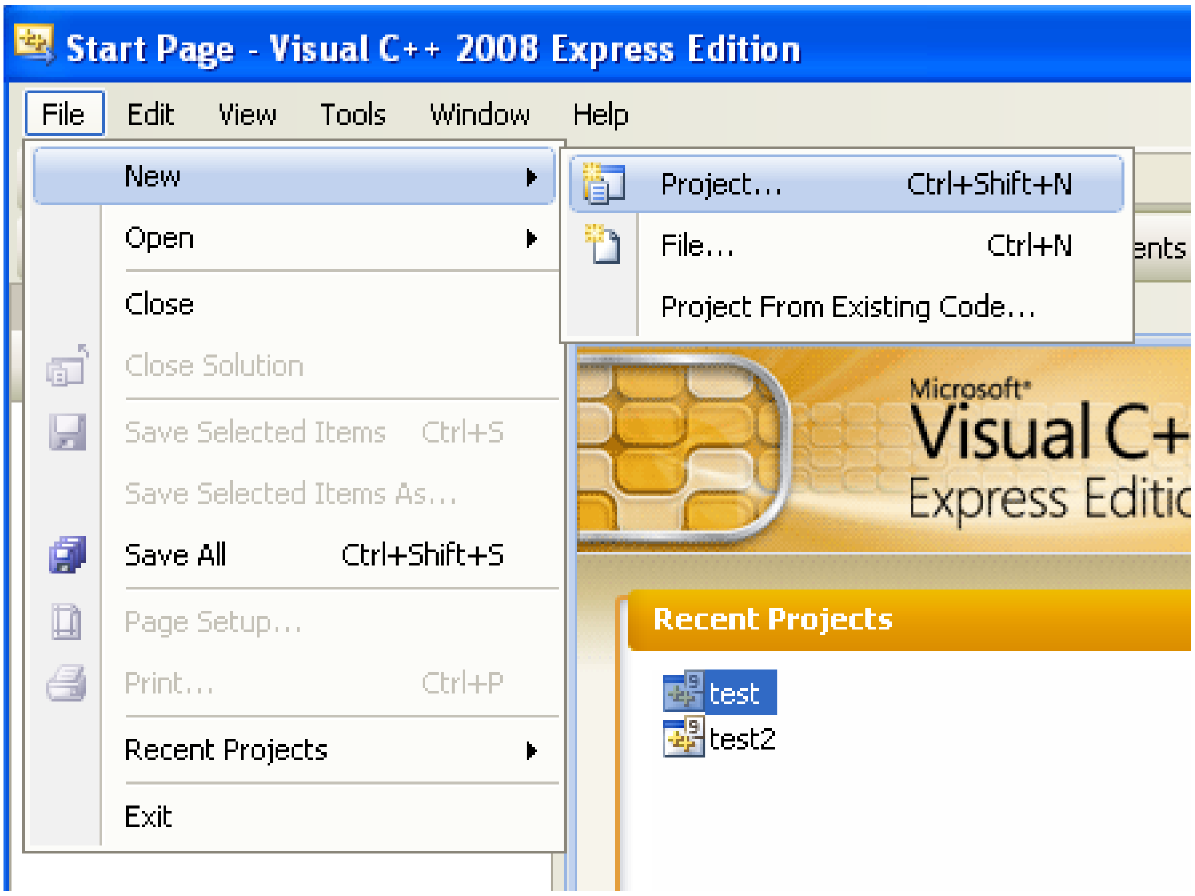
The MySQL Connector/C++ dynamic library file is `mysqlcppconn.dll`. In order to build your client application you need to link it with the file `mysqlcppconn.lib`. At run time the application will require access to the files `mysqlcppconn.dll` and `libmysql.dll`.

Building a MySQL Connector/C++ application with Microsoft Visual Studio

Initially, the procedure for building an application to use either the static or dynamic library is the same. You then carry out some additional steps depending on whether you want to build your application to use the static or dynamic library.

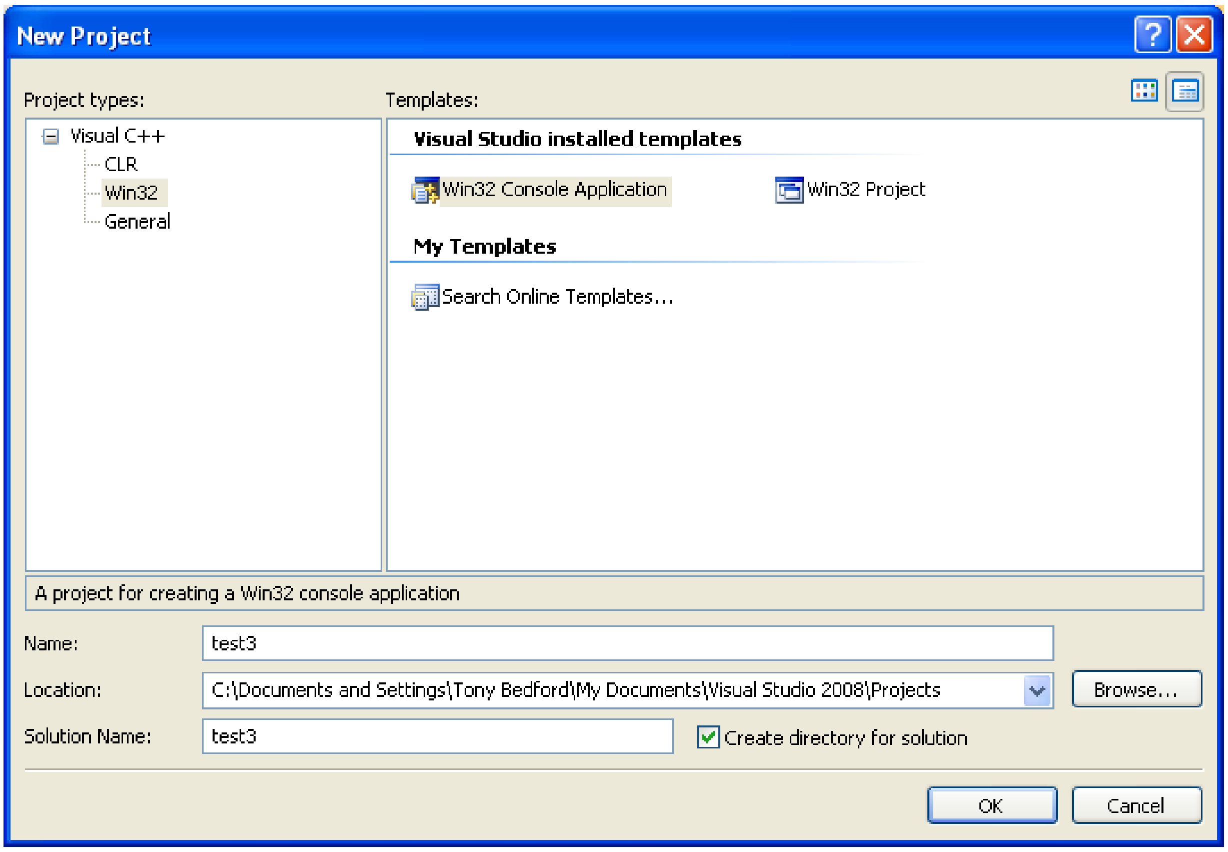
1. Select FILE, NEW, PROJECT from the main menu.

Figure 6.4. 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, and then click OK, to move to the Win32 Application Wizard.

Figure 6.5. The New Project dialog box



A project for creating a Win32 console application

Name: test3

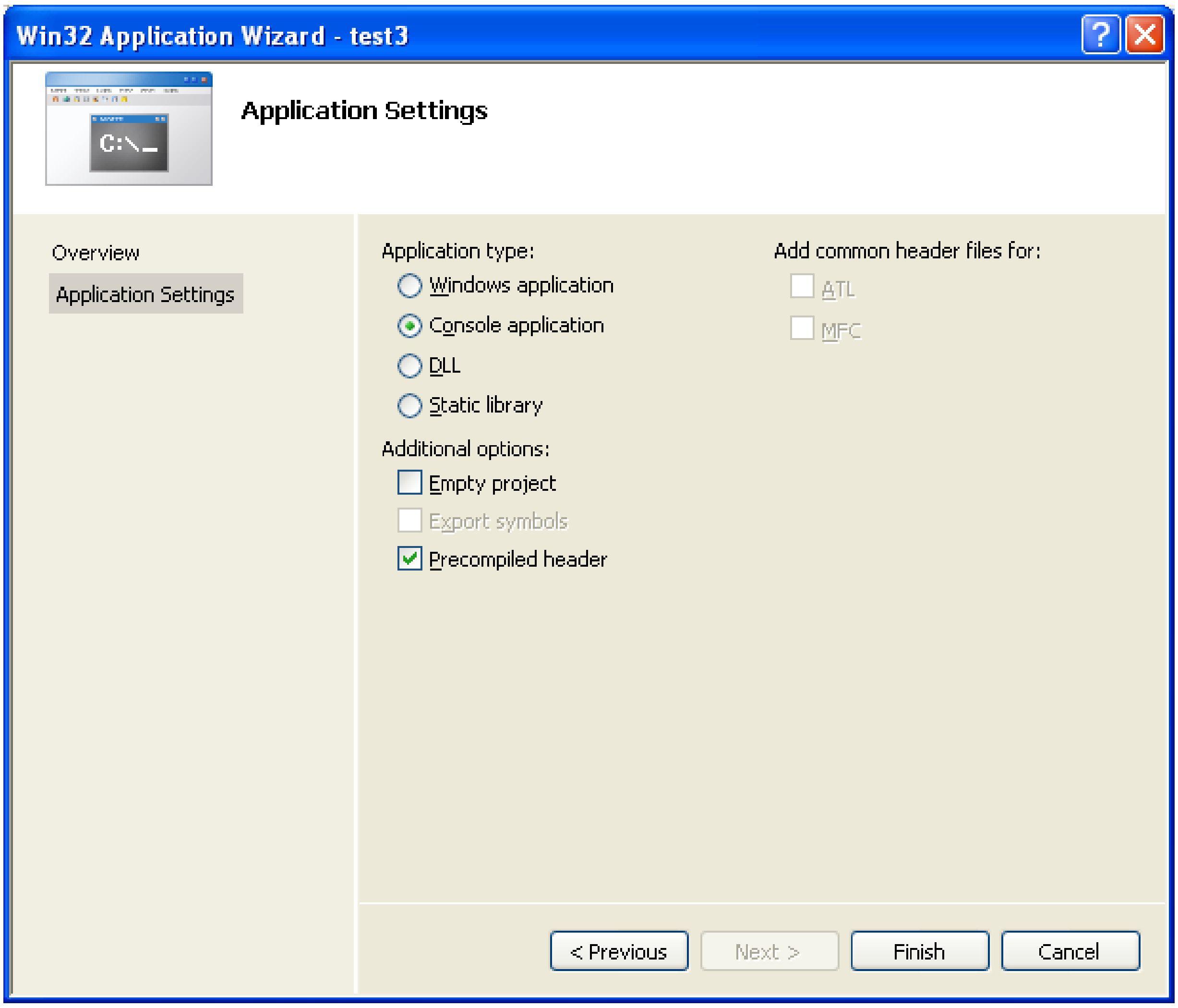
Location: C:\Documents and Settings\Tony Bedford\My Documents\Visual Studio 2008\Projects

Solution Name: test3

✓ Create directory for solution

3. In the Win32 Application Wizard, click on **APPLICATION SETTINGS** and ensure the defaults are selected. The radio button **CONSOLE APPLICATION**, and the checkbox **PRECOMPILED HEADERS** will be selected. Click FINISH to close the wizard.

Figure 6.6. The Win32 Application Wizard



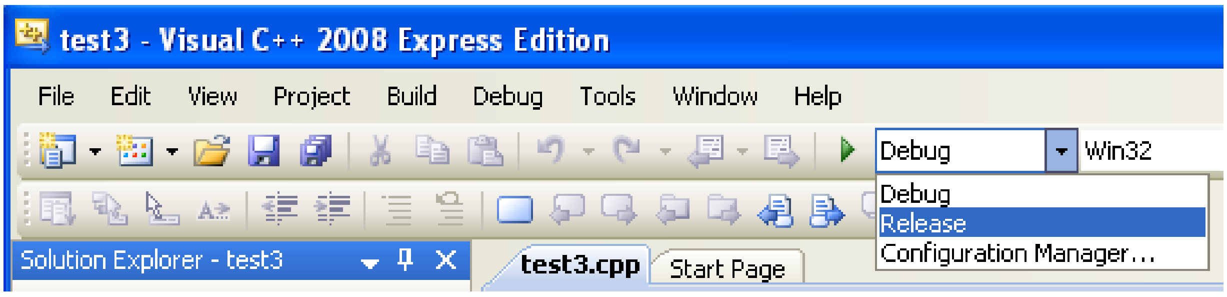
Application type:
- ( ) Windows application
- (●) Console application
- ( ) DLL
- ( ) Static library

Additional options:
- [ ] Empty project
- [ ] Export symbols
- [✓] Precompiled header

Add common header files for:
- [ ] ATL
- [ ] MFC

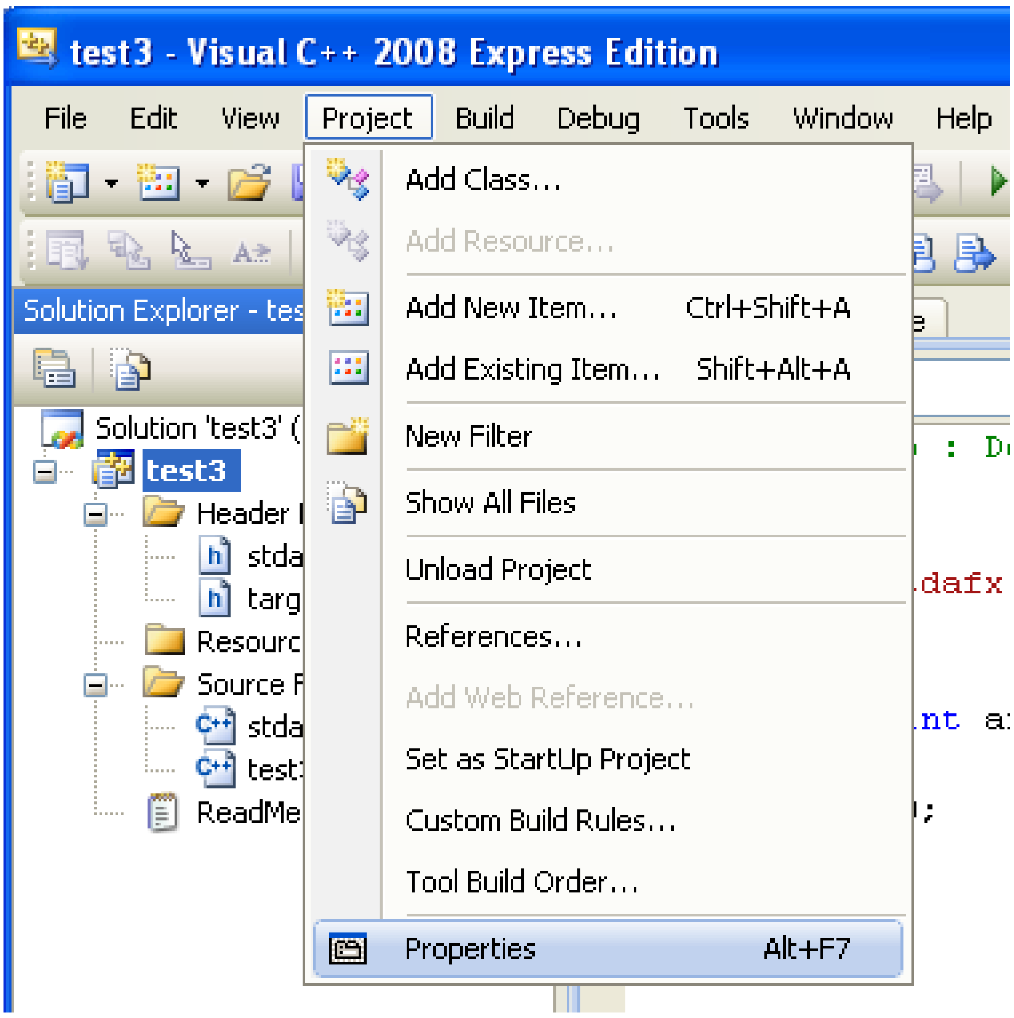
4. From the drop down list box on the toolbar, change from the default **DEBUG** build to the **RELEASE** build.

Figure 6.7. Selecting the Release build



5. From the main menu select PROJECT, PROPERTIES. This can also be accessed using the hot key ALT + F7.

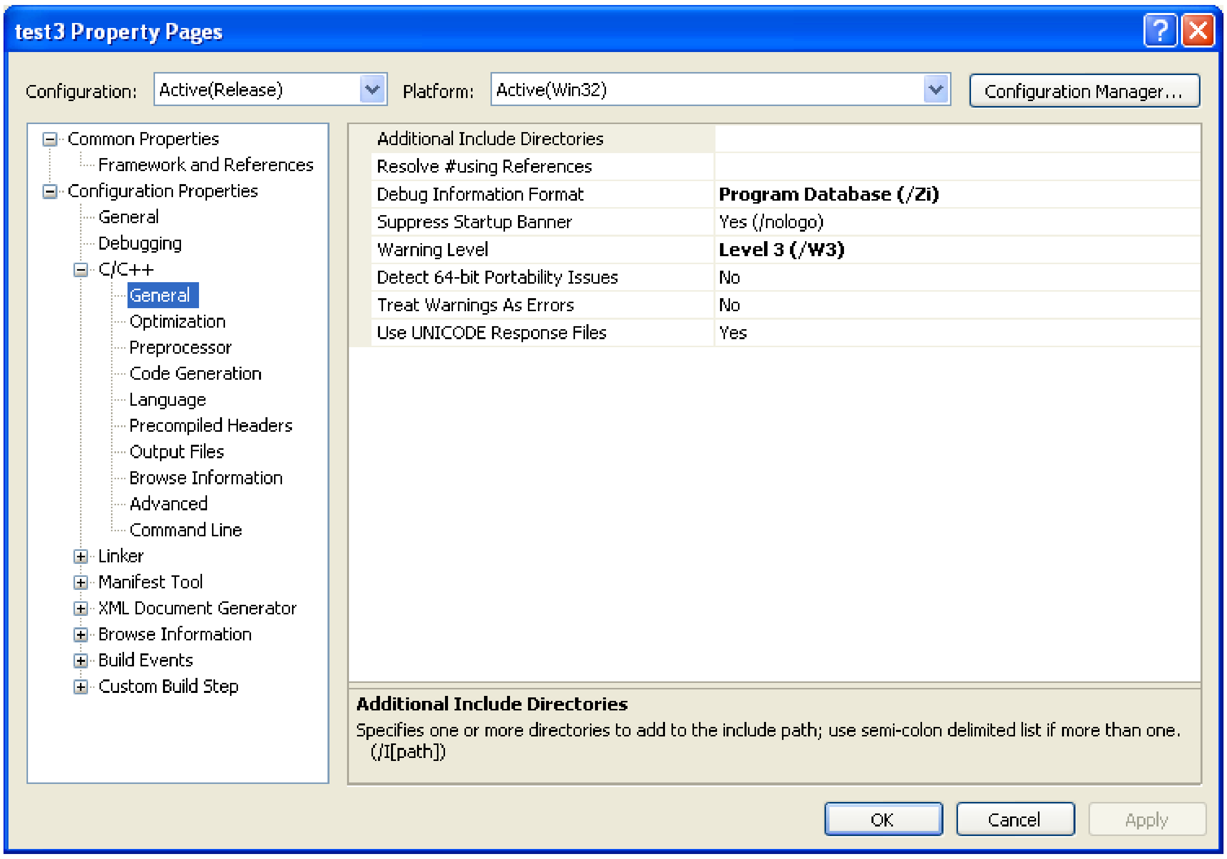
Figure 6.8. Selecting Project Properties from the main menu



6. Under **Configuration Properties**, open the tree view.

7. Select C++, **General** in the tree view.

Figure 6.9. Setting properties


test3 Property Pages

Configuration: Active(Release)   Platform: Active(Win32)   Configuration Manager...

- Common Properties
  - Framework and References
- Configuration Properties
  - General
  - Debugging
  - C/C++
    - General
    - Optimization
    - Preprocessor
    - Code Generation
    - Language
    - Precompiled Headers
    - Output Files
    - Browse Information
    - Advanced
    - Command Line
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step

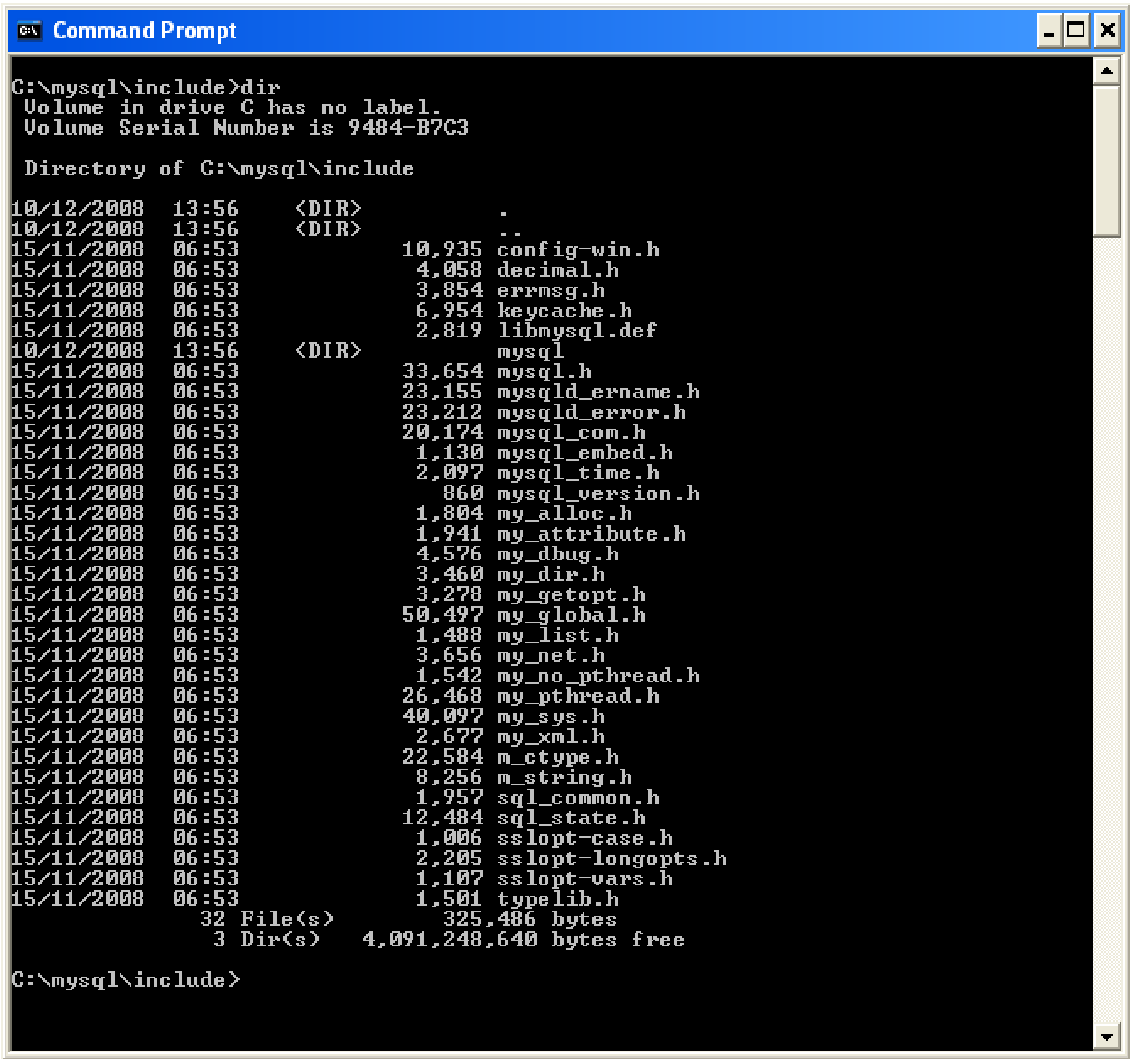
Additional Include Directories	
Resolve #using References	
Debug Information Format	**Program Database (/Zi)**
Suppress Startup Banner	Yes (/nologo)
Warning Level	**Level 3 (/W3)**
Detect 64-bit Portability Issues	No
Treat Warnings As Errors	No
Use UNICODE Response Files	Yes

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

OK   Cancel   Apply

8. You now need to ensure that Visual Studio can find the MySQL include directory. This directory includes header files that can optionally be installed when installing MySQL Server.

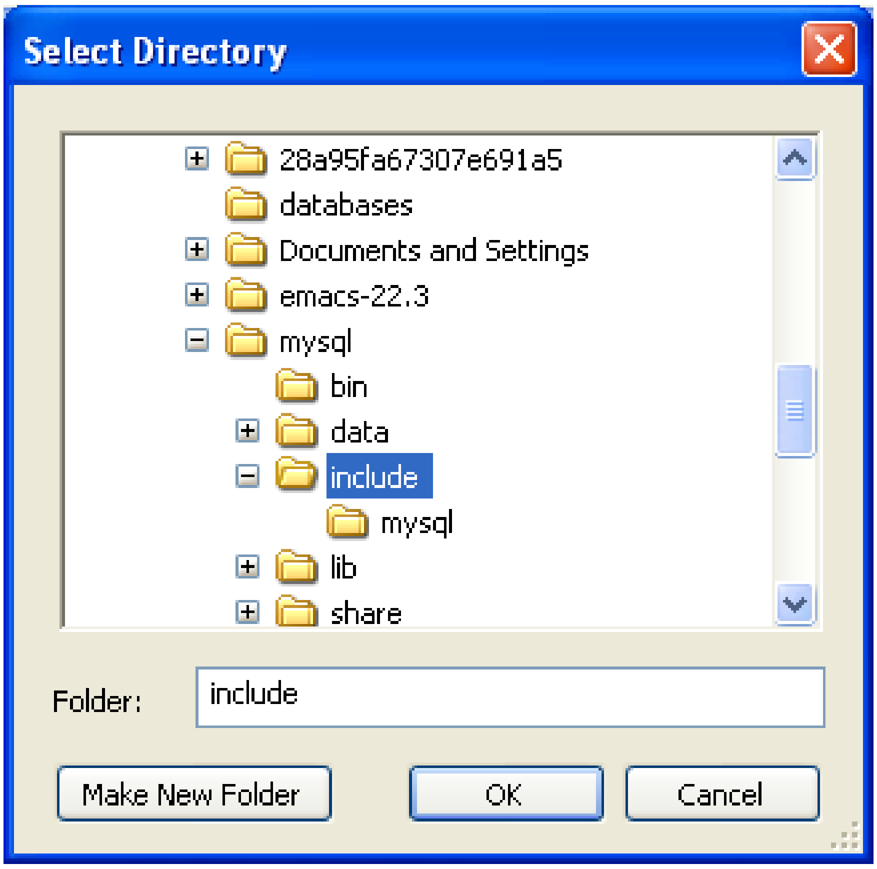
Figure 6.10. MySQL include directory



```
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_ername.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_dbug.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 m_ctype.h
15/11/2008  06:53             8,256 m_string.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>
```

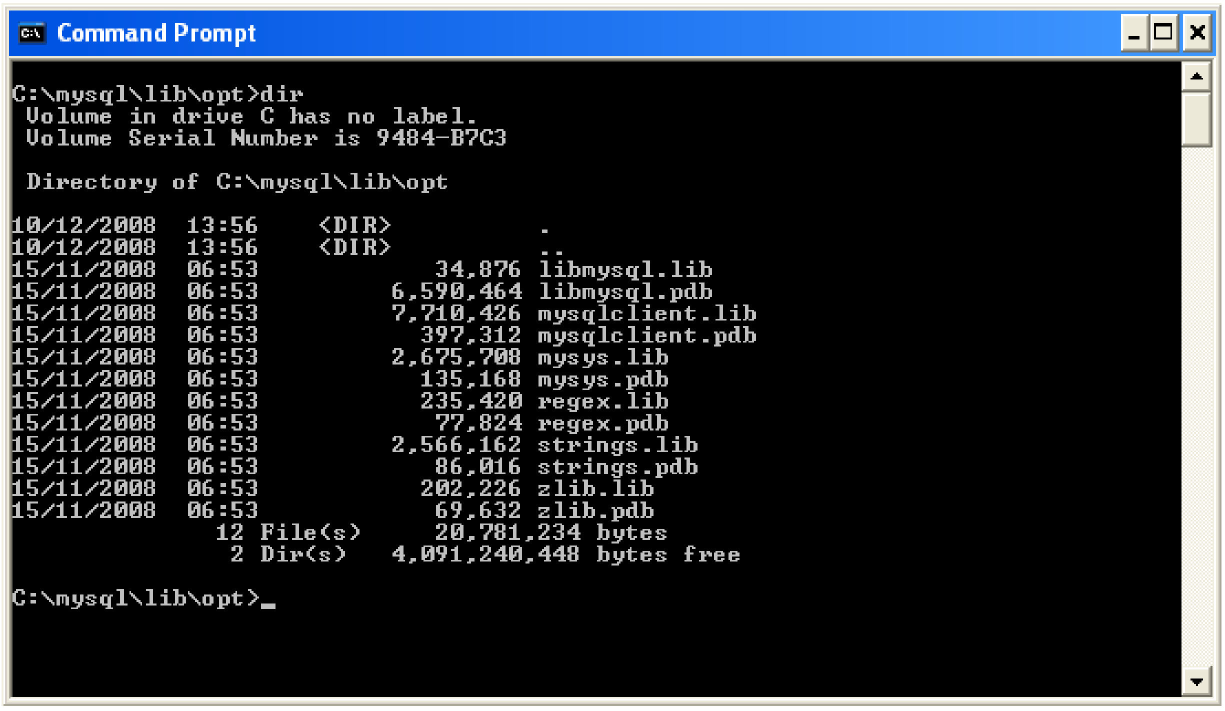
9. Then in the **Additional Include Directories** text field, add the MySQL `include/` directory.

Figure 6.11. Select Directory dialog



10. You will also need to set the location of additional libraries that Visual Studio will need in order to build the application. These are located in the MySQL `lib/opt` directory, a sub-directory of the MySQL Server installation directory.

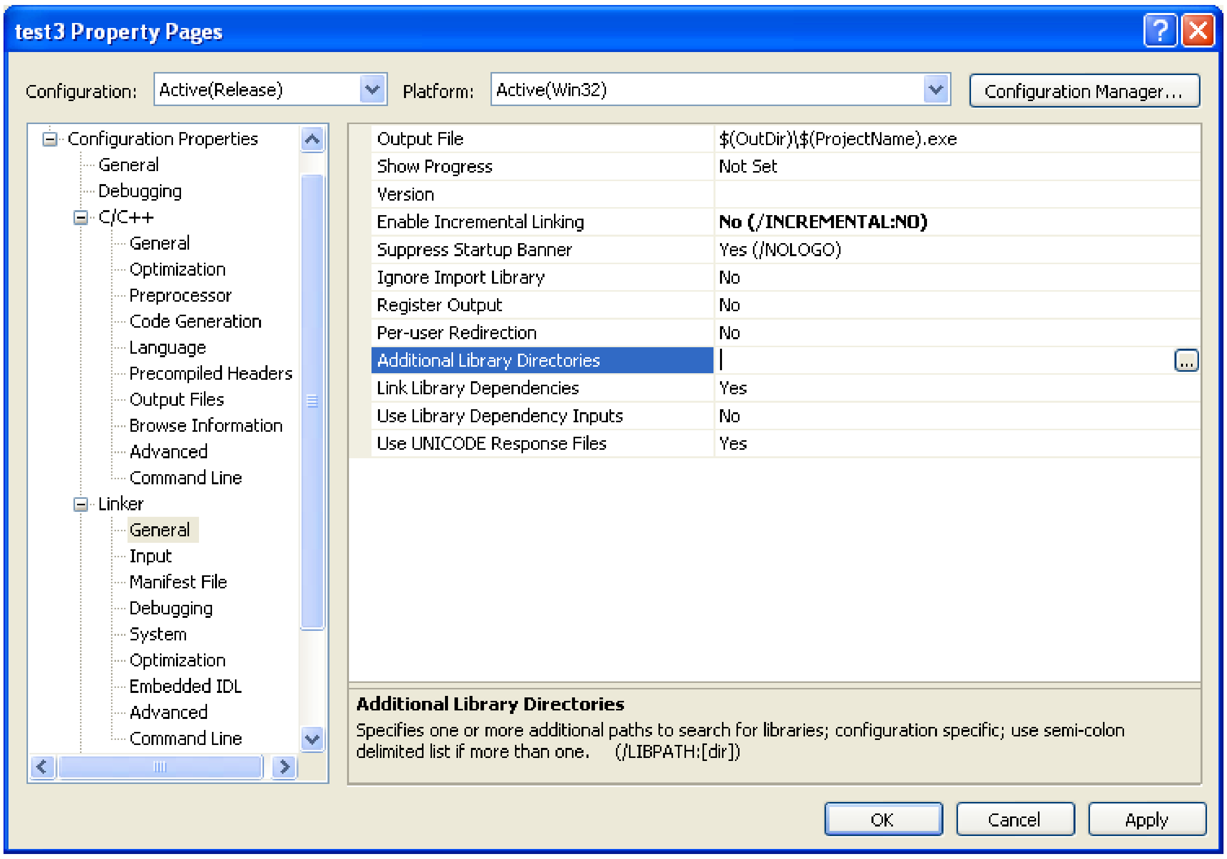
Figure 6.12. Typical contents of MySQL lib/opt directory



```
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 6.13. Additional Library Directories



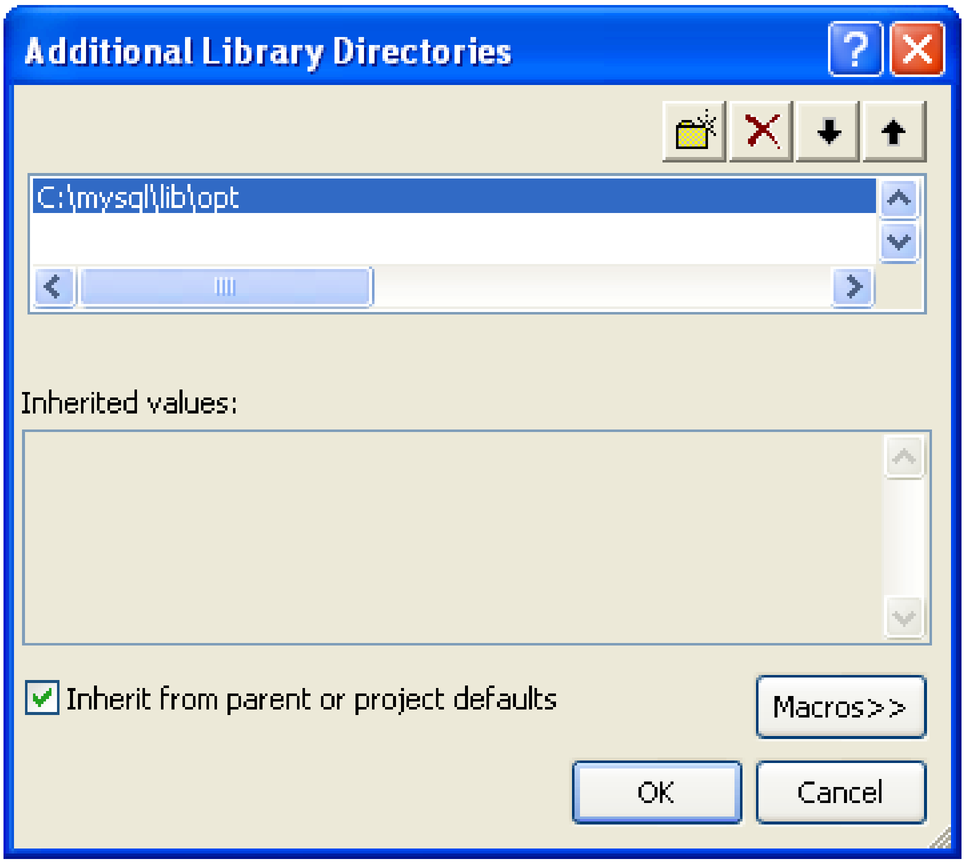
Configuration: Active(Release) Platform: Active(Win32)

Output File	$(OutDir)\$(ProjectName).exe
Show Progress	Not Set
Version	
Enable Incremental Linking	**No (/INCREMENTAL:NO)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Use UNICODE Response Files	Yes

Additional Library Directories
Specifies one or more additional paths to search for libraries; configuration specific; use semi-colon delimited list if more than one.   (/LIBPATH:[dir])

12. Add the `lib/opt` directory into the **Additional Library Directories** text field. This allows the library file `libmysql.lib` to be found.

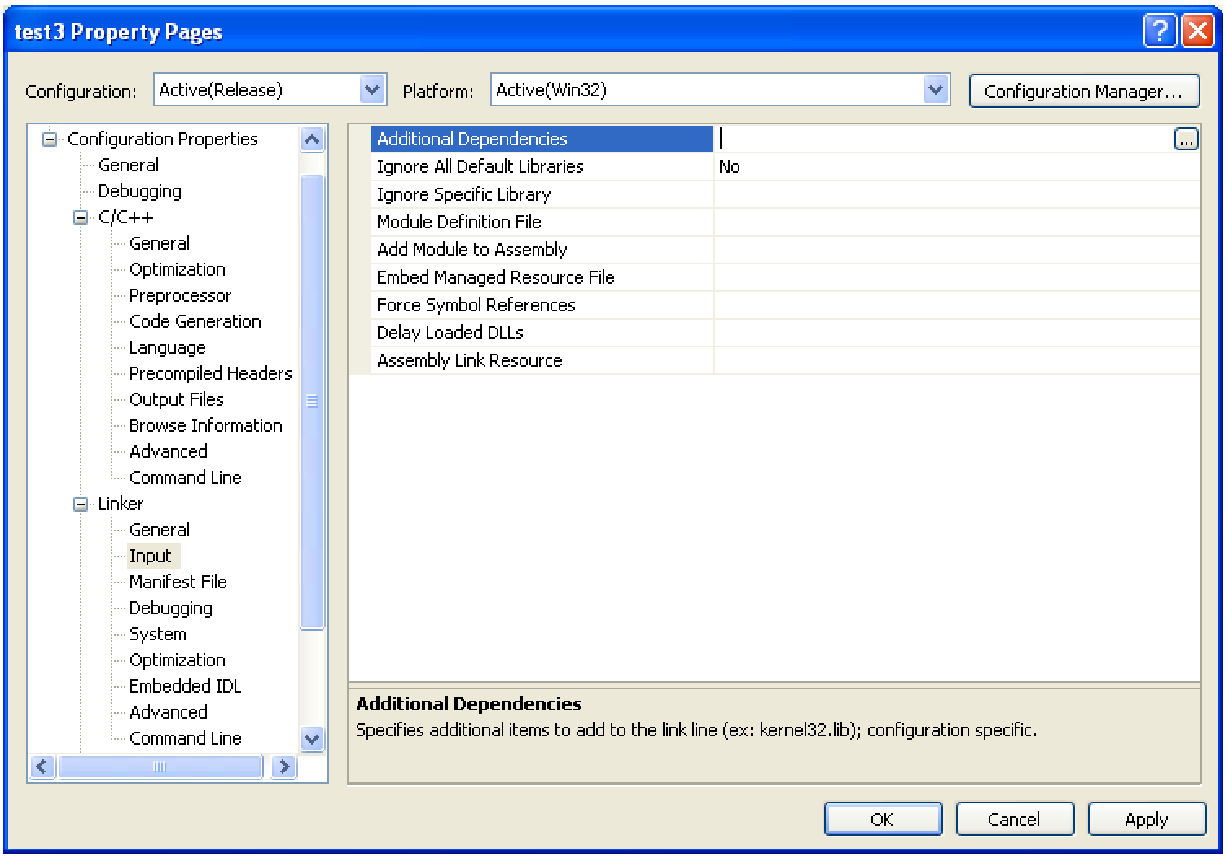
Figure 6.14. Additional Library Directories dialog


Additional Library Directories
C:\mysql\lib\opt
Inherited values:
Inherit from parent or project defaults
Macros>>
OK
Cancel

The remaining steps depend on whether you are building an application to use the MySQL Connector/C++ static or dynamic library. If you are building your application to use the dynamic library go here. If you are building your application to use the static library, carry out the following steps:

1. Then open **LINKER**, **INPUT**, **ADDITIONAL DEPENDENCIES**.

   Figure 6.15.

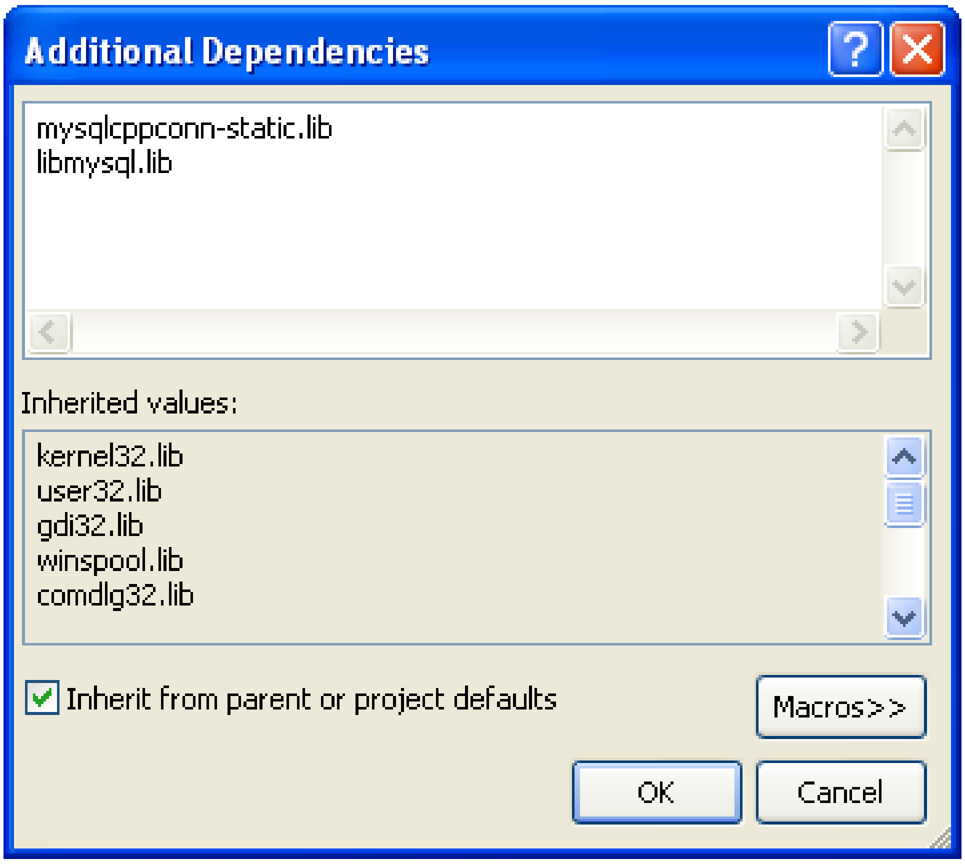


Additional Dependencies	
Ignore All Default Libraries	No
Ignore Specific Library	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

2. Enter `mysqlcppconn-static.lib` and `libmysql.lib`.

Figure 6.16. Adding additional dependencies



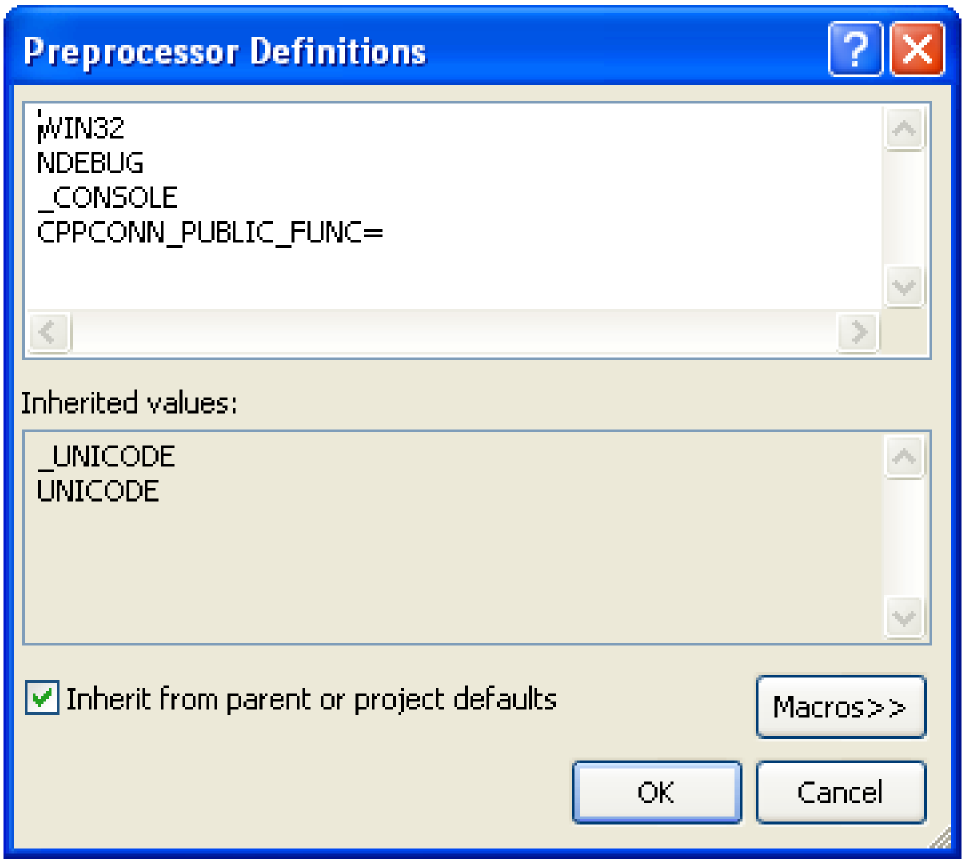
```
mysqlcppconn-static.lib
libmysql.lib
```

```
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
```

3. By default `CPPCONN_PUBLIC_FUNC` is defined to declare functions to be compatible with an application that calls a DLL. If building an application to call the static library, you must ensure that function prototypes are compatible with this. In this case `CPPCONN_PUBLIC_FUNC` needs to be defined 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 6.17. Setting the CPPCONN_PUBLIC_FUNC define


Preprocessor Definitions

```
WIN32
NDEBUG
_CONSOLE
CPPCONN_PUBLIC_FUNC=
```

Inherited values:

```
_UNICODE
UNICODE
```

☑ Inherit from parent or project defaults

> **Note**
>
> Make sure you enter `CPPCONN_PUBLIC_FUNC=` and not `CPPCONN_PUBLIC_FUNC`, as it needs to be defined as an empty string.

If building an application to use the MySQL Connector/C++ dynamically linked library carry out these steps:

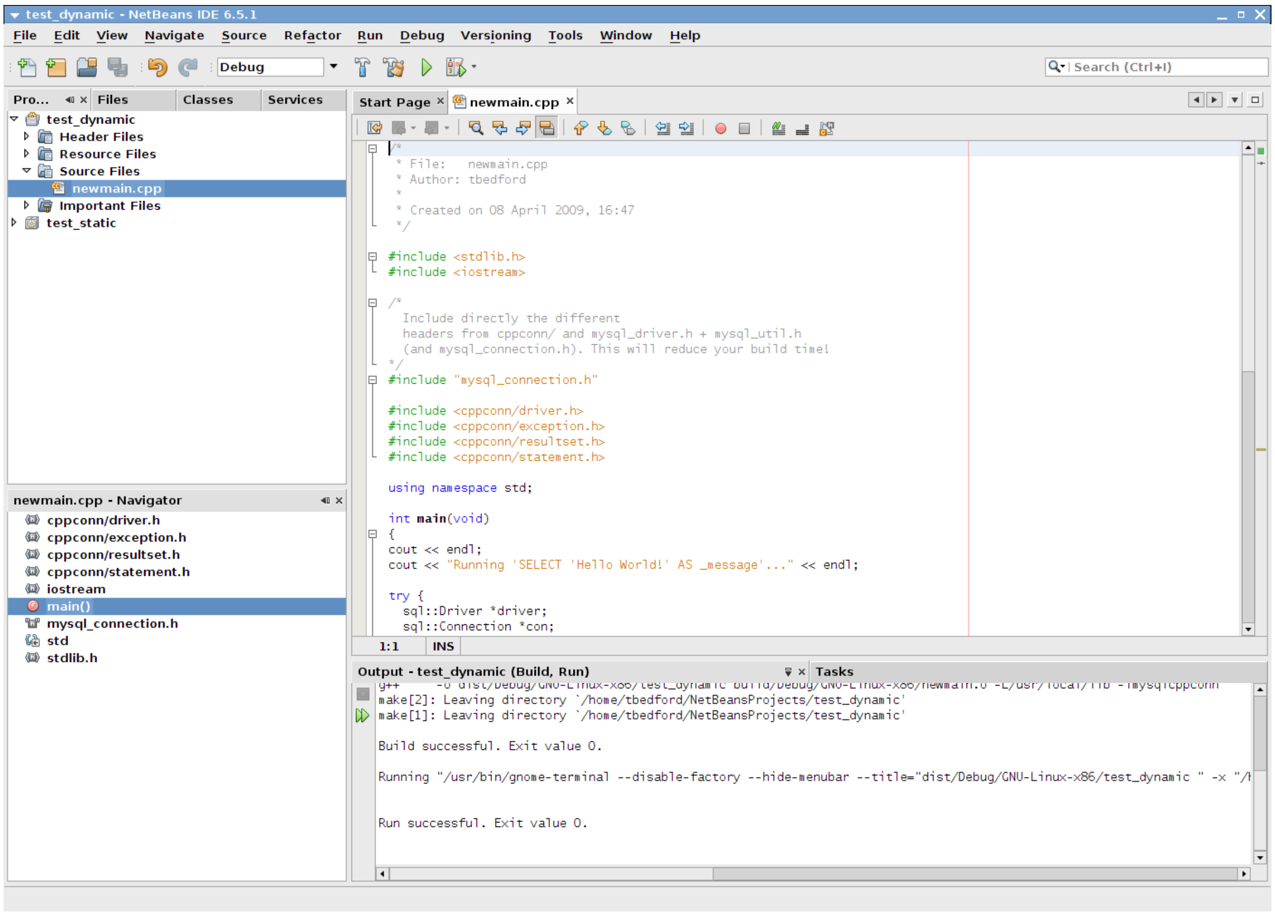
1. Under **Linker**, **Input**, add `mysqlcppconn.lib` into the **Additional Dependencies** text field.
2. The application will need to access the MySQL Connector/C++ Dynamic Linked Library at run time. Therefore, `mysqlcppconn.dll` needs to be in the same directory as the application executable, or somewhere on the system's path.

   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`.

6.4. MySQL Connector/C++ Building Linux applications with NetBeans

This section describes how you can build MySQL Connector/C++ applications for Linux using the NetBeans IDE.

Figure 6.18. The NetBeans IDE



```
/*
 * File:   newmain.cpp
 * Author: tbedford
 *
 * Created on 08 April 2009, 16:47
 */

#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;
```

```
g++     -o dist/Debug/GNU-Linux-x86/test_dynamic build/Debug/GNU-Linux-x86/newmain.o -L/usr/local/lib -lmysqlcppconn
make[2]: Leaving directory `/home/tbedford/NetBeansProjects/test_dynamic'
make[1]: Leaving directory `/home/tbedford/NetBeansProjects/test_dynamic'

Build successful. Exit value 0.

Running "/usr/bin/gnome-terminal --disable-factory --hide-menubar --title="dist/Debug/GNU-Linux-x86/test_dynamic " -x "/

Run successful. Exit value 0.
```

Note

To avoid potential crashes the build configuration of MySQL Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of MySQL Connector/C++ with a debug build of the client application.

1. The first step of building your application is to 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 on **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. You now need to add some working code to you main source file. Explore your MySQL 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 MySQL Connector/C++.

7. You will notice that at this point NetBeans is showing some errors in the source code. This is because you need to direct NetBeans to the necessary header files that need to be included. Select FILE, PROJECT PROPERTIES from the main menu.

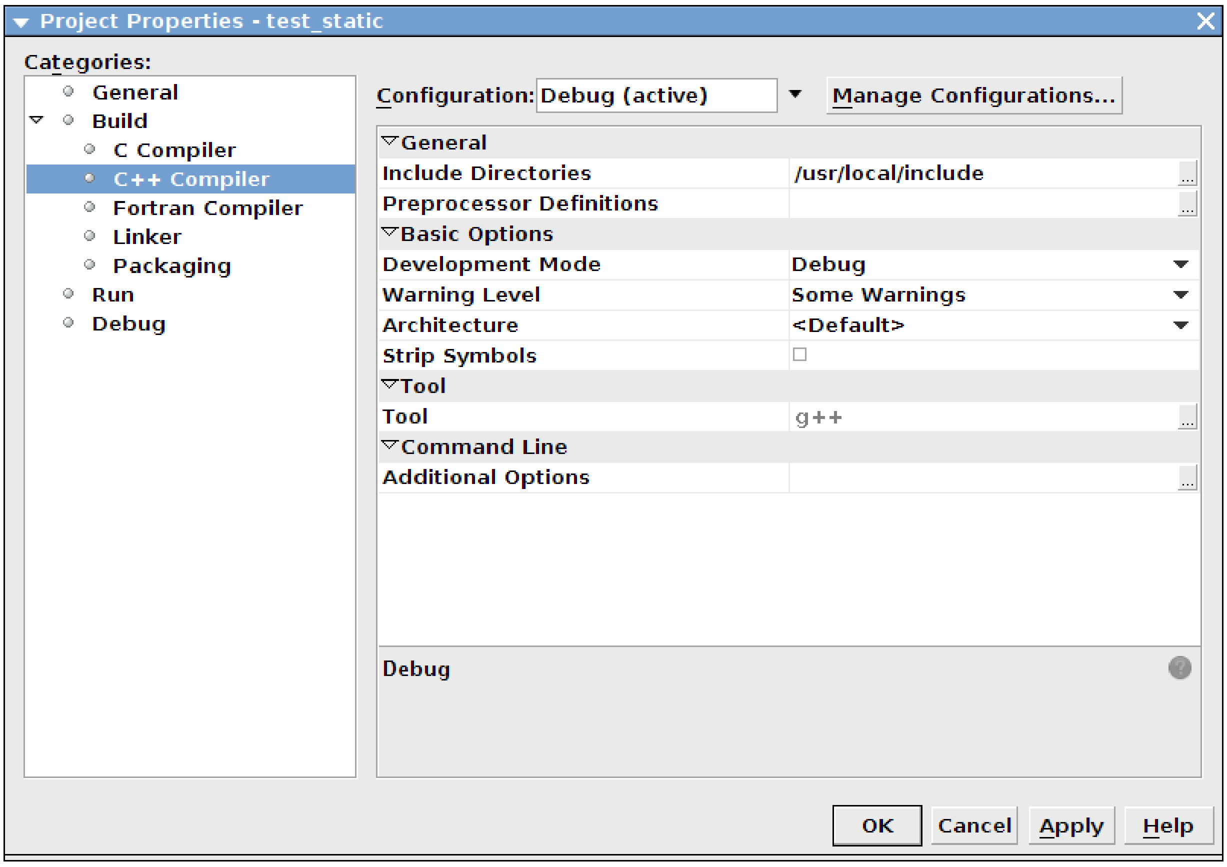
8. In the **CATEGORIES:** tree view panel, navigate to **BUILD**, **C++ COMPILER**.

9. In the **GENERAL** panel, select **INCLUDE DIRECTORIES**.

10. Click the ... button.

11. Click ADD and then navigate to the directory where the MySQL Connector/C++ header files are located. This will be `/usr/local/include` unless you have installed the files to a different location. Click SELECT. Click OK.

Figure 6.19. Setting the header include directory



Configuration: Debug (active)

General	
Include Directories	/usr/local/include
Preprocessor Definitions	
Basic Options	
Development Mode	Debug
Warning Level	Some Warnings
Architecture	<Default>
Strip Symbols	☐
Tool	
Tool	g++
Command Line	
Additional Options	

Debug

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, you need to decide whether your project is to use the MySQL Connector/C++ static or dynamic library. The project settings are slightly different in each case, as you need to link against a different library.

Using the static library

If you intend to use the static library you will need to link against two library files, `libmysqlcppconn-static.a` and `libmysqlclient.a`. The locations of the files will depend on your setup, but typically the former will be found in `/usr/local/lib` and the latter in `/usr/lib`. Note the file `libmysqlclient.a` is not part of MySQL 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. Note the MySQL Client Library is also available as part of the new MySQL Connector/C distribution.

1. The first step is to set the project to link the necessary library files. Select FILE, PROJECT PROPERTIES from the main menu.

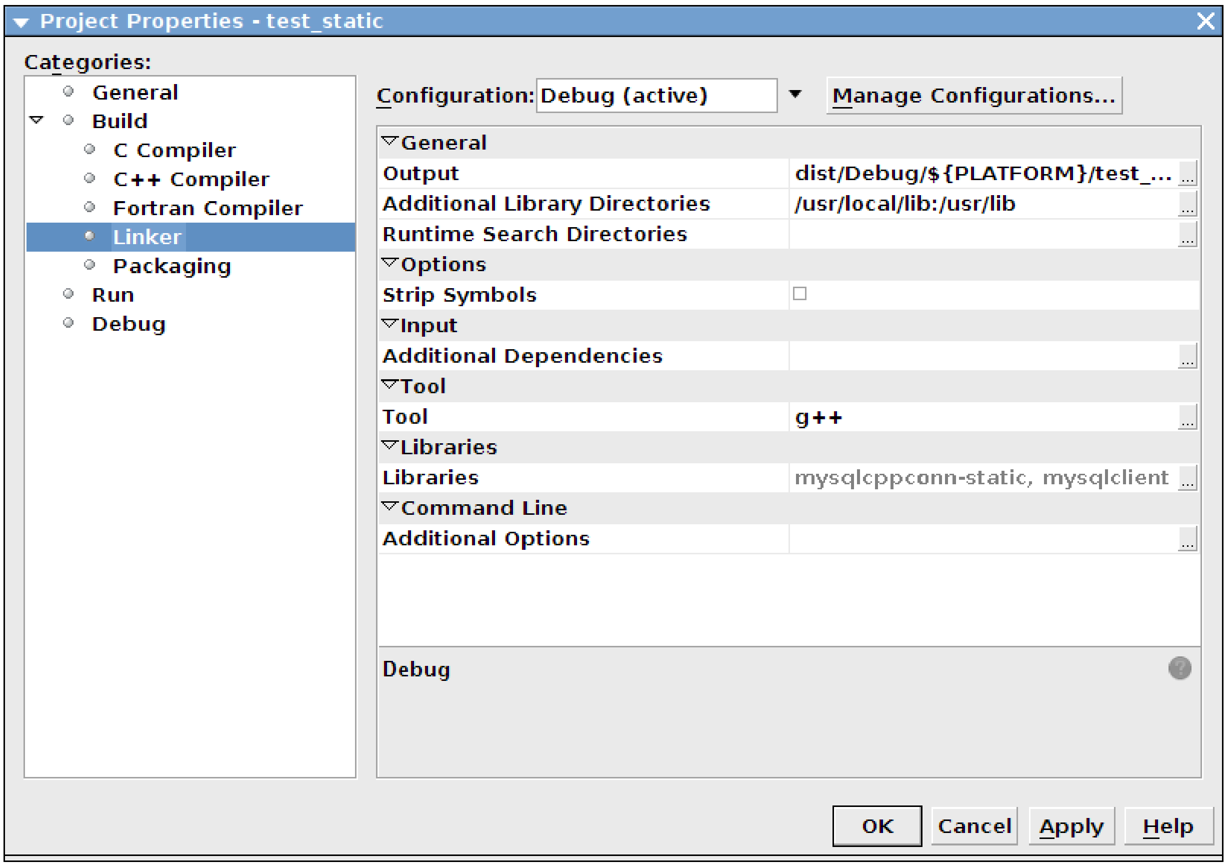
2. In the **CATEGORES:** tree view navigate to **LINKER**.

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 6.20. Setting the static library directories and file names


Project Properties - test_static

Categories:
- General
- Build
  - C Compiler
  - C++ Compiler
  - Fortran Compiler
  - Linker
  - Packaging
- Run
- Debug

Configuration: Debug (active) | Manage Configurations...

General	
Output	dist/Debug/${PLATFORM}/test_...
Additional Library Directories	/usr/local/lib:/usr/lib
Runtime Search Directories	
Options	
Strip Symbols	☐
Input	
Additional Dependencies	
Tool	
Tool	g++
Libraries	
Libraries	mysqlcppconn-static, mysqlclient
Command Line	
Additional Options	

Debug

OK | Cancel | Apply | Help

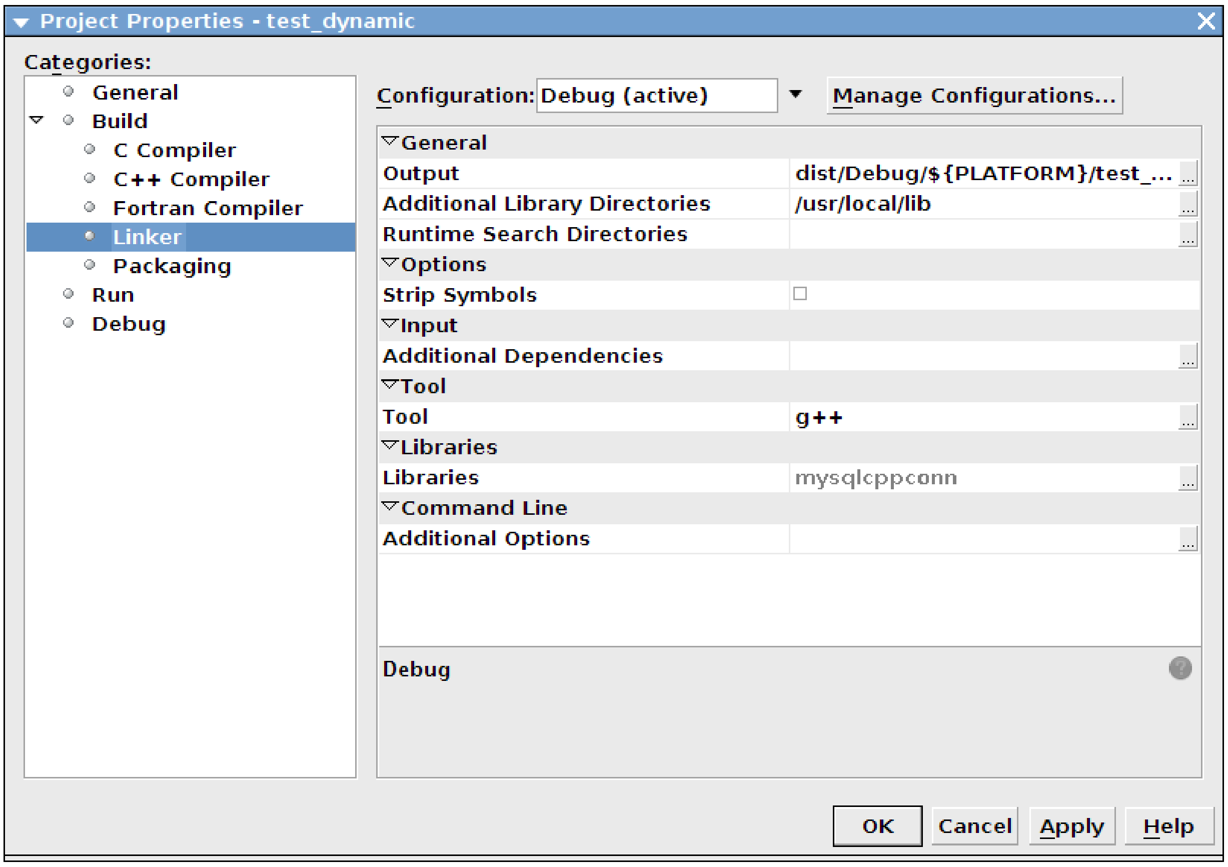
6. Click OK to close the **PROJECT PROPERTIES** dialog.

Using the dynamic library

If you require your application to use the MySQL Connector/C++ dynamic library, you simply need to link your project with a single library file, `libmysqlcppconn.so`. The location of this file will depend on how you configured your installation of MySQL Connector/C++, but will typically be `/usr/local/lib`.

1. The first step is to 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 then look similar to the following screenshot:

Figure 6.21. Setting the dynamic library directory and file name



Project Properties - test_dynamic

Categories:
- General
- Build
  - C Compiler
  - C++ Compiler
  - Fortran Compiler
  - Linker
  - Packaging
- Run
- Debug

Configuration: Debug (active)   Manage Configurations...

General	
Output	dist/Debug/${PLATFORM}/test_...
Additional Library Directories	/usr/local/lib
Runtime Search Directories	
Options	
Strip Symbols	☐
Input	
Additional Dependencies	
Tool	
Tool	g++
Libraries	
Libraries	mysqlcppconn
Command Line	
Additional Options	

Debug

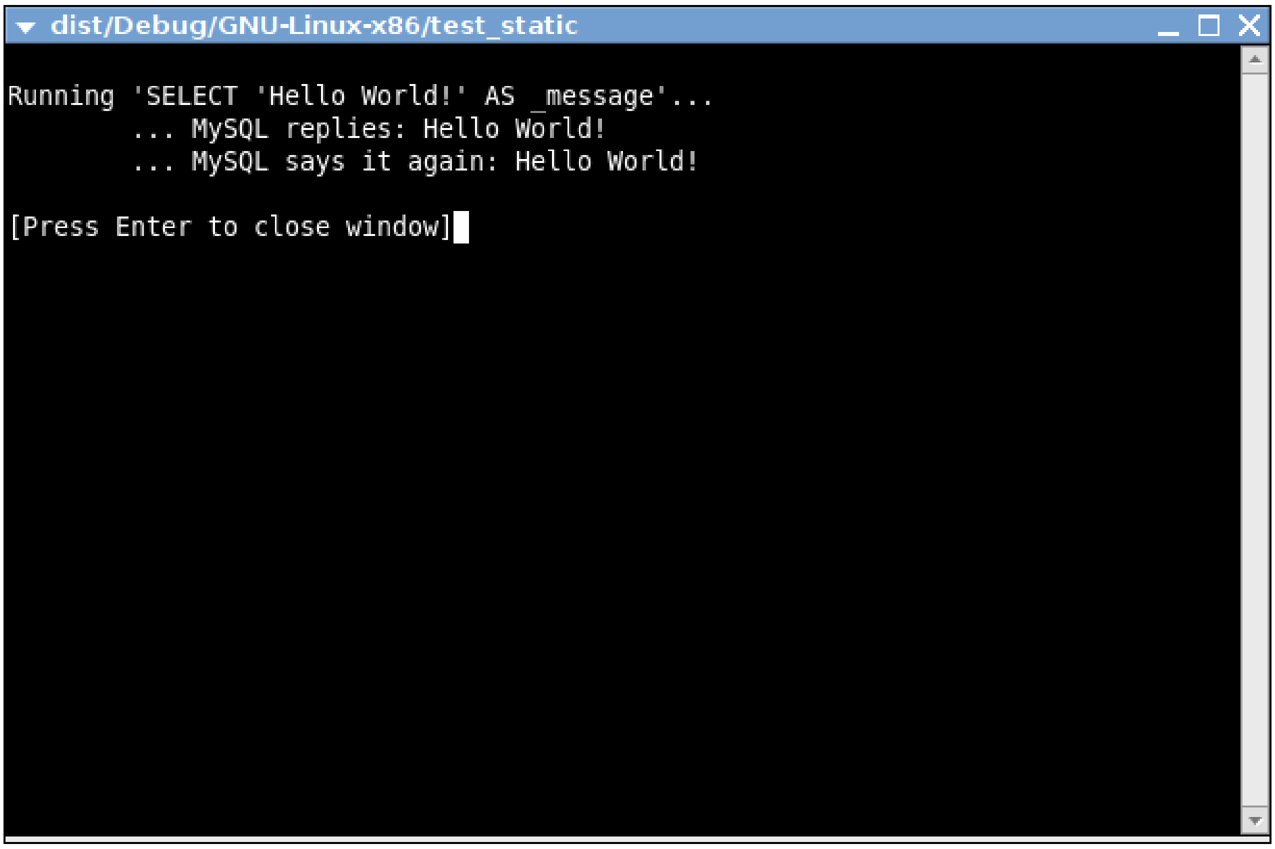
OK   Cancel   Apply   Help

6. Click OK to close the Project Properties dialog.

Having configured your project you can build it by selecting RUN, BUILD MAIN PROJECT from the main menu. You can 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 6.22. The example application running



```
Running 'SELECT 'Hello World!' AS _message'...
        ... MySQL replies: Hello World!
        ... MySQL says it again: Hello World!

[Press Enter to close window]
```

Note

Note the above settings and procedures were carried out for the default `Debug` configuration. If you want to create a `Release` configuration you will need to select that configuration before setting the Project Properties.

6.5. MySQL Connector/C++ Getting Started: Usage Examples

Caution

Please note, official support is not available for the beta version of MySQL Connector/C++.

The download package contains usage examples in the directory `examples/`. The examples explain the basic usage of 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 meta data

The examples in this document are only code snippets. The code snippets provide a brief overview on the API. They are not complete programs. Please check the `examples/` directory of your MySQL Connector/C++ installation for complete programs. Please also read the `README` file in the `examples/` directory. Note to test the example code you will first need to edit the `examples.h` file in the `examples/` directory, to add your connection information. Then simply rebuild the code by issuing a `make` command.

The examples in the `examples/` directory include:

- `examples/connect.cpp`:

  How to create a connection, insert data into MySQL and handle exceptions.

- `examples/connection_meta_schemaobj.cpp`:

  How to obtain meta data associated with a connection object, for example, a list of tables, databases, MySQL version, connector version.

- `examples/debug.cpp`:

  How to activate and deactivate the MySQL Connector/C++ debug protocol.

- `examples/exceptions.cpp`:

  A closer look at the exceptions thrown by the connector and how to fetch error information.

- `examples/prepared_statements.cpp`:

  How to run Prepared Statements including an example how to handle SQL commands that cannot be prepared by the MySQL Server.

- `examples/resultset.cpp`:

  How to fetch data and iterate over the result set (cursor).

- `examples/resultset_meta.cpp`:

  How to obtain meta data associated with a result set, for example, number of columns and column types.

- `examples/resultset_types.cpp`:

  Result sets returned from meta data methods - this is more a test than much of an example.

- `examples/standalone_example.cpp`:

  Simple standalone program not integrated into regular CMake builds.

- `examples/statements.cpp`:

  How to run SQL commands without using Prepared Statements.

- `examples/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.

6.5.1. MySQL Connector/C++ Connecting to MySQL

A connection to MySQL is established by retrieving an instance of `sql::Connection` from a `sql::mysql::MySQL_Driver` object. A `sql::mysql::MySQL_Driver` object is returned by `sql::mysql::MySQL_Driver::get_mysql_driver_instance()`.

```
sql::mysql::MySQL_Driver *driver;
sql::Connection *con;
driver = sql::mysql::MySQL_Driver::get_mysql_driver_instance();
con = driver->connect("tcp://127.0.0.1:3306", "user", "password");
delete con;
```

Make sure that you free the `sql::Connection` object as soon as you do not need it any more. But do not explicitly free the connector object!

6.5.2. MySQL Connector/C++ Running a simple query

For running simple queries you can use the methods `sql::Statement::execute()`, `sql::Statement::executeQuery()` and `sql::Statement::executeUpdate()`. The method `sql::Statement::execute()` should be used 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 on this.

```
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 that you have to free `sql::Statement` and `sql::Connection` objects explicitly using delete.

6.5.3. MySQL Connector/C++ Fetching results

The API for fetching result sets is identical for (simple) statments and prepared statements. If your query returns one result set you should use `sql::Statement::executeQuery()` or `sql::PreparedStatement::executeQuery()` to run your query. Both methods return `sql::ResultSet` objects. The preview version does buffer 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 = " <&;t; res->getInt(0);
  // ... or column names for accessing results. »
    The latter is recommended.
  cout << ", label = '" << »
    res->getString("label") << "'" << endl;
}
delete res;
delete stmt;
delete con;
```

Note that you have to free `sql::Statement`, `sql::Connection` and `sql::ResultSet` objects explicitly using delete.

The usage of cursors is demonstrated in the examples contained in the download package.

6.5.4. MySQL Connector/C++ Using Prepared Statements

If you are not familiar with Prepared Statements on MySQL have an extra look at the source code comments and explanations in the file `examples/prepared_statement.cpp`.

`sql::PreparedStatement` is created by passing a 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;
```

As usual, you have to free `sql::PreparedStatement` and `sql::Connection` objects explicitly.

6.5.5. MySQL Connector/C++ Complete Example 1

The following code shows a complete example of how to use MySQL Connector/C++:

```
/* Copyright 2008 Sun Microsystems, Inc.
This program is free software; you can redistribute it and/or modify
it under only 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., 59 Temple Place, Suite 330, Boston, MA  02111-1307 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 fata 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;
}
```

6.5.6. MySQL Connector/C++ Complete Example 2

The following code shows a complete example of how to use MySQL Connector/C++:

```
/* Copyright 2008 Sun Microsystems, Inc.
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., 59 Temple Place, Suite 330, Boston, MA  02111-1307 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;
}
```

6.6. MySQL Connector/C++ Debug Tracing

Caution

Please note, official support is not available for the beta version of MySQL Connector/C++.

Although a debugger can be used to debug your application, you may find it beneficial to turn on 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. Unfortunately, DTrace is currently only available on OpenSolaris, Solaris, MacOS 10.5 and FreeBSD.

The MySQL Connector/C++ can write two trace files:

1. Trace file generated by the MySQL Client Library

2. Trace file generated internally by MySQL Connector/C++

The first trace file can be generated by the underlying MySQL Client Library (libmysql). To enable this trace the connector will call the C-API function `mysql_debug()` internally. Only debug versions of the MySQL Client Library are capable of writing a trace file. Therefore you need to compile MySQL Connector/C++ against a debug version of the library, if you want utilize this trace. The trace shows the internal function calls and the addresses of internal objects as you can see below:

```
>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 MySQL Connector/C++ internal trace. It is available with debug and non-debug builds of the connector as long as you have enabled the tracing module at compile time using `cmake -DMYSQLCPPCONN_TRACE_ENABLE:BOOL=1`. By default, the tracing functionality is not available and calls to trace functions are removed by the preprocessor.

Compiling the connector with tracing functionality enabled will cause two additional tracing function calls per each connector function call. You will need to 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 run time 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 IO 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 run time of your application is not impacted significantly.

```
|  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
[...]
```

The example from `examples/debug.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 `setClientOptions()` method of a connection object to activate and deactivate the generation of a trace. The MySQL Client Library trace is always written into a file, whereas the connector's protocol messages are printed to standard out.

```
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:O,client.trace");
/*
Tracing is available if you have compiled the driver using
cmake -DMYSQLCPPCONN_TRACE_ENABLE:BOOL=1
*/
con->setClientOption("client_trace", &on_off);
```

6.7. MySQL Connector/C++ References

Caution

Please note, official support is not available for the beta version of MySQL Connector/C++.

See the JDBC overview for information on JDBC 4.0. Please also check the `examples/` directory of the download package.

Notes on using the MySQL Connector/C++ API

- `DatabaseMetaData::supportsBatchUpdates()` returns `true` because MySQL supports batch updates in general. However, no API calls for batch updates are provided by the MySQL Connector/C++ API.

- Two non-JDBC methods have been introduced for fetching and setting unsigned integers: `getUInt64()` and `getUInt()`. These are available for `ResultSet` and `Prepared_Statement`:

  - `ResultSet::getUInt64()`
  - `ResultSet::getUInt()`
  - `Prepared_Statement::setUInt64()`
  - `Prepared_Statement::setUInt()`

  The corresponding `getLong()` and `setLong()` methods have been removed.

- The method `DatabaseMetaData::getColumns()` 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`: String which is “YES” if the column is an auto-increment column. Otherwise the string contains “NO”.

- MySQL Connector/C++ may return different metadata for the same column.

  When you have any column that accepts a charset and a collation in its specification and you specify a binary collation, such as:

  ```
   CHAR(250) CHARACTER SET 'latin1' COLLATE 'latin1_bin'
  ```

  The server sets the `BINARY` flag in the result set metadata of this column. The method `ResultSet-Metadata::getColumnTypeName()` uses the metadata and will report, due to the `BINARY` flag, that the column type name is `BINARY`. This is illustrated below:

  ```
  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::getColumns()` uses `INFORMATION_SCHEMA`. It will report the type name `CHAR` for the same column. Note, a different type code is also returned.

- The MySQL Connector/C++ class `sql::DataType` 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`, `SQL-NULL`.

  However, the following JDBC standard data types are *not* supported by MySQL Connector/C++: `ARRAY`, `BLOB`, `CLOB`, `DIS-TINCT`, `FLOAT`, `OTHER`, `REF`, `STRUCT`.

- When inserting or updating `BLOB` or `TEXT` columns, MySQL Connector/C++ developers are advised not to use `set-`

String(). Instead it is recommended that the dedicated API function setBlob() be used instead.

The use of setString() can cause a Packet too large error message. The error will occur 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 MySQL Connector/C++, as this could lead to security issues, such as extremely large memory allocation requests due to malevolently long strings.

However, if setBlob() is used, this problem does not arise. This is because setBlob() takes a streaming approach based on std::istream. When sending the data from the stream to MySQL Server, MySQL Connector/C++ will split the stream into chunks appropriate for MySQL Server and observe the max_allowed_packet setting currently being used.

> **Caution**
>
> When using setString() it is not possible to set max_packet_size to a value large enough for the string, prior to passing it to MySQL Connector/C++. The MySQL 6.0 documentation for max_packet_size states: “As of MySQL 6.0.9, the session value of this variable is read only. Before 6.0.9, setting the session value is allowed but has no effect.”

This difference with the JDBC specification ensures that MySQL Connector/C++ is not vulnerable to memory flooding attacks.

- In general MySQL 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 5.1. As a new product, MySQL Connector/C++ is primarily targeted at the MySQL Server GA version that was available on its release.

  The following methods will throw a sql::MethodNotImplemented exception when you connect to MySQL earlier than 5.1.0:

  - DatabaseMetadata::getCrossReference()
  - DatabaseMetadata::getExportedKeys()

- MySQL Connector/C++ includes a method Connection::getClientOption() which 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 connection option metadataUseInfoSchema controls whether to use the Information_Schemata for returning the meta data of SHOW commands. In the case of metadataUseInfoSchema the optionValue argument should be interpreted as a boolean upon return.

  In the case of both defaultStatementResultType and defaultPreparedStatementResultType, the optionValue argument should be interpreted as an integer upon return.

  The connection property can be either set 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 are given below:

```
int defaultStmtResType;
int defaultPStmtResType;
conn->getClientOption("defaultStatementResultType", (void *) &defaultStmtResType);
conn->getClientOption("defaultPreparedStatementResultType", (void *) &defaultPStmtResType);
bool isInfoSchemaUsed;
conn->getClientOption("metadataUseInfoSchema", (void *) &isInfoSchemaUsed);
```

- MySQL Connector/C++ also supports the following methods 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)
```

Note that both methods are members of the `MySQL_Connection` class. The methods get and set MySQL session variables.

`setSessionVariable()` is equivalent to executing:

```
SET SESSION <varname> = <value>
```

`getSessionVariable()` is equivalent to executing the following and fetching the first return value:

```
SHOW SESSION VARIABLES LIKE "<varname>"
```

You can use “%” and other placeholders in <varname>, if the underlying MySQL server supports this.

6.8. MySQL Connector/C++ Known Bugs and Issues

Caution

Please note, official support is not available for the beta version of MySQL Connector/C++.

Note

Please report bugs through MySQL Bug System .

Known bugs:

None.

Known issues:

- When linking against a static library for 1.0.3 on Windows you need to 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, but does not have to, cause problems. If you obtain error messages, that you suspect are related to binary incompatibilities, build MySQL Connector/C++ from source, using the same compiler and linker that you will 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 MySQL 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 MySQL Connector/C++ should match the build configuration of the application using it. For example, do not use the release build of MySQL Connector/C++ with a debug build of the client application.

See also the MySQL Connector/C++ Changelogs which can be found here Appendix G, *MySQL Connector/C++ Change History*.

6.9. MySQL Connector/C++ Feature requests

Caution

Please note, official support is not available for the beta version of MySQL Connector/C++.

You can suggest new features in the first instance by joining the mailing list or forum and talking with the developers directly. See Section 6.10, “MySQL Connector/C++ Support”

The following feature requests are currently being worked on:

- C++ references for `Statements`, `ResultSets`, and exceptions, are being considered, instead of pointers to heap memory. This reduces the exception handling burden for the programmer.
- Adopt STL (suggestions are welcome).
- JDBC compliance: datatype interfaces and support through `ResultSet:getType()` and `PreparedStatement:bind()`. Introduce `sql::Blob`, `sql::Clob`, `sql::Date`, `sql::Time`, `sql::Timestamp`, `sql::URL`. Support `get|setBlob()`, `get|setClob()`, `get|setDate()`, `get|setTime()`, `get|setTimestamp()`, `get|setURL()`
- Add support for all C-API connection options. Improved support for `mysql_options`.
- Add connect method which supports passing options using HashMaps.
- Create Windows installer.

6.10. MySQL Connector/C++ Support

Caution

Please note, official support is not available for the beta version of MySQL Connector/C++.

For general discussion of the MySQL Connector/C++ please use the C/C++ community forum or join the MySQL Connector/C++ mailing list.

Bugs can be reported at the MySQL bug website.

For Licensing questions, and to purchase MySQL Products and Services, please Contact MySQL

The MySQL Connector/C++ Changelogs can be found here Appendix G, *MySQL Connector/C++ Change History*

6.11. MySQL Connector/C++ FAQ

Questions

- 6.11.1: What is MySQL Connector/C++?
- 6.11.2: Which MySQL Server version(s) is MySQL Connector/C++ compatible with?
- 6.11.3: Does MySQL Connector/C++ implement the client-server protocol?
- 6.11.4: Are any MySQL products using MySQL Connector/C++?

Questions and Answers

6.11.1: What is MySQL Connector/C++?

MySQL Connector/C++ is a MySQL database connector for C++. It allows you develop applications in C++ that connect to the MySQL Server. MySQL Connector/C++ is compatible with the JDBC 4.0 API.

6.11.2: Which MySQL Server version(s) is MySQL Connector/C++ compatible with?

MySQL Connector/C++ fully supports MySQL Server version 5.1 and later.

6.11.3: Does MySQL Connector/C++ implement the client-server protocol?

No. MySQL Connector/C++ uses the MySQL Client Library for the client-server communication.

6.11.4: Are any MySQL products using MySQL Connector/C++?

Yes, MySQL Workbench and MySQL Connector/OpenOffice.org.

Chapter 7. MySQL Connector/OpenOffice.org

MySQL Connector/OpenOffice.org is a native MySQL database connector for OpenOffice.org. Currently, it is in preview status and supports OpenOffice.org 2.4 only. It can be used to connect OpenOffice.org applications to a MySQL server.

Before MySQL Connector/OpenOffice.org became available you would have to use MySQL Connector/J (JDBC) or MySQL Connector/ODBC to connect to a MySQL server.

Connector/OpenOffice.org is a community project, although Sun Microsystems actively contributes code. The source code for Connector/OpenOffice.org is available under GPL with the FLOSS License Exception.

In the future a closed-source StarOffice version of Connector/OpenOffice.org will be made available.

Advantages

Using MySQL Connector/OpenOffice.org has the following advantages:

- Easy installation through the OpenOffice.org Extension Manager.
- Seamless integration into OpenOffice.org.
- No need to go through an additional Connector installation routine (ODBC/JDBC)
- No need to configure or register an additional Connector (ODBC)
- No need to install or configure a driver manager (ODBC)

Status

MySQL Connector/OpenOffice.org is available as a development preview version. We kindly ask users and developers to try it out and provide us with feedback. We do not encourage you to use it in production environments, though.

Note

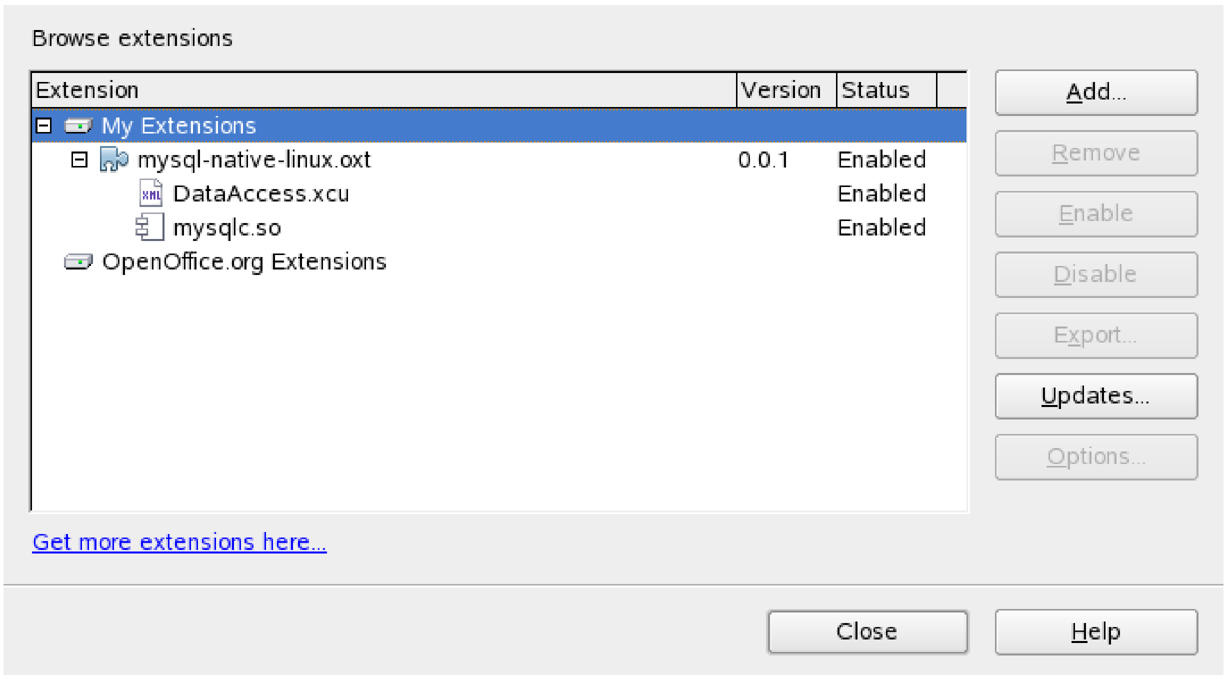
Sun Microsystems does not provide formal support for Connector/OpenOffice.org.

If you have any queries please contact us through our mailing list at `<users@dba.openoffice.org>`

7.1. Installation

1. Install or upgrade to OpenOffice.org 2.4.
2. Download MySQL Connector/OpenOffice.org from The OpenOffice.org download site. Save the file, `mysql-native-win32.oxt` or `mysql-native-linux.oxt`, respectively, to a location of your choice, for example `My Documents` or `~/Documents`.
3. Add the `.oxt` extension through the Extension Manager of OpenOffice.org. In OpenOffice.org, select Tools, Extension Manager... and specify the `.oxt` file as a new extension. When done, MySQL Connector/OpenOffice.org will show up as a new extension under **My Extensions**.

Figure 7.1. Adding an extension



Extension	Version	Status
My Extensions		
mysql-native-linux.oxt	0.0.1	Enabled
DataAccess.xcu		Enabled
mysqlc.so		Enabled
OpenOffice.org Extensions		

4. Restart OpenOffice.org.

7.2. Getting Started: Connecting to MySQL

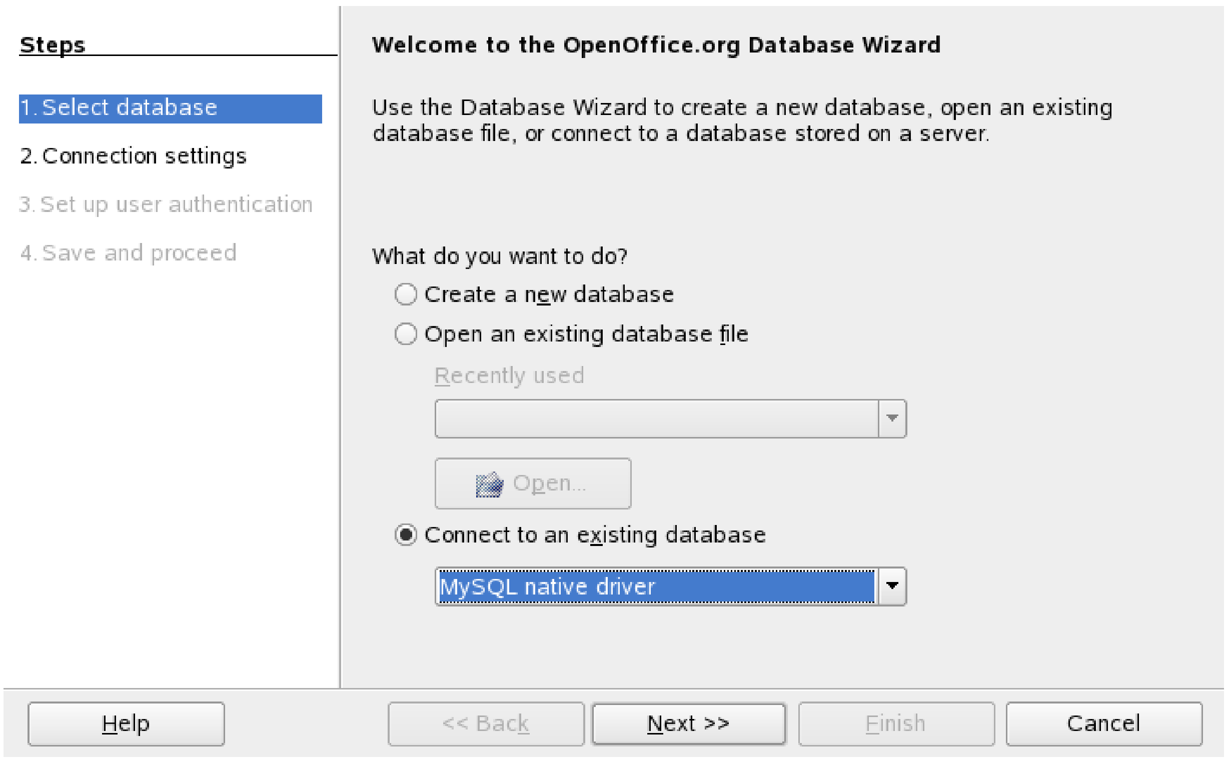
MySQL Connector/OpenOffice.org allows you to access the MySQL Server and its schemata from the OpenOffice.org suite. Currently the connector is in preview status, and only OpenOffice.org 2.4 is supported.

The following example demonstrates the creation of a new OpenOffice.org Base database which uses a local MySQL Server for storage and the new connector for connecting.

1. Select the database

   Create a new database by selecting FILE, NEW, DATABASE. This starts a wizard that allows you to create a new, open an existing, or connect to existing database. Select the latter option. From the drop-down list, select MySQL native driver. Click NEXT >>.

Figure 7.2. Selecting the database



Welcome to the OpenOffice.org Database Wizard

Use the Database Wizard to create a new database, open an existing database file, or connect to a database stored on a server.

What do you want to do?

( ) Create a new database

( ) Open an existing database file

Recently used

[✓ Connect to an existing database]

MySQL native driver

2. Fill in the connection settings

   Under **MYSQL NATIVE DRIVER**, fill in the host name, and optionally a database name, and port name, for example:

   ```
   localhost/test
   ```

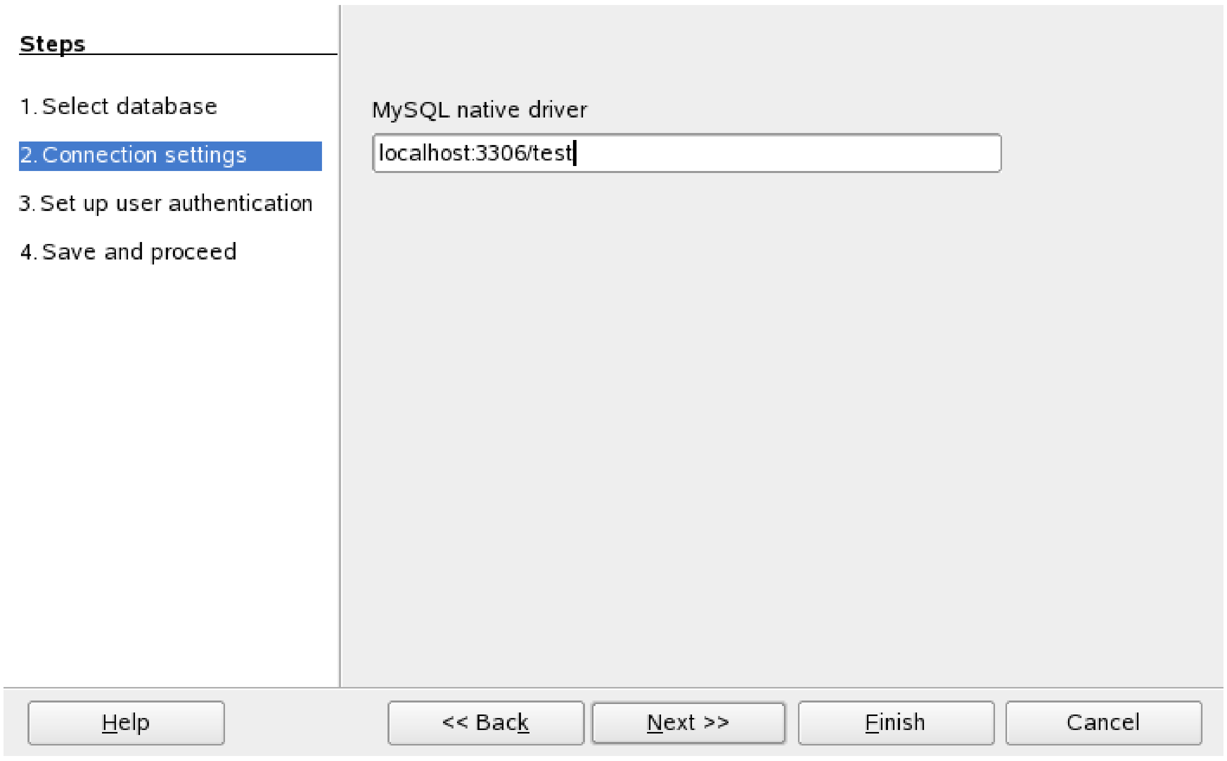
   This will connect to a MySQL server running on the local host and select the test database. Note that is you do not specify a database the process below will still work, and all databases will be available for selection.

   On Linux, you may have to specify an IP number and a port number instead, due to a limitation in Connector/OpenOffice.org. You have to do so if your MySQL socket file is not `/tmp/mysql.sock`. Enter your data using the format illustrated in the following example:

   ```
   127.0.0.1:3006/test
   ```

   This will connect to a MySQL server running on the local host, but do so via TCP/IP using 3306 as the port number. Click NEXT >>.

Figure 7.3. Entering connection settings



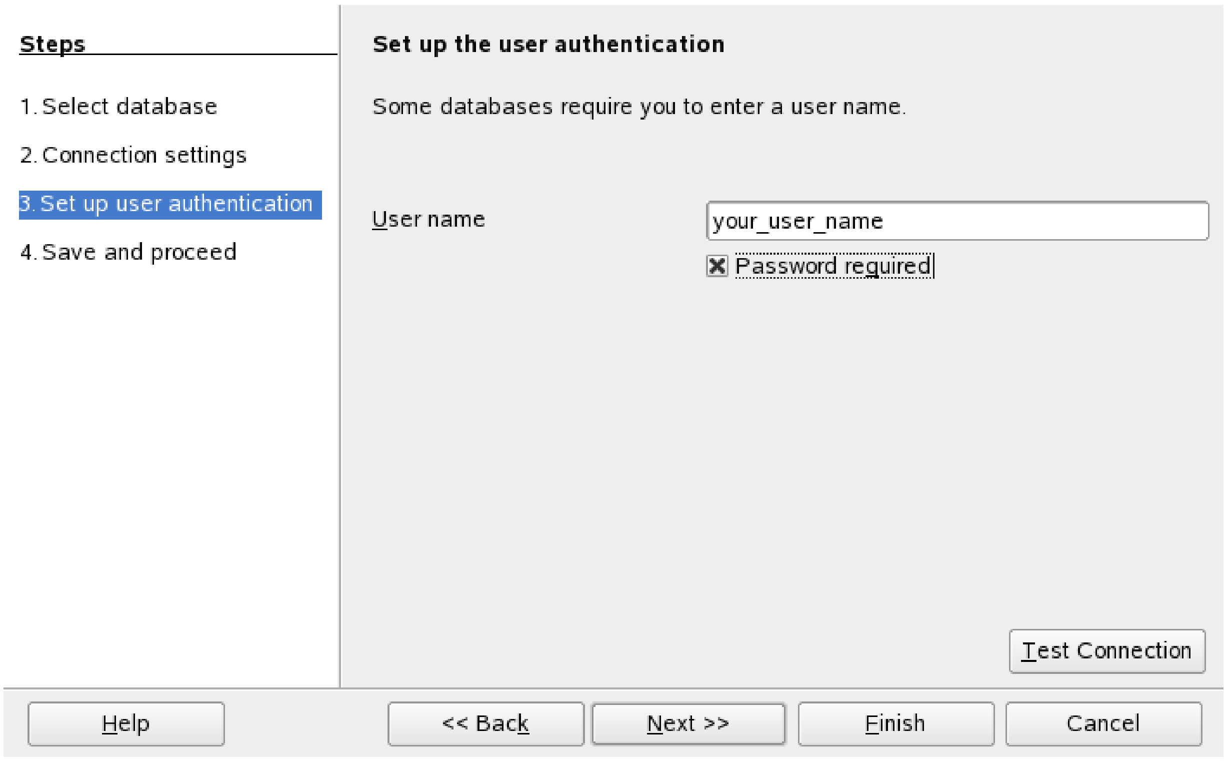
MySQL native driver

localhost:3306/test

3. Fill in credentials

   If you are using MySQL server's anonymous account without a password, you do not have to fill in anything in this step. Otherwise, fill in your MySQL user name and check the password checkbox. Note, for security reasons, you should not normally use the anonymous account without a password.

Figure 7.4. Setting up user authentication



Set up the user authentication

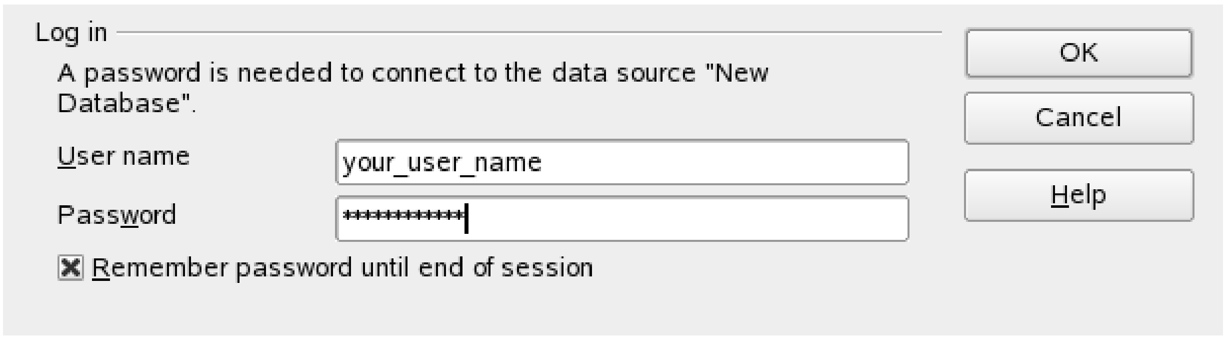
Some databases require you to enter a user name.

User name: your_user_name

[✓] Password required

You can now test your connection to the MySQL database server by clicking the TEST CONNECTION button. Check the check-box if you do not want OpenOffice.org to ask you for your password again in the current session. Testing the connection is op-tional, although recommended.

Figure 7.5. Entering user credentials



A password is needed to connect to the data source "New Database".

User name: your_user_name

Password: ************

☒ Remember password until end of session

Click NEXT >>.

4. Finish the wizard

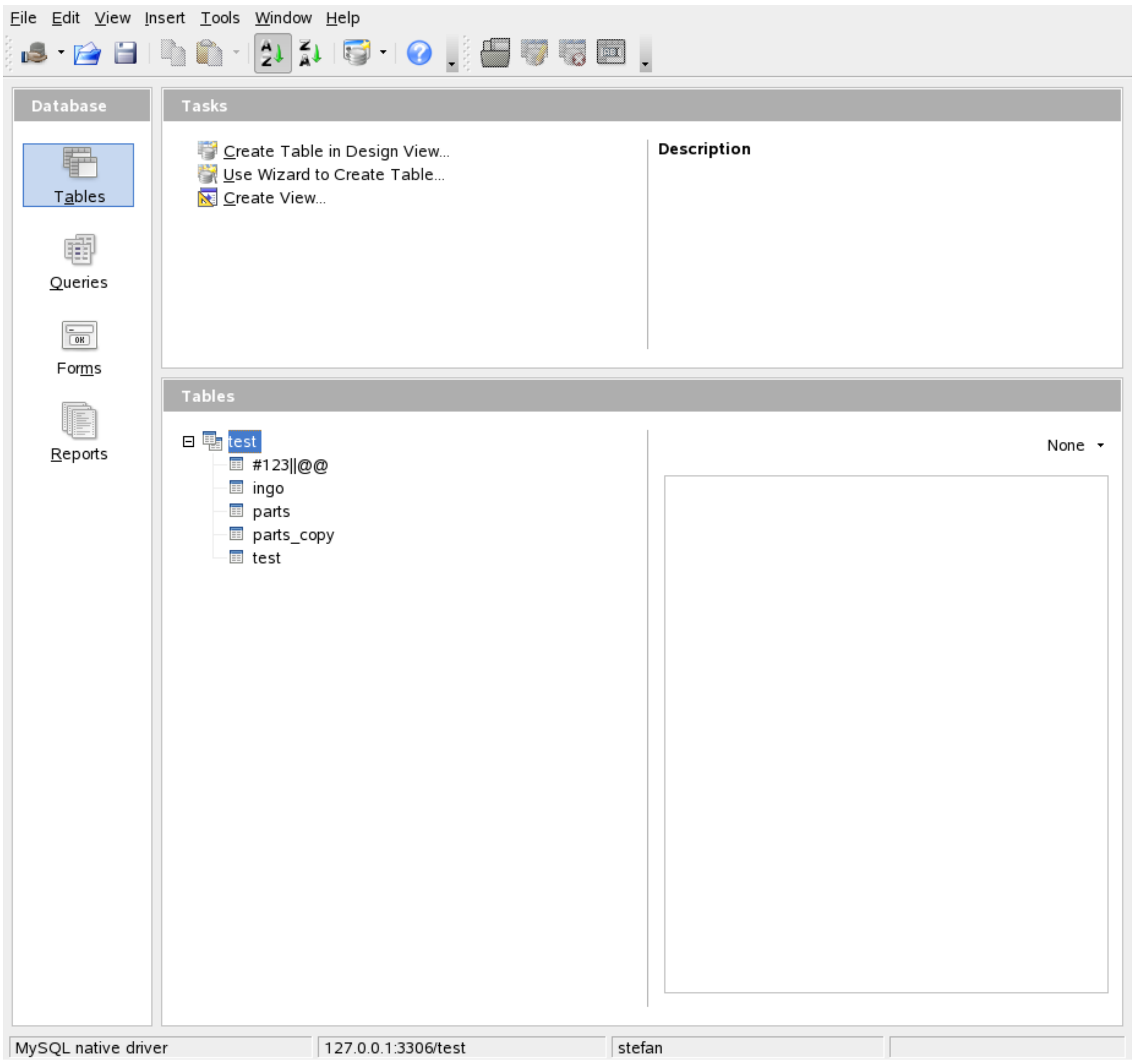
   Leave the default settings and click FINISH. You will be forwarded to the OpenOffice.org Base main window. Note that you can invoke the wizard again at any point by right-clicking in the **TABLES** section of the **BASE** main window and selecting DATABASE, CONNECTION TYPE.

7.3. Getting Started: Usage Examples

Listing Tables

In the **DATABASE** area of the **BASE** main window, select **TABLES**. If this is the first time you are accessing the database you will be prompted for your credentials (user name and password); you can store these settings for your current Base session.

Figure 7.6. Listing tables


File Edit View Insert Tools Window Help
Database
Tables
Queries
Forms
Reports
Tasks
Create Table in Design View...
Use Wizard to Create Table...
Create View...
Description
Tables
None

test
- #123||@@
- ingo
- parts
- parts_copy
- test

MySQL native driver | 127.0.0.1:3306/test | stefan

Depending on your connection settings you will now see all databases with all their tables, or just the database you have specified in the connection settings.

7.4. References

See the OpenOffice.org website for documentation of the office suite and its Extension Manager.

7.5. Known Bugs

If you discover a bug in Connector/OpenOffice.org please add it to this list and send an email to `<users@dba.openoffice.org>`. You need to be logged in with an OpenOffice.org account for both; see the project mailing list for details.

7.6. Contact

To discuss the new MySQL Connector/OpenOffice.org, please subscribe to the mailing list `<users@dba.openoffice.org>`. It is a low-volume list with less than 10 mails per day.

Chapter 8. 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 the Apache Web server.

PHP actually provides two different MySQL API extensions:

- `mysql`: 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. If you wish to use this extension with MySQL 4.1, you will likely want to configure the MySQL server to use the `--old-passwords` option (see `Client does not support authentication protocol`). This extension is documented on the PHP Web site at http://php.net/mysql.
- Section 8.2, "MySQL Improved Extension (`Mysqli`)" - Stands for "MySQL, Improved"; this extension is available only in PHP 5. 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. You can read the documentation for the `mysqli` extension at http://php.net/mysqli. Helpful article can be found at http://devzone.zend.com/node/view/id/686 and http://devzone.zend.com/node/view/id/687.

If you're experiencing problems with enabling both the `mysql` and the `mysqli` extension when building PHP on Linux yourself, see Section 8.6, "Enabling Both `mysql` and `mysqli` in PHP".

The PHP distribution and documentation are available from the PHP Web site.

MySQL Enterprise
MySQL Enterprise subscribers will find more information about MySQL and PHP in the Knowledge Base articles found at PHP. Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information, see http://www.mysql.com/products/enterprise/knowledgebase.html.

8.1. MySQL

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/.

8.1.1. Installing/Configuring

8.1.1.1. Requirements

In order to have these functions available, you must compile PHP with MySQL support.

8.1.1.2. Installation

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.

8.1.1.2.1. Installation on Linux Systems

8.1.1.2.1.1. PHP 4

The option `--with-mysql` is enabled by default. This default behavior may be disabled with the `--without-mysql` configure option. If MySQL is enabled without specifying the path to the MySQL install DIR, PHP will use the bundled MySQL client libraries.

Users who run other applications that use MySQL (for example, auth-mysql) should not use the bundled library, but rather specify the path to MySQL's install directory, like so: `--with-mysql=/path/to/mysql`. This will force PHP to use the client libraries installed by MySQL, thus avoiding any conflicts.

8.1.1.2.1.2. PHP 5+

MySQL is not enabled by default, nor is the MySQL library bundled with PHP. Read this FAQ for details on why. Use the `--with-mysql[=DIR]` configure option to include MySQL support. You can download *headers and libraries* from MySQL.

8.1.1.2.2. Installation on Windows Systems

8.1.1.2.2.1. PHP 4

The PHP MySQL extension is compiled into PHP.

8.1.1.2.2.2. PHP 5+

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.

8.1.1.2.3. MySQL Installation Notes

> **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) libmysql with compiled charset support.

8.1.1.3. Runtime Configuration

The behaviour of these functions is affected by settings in `php.ini`.

Table 8.1. MySQL Configuration Options

Name	Default	Changeable	Changelog
mysql.allow_persistent	"1"	PHP_INI_SYSTEM	
mysql.max_persistent	"-1"	PHP_INI_SYSTEM	
mysql.max_links	"-1"	PHP_INI_SYSTEM	
mysql.trace_mode	"0"	PHP_INI_ALL	Available since PHP 4.3.0.
mysql.default_port	NULL	PHP_INI_ALL	
mysql.default_socket	NULL	PHP_INI_ALL	Available since PHP 4.0.1.
mysql.default_host	NULL	PHP_INI_ALL	
mysql.default_user	NULL	PHP_INI_ALL	
mysql.default_password	NULL	PHP_INI_ALL	
mysql.connect_timeout	"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_* constants, see the ini.

Here's a short explanation of the configuration directives.

`mysql.allow_persistent` boolean	Whether to allow persistent connections to MySQL.
`mysql.max_persistent` integer	The maximum number of persistent MySQL connections per process.
`mysql.max_links` integer	The maximum number of MySQL connections per process, including persistent connections.
`mysql.trace_mode` boolean	Trace mode. When `mysql.trace_mode` is enabled, warnings for table/index scans, non free result sets, and SQL-Errors will be displayed. (Introduced in PHP 4.3.0)
`mysql.default_port` 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 `MYSQL_TCP_PORT` environment variable, the `mysql-tcp` entry in `/etc/services` or the compile-time `MYSQL_PORT` constant, in that order. Win32 will only use the `MYSQL_PORT` constant.
`mysql.default_socket` string	The default socket name to use when connecting to a local database server if no other socket name is specified.
`mysql.default_host` 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.
`mysql.default_user` 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.
`mysql.default_password` 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.
`mysql.connect_timeout` integer	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.

8.1.1.4. Resource Types

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.

8.1.2. Predefined Constants

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 8.2. 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 8.3. MySQL fetch constants

Constant	Description
MYSQL_ASSOC	Columns are returned into the array having the fieldname as the array index.
MYSQL_BOTH	Columns are returned into the array having both a numerical index and the fieldname as the array index.
MYSQL_NUM	Columns are returned into the array having a numerical index to the fields. This index starts with 0, the first field in the result.

8.1.3. Examples

8.1.3.1. Basic

This simple example shows how to connect, execute a query, print resulting rows and disconnect from a MySQL database.

Example 8.1. 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);
?>
```

8.1.4. MySQL Functions

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` .

8.1.4.1. `mysql_affected_rows`

- `mysql_affected_rows`

  Get number of affected rows in previous MySQL operation

Description

```
  int mysql_affected_rows(resource link_identifier);
```

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` was called with no arguments. If by chance 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.

Examples

Example 8.2. `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 8.3. `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`.

See Also

`mysql_num_rows`
`mysql_info`

8.1.4.2. `mysql_change_user`

- `mysql_change_user`

Change logged in user of the active connection

Description

```
int mysql_change_user(string user,
                      string password,
                      string database,
                      resource link_identifier);
```

`mysql_change_user` changes the logged in user of the current active connection, or the connection given by the optional *`link_identifier`* parameter. If a database is specified, this will be the current database after the user has been changed. If the new user and password authorization fails, the current connected user stays active.

This function is deprecated and no longer exists in PHP.

Parameters

`user`	The new MySQL username.
`password`	The new MySQL password.
`database`	The MySQL database. If not specified, the current selected database is used.
`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` was called with no arguments. If by chance no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

ChangeLog

Version	Description
3.0.14	This function was removed from PHP.

Notes

Requirements

This function requires MySQL 3.23.3 or higher.

See Also

`mysql_connect`
`mysql_select_db`
`mysql_query`

8.1.4.3. `mysql_client_encoding`

- `mysql_client_encoding`

  Returns the name of the character set

Description

```
string mysql_client_encoding(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.4. `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`

8.1.4.4. `mysql_close`

- `mysql_close`

  Close MySQL connection

Description

```
 bool mysql_close(resource link_identifier);
```

`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.

Using `mysql_close` isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution. See also 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 such link is found, it will try to create one as if mysql_connect was called with no arguments. If by chance 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 8.5. 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.

See Also

mysql_connect
mysql_free_result

8.1.4.5. mysql_connect

- mysql_connect

  Open a connection to a MySQL Server

Description

```
  resource mysql_connect(string server,
                         string username,
                         string password,
                         bool new_link,
                         int client_flags);
```

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. ":/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.
username	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.
password	The password. Default value is defined by mysql.default_password. In SQL safe mode, this parameter is ignored and empty password is used.
new_link	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 *new_link* 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.
client_flags	The *client_flags* 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 8.2, “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
4.3.0	Added the *client_flags* parameter.
4.2.0	Added the *new_link* parameter.
3.0.10	Added support for ":/path/to/socket" with *server*.
3.0.0	Added support for ":port" with *server*.

Examples

Example 8.6. `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 8.7. `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 8.8. `mysql_connect` example using ":/path/to/socket" syntax

```
<?php
// we connect to localhost and socket e.g. /tmp/mysql.sock

//variant 1: ommit 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 ini.mysql.default-host 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`

8.1.4.6. `mysql_create_db`

- `mysql_create_db`

  Create a MySQL database

Description

```
bool mysql_create_db(string database_name,
                     resource link_identifier);
```

`mysql_create_db` attempts to create a new database on the server associated with the specified link identifier.

Parameters

`database_name`	The name of the database being created.
`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` was called with no arguments. If by chance 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 8.9. `mysql_create_db` alternative example

The function `mysql_create_db` is deprecated. It is preferable to use `mysql_query` to issue a 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`

8.1.4.7. mysql_data_seek

- mysql_data_seek

  Move internal result pointer

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

result	The result resource that is being evaluated. This result comes from a call to mysql_query.
row_number	The desired row number of the new result pointer.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 8.10. 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`

8.1.4.8. `mysql_db_name`

- `mysql_db_name`

  Get result data

Description

```
  string mysql_db_name(resource result,
                       int row,
                       mixed field);
```

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.

Examples

Example 8.11. `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

8.1.4.9. mysql_db_query

- mysql_db_query

  Send a MySQL query

Description

```
resource mysql_db_query(string database,
                        string query,
                        resource link_identifier);
```

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.
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 was called with no arguments. If by chance 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.
4.0.6	This function is deprecated, do not use this function. Use mysql_select_db and mysql_query instead.

Examples

Example 8.12. 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`

8.1.4.10. `mysql_drop_db`

- `mysql_drop_db`

  Drop (delete) a MySQL database

Description

```
  bool mysql_drop_db(string database_name,
                     resource link_identifier);
```

`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 a 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` was called with no arguments. If by chance 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 8.13. `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`

8.1.4.11. `mysql_errno`

- `mysql_errno`

  Returns the numerical value of the error message from previous MySQL operation

Description

```
  int mysql_errno(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.14. `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

8.1.4.12. `mysql_error`

- `mysql_error`

  Returns the text of the error message from previous MySQL operation

Description

```
  string mysql_error(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.15. `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

8.1.4.13. `mysql_escape_string`

- `mysql_escape_string`

  Escapes a string for use in a mysql_query

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.
4.3.0	This function became deprecated, do not use this function. Instead, use `mysql_real_escape_string`.

Examples

Example 8.16. `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.

8.1.4.14. `mysql_fetch_array`

- `mysql_fetch_array`

  Fetch a result row as an associative array, a numeric array, or both

Description

```
  array mysql_fetch_array(resource result,
                          int result_type);
```

Returns an 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`.
`result_type`	The type of array that is to be fetched. It's a constant and can take the following values: `MYSQL_ASSOC` , `MYSQL_NUM` , and the default value of `MYSQL_BOTH` .

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 8.17. Query with aliased duplicate field names

```
SELECT table1.field AS foo, table2.field AS bar FROM table1, table2
```

Example 8.18. `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 8.19. `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 8.20. `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`

8.1.4.15. `mysql_fetch_assoc`

- `mysql_fetch_assoc`

  Fetch a result row as an associative array

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 8.21. 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`

8.1.4.16. `mysql_fetch_field`

- `mysql_fetch_field`

  Get column information from a result and return as an object

Description

```
  object mysql_fetch_field(resource result,
                           int field_offset);
```

Returns an object containing field information. This function can be used to obtain information about fields in the provided query result.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.
`field_offset`	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 *`field_offset`* 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
- def - default value of the column
- 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 8.22. `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
default:      $meta->def
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*.

See Also

mysql_field_seek

8.1.4.17. mysql_fetch_lengths

- mysql_fetch_lengths

  Get the length of each output in a result

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 8.23. 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`

8.1.4.18. `mysql_fetch_object`

- `mysql_fetch_object`

  Fetch a result row as an object

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.

`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.

ChangeLog

Version	Description
5.0.0	Added the ability to return as a different object.

Examples

Example 8.24. `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 8.25. `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`

8.1.4.19. `mysql_fetch_row`

- `mysql_fetch_row`

  Get a result row as an enumerated array

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 8.26. 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`

8.1.4.20. `mysql_field_flags`

- `mysql_field_flags`

  Get the flags associated with the specified field in a result

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 8.27. 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`

8.1.4.21. `mysql_field_len`

Copyright 1997-2008 the PHP Documentation Group.

- `mysql_field_len`

  Returns the length of the specified field

Description

```
  int mysql_field_len(resource result,
                      int field_offset);
```

`mysql_field_len` returns the length of the specified field.

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 length of the specified field index on success, or FALSE on failure.

Examples

Example 8.28. 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

8.1.4.22. mysql_field_name

- mysql_field_name

  Get the name of the specified field in a result

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 8.29. 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

8.1.4.23. mysql_field_seek

- mysql_field_seek

  Set result pointer to a specified field offset

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`

8.1.4.24. `mysql_field_table`

- `mysql_field_table`

  Get name of the table the specified field is in

Description

```
  string mysql_field_table(resource result,
                           int field_offset);
```

Returns the name of the table that the specified field is in.

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 table on success.

Examples

Example 8.30. A `mysql_field_table` example

```
<?php

$query = "SELECT account.*, country.* FROM account, country WHERE country.name = 'Portugal' AND account.country_id = co

// 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_fieldtable`

See Also

`mysql_list_tables`

8.1.4.25. `mysql_field_type`

- `mysql_field_type`

  Get the type of the specified field in a result

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 8.31. `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`

8.1.4.26. `mysql_free_result`

- `mysql_free_result`

  Free result memory

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 8.32. 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`

8.1.4.27. `mysql_get_client_info`

- `mysql_get_client_info`

  Get MySQL client info

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 8.33. `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`

8.1.4.28. `mysql_get_host_info`

- `mysql_get_host_info`

  Get MySQL host info

Description

```
string mysql_get_host_info(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.34. `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`

8.1.4.29. `mysql_get_proto_info`

- `mysql_get_proto_info`

  Get MySQL protocol info

Description

```
int mysql_get_proto_info(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.35. `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`

8.1.4.30. `mysql_get_server_info`

- `mysql_get_server_info`

  Get MySQL server info

Description

```
string mysql_get_server_info(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.36. `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`

8.1.4.31. `mysql_info`

- `mysql_info`

  Get information about the most recent query

Description

```
string mysql_info(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.37. 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`

8.1.4.32. `mysql_insert_id`

- `mysql_insert_id`

  Get the ID generated from the previous INSERT operation

Description

```
  int mysql_insert_id(resource link_identifier);
```

Retrieves the ID generated for an AUTO_INCREMENT column by the previous INSERT 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` was called with no arguments. If by chance 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 INSERT 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 8.38. `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` converts 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, the value returned by `mysql_insert_id` will be incorrect. Instead, use the internal MySQL SQL function `LAST_INSERT_ID()` in an SQL query.

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`

8.1.4.33. `mysql_list_dbs`

- `mysql_list_dbs`

  List databases available on a MySQL server

Description

```
  resource mysql_list_dbs(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.39. `mysql_list_dbs` example

```
<?php
$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`

8.1.4.34. `mysql_list_fields`

- `mysql_list_fields`

  List MySQL table fields

Description

```
  resource mysql_list_fields(string database_name,
                             string table_name,
                             resource link_identifier);
```

Retrieves information about the given table name.

This function is deprecated. It is preferable to use `mysql_query` to issue a SQL `SHOW COLUMNS FROM table [LIKE 'name']` statement instead.

Parameters

`database_name`	The name of the database that's being queried.
`table_name`	The name of the table that's being queried.

`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` was called with no arguments. If by chance 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.

The returned result can be used with `mysql_field_flags`, `mysql_field_len`, `mysql_field_name` and `mysql_field_type`.

Examples

Example 8.40. 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`

8.1.4.35. `mysql_list_processes`

- `mysql_list_processes`

  List MySQL processes

Description

```
resource mysql_list_processes(resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.41. `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`

8.1.4.36. `mysql_list_tables`

- `mysql_list_tables`

  List tables in a MySQL database

Description

```
resource mysql_list_tables(string database,
                           resource link_identifier);
```

Retrieves a list of table names from a MySQL database.

This function is deprecated. It is preferable to use `mysql_query` to issue a 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` was called with no arguments. If by chance 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 8.42. `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

8.1.4.37. mysql_num_fields

- mysql_num_fields

  Get number of fields in result

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 8.43. 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

8.1.4.38. mysql_num_rows

- mysql_num_rows

Get number of rows in result

Description

```
int mysql_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 `mysql_affected_rows`.

Parameters

`result`	The result resource that is being evaluated. This result comes from a call to `mysql_query`.

Return Values

The number of rows in a result set on success, or `FALSE` on failure.

Examples

Example 8.44. `mysql_num_rows` example

```
<?php

$link = mysql_connect("localhost", "mysql_user", "mysql_password");
mysql_select_db("database", $link);

$result = mysql_query("SELECT * FROM table1", $link);
$num_rows = mysql_num_rows($result);

echo "$num_rows Rows\n";

?>
```

Notes

> **Note**
>
> If you use `mysql_unbuffered_query`, `mysql_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: `mysql_numrows`

See Also

`mysql_affected_rows`
`mysql_connect`
`mysql_data_seek`
`mysql_select_db`
`mysql_query`

8.1.4.39. `mysql_pconnect`

- `mysql_pconnect`

  Open a persistent connection to a MySQL server

Description

```
resource mysql_pconnect(string server,
                        string username,
                        string password,
                        int client_flags);
```

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

`server`	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'
`username`	The username. Default value is the name of the user that owns the server process.
`password`	The password. Default value is an empty password.
`client_flags`	The *`client_flags`* 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
4.3.0	Added the *`client_flags`* parameter.
3.0.10	Added support for ":/path/to/socket" with *`server`*.
3.0.0	Added support for ":port" with *`server`*.

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

8.1.4.40. `mysql_ping`

- `mysql_ping`

  Ping a server connection or reconnect if there is no connection

Description

```
bool mysql_ping(resource link_identifier);
```

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**
>
> Since MySQL 5.0.13, automatic reconnection feature is disabled.

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` was called with no arguments. If by chance 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 8.45. 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`

8.1.4.41. `mysql_query`

- `mysql_query`

  Send a MySQL query

Description

```
resource mysql_query(string query,
                     resource link_identifier);
```

`mysql_query` sends an unique query (multiple queries are not supported) to the currently active database on the server that's associated with the specified *`link_identifier`*.

Parameters

`query`	A SQL query The query string should not end with a semicolon.
`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` was called with no arguments. If by chance 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 8.46. 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 8.47. 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 a 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`

8.1.4.42. `mysql_real_escape_string`

- `mysql_real_escape_string`

  Escapes special characters in a string for use in a SQL statement

Description

```
  string mysql_real_escape_string(string unescaped_string,
                                  resource link_identifier);
```

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.

Parameters

`unescaped_string`	The string that is to be 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` was called with no arguments. If by chance no connection is found or established, an `E_WARNING` level error is generated.

Return Values

Returns the escaped string, or `FALSE` on error.

Examples

Example 8.48. 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 8.49. An example SQL Injection Attack

```
<?php
// 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);

// We didn't check $_POST['password'], it could be anything the user wanted! For example:
$_POST['username'] = 'aidan';
$_POST['password'] = "' OR ''='";

// 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.

Example 8.50. A "Best Practice" query

Using `mysql_real_escape_string` around each variable prevents SQL Injection. This example demonstrates the "best practice" method for querying a database, independent of the Magic Quotes setting.

```
<?php

if (isset($_POST['product_name']) && isset($_POST['product_description']) && isset($_POST['user_id'])) {
    // Connect

    $link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password');

    if(!is_resource($link)) {

        echo "Failed to connect to the server\n";
        // ... log the error properly

    } else {

        // Reverse magic_quotes_gpc/magic_quotes_sybase effects on those vars if ON.

        if(get_magic_quotes_gpc()) {
            $product_name        = stripslashes($_POST['product_name']);
            $product_description = stripslashes($_POST['product_description']);
        } else {
            $product_name        = $_POST['product_name'];
            $product_description = $_POST['product_description'];
        }

        // Make a safe query
        $query = sprintf("INSERT INTO products (`name`, `description`, `user_id`) VALUES ('%s', '%s', %d)",
                    mysql_real_escape_string($product_name, $link),
                    mysql_real_escape_string($product_description, $link),
                    $_POST['user_id']);

        mysql_query($query, $link);

        if (mysql_affected_rows($link) > 0) {
            echo "Product inserted\n";
        }
    }
} else {
    echo "Fill the form properly\n";
}
?>
```

The query will now execute correctly, and SQL Injection attacks will not work.

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_client_encoding`
`addslashes`
`stripslashes`
The magic_quotes_gpc directive
The magic_quotes_runtime directive

8.1.4.43. `mysql_result`

- `mysql_result`

  Get result data

Description

```
string mysql_result(resource result,
                    int row,
                    mixed field);
```

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 8.51. `mysql_result` example

```
<?php
$link = mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
    die('Could not connect: ' . 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`

8.1.4.44. `mysql_select_db`

- `mysql_select_db`

  Select a MySQL database

Description

```
bool mysql_select_db(string database_name,
                     resource link_identifier);
```

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` was called with no arguments. If by chance 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 8.52. `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`

8.1.4.45. `mysql_set_charset`

- `mysql_set_charset`

  Sets the client character set

Description

```
bool mysql_set_charset(string charset,
                       resource link_identifier);
```

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` was called with no arguments. If by chance 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 execute `SET NAMES ..` is not reccomended.

See Also

`mysql_client_encoding`
List of character sets that MySQL supports

8.1.4.46. `mysql_stat`

- `mysql_stat`

  Get current system status

Description

```
string mysql_stat(resource link_identifier);
```

`mysql_stat` returns the current server status.

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` was called with no arguments. If by chance 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 8.53. `mysql_stat` example

```
<?php
$link   = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$status = explode('  ', mysql_stat($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 8.54. Alternative `mysql_stat` example

```
<?php
$link   = mysql_connect('localhost', 'mysql_user', 'mysql_password');
$result = mysql_query('SHOW VARIABLES', $link);
while ($row = mysql_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`

8.1.4.47. `mysql_tablename`

- `mysql_tablename`

  Get table name of field

Description

```
  string mysql_tablename(resource result,
                         int i);
```

Retrieves the table name from a *`result`*.

This function deprecated. It is preferable to use `mysql_query` to issue a 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`.

Examples

Example 8.55. `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`

8.1.4.48. `mysql_thread_id`

- mysql_thread_id

  Return the current thread ID

Description

```
int mysql_thread_id(resource link_identifier);
```

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 was called with no arguments. If by chance 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 8.56. 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

8.1.4.49. mysql_unbuffered_query

- mysql_unbuffered_query

  Send an SQL query to MySQL, without fetching and buffering the result rows

Description

```
resource mysql_unbuffered_query(string query,
                                resource link_identifier);
```

`mysql_unbuffered_query` sends a SQL query *`query`* to MySQL, without fetching and buffering the result rows automatically, as `mysql_query` does. On the one hand, this saves a considerable amount of memory with SQL queries that produce large result sets. On the other hand, you can start working on the result set immediately after the first row has been retrieved: you don't have to wait until the complete SQL query has been performed. When using multiple DB-connects, you have to specify the optional parameter *`link_identifier`*.

Parameters

`query`	A SQL query
`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` was called with no arguments. If by chance 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`. You also have to fetch all result rows from an unbuffered SQL query, before you can send a new SQL query to MySQL.

See Also

`mysql_query`

8.2. MySQL Improved Extension (`Mysqli`)

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 8.2.2, “Overview”

Documentation for MySQL can be found at http://dev.mysql.com/doc/.

Parts of this documentation included from MySQL manual with permissions of MySQL AB.

8.2.1. Examples

All Examples in the MySQLI documentation use the world database from MySQL AB. The world database can be found at http://dev.mysql.com/get/Downloads/Manual/world.sql.gz/from/pick

8.2.2. Overview

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 organised 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.

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 8.1, "MySQL".

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 frameowrk, its source code is located in the directory `ext/mysqli`.

For further information on the `mysqli` extension, see Section 8.2, "MySQL Improved Extension (`Mysqli`)".

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 Section 8.3, "MySQL Functions (PDO_MYSQL)".

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 8.3, "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 `libmysql`.

However, the interface presented by `libmysql` was not optimized for communication with PHP applications, as `libmysql` was originally designed with C applications in mind. For this reason the MySQL Native Driver, `mysqlnd`, was developed as an alternative to `libmysql` for PHP applications.

The `mysql` extension, the `mysqli` extension and the PDO MySQL driver can each be individually configured to use either `libmysql` or `mysqlnd`. As `mysqlnd` is designed specifically to be utilised in the PHP system it has numerous memory and speed enhancements over `libmysql`. 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:

	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
Comes with PHP 6.0	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

8.2.3. Installing/Configuring

8.2.3.1. Requirements

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 the version 4.1.3 or above of MySQL. For previous versions, please see the MySQL extension documentation.

8.2.3.2. Installation

To install the mysqli extension for PHP, use the `--with-mysqli=mysql_config_path/mysql_config` configuration option where `mysql_config_path` represents the location of the `mysql_config` program that comes with MySQL versions greater than 4.1.

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.

8.2.3.2.1. Installation on Windows Systems

MySQLi is not enabled by default, so the `php_mysqli.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_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.

8.2.3.3. Runtime Configuration

The behaviour of these functions is affected by settings in `php.ini`.

Table 8.4. MySQLi Configuration Options

Name	Default	Changeable	Changelog
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.

For further details and definitions of the above PHP_INI_* constants, see the chapter on configuration changes.

Here's a short explanation of the configuration directives.

`mysqli.max_links` integer	The maximum number of MySQL connections per process.
`mysqli.default_port` 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 `MYSQL_TCP_PORT` environment variable, the `mysql-tcp` entry in `/etc/services` or the compile-time `MYSQL_PORT` constant, in that order. Win32 will only use the `MYSQL_PORT` constant.
`mysqli.default_socket` string	The default socket name to use when connecting to a local database server if no other socket name is specified.
`mysqli.default_host` 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.
`mysqli.default_user` 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.
`mysqli.default_pw` string	The default password to use when connecting to the database server if no other password is specified. Doesn't apply in safe mode.

8.2.3.4. Resource Types

This extension has no resource types defined.

8.2.4. Predefined Constants

`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
`MYSQLI_ASSOC`	Columns are returned into the array having the fieldname as the array index.
`MYSQLI_NUM`	Columns are returned into the array having an enumerated index.
`MYSQLI_BOTH`	Columns are returned into the array having both a numerical index and the fieldname as the associative index.
`MYSQLI_NOT_NULL_FLAG`	Indicates that a field is defined as `NOT NULL`
`MYSQLI_PRI_KEY_FLAG`	Field is part of a primary index
`MYSQLI_UNIQUE_KEY_FLAG`	Field is part of a unique index.
`MYSQLI_MULTIPLE_KEY_FLAG`	Field is part of an index.
`MYSQLI_BLOB_FLAG`	Field is defined as `BLOB`
`MYSQLI_UNSIGNED_FLAG`	Field is defined as `UNSIGNED`
`MYSQLI_ZEROFILL_FLAG`	Field is defined as `ZEROFILL`
`MYSQLI_AUTO_INCREMENT_FLAG`	Field is defined as `AUTO_INCREMENT`
`MYSQLI_TIMESTAMP_FLAG`	Field is defined as `TIMESTAMP`
`MYSQLI_SET_FLAG`	Field is defined as `SET`
`MYSQLI_NUM_FLAG`	Field is defined as `NUMERIC`

MYSQLI_PART_KEY_FLAG	Field is part of an multi-index
MYSQLI_GROUP_FLAG	Field is part of GROUP BY
MYSQLI_TYPE_DECIMAL	Field is defined as DECIMAL
MYSQLI_TYPE_NEWDECIMAL	Precision math DECIMAL or NUMERIC field (MySQL 5.0.3 and up)
MYSQLI_TYPE_BIT	Field is defined as BIT (MySQL 5.0.3 and up)
MYSQLI_TYPE_TINY	Field is defined as TINYINT
MYSQLI_TYPE_SHORT	Field is defined as SMALLINT
MYSQLI_TYPE_LONG	Field is defined as INT
MYSQLI_TYPE_FLOAT	Field is defined as FLOAT
MYSQLI_TYPE_DOUBLE	Field is defined as DOUBLE
MYSQLI_TYPE_NULL	Field is defined as DEFAULT NULL
MYSQLI_TYPE_TIMESTAMP	Field is defined as TIMESTAMP
MYSQLI_TYPE_LONGLONG	Field is defined as BIGINT
MYSQLI_TYPE_INT24	Field is defined as MEDIUMINT
MYSQLI_TYPE_DATE	Field is defined as DATE
MYSQLI_TYPE_TIME	Field is defined as TIME
MYSQLI_TYPE_DATETIME	Field is defined as DATETIME
MYSQLI_TYPE_YEAR	Field is defined as YEAR
MYSQLI_TYPE_NEWDATE	Field is defined as DATE
MYSQLI_TYPE_ENUM	Field is defined as ENUM
MYSQLI_TYPE_SET	Field is defined as SET
MYSQLI_TYPE_TINY_BLOB	Field is defined as TINYBLOB
MYSQLI_TYPE_MEDIUM_BLOB	Field is defined as MEDIUMBLOB
MYSQLI_TYPE_LONG_BLOB	Field is defined as LONGBLOB
MYSQLI_TYPE_BLOB	Field is defined as BLOB
MYSQLI_TYPE_VAR_STRING	Field is defined as VARCHAR
MYSQLI_TYPE_STRING	Field is defined as CHAR
MYSQLI_TYPE_GEOMETRY	Field is defined as GEOMETRY
MYSQLI_NEED_DATA	More data available for bind variable
MYSQLI_NO_DATA	No more data available for bind variable
MYSQLI_DATA_TRUNCATED	Data truncation occurred. Available since PHP 5.1.0 and MySQL 5.0.5.
MYSQLI_ENUM_FLAG	Field is defined as ENUM. Available since PHP 5.3.0.

8.2.5. The MySQLi Extension Function Summary

MySQLi Class			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Properties			
$mysqli->affected_rows	`mysqli_affected_rows`	N/A	Gets the number of affected rows in a previous MySQL operation
$mysqli->connect_errno	`mysqli_connect_errno`	N/A	Returns the error code from last connect call
$mysqli->connect_error	`mysqli_connect_error`	N/A	Returns a string description of the last connect error
$mysqli->errno	`mysqli_errno`	N/A	Returns the error code for the most recent function call
$mysqli->error	`mysqli_error`	N/A	Returns a string description of the last error
$mysqli->field_count	`mysqli_field_count`	N/A	Returns the number of columns for the most recent query
$mysqli->host_info	`mysqli_get_host_info`	N/A	Returns a string representing the type of connection used
$mysqli->protocol_version	`mysqli_get_proto_info`	N/A	Returns the version of the MySQL protocol used
$mysqli->server_info	`mysqli_get_server_info`	N/A	Returns the version of the MySQL server
$mysqli->server_version	`mysqli_get_server_version`	N/A	Returns the version of the MySQL server as an integer
$mysqli->info	`mysqli_info`	N/A	Retrieves information about the most recently executed query
$mysqli->insert_id	`mysqli_insert_id`	N/A	Returns the auto generated id used in the last query
$mysqli->sqlstate	`mysqli_sqlstate`	N/A	Returns the SQLSTATE error from previous MySQL operation
$mysqli->warning_count	`mysqli_warning_count`	N/A	Returns the number of warnings from the last query for the given link
Methods			
`mysqli->autocommit`	`mysqli_autocommit`	N/A	Turns on or off auto-commiting database modifications
`mysqli->change_user`	`mysqli_change_user`	N/A	Changes the user of the specified database connection
`mysqli->character_set_name`, mysqli->client_encoding	`mysqli_character_set_name`	`mysqli_client_encoding`	Returns the default character set for the database connection
`mysqli->close`	`mysqli_close`	N/A	Closes a previously opened database connection
`mysqli->commit`	`mysqli_commit`	N/A	Commits the current transaction
`mysqli::__construct`	`mysqli_connect`	N/A	Open a new connection to the MySQL server [Note: static (i.e. class) method]
`mysqli->debug`	`mysqli_debug`	N/A	Performs debugging operations
`mysqli->dump_debug_info`	`mysqli_dump_debug_info`	N/A	Dump debugging information into the log
`mysqli->get_charset`	`mysqli_get_charset`	N/A	Returns a character set object
`mysqli->get_client_info`	`mysqli_get_client_info`	N/A	Returns the MySQL client version as a string
`mysqli->get_client_version`	`mysqli_get_client_version`	N/A	Get MySQL client info

MySQLi Class			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
$mysqli->get_connection_stats ()	mysqli_get_connection_stats()	N/A	NOT DOCUMENTED [mysqlnd only]
`mysqli->get_server_info`	`mysqli_get_server_info`	N/A	NOT DOCUMENTED
`mysqli->get_warnings`	`mysqli_get_warnings`	N/A	NOT DOCUMENTED
`mysqli_init`	`mysqli_init`	N/A	Initializes MySQLi and returns a resource for use with 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 a 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`	Escapes special characters in a string for use in a SQL statement, taking into account the current charset of the connection
`mysqli->real_query`	`mysqli_real_query`	N/A	Execute an SQL query
`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->set_local_infile_handler`	`mysqli_set_local_infile_handler`	N/A	Set callback function for LOAD DATA LOCAL INFILE command
`mysqli->ssl_set`	`mysqli_ssl_set`	N/A	Used for establishing secure connections using SSL
`mysqli->stat`	`mysqli_stat`	N/A	Gets the current system status
`mysqli->stmt_init`	`mysqli_stmt_init`	N/A	Initializes a statement and returns an object for use with mysqli_stmt_prepare
`mysqli->store_result`	`mysqli_store_result`	N/A	Transfers a result set from the last query
`mysqli->thread_id`	`mysqli_thread_id`	N/A	Returns the thread ID for the current connection

MySQLi Class			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
`mysqli->thread_safe`	`mysqli_thread_safe`	N/A	Returns whether thread safety is given or not
`mysqli->use_result`	`mysqli_use_result`	N/A	Initiate a result set retrieval

MySQL_STMT			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Properties			
$mysqli_stmt->affected_rows	`mysqli_stmt_affected_rows`	N/A	Returns the total number of rows changed, deleted, or inserted by the last executed statement
$mysqli_stmt->errno	`mysqli_stmt_errno`	N/A	Returns the error code for the most recent statement call
$mysqli_stmt->error	`mysqli_stmt_error`	N/A	Returns a string description for last statement error
$mysqli_stmt->field_count	`mysqli_stmt_field_count`	N/A	Returns the number of field in the given statement - not documented
$mysqli_stmt->insert_id	`mysqli_stmt_insert_id`	N/A	Get the ID generated from the previous INSERT operation
$mysqli_stmt->num_rows	`mysqli_stmt_num_rows`	N/A	Return the number of rows in statements result set
$mysqli_stmt->param_count	`mysqli_stmt_param_count`	`mysqli_param_count`	Returns the number of parameter for the given statement
$mysqli_stmt->sqlstate	`mysqli_stmt_sqlstate`	N/A	Returns SQLSTATE error from previous statement operation
Methods			
`mysqli_stmt->attr_get`	`mysqli_stmt_attr_get`	N/A	NOT DOCUMENTED
`mysqli_stmt->attr_set`	`mysqli_stmt_attr_set`	N/A	NOT DOCUMENTED
`mysqli_stmt->bind_param`	`mysqli_stmt_bind_param`	`mysqli_bind_param`	Binds variables to a prepared statement as parameters
`mysqli_stmt->bind_result`	`mysqli_stmt_bind_result`	`mysqli_bind_result`	Binds variables to a prepared statement for result storage
`mysqli_stmt->close`	`mysqli_stmt_close`	N/A	Closes a prepared statement
`mysqli_stmt->data_seek`	`mysqli_stmt_data_seek`	N/A	Seeks to an arbitrary row in statement result set
`mysqli_stmt->execute`	`mysqli_stmt_execute`	`mysqli_execute`	Executes a prepared Query
`mysqli_stmt->fetch`	`mysqli_stmt_fetch`	`mysqli_fetch`	Fetch results from a prepared statement into the bound variables
`mysqli_stmt->free_result`	`mysqli_stmt_free_result`	N/A	Frees stored result memory for the given statement handle
$mysqli_stmt->get_result()	mysqli_stmt_get_result	N/A	NOT DOCUMENTED [mysqlnd only]
`mysqli_stmt->get_warnings`	`mysqli_stmt_get_warnings`	N/A	NOT DOCUMENTED
$mysqli_stmt->more_results()	mysqli_stmt_more_results()	N/A	NOT DOCUMENTED [mysqlnd only]
$mysqli_stmt->next_result()	mysqli_stmt_next_result()	N/A	NOT DOCUMENTED [mysqlnd only]
`mysqli_stmt->num_rows`	`mysqli_stmt_num_rows`	N/A	NOT DOCUMENTED [see also num_rows property]
`mysqli_stmt->prepare`	`mysqli_stmt_prepare`	N/A	Prepare a SQL statement for

MySQL_STMT			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
			execution
`mysqli_stmt->reset`	`mysqli_stmt_reset`	N/A	Resets a prepared statement
`mysqli_stmt->result_metadata`	`mysqli_stmt_result_metadata`	`mysqli_get_metadata`	Returns result set metadata from a prepared statement
`mysqli_stmt->send_long_data`	`mysqli_stmt_send_long_data`	`mysqli_send_long_data`	Send data in blocks
`mysqli_stmt->store_result`	`mysqli_stmt_store_result`	N/A	Transfers a result set from a prepared statement

MySQLi_RESULT			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Properties			
$mysqli_result->current_field	`mysqli_field_tell`	N/A	Get current field offset of a result pointer
$mysqli_result->field_count	`mysqli_num_fields`	N/A	Get the number of fields in a result
$mysqli_result->lengths	`mysqli_fetch_lengths`	N/A	Returns the lengths of the columns of the current row in the result set
$mysqli_result->num_rows	`mysqli_num_rows`	N/A	Gets the number of rows in a result
Methods			
`mysqli_result->data_seek`	`mysqli_data_seek`	N/A	Adjusts the result pointer to an arbitary row in the result
$mysqli_result->fetch_all()	mysqli_fetch_all()	N/A	NOT DOCUMENTED [mysqlnd only]
`mysqli_result->fetch_array`	`mysqli_fetch_array`	N/A	Fetch a result row as an associative, a numeric array, or both
`mysqli_result->fetch_assoc`	`mysqli_fetch_assoc`	N/A	Fetch a result row as an associative array
`mysqli_result->fetch_field_direct`	`mysqli_fetch_field_direct`	N/A	Fetch meta-data for a single field
`mysqli_result->fetch_field`	`mysqli_fetch_field`	N/A	Returns the next field in the result set
`mysqli_result->fetch_fields`	`mysqli_fetch_fields`	N/A	Returns an array of objects representing the fields in a result set
`mysqli_result->fetch_object`	`mysqli_fetch_object`	N/A	Returns the current row of a result set as an object
`mysqli_result->fetch_row`	`mysqli_fetch_row`	N/A	Get a result row as an enumerated array
`mysqli_result->field_seek`	`mysqli_field_seek`	N/A	Set result pointer to a specified field offset
`mysqli_result->free`, mysqli_result->close, mysqli_result->free_result	`mysqli_free_result`	N/A	Frees the memory associated with a result

MySQL_Driver			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
Properties			
N/A			
Methods			

MySQL_Driver			
OOP Interface	**Procedural Interface**	**Alias (Do not use)**	**Description**
mysqli_driver->embedded_server_end	mysqli_embedded_server_end	N/A	NOT DOCUMENTED
mysqli_driver->embedded_server_start	mysqli_embedded_server_start	N/A	NOT DOCUMENTED

Note

Alias functions are provided for backward compatibility purposes only. Do not use them in new projects.

8.2.6. The MySQLi class (MySQLi)

Represents a connection between PHP and a MySQL database.

```
 MySQLi {
 MySQLi

      Properties

  int affected_rows ;


  string connect_errno ;


  string connect_error ;


  int errno ;


  string error ;


  int field_count ;


  string host_info ;


  string protocol_version ;


  string server_info ;


  int server_version ;


  string info ;


  int insert_id ;


  string sqlstate ;


  int thread_id ;


  int warning_count ;
Methods

  int mysqli_affected_rows(mysqli link);


  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 mysqli_connect_errno();


string mysqli_connect_error();


mysqli mysqli_connect(string host,
                      string username,
                      string passwd,
                      string dbname,
                      int port,
                      string socket);


bool mysqli::debug(string message);


bool mysqli::dump_debug_info();


int mysqli_errno(mysqli link);


string mysqli_error(mysqli link);


int mysqli_field_count(mysqli link);


object mysqli::get_charset();


string mysqli::get_client_info();


int mysqli::get_client_version();


string mysqli_get_host_info(mysqli link);


int mysqli_get_proto_info(mysqli link);


string mysqli_get_server_info(mysqli link);


int mysqli_get_server_version(mysqli link);


object mysqli::get_warnings();


string mysqli_info(mysqli link);


mysqli init();


int mysqli_insert_id(mysqli link);


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();


mysqli_stmt prepare(string query);
```

```
  mixed mysqli::query(string query,
                      int resultmode);


  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 real_query(string query);


  bool mysqli::rollback();


  bool mysqli::select_db(string dbname);


  bool mysqli::set_charset(string charset);


  void mysqli_set_local_infile_default(mysqli link);


  bool mysqli_set_local_infile_handler(mysqli link,
                                       callback read_func);


  string mysqli_sqlstate(mysqli link);


  bool mysqli::ssl_set(string key,
                       string cert,
                       string ca,
                       string capath,
                       string cipher);


  string mysqli::stat();


  mysqli_stmt stmt_init();


  mysqli_result store_result();


  int mysqli_thread_id(mysqli link);


  bool mysqli_thread_safe();


  mysqli_result use_result();


  int mysqli_warning_count(mysqli link);
}
```

8.2.6.1. `mysqli->affected_rows`, `mysqli_affected_rows`

- `mysqli->affected_rows`

  `mysqli_affected_rows`

  Gets the number of affected rows in a previous MySQL operation

Description

Object oriented style (property):

```
mysqli {
  int 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 where 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 maximal int value, the number of affected rows will be returned as a string.

Examples

Example 8.57. 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();
?>
```

Example 8.58. 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 example 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`

8.2.6.2. `mysqli::autocommit`, `mysqli_autocommit`

- `mysqli::autocommit`

  `mysqli_autocommit`

  Turns on or off auto-commiting database modifications

Description

Object oriented style (method)

```
  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

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
`mode`	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 8.59. 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();
?>
```

Example 8.60. 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 example will output:

```
Autocommit is 1
```

See Also

mysqli_commit
mysqli_rollback

8.2.6.3. mysqli::change_user, mysqli_change_user

- mysqli::change_user

  mysqli_change_user

  Changes the user of the specified database connection

Description

Object oriented style (method):

```
  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

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
user	The MySQL user name.
password	The MySQL password.
database	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 8.61. 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();
?>
```

Example 8.62. 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 example will output:

```
Default database: world
Value of variable a is NULL
```

See Also

`mysqli_connect`
`mysqli_select_db`

8.2.6.4. `mysqli::character_set_name`, `mysqli_character_set_name`

- `mysqli::character_set_name`

  `mysqli_character_set_name`

  Returns the default character set for the database connection

Description

Object oriented style (method):

```
  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 8.63. 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();
?>
```

Example 8.64. 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 example will output:

```
Current character set is latin1_swedish_ci
```

See Also

`mysqli_client_encoding`
`mysqli_real_escape_string`

8.2.6.5. `mysqli::close`, `mysqli_close`

- `mysqli::close`

  `mysqli_close`

  Closes a previously opened database connection

Description

Object oriented style (method):

```
bool mysqli::close();
```

Procedural style:

```
bool mysqli_close(mysqli link);
```

Closes a previously opened database 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_connect`
`mysqli_init`
`mysqli_real_connect`

8.2.6.6. `mysqli::commit`, `mysqli_commit`

- `mysqli::commit`

  `mysqli_commit`

  Commits the current transaction

Description

Object oriented style (method)

```
bool mysqli::commit();
```

Procedural style:

```
bool mysqli_commit(mysqli link);
```

Commits the current transaction for the database 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.

Examples

Example 8.65. 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 Type=InnoDB");

/* 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 */
$mysqli->commit();

/* drop table */
$mysqli->query("DROP TABLE Language");

/* close connection */
$mysqli->close();
?>
```

Example 8.66. 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 Type=InnoDB");

/* 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 */
mysqli_commit($link);

/* close connection */
mysqli_close($link);
?>
```

See Also

`mysqli_autocommit`
`mysqli_rollback`

8.2.6.7. `mysqli->connect_errno`, `mysqli_connect_errno`

- `mysqli->connect_errno`

  `mysqli_connect_errno`

  Returns the error code from last connect call

Description

```
mysqli {

  string connect_errno ;

}
```

```
  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 8.67. `mysqli_connect_errno` example

```
<?php
$link = @mysqli_connect("localhost", "nonexisting_user", "");

if (!$link) {
    printf("Can't connect to localhost. Errorcode: %d\n", mysqli_connect_errno());
}
?>
```

See Also

mysqli_connect
mysqli_connect_error
mysqli_errno
mysqli_error
mysqli_sqlstate

8.2.6.8. mysqli->connect_error, mysqli_connect_error

- mysqli->connect_error

  mysqli_connect_error

  Returns a string description of the last connect error

Description

```
 mysqli {

  string connect_error ;

}
```

```
  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. An empty string if no error occurred.

Examples

Example 8.68. mysqli_connect_error example

```
<?php
$link = @mysqli_connect("localhost", "nonexisting_user", "");

if (!$link) {
    printf("Can't connect to localhost. Error: %s\n", mysqli_connect_error());
}
?>
```

See Also

mysqli_connect
mysqli_connect_errno
mysqli_errno

`mysqli_error`
`mysqli_sqlstate`

8.2.6.9. `mysqli::__construct`, `mysqli_connect`

- `mysqli::__construct`

  `mysqli_connect`

  Open a new connection to the MySQL server

Description

Object oriented style (constructor):

```
mysqli::__construct(string host,
                    string username,
                    string passwd,
                    string dbname,
                    int port,
                    string socket);
```

Procedural style

```
mysqli mysqli_connect(string host,
                      string username,
                      string passwd,
                      string dbname,
                      int port,
                      string 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.
`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 a object which represents the connection to a MySQL Server.

Examples

Example 8.69. Object oriented style

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");
```

```
/* check connection */
if ($mysqli->connect_error) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

printf("Host information: %s\n", $mysqli->host_info);

/* close connection */
$mysqli->close();
?>
```

Example 8.70. 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();
}

printf("Host information: %s\n", mysqli_get_host_info($link));

/* close connection */
mysqli_close($link);
?>
```

The above example will output:

```
Host information: Localhost via UNIX socket
```

Notes

Note

OO syntax only: If a connection fails an object is still returned. To check if the connection failed then use the mysqli->connect_error property like in the examples above.

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.

8.2.6.10. `mysqli::debug`, `mysqli_debug`

- `mysqli::debug`

  `mysqli_debug`

  Performs debugging operations

Description

Object oriented style (method):

```
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 complile the MySQL client library to support debugging.

Examples

Example 8.71. Generating a Trace File

```
<?php

/* Create a trace file in '/tmp/client.trace' on the local (client) machine: */
mysqli_debug("d:t:0,/tmp/client.trace");

?>
```

See Also

`mysqli_dump_debug_info`
`mysqli_report`

8.2.6.11. `mysqli::dump_debug_info`, `mysqli_dump_debug_info`

- `mysqli::dump_debug_info`

  `mysqli_dump_debug_info`

  Dump debugging information into the log

Description

Object oriented style (method):

```
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

8.2.6.12. mysqli->errno, mysqli_errno

- mysqli->errno

  mysqli_errno

  Returns the error code for the most recent function call

Description

Object oriented style (property):

```
 mysqli {

  int 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 8.72. 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();
?>
```

Example 8.73. 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 example will output:

```
Errorcode: 1193
```

See Also

`mysqli_connect_errno`
`mysqli_connect_error`
`mysqli_error`
`mysqli_sqlstate`

8.2.6.13. `mysqli->error`, `mysqli_error`

- `mysqli->error`

  `mysqli_error`

  Returns a string description of the last error

Description

Object oriented style (property):

```
 mysqli {

  string 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 8.74. 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();
?>
```

Example 8.75. 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 example will output:

```
Errormessage: Unknown system variable 'a'
```

See Also

`mysqli_connect_errno`
`mysqli_connect_error`
`mysqli_errno`
`mysqli_sqlstate`

8.2.6.14. `mysqli->field_count`, `mysqli_field_count`

- `mysqli->field_count`

  `mysqli_field_count`

  Returns the number of columns for the most recent query

Description

Object oriented style (property):

```
 mysqli_result {
  int 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 8.76. 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();
?>
```

Example 8.77. 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);
?>
```

8.2.6.15. `mysqli::get_charset`, `mysqli_get_charset`

- `mysqli::get_charset`

  `mysqli_get_charset`

  Returns a character set object

Description

```
object mysqli::get_charset();
```

```
object mysqli_get_charset(mysqli link);
```

Returns a character set object providing several properties of the current active characer 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:

`charset`	Character set name
`collation`	Collation name
`dir`	Directory the charset description was fetched from (?) or "" for builtin character sets
`min_length`	Minimum character lenght in bytes
`max_length`	Maximum character length in bytes
`number`	Internal character set number
`state`	Characer set status (?)

Examples

Example 8.78. Object oriented style

```
<?php
  $db = mysqli_init();
  $db->real_connect("localhost","root","","test");
  var_dump($db->get_charset());
?>
```

Example 8.79. Procedural style

```
<?php
  $db = mysqli_init();
  mysqli_real_connect($db, "localhost","root","","test");
  var_dump($db->get_charset());
?>
```

The above example 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_characters_set_name`
`mysqli_set_charset`

8.2.6.16. `mysqli::get_client_info`, `mysqli_get_client_info`

- `mysqli::get_client_info`

  `mysqli_get_client_info`

  Returns the MySQL client version as a string

Description

```
string mysqli::get_client_info();
```

```
string mysqli_get_client_info();
```

The `mysqli_get_client_info` function is used to return a string representing the client version being used in the MySQLi extension.

Return Values

A string that represents the MySQL client library version

Examples

Example 8.80. 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

8.2.6.17. mysqli::get_client_version, mysqli_get_client_version

- mysqli::get_client_version

  mysqli_get_client_version

  Get MySQL client info

Description

```
int mysqli::get_client_version();
```

```
int mysqli_get_client_version();
```

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 exits.

Examples

Example 8.81. 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`

8.2.6.18. `mysqli->host_info`, `mysqli_get_host_info`

- `mysqli->host_info`

  `mysqli_get_host_info`

  Returns a string representing the type of connection used

Description

Object oriented style (property):

```
 mysqli {

  string host_info ;

}
```

Procdural style:

```
  string mysqli_get_host_info(mysqli link);
```

The `mysqli_get_host_info` function returns a string describing the connection represented by the *`link`* parameter is using (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 8.82. 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();
?>
```

Example 8.83. 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 example will output:

```
Host info: Localhost via UNIX socket
```

See Also

`mysqli_get_proto_info`

8.2.6.19. `mysqli->protocol_version`, `mysqli_get_proto_info`

- `mysqli->protocol_version`

  `mysqli_get_proto_info`

  Returns the version of the MySQL protocol used

Description

Object oriented style (property):

```
 mysqli {

  string 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 8.84. 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();
?>
```

Example 8.85. 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 example will output:

```
Protocol version: 10
```

See Also

`mysqli_get_host_info`

8.2.6.20. `mysqli->server_info`, `mysqli_get_server_info`

- `mysqli->server_info`

  `mysqli_get_server_info`

  Returns the version of the MySQL server

Description

Object oriented style (property):

```
 mysqli {

  string 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 8.86. 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();
?>
```

Example 8.87. 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 example will output:

```
Server version: 4.1.2-alpha-debug
```

See Also

mysqli_get_client_info
mysqli_get_client_version
mysqli_get_server_version

8.2.6.21. mysqli->server_version, mysqli_get_server_version

- mysqli->server_version

  mysqli_get_server_version

  Returns the version of the MySQL server as an integer

Description

Object oriented style (property):

```
mysqli {

  int 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 8.88. 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();
?>
```

Example 8.89. 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 example will output:

```
Server version: 40102
```

See Also

mysqli_get_client_info
mysqli_get_client_version
mysqli_get_server_info

8.2.6.22. mysqli::get_warnings, mysqli_get_warnings

- mysqli::get_warnings

  mysqli_get_warnings

Description

```
object mysqli::get_warnings();
```

```
object mysqli_get_warnings(mysqli link);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

8.2.6.23. mysqli->info, mysqli_info

- mysqli->info

  mysqli_info

  Retrieves information about the most recently executed query

Description

Object oriented style (property)

```
mysqli {

  string 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 8.5. 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 above 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 8.90. 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();
?>
```

Example 8.91. 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 example will output:

```
Records: 150  Duplicates: 0  Warnings: 0
```

See Also

`mysqli_affected_rows`
`mysqli_warning_count`
`mysqli_num_rows`

8.2.6.24. `mysqli::init`, `mysqli_init`

- `mysqli::init`

  `mysqli_init`

  Initializes MySQLi and returns a resource for use with mysqli_real_connect()

Description

Object oriented style (method):

```
  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.

See Also

`mysqli_options`
`mysqli_close`
`mysqli_real_connect`
`mysqli_connect`

8.2.6.25. `mysqli->insert_id`, `mysqli_insert_id`

- `mysqli->insert_id`

  `mysqli_insert_id`

  Returns the auto generated id used in the last query

Description

Object oriented style (property):

```
mysqli {
  int insert_id ;
}
```

Procedural style:

```
int 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 8.92. 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();
?>
```

Example 8.93. 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 example will output:

```
New Record has id 1.
```

8.2.6.26. `mysqli::kill`, `mysqli_kill`

- `mysqli::kill`

  `mysqli_kill`

  Asks the server to kill a MySQL thread

Description

Object oriented style (method)

```
  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 8.94. 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();
?>
```

Example 8.95. 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 example will output:

```
Error: MySQL server has gone away
```

See Also

mysqli_thread_id

8.2.6.27. mysqli::more_results, mysqli_more_results

- mysqli::more_results

  mysqli_more_results

  Check if there are any more query results from a multi query

Description

```
bool mysqli::more_results();
```

```
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 on success or FALSE on failure.

Examples

See mysqli_multi_query.

See Also

mysqli_multi_query
mysqli_next_result
mysqli_store_result
mysqli_use_result

8.2.6.28. mysqli::multi_query, mysqli_multi_query

- mysqli::multi_query

  mysqli_multi_query

  Performs a query on the database

Description

Object oriented style (method):

```
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

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
`query`	The query, as a string.

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 8.96. 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();
?>
```

Example 8.97. 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 example will output something similar to:

```
my_user@localhost
-----------------
Amersfoort
Maastricht
Dordrecht
Leiden
Haarlemmermeer
```

See Also

`mysqli_use_result`
`mysqli_store_result`
`mysqli_next_result`
`mysqli_more_results`

8.2.6.29. `mysqli::next_result`, `mysqli_next_result`

- `mysqli::next_result`

  `mysqli_next_result`

  Prepare next result from multi_query

Description

```
bool mysqli::next_result();
```

```
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

8.2.6.30. mysqli::options, mysqli_options

- mysqli::options

  mysqli_options

  Set options

Description

Object oriented style (method)

```
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 8.6. Valid options

Name	Description
MYSQLI_OPT_CONNECT_TIMEOUT	connection timeout in seconds
MYSQLI_OPT_LOCAL_INFILE	enable/disable use of LOAD LOCAL INFILE
MYSQLI_INIT_COMMAND	command to execute after when connecting to MySQL server
MYSQLI_READ_DEFAULT_FILE	Read options from named option file instead of my.cnf
MYSQLI_READ_DEFAULT_GROUP	Read options from the named group from my.cnf or the file specified with MYSQL_READ_DEFAULT_FILE .

value	The value for the option.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

See mysqli_real_connect.

See Also

mysqli_init
mysqli_real_connect

8.2.6.31. mysqli::ping, mysqli_ping

- mysqli::ping

  mysqli_ping

  Pings a server connection, or tries to reconnect if the connection has gone down

Description

Object oriented style (method):

```
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 re-connect 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 8.98. 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();
}

/* 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();
?>
```

Example 8.99. 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();
}

/* 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 example will output:

```
Our connection is ok!
```

8.2.6.32. `mysqli::prepare`, `mysqli_prepare`

- `mysqli::prepare`

  `mysqli_prepare`

  Prepare a SQL statement for execution

Description

Object oriented style (method)

```
  mysqli_stmt prepare(string query);
```

Procedure style:

```
  mysqli_stmt mysqli_prepare(mysqli link,
                             string query);
```

Prepares the SQL query pointed to by the null-terminated string 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 Languange (DML) statements, and not in Data Defination Language (DDL) statements.

Return Values

`mysqli_prepare` returns a statement object or `FALSE` if an error occured.

Examples

Example 8.100. 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();
}

$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();
?>
```

Example 8.101. 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();
}

$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 example 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`

8.2.6.33. `mysqli::query`, `mysqli_query`

- `mysqli::query`

  `mysqli_query`

  Performs a query on the database

Description

Object oriented style (method):

```
  mixed mysqli::query(string query,
                      int resultmode);
```

Procedural style:

```
  mixed mysqli_query(mysqli link,
                     string query,
                     int resultmode);
```

Performs a *`query`* against the database.

Functionally, using this function is identical to calling `mysqli_real_query` followed either by `mysqli_use_result` or `mysqli_store_result`.

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
`query`	The query string.

`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`

Return Values

Returns `TRUE` on success or `FALSE` on failure. For `SELECT, SHOW, DESCRIBE` or `EXPLAIN` `mysqli_query` will return a result object.

Examples

Example 8.102. 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();
?>
```

Example 8.103. 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 example 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`

8.2.6.34. `mysqli::real_connect`, `mysqli_real_connect`

- `mysqli::real_connect`

  `mysqli_real_connect`

  Opens a connection to a mysql server

Description

Object oriented style (method)

```
  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 function `mysqli_options` you can set various options for connection.

- There is a *`flags`* parameter.

Parameters

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
`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.
`username`	The MySQL user name.
`passwd`	If 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.

`flags` With the parameter *`flags`* you can set different connection options:

Table 8.7. Supported flags

Name	Description
`MYSQLI_CLIENT_COMPRESS`	Use compression protocol
`MYSQLI_CLIENT_FOUND_ROWS`	return number of matched rows, not the number of affected rows
`MYSQLI_CLIENT_IGNORE_SPACE`	Allow spaces after function names. Makes all function names reserved words.
`MYSQLI_CLIENT_INTERACTIVE`	Allow `interactive_timeout` seconds (instead of `wait_timeout` seconds) of inactivity before closing the connection
`MYSQLI_CLIENT_SSL`	Use SSL (encryption)

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.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 8.104. Object oriented style

```
<?php

/* create a connection object which is not connected */
$mysqli = mysqli_init();

/* set connection options */
$mysqli->options(MYSQLI_INIT_COMMAND, "SET AUTOCOMMIT=0");
$mysqli->options(MYSQLI_OPT_CONNECT_TIMEOUT, 5);

/* connect to server */
```

```
$mysqli->real_connect('localhost', 'my_user', 'my_password', 'world');

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

printf ("Connection: %s\n.", $mysqli->host_info);

$mysqli->close();
?>
```

Example 8.105. Procedural style

```
<?php

/* create a connection object which is not connected */
$link = mysqli_init();

/* set connection options */
mysqli_options($link, MYSQLI_INIT_COMMAND, "SET AUTOCOMMIT=0");
mysqli_options($link, MYSQLI_OPT_CONNECT_TIMEOUT, 5);

/* connect to server */
mysqli_real_connect($link, 'localhost', 'my_user', 'my_password', 'world');

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

printf ("Connection: %s\n.", mysqli_get_host_info($link));

mysqli_close($link);
?>
```

The above example will output:

```
Connection: Localhost via UNIX socket
```

See Also

`mysqli_connect`
`mysqli_init`
`mysqli_options`
`mysqli_ssl_set`
`mysqli_close`

8.2.6.35. `mysqli::real_escape_string`, `mysqli_real_escape_string`

- `mysqli::real_escape_string`

  `mysqli_real_escape_string`

  Escapes special characters in a string for use in a SQL statement, taking into account the current charset of the connection

Description

Object oriented style (both methods are equivalent):

```
string mysqli::escape_string(string escapestr);
```

```
string 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.

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 8.106. 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();
}

$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();
?>
```

Example 8.107. 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();
}

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 example will output:

```
Error: 42000
1 Row inserted.
```

See Also

`mysqli_character_set_name`

8.2.6.36. `mysqli::real_query`, `mysqli_real_query`

- `mysqli::real_query`

  `mysqli_real_query`

  Execute an SQL query

Description

Object oriented style (method):

```
  bool 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

`link`	Procedural style only: A link identifier returned by `mysqli_connect` or `mysqli_init`
`query`	The query, as a string.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

See Also

mysqli_query
mysqli_store_result
mysqli_use_result

8.2.6.37. `mysqli::rollback`, `mysqli_rollback`

- `mysqli::rollback`

  `mysqli_rollback`

  Rolls back current transaction

Description

Object oriented style (method):

```
bool mysqli::rollback();
```

Procedural style:

```
bool mysqli_rollback(mysqli link);
```

Rollbacks the current transaction for the database.

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 8.108. 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();
}

/* 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();
?>
```

Example 8.109. 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();
}

/* 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 example will output:

```
0 rows in table myCity.
50 rows in table myCity (after rollback).
```

See Also

`mysqli_commit`
`mysqli_autocommit`

8.2.6.38. `mysqli::select_db`, `mysqli_select_db`

- `mysqli::select_db`

  `mysqli_select_db`

  Selects the default database for database queries

Description

Object oriented style (method):

```
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 8.110. 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();
?>
```

Example 8.111. 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 example will output:

```
Default database is test.
Default database is world.
```

See Also

`mysqli_connect`
`mysqli_real_connect`

8.2.6.39. `mysqli::set_charset`, `mysqli_set_charset`

- `mysqli::set_charset`

  `mysqli_set_charset`

  Sets the default client character set

Description

Object oriented style (method):

```
  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 execute SET NAMES .. is not rec-comended.

Examples

Example 8.112. 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();
}

/* change character set to utf8 */
if (!$mysqli->set_charset("utf8")) {
    printf("Error loading character set utf8: %s\n", $mysqli->error);
} else {
    printf("Current character set: %s\n", $mysqli->character_set_name());
}

$mysqli->close();
?>
```

Example 8.113. 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();
}

/* change character set to utf8 */
if (!mysqli_set_charset($link, "utf8")) {
    printf("Error loading character set utf8: %s\n", mysqli_error($link));
} else {
    printf("Current character set: %s\n", mysqli_character_set_name($link));
}

mysqli_close($link);
?>
```

The above example will output:

```
Current character set: utf8
```

See Also

mysqli_character_set_name
mysqli_real_escape_string
List of character sets that MySQL supports

8.2.6.40. mysqli::set_local_infile_default, mysqli_set_local_infile_default

- 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

8.2.6.41. mysqli::set_local_infile_handler, mysqli_set_local_infile_handler

- mysqli::set_local_infile_handler

  mysqli_set_local_infile_handler

  Set callback function for LOAD DATA LOCAL INFILE command

Description

```
bool mysqli_set_local_infile_handler(mysqli link,
                                     callback read_func);
```

Object oriented style (method)

```
mysqli {

  bool set_local_infile_handler(mysqli link,
                                callback read_func);

}
```

Set callback function for LOAD DATA LOCAL INFILE command

The callbacks 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 speficied 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
	`&errormsg`	If an error occures 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 occured.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 8.114. 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";
  }
?>
```

Example 8.115. Procedural style

```
<?php
  $db = mysqli_init();
  mysqli_real_connect($db, "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";

  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 example will output:

```
Input:
23,foo
42,bar

Output:
23,FOO
42,BAR
```

See Also

`mysqli_set_local_infile_default`

8.2.6.42. `mysqli->sqlstate`, `mysqli_sqlstate`

- `mysqli->sqlstate`

  `mysqli_sqlstate`

  Returns the SQLSTATE error from previous MySQL operation

Description

Object oriented style (property):

```
 mysqli {

  string 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 8.116. 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();
?>
```

Example 8.117. 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 example will output:

```
Error - SQLSTATE 42S01.
```

See Also

mysqli_errno
mysqli_error

8.2.6.43. mysqli::ssl_set, mysqli_ssl_set

- mysqli::ssl_set

  mysqli_ssl_set

  Used for establishing secure connections using SSL

Description

Object oriented style (method):

```
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.

Parameters

link	Procedural style only: A link identifier returned by mysqli_connect or mysqli_init
key	The path name to the key file.
cert	The path name to the certificate file.
ca	The path name to the certificate authority file.
capath	The pathname to a directory that contains trusted SSL CA certificates in PEM format.
cipher	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

8.2.6.44. mysqli::stat, mysqli_stat

- mysqli::stat

`mysqli_stat`

Gets the current system status

Description

Object oriented style (method):

```
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 8.118. 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();
?>
```

Example 8.119. 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 example 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`

8.2.6.45. `mysqli::stmt_init`, `mysqli_stmt_init`

- `mysqli::stmt_init`

  `mysqli_stmt_init`

  Initializes a statement and returns an object for use with mysqli_stmt_prepare

Description

Object oriented style (property):

```
 mysqli {
  mysqli_stmt 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`

8.2.6.46. `mysqli::store_result`, `mysqli_store_result`

- `mysqli::store_result`

  `mysqli_store_result`

  Transfers a result set from the last query

Description

Object oriented style (method):

```
mysqli_result store_result();
```

Procedural style:

```
mysqli_result mysqli_store_result(mysqli link);
```

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

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 successfull 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 transfering 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

8.2.6.47. mysqli::thread_id, mysqli_thread_id

- mysqli::thread_id

  mysqli_thread_id

  Returns the thread ID for the current connection

Description

Object oriented style (property):

```
mysqli {
  int 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 8.120. 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();
?>
```

Example 8.121. 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 example will output:

```
Error: MySQL server has gone away
```

See Also

`mysqli_kill`

8.2.6.48. `mysqli::thread_safe`, `mysqli_thread_safe`

- `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` .

8.2.6.49. `mysqli::use_result`, `mysqli_use_result`

- `mysqli::use_result`

  `mysqli_use_result`

  Initiate a result set retrieval

Description

Object oriented style (method):

```
mysqli_result 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 8.122. 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();
?>
```

Example 8.123. 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 example will output:

```
my_user@localhost
-----------------
Amersfoort
Maastricht
Dordrecht
Leiden
Haarlemmermeer
```

See Also

`mysqli_real_query`
`mysqli_store_result`

8.2.6.50. `mysqli::warning_count`, `mysqli_warning_count`

- `mysqli::warning_count`

  `mysqli_warning_count`

  Returns the number of warnings from the last query for the given link

Description

Object oriented style (property):

```
 mysqli {

  int 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 8.124. 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',
        'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch')";

$mysqli->query($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();
?>
```

Example 8.125. 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',
        'Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch')";

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 example will output:

```
Warning (1264): Data truncated for column 'Name' at row 1
```

See Also

mysqli_errno
mysqli_error
mysqli_sqlstate

8.2.7. The MySQLi_STMT class (MySQLi_STMT)

Represents a prepared statement.

```
 MySQLi_STMT {
 MySQLi_STMT

      Properties

  int affected_rows ;


  int errno ;


  string error ;


  int field_count ;


  int insert_id ;


  int num_rows ;


  int param_count ;


  string sqlstate ;
Methods

  int mysqli_stmt_affected_rows(mysqli_stmt stmt);


  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);


  int mysqli_stmt_errno(mysqli_stmt stmt);


  string mysqli_stmt_error(mysqli_stmt stmt);


  bool mysqli_stmt::execute();


  bool mysqli_stmt::fetch();


  int mysqli_stmt_field_count(mysqli_stmt stmt);


  void mysqli_stmt::free_result();


  object mysqli_stmt::get_warnings(mysqli_stmt stmt);
  mixed mysqli_stmt_insert_id(mysqli_stmt stmt);

  int mysqli_stmt_num_rows(mysqli_stmt stmt);

  int mysqli_stmt_param_count(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);

  string mysqli_stmt_sqlstate(mysqli_stmt stmt);

  bool mysqli_stmt::store_result();
}
```

8.2.7.1. `mysqli_stmt->affected_rows`, `mysqli_stmt_affected_rows`

- `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 (property):

```
mysqli_stmt {
  int 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 where 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 8.126. 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();
}

/* 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 8.127. 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();
}

/* 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 example will output:

```
rows inserted: 17
```

See Also

`mysqli_stmt_num_rows`
`mysqli_prepare`

8.2.7.2. `mysqli_stmt::attr_get`, `mysqli_stmt_attr_get`

- `mysqli_stmt::attr_get`

  `mysqli_stmt_attr_get`

  Used to get the current value of a statement attribute

Description

Object oriented style (method):

```
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.

8.2.7.3. `mysqli_stmt::attr_set`, `mysqli_stmt_attr_set`

- `mysqli_stmt::attr_set`

  `mysqli_stmt_attr_set`

  Used to modify the behavior of a prepared statement

Description

Object oriented style (method):

```
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 8.8. Attribute values

Character	Description
MYSQLI_STMT_ATTR_UPDATE_MAX_LENGTH	If set to 1, causes mysqli_stmt_store_result to update the metadata MYSQL_FIELD->max_length value.
MYSQLI_STMT_ATTR_CURSOR_TYPE	Type of cursor to open for statement when mysqli_stmt_execute is invoked. *mode* can be MYSQLI_CURSOR_TYPE_NO_CURSOR (the default) or MYSQLI_CURSOR_TYPE_READ_ONLY.
MYSQLI_STMT_ATTR_PREFETCH_ROWS	Number of rows to fetch from server at a time when using a cursor. *mode* 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.

8.2.7.4. mysqli_stmt::bind_param, mysqli_stmt_bind_param

- mysqli_stmt::bind_param

  mysqli_stmt_bind_param

  Binds variables to a prepared statement as parameters

Description

Object oriented style (method):

```
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.

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 8.9. 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 8.128. 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();
}

$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 8.129. 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 example 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`

8.2.7.5. `mysqli_stmt::bind_result`, `mysqli_stmt_bind_result`

- `mysqli_stmt::bind_result`

  `mysqli_stmt_bind_result`

  Binds variables to a prepared statement for result storage

Description

Object oriented style (method):

```
  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 8.130. 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 8.131. 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 example will output:

```
AFG Afghanistan
ALB Albania
DZA Algeria
ASM American Samoa
AND Andorra
```

See Also

`mysqli_stmt_bind_param`
`mysqli_stmt_execute`
`mysqli_stmt_fetch`
`mysqli_prepare`
`mysqli_stmt_prepare`
`mysqli_stmt_init`
`mysqli_stmt_errno`
`mysqli_stmt_error`

8.2.7.6. `mysqli_stmt::close`, `mysqli_stmt_close`

- `mysqli_stmt::close`

  `mysqli_stmt_close`

  Closes a prepared statement

Description

Object oriented style (method):

```
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

8.2.7.7. mysqli_stmt::data_seek, mysqli_stmt_data_seek

- mysqli_stmt::data_seek

  mysqli_stmt_data_seek

  Seeks to an arbitrary row in statement result set

Description

Object oriented style (method):

```
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 8.132. 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 8.133. 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 example will output:

```
City: Benin City  Countrycode: NGA
```

See Also

`mysqli_prepare`

8.2.7.8. `mysqli_stmt->errno`, `mysqli_stmt_errno`

- `mysqli_stmt->errno`

mysqli_stmt_errno

Returns the error code for the most recent statement call

Description

Object oriented style (property):

```
mysqli_stmt {

  int 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 8.134. 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 8.135. 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 example will output:

```
Error: 1146.
```

See Also

mysqli_stmt_error
mysqli_stmt_sqlstate

8.2.7.9. mysqli_stmt->error, mysqli_stmt_error

- mysqli_stmt->error

  mysqli_stmt_error

  Returns a string description for last statement error

Description

Object oriented style (property):

```
 mysqli_stmt {

  string error ;

}
```

Procedural style:

```
  string mysqli_stmt_error(mysqli_stmt stmt);
```

Returns a 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 8.136. 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 8.137. 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 example will output:

```
Error: Table 'world.myCountry' doesn't exist.
```

See Also

`mysqli_stmt_errno`
`mysqli_stmt_sqlstate`

8.2.7.10. `mysqli_stmt->execute`, `mysqli_stmt_execute`

- `mysqli_stmt->execute`

  `mysqli_stmt_execute`

  Executes a prepared Query

Description

Object oriented style (method):

```
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 appropiate 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 8.138. 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();
}

$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 8.139. 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();
}

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 example will output:

```
Stuttgart (DEU,Baden-Wuerttemberg)
Bordeaux (FRA,Aquitaine)
```

See Also

`mysqli_prepare`
`mysqli_stmt_bind_param`

8.2.7.11. `mysqli_stmt::fetch`, `mysqli_stmt_fetch`

- `mysqli_stmt::fetch`

  `mysqli_stmt_fetch`

  Fetch results from a prepared statement into the bound variables

Description

Object oriented style (method):

```
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 8.10. Return Values

Value	Description
TRUE	Success. Data has been fetched
FALSE	Error occured
NULL	No more rows/data exists or data truncation occurred

Examples

Example 8.140. 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 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 8.141. 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 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 example 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`

8.2.7.12. `mysqli_stmt->field_count`, `mysqli_stmt_field_count`

- `mysqli_stmt->field_count`

  `mysqli_stmt_field_count`

  Returns the number of field in the given statement

Description

```
 mysqli_stmt {
  int field_count ;
}
```

```
  int mysqli_stmt_field_count(mysqli_stmt stmt);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

8.2.7.13. `stmt::free_result`, `mysqli_stmt_free_result`

- `stmt::free_result`

  `mysqli_stmt_free_result`

  Frees stored result memory for the given statement handle

Description

Object oriented style (method):

```
  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`

8.2.7.14. `mysqli_stmt::get_warnings`, `mysqli_stmt_get_warnings`

- `mysqli_stmt::get_warnings`

  `mysqli_stmt_get_warnings`

Description

```
object mysqli_stmt::get_warnings(mysqli_stmt stmt);
```

```
object mysqli_stmt_get_warnings(mysqli_stmt stmt);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

8.2.7.15. `mysqli_stmt->insert_id`, `mysqli_stmt_insert_id`

- `mysqli_stmt->insert_id`

  `mysqli_stmt_insert_id`

  Get the ID generated from the previous INSERT operation

Description

```
 mysqli_stmt {
  int insert_id ;
}
```

```
mixed mysqli_stmt_insert_id(mysqli_stmt stmt);
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

8.2.7.16. `mysqli_stmt::num_rows`, `mysqli_stmt_num_rows`

- `mysqli_stmt::num_rows`

  `mysqli_stmt_num_rows`

  Return the number of rows in statements result set

Description

Object oriented style (property):

```
mysqli_stmt {

  int 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 8.142. 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 8.143. 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 example will output:

```
Number of rows: 20.
```

See Also

`mysqli_stmt_affected_rows`
`mysqli_prepare`
`mysqli_stmt_store_result`

8.2.7.17. `mysqli_stmt->param_count`, `mysqli_stmt_param_count`

- `mysqli_stmt->param_count`

  `mysqli_stmt_param_count`

  Returns the number of parameter for the given statement

Description

Object oriented style (property):

```
 mysqli_stmt {

  int 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 8.144. 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();
}

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 8.145. 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 example will output:

```
Statement has 2 markers.
```

See Also

mysqli_prepare

8.2.7.18. `mysqli_stmt::prepare`, `mysqli_stmt_prepare`

- `mysqli_stmt::prepare`

  `mysqli_stmt_prepare`

  Prepare a SQL statement for execution

Description

Object oriented style (method)

```
mixed mysqli_stmt::prepare(string query);
```

Procedure 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.

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 Languange (DML) statements, and not in Data Definition Language (DDL) statements.

Return Values

Returns `TRUE` on success or `FALSE` on failure.

Examples

Example 8.146. 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 8.147. 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 example 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_close.

8.2.7.19. mysqli_stmt::reset, mysqli_stmt_reset

- mysqli_stmt::reset

  mysqli_stmt_reset

  Resets a prepared statement

Description

Object oriented style (method):

```
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.

For now this is mainly used to reset data sent with mysqli_stmt_send_long_data.

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

8.2.7.20. mysqli_stmt::result_metadata, mysqli_stmt_result_metadata

- mysqli_stmt::result_metadata

  mysqli_stmt_result_metadata

  Returns result set metadata from a prepared statement

Description

Object oriented style (method):

```
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 occured.

Examples

Example 8.148. 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 8.149. 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`

8.2.7.21. `mysqli_stmt::send_long_data`, `mysqli_stmt_send_long_data`

- `mysqli_stmt::send_long_data`

  `mysqli_stmt_send_long_data`

  Send data in blocks

Description

Object oriented style (method)

```
  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

`stmt`	Procedural style only: A statement identifier returned by `mysqli_stmt_init`.
`param_nr`	Indicates which parameter to associate the data with. Parameters are numbered beginning with 0.
`data`	A string containing data to be sent.

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 8.150. 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

8.2.7.22. mysqli_stmt::sqlstate, mysqli_stmt_sqlstate

- mysqli_stmt::sqlstate

  mysqli_stmt_sqlstate

  Returns SQLSTATE error from previous statement operation

Description

Object oriented style (property):

```
 mysqli_stmt {

  string 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 8.151. 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 8.152. 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 example will output:

```
Error: 42S02.
```

See Also

mysqli_stmt_errno
mysqli_stmt_error

8.2.7.23. mysqli_stmt::store_result, mysqli_stmt_store_result

- mysqli_stmt::store_result

  mysqli_stmt_store_result

  Transfers a result set from a prepared statement

Description

Object oriented style (method):

```
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`), 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 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 8.153. 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 8.154. 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 example will output:

```
Number of rows: 20.
```

See Also

`mysqli_prepare`
`mysqli_stmt_result_metadata`
`mysqli_stmt_fetch`

8.2.8. The MySQLi_Result class (`MySQLi_Result`)

Represents the result set obtained from a query against the database.

```
 MySQLi_Result {
 MySQLi_Result

      Properties

  int current_field ;


  int field_count ;


  array lengths ;


  int num_rows ;

Methods

  int mysqli_field_tell(mysqli_result result);


  bool mysqli_result::data_seek(int offset);


  mixed mysqli_result::fetch_all(int resulttype);


  mixed mysqli_result::fetch_array(int resulttype);


  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,
                                     array params);


  mixed mysqli_result::fetch_row();


  int mysqli_num_fields(mysqli_result result);


  bool mysqli_result::field_seek(int fieldnr);


  void mysqli_result::free();


  array mysqli_fetch_lengths(mysqli_result result);


  int mysqli_num_rows(mysqli_result result);

}
```

8.2.8.1. `mysqli_result->current_field`, `mysqli_field_tell`

- `mysqli_result->current_field`

  `mysqli_field_tell`

  Get current field offset of a result pointer

Description

Object oriented style (property):

```
mysqli_result {

  int 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

`result`	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.

Return Values

Returns current offset of field cursor.

Examples

Example 8.155. 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 8.156. 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 example 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`

8.2.8.2. `mysqli_result::data_seek`, `mysqli_data_seek`

- `mysqli_result::data_seek`

  `mysqli_data_seek`

  Adjusts the result pointer to an arbitary row in the result

Description

Object oriented style (method):

```
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 8.157. 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 8.158. 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 example 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

8.2.8.3. mysqli_result::fetch_all, mysqli_fetch_all

- mysqli_result::fetch_all

  mysqli_fetch_all

  Fetches all result rows as an associative array, a numeric array, or both

Description

Object oriented style (method):

```
  mixed mysqli_result::fetch_all(int resulttype);
```

Procedural style:

```
  mixed mysqli_fetch_all(mysqli_result result,
                         int resulttype);
```

Available only with mysqlnd.

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 . Defaults to MYSQLI_NUM .

Return Values

Returns an array of associative or numeric arrays holding result rows.

See Also

mysqli_fetch_array
mysqli_query

8.2.8.4. mysqli_result::fetch_array, mysqli_fetch_array

- mysqli_result::fetch_array

  mysqli_fetch_array

  Fetch a result row as an associative, a numeric array, or both

Description

Object oriented style (method):

```
mixed mysqli_result::fetch_array(int resulttype);
```

Procedural style:

```
mixed mysqli_fetch_array(mysqli_result result,
                         int resulttype);
```

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 . Defaults to 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 8.159. 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->close();

/* close connection */
$mysqli->close();
?>
```

Example 8.160. 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 example will output:

```
Kabul (AFG)
Qandahar (AFG)
Herat (AFG)
```

See Also

mysqli_fetch_assoc
mysqli_fetch_row
mysqli_fetch_object
mysqli_query
mysqli_data_seek

8.2.8.5. `mysqli_result::fetch_assoc`, `mysqli_fetch_assoc`

- `mysqli_result::fetch_assoc`

  `mysqli_fetch_assoc`

  Fetch a result row as an associative array

Description

Object oriented style (method):

```
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 8.161. 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->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 8.162. 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 example will output:

```
Pueblo (USA)
Arvada (USA)
Cape Coral (USA)
Green Bay (USA)
Santa Clara (USA)
```

See Also

`mysqli_fetch_array`
`mysqli_fetch_row`
`mysqli_fetch_object`
`mysqli_query`
`mysqli_data_seek`

8.2.8.6. `mysqli_result::fetch_field_direct`, `mysqli_fetch_field_direct`

- `mysqli_result::fetch_field_direct`

  `mysqli_fetch_field_direct`

  Fetch meta-data for a single field

Description

Object oriented style (method):

```
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 informations from specified resultset.

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 8.11. 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 tabl 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 8.163. 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 8.164. 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 example will output:

```
Name:     SurfaceArea
Table:    Country
max. Len: 10
Flags:    32769
Type:     4
```

See Also

`mysqli_num_fields`
`mysqli_fetch_field`
`mysqli_fetch_fields`

8.2.8.7. `mysqli_result::fetch_field`, `mysqli_fetch_field`

- `mysqli_result::fetch_field`

  `mysqli_fetch_field`

  Returns the next field in the result set

Description

Object oriented style (method):

```
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 8.12. 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	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 integer fields)

Examples

Example 8.165. 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 8.166. 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 example 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

8.2.8.8. mysqli_result::fetch_fields, mysqli_fetch_fields

- mysqli_result::fetch_fields

  mysqli_fetch_fields

  Returns an array of objects representing the fields in a result set

Description

Object oriented style (method):

```
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 8.13. 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	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 integer fields)

Examples

Example 8.167. 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 */
    $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("Flags:    %d\n", $val->flags);
        printf("Type:     %d\n\n", $val->type);
    }
    $result->close();
}

/* close connection */
$mysqli->close();
?>
```

Example 8.168. 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 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("Flags:    %d\n", $val->flags);
        printf("Type:     %d\n\n", $val->type);
    }
    mysqli_free_result($result);
}

/* close connection */
mysqli_close($link);
?>
```

The above example 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_field`

8.2.8.9. `mysqli_result::fetch_object`, `mysqli_fetch_object`

- `mysqli_result::fetch_object`

  `mysqli_fetch_object`

Returns the current row of a result set as an object

Description

Object oriented style (method):

```
object mysqli_result::fetch_object(string class_name,
                                   array params);
```

Procedural style:

```
object mysqli_fetch_object(mysqli_result result,
                           string class_name,
                           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.

Parameters

`result`	Procedural style only: A result set identifier returned by `mysqli_query`, `mysqli_store_result` or `mysqli_use_result`.
`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 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.

ChangeLog

Version	Description
5.0.0	Added the ability to return as a different object.

Examples

Example 8.169. 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 8.170. 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 example 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`

8.2.8.10. `mysqli_result::fetch_row`, `mysqli_fetch_row`

- `mysqli_result::fetch_row`

  `mysqli_fetch_row`

  Get a result row as an enumerated array

Description

Object oriented style (method):

```
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 8.171. 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 8.172. 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 example 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`

8.2.8.11. `mysqli_result->field_count`, `mysqli_num_fields`

- `mysqli_result->field_count`

  `mysqli_num_fields`

  Get the number of fields in a result

Description

Object oriented style (property):

```
 mysqli_result {

  int 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 8.173. 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 8.174. 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 example will output:

```
Result set has 5 fields.
```

See Also

`mysqli_fetch_field`

8.2.8.12. `mysqli_result::field_seek`, `mysqli_field_seek`

- `mysqli_result::field_seek`

  `mysqli_field_seek`

  Set result pointer to a specified field offset

Description

Object oriented style (method):

```
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 8.175. 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 8.176. 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 example will output:

```
Name:     SurfaceArea
Table:    Country
max. Len: 10
Flags:    32769
Type:     4
```

See Also

`mysqli_fetch_field`

8.2.8.13. `mysqli_result::free`, `mysqli_free_result`

- `mysqli_result::free`

  `mysqli_free_result`

  Frees the memory associated with a result

Description

Object oriented style (all methods are equivalent):

```
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`

8.2.8.14. `mysqli_result->lengths`, `mysqli_fetch_lengths`

- `mysqli_result->lengths`

  `mysqli_fetch_lengths`

  Returns the lengths of the columns of the current row in the result set

Description

Object oriented style (property):

```
 mysqli_result {
  array 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 8.177. 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 8.178. 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 example 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
```

8.2.8.15. `mysqli_result->num_rows`, `mysqli_num_rows`

- `mysqli_result->num_rows`

  `mysqli_num_rows`

  Gets the number of rows in a result

Description

Object oriented style (property):

```
 mysqli_result {

  int num_rows ;

}
```

Procedural style:

```
  int mysqli_num_rows(mysqli_result result);
```

Returns the number of rows in the result set.

The use of `mysqli_num_rows` depends on whether you use buffered or unbuffered result sets. In case you use unbuffered result-sets `mysqli_num_rows` will not correct 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 maximal int value, the number will be returned as a string.

Examples

Example 8.179. 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 8.180. 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 example will output:

```
Result set has 239 rows.
```

See Also

`mysqli_affected_rows`
`mysqli_store_result`
`mysqli_use_result`
`mysqli_query`

8.2.9. The MySQLi_Driver class (`MySQLi_Driver`)

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);
}
```

`client_info`	The Client API header version
`client_version`	The Client version
`driver_version`	The MySQLi Driver version
`embedded`	Wether MySQLi Embedded support is enabled
`reconnect`	Allow or prevent reconnect (see the mysqli.reconnect INI directive)
`report_mode`	Set to `MYSQLI_REPORT_STRICT` to throw Exceptions for errors

8.2.9.1. `mysqli_driver::embedded_server_end`, `mysqli_embedded_server_end`

- `mysqli_driver::embedded_server_end`

  `mysqli_embedded_server_end`

  Stop embedded server

Description

```
  void mysqli_driver::embedded_server_end();
```

```
  void mysqli_embedded_server_end();
```

> **Warning**
>
> This function is currently not documented; only its argument list is available.

8.2.9.2. `mysqli_driver::embedded_server_start`, `mysqli_embedded_server_start`

- `mysqli_driver::embedded_server_start`

  `mysqli_embedded_server_start`

  Initialize and start embedded server

Description

```
  bool mysqli_driver::embedded_server_start(bool start,
                                            array arguments,
                                            array groups);
```

```
  bool mysqli_embedded_server_start(bool start,
                                    array arguments,
                                    array groups);
```

> **Warning**

> This function is currently not documented; only its argument list is available.

8.2.10. Aliases and deprecated Mysqli Functions

8.2.10.1. `mysqli_bind_param`

- `mysqli_bind_param`

  Alias for `mysqli_stmt_bind_param`

Description

This function is an alias of `mysqli_stmt_bind_param`.

Notes

> **Note**
>
> `mysqli_bind_param` is deprecated and will be removed.

See Also

`mysqli_stmt_bind_param`

8.2.10.2. `mysqli_bind_result`

- `mysqli_bind_result`

  Alias for `mysqli_stmt_bind_result`

Description

This function is an alias of `mysqli_stmt_bind_result`.

Notes

> **Note**
>
> `mysqli_bind_result` is deprecated and will be removed.

See Also

`mysqli_stmt_bind_result`

8.2.10.3. `mysqli_client_encoding`

- `mysqli_client_encoding`

  Alias of `mysqli_character_set_name`

Description

This function is an alias of `mysqli_character_set_name`.

See Also

`mysqli_real_escape_string`

8.2.10.4. `mysqli_disable_reads_from_master`, `mysqli->disable_reads_from_master`

- `mysqli_disable_reads_from_master`

  `mysqli->disable_reads_from_master`

  Disable reads from master

Description

Procedural style:

```
bool mysqli_disable_reads_from_master(mysqli link);
```

Object oriented style (method):

```
mysqli {
  void disable_reads_from_master();
}
```

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.

8.2.10.5. `mysqli_disable_rpl_parse`

- `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.

8.2.10.6. `mysqli_enable_reads_from_master`

- `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.

8.2.10.7. `mysqli_enable_rpl_parse`

- `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.

8.2.10.8. `mysqli_escape_string`

- `mysqli_escape_string`

  Alias of `mysqli_real_escape_string`

Description

This function is an alias of `mysqli_real_escape_string`.

See Also

`mysqli_real_escape_string`

8.2.10.9. `mysqli_execute`

- `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`

8.2.10.10. `mysqli_fetch`

- `mysqli_fetch`

  Alias for `mysqli_stmt_fetch`

Description

This function is an alias of `mysqli_stmt_fetch`.

Notes

> **Note**
>
> `mysqli_fetch` is deprecated and will be removed.

See Also

`mysqli_stmt_fetch`

8.2.10.11. `mysqli_get_metadata`

- `mysqli_get_metadata`

  Alias for `mysqli_stmt_result_metadata`

Description

This function is an alias of `mysqli_stmt_result_metadata`.

Notes

> **Note**
>
> `mysqli_get_metadata` is deprecated and will be removed.

See Also

`mysqli_stmt_result_metadata`

8.2.10.12. `mysqli_master_query`

- `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.

8.2.10.13. `mysqli_param_count`

- `mysqli_param_count`

  Alias for `mysqli_stmt_param_count`

Description

This function is an alias of `mysqli_stmt_param_count`.

Notes

Note

`mysqli_param_count` is deprecated and will be removed.

See Also

`mysqli_stmt_param_count`

8.2.10.14. `mysqli_report`

- `mysqli_report`

  Enables or disables internal report functions

Description

```
bool mysqli_report(int flags);
```

`mysqli_report` is a powerful function to improve your queries and code during development and testing phase. Depending on the flags it reports errors from mysqli function calls or queries which don't use an index (or use a bad index).

Parameters

`flags`

Table 8.14. Supported flags

Name	Description
`MYSQLI_REPORT_OFF`	Turns reporting off
`MYSQLI_REPORT_ERROR`	Report errors from mysqli function calls
`MYSQLI_REPORT_STRICT`	Report warnings from mysqli function calls
`MYSQLI_REPORT_INDEX`	Report if no index or bad index was used in a query
`MYSQLI_REPORT_ALL`	Set all options (report all)

Return Values

Returns TRUE on success or FALSE on failure.

Examples

Example 8.181. Object oriented style

```
<?php
/* activate reporting */
mysqli_report(MYSQLI_REPORT_ALL);

$mysqli = new mysqli("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 warning */
$result = $mysqli->query("SELECT Name FROM City WHERE population > 50000");
$result->close();

$mysqli->close();
?>
```

See Also

`mysqli_debug`
`mysqli_dump_debug_info`

8.2.10.15. `mysqli_rpl_parse_enabled`

- `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.

8.2.10.16. `mysqli_rpl_probe`

- `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.

8.2.10.17. `mysqli_rpl_query_type`, `mysqli->rpl_query_type`

- `mysqli_rpl_query_type`

  `mysqli->rpl_query_type`

  Returns RPL query type

Description

Procedural style:

```
int mysqli_rpl_query_type(mysqli link,
                          string query);
```

Object oriented style (method)

```
mysqli {

 int rpl_query_type(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.

8.2.10.18. `mysqli_send_long_data`

- `mysqli_send_long_data`

  Alias for `mysqli_stmt_send_long_data`

Description

This function is an alias of `mysqli_stmt_send_long_data`.

Notes

Note

`mysqli_send_long_data` is deprecated and will be removed.

See Also

`mysqli_stmt_send_long_data`

8.2.10.19. mysqli_send_query, mysqli->send_query

- mysqli_send_query

  mysqli->send_query

  Send the query and return

Description

Procedural style:

```
  bool mysqli_send_query(mysqli link,
                         string query);
```

Object oriented style (method)

```
 mysqli {
  bool send_query(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.

8.2.10.20. mysqli_set_opt

- mysqli_set_opt

  Alias of mysqli_options

Description

This function is an alias of mysqli_options.

8.2.10.21. mysqli_slave_query

- 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.

8.3. MySQL Functions (PDO_MYSQL)

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 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 mysql-specific attributes with the postgres 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 8.182. 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() instead
}
?>
```

`PDO::MYSQL_ATTR_LOCAL_INFILE` (integer)	Enable `LOAD LOCAL INFILE`.
`PDO::MYSQL_ATTR_INIT_COMMAND` (integer)	Command to execute when connecting to the MySQL server. Will automatically be re-executed when reconnecting.
`PDO::MYSQL_ATTR_READ_DEFAULT_FILE` (integer)	Read options from the named option file instead of from `my.cnf`.
`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` .
`PDO::MYSQL_ATTR_MAX_BUFFER_SIZE` (integer)	Maximum buffer size. Defaults to 1 MiB.
`PDO::MYSQL_ATTR_DIRECT_QUERY` (integer)	Perform direct queries, don't use prepared statements.

8.3.1. `PDO_MYSQL DSN`

- `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 **`mysql:`**.
`host`	The hostname on which the database server resides.
`port`	The port number where the database server is listening.
`dbname`	The name of the database.
`unix_socket`	The MySQL Unix socket (shouldn't be used with `host` or `port`).

Examples

Example 8.183. 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
```

8.4. Connector/PHP

The MySQL Connector/PHP is a version of the `mysql` and `mysqli` extensions for PHP optimized for the Windows operating system. Later versions of the main PHP `mysql`/`mysqli` drivers are compatible with Windows and a separate, Windows specific driver is no longer required.

For PHP for all platforms, including Windows, you should use the `mysql` or `mysqli` extensions shipped with the PHP sources. See Chapter 8, *MySQL PHP API*.

8.5. 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 `--old-passwords`. (See `Client does not support authentication protocol`, for more information.)

Those with PHP4 legacy code can make use of a compatibility layer for the old and new MySQL libraries, such as this one: http://www.coggeshall.org/oss/mysql2i.

8.6. Enabling Both `mysql` and `mysqli` in PHP

If you're experiencing problems with enabling both the `mysql` and the `mysqli` extension when building PHP on Linux yourself, you should try the following procedure.

1. Configure PHP like this:

   ```
   ./configure --with-mysqli=/usr/bin/mysql_config --with-mysql=/usr
   ```

2. Edit the `Makefile` and search for a line that starts with `EXTRA_LIBS`. It might look like this (all on one line):

   ```
   EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl
   -lxml2 -lz -lm -lxml2 -lz -lm -lmysqlclient -lz -lcrypt -lnsl -lm
   -lxml2 -lz -lm -lcrypt -lxml2 -lz -lm -lcrypt
   ```

   Remove all duplicates, so that the line looks like this (all on one line):

   ```
   EXTRA_LIBS = -lcrypt -lcrypt -lmysqlclient -lz -lresolv -lm -ldl -lnsl
   -lxml2
   ```

3. Build and install PHP:

   ```
   make
   make install
   ```

MySQL Enterprise
MySQL Enterprise subscribers will find more information about the mysqli extension in the Knowledge Base articles found at mysqli. Access to the MySQL Knowledge Base collection of articles is one of the advantages of subscribing to MySQL Enterprise. For more information, see http://www.mysql.com/products/enterprise/advisors.html.

Appendix A. MySQL Connector/ODBC (MyODBC) Change History

A.1. Changes in MySQL Connector/ODBC 5.1.6 (Not yet released)

Bugs fixed:

- Connector/ODBC failed to build with MySQL 5.1.30 due to incorrect use of the data type `bool`. (Bug#42120)
- Calling `SQLDescribeCol()` with a NULL buffer and nonzero buffer length caused a crash. (Bug#41942)
- MySQL Connector/ODBC updated some fields with random values, rather than with `NULL`. (Bug#41256)
- Calling `SQLDriverConnect()` with a `NULL` pointer for the output buffer caused a crash if `SQL_DRIVER_NOPROMPT` was also specified:

  ```
  SQLDriverConnect(dbc, NULL, "DSN=myodbc5", SQL_NTS, NULL, 0, NULL, SQL_DRIVER_NOPROMPT)
  ```

  (Bug#40316)
- Setting the ADO `Recordset` decimal field value to 44.56 resulted in an incorrect value of 445600.0000 being stored when the record set was updated with the `Update` method. (Bug#39961)
- The `SQLTablesW` API gave incorrect results. For example, table name and table type were returned as `NULL` rather than as the correct values. (Bug#39957)
- MyODBC would crash when a character set was being used on the server that was not supported in the client, for example cp1251:

  ```
  [MySQL][ODBC 5.1 Driver][mysqld-5.0.27-community-nt]Restricted data type attribute violation
  ```

  The fix causes MyODBC to return an error message instead of crashing. (Bug#39831)
- When the `SQLTables` method was called with `NULL` passed as the `tablename` parameter, only one row in the `resultset`, with table name of `NULL` was returned, instead of all tables for the given database. (Bug#39561)
- The `SQLGetInfo()` function returned 0 for `SQL_CATALOG_USAGE` information. (Bug#39560)
- MyODBC Driver 5.1.5 was not able to connect if the connection string parameters contained spaces or tab symbols. For example, if the `SERVER` parameter was specified as “SERVER= localhost” instead of “SERVER=localhost” the following error message will be displayed:

  ```
  [MySQL][ODBC 5.1 Driver] Unknown MySQL server host ' localhost' (11001).
  ```

  (Bug#39085)
- The pointer passed to the `SQLDriverConnect` method to retrieve the output connection string length was one greater than it should have been due to the inclusion of the NULL terminator. (Bug#38949)
- Data-at-execution parameters were not supported during positioned update. This meant updating a long text field with a cursor update would erroneously set the value to null. This would lead to the error `Column 'column_name' cannot be null` while updating the database, even when `column_name` had been assigned a valid non-null string. (Bug#37649)
- The `SQLDriverConnect` method truncated the `OutputConnectionString` parameter to 52 characters. (Bug#37278)
- The connection string option `Enable Auto-reconnect` did not work. When the connection failed, it could not be restored, and the errors generated were the same as if the option had not been selected. (Bug#37179)
- Insertion of data into a `LONGTEXT` table field did not work. If such an attempt was made the corresponding field would be found to be empty on examination, or contain random characters. (Bug#36071)
- No result record was returned for `SQLGetTypeInfo` for the `TIMESTAMP` data type. An application would receive the result `return code 100 (SQL_NO_DATA_FOUND)`. (Bug#30626)
- It was not possible to use Connector/ODBC to connect to a server using SSL. The following error was generated:

  ```
  Runtime error '-2147467259 (80004005)':
  ```

```
[MySQL][ODBC 3.51 Driver]SSL connection error.
```

(Bug#29955)

- When the `recordSet.Update` function was called to update an `adLongVarChar` field, the field was updated but the recordset was immediately lost. This happened with driver cursors, whether the cursor was opened in optimistic or pessimistic mode.

  When the next update was called the test code would exit with the following error:

  ```
  -2147467259 : Query-based update failed because the row to update could not be found.
  ```

  (Bug#26950)

A.2. Changes in MySQL Connector/ODBC 5.1.5 (18 August 2008)

Bugs fixed:

- ODBC `TIMESTAMP` string format is not handled properly by the MyODBC driver. When passing a `TIMESTAMP` or `DATE` to MyODBC, in the ODBC format: {d <date>} or {ts <timestamp>}, the string that represents this is copied once into the SQL statement, and then added again, as an escaped string. (Bug#37342)
- The connector failed to prompt for additional information required to create a DSN-less connection from an application such as Microsoft Excel. (Bug#37254)
- `SQLDriverConnect` does not return `SQL_NO_DATA` on cancel. The ODBC documentation specifies that this method should return `SQL_NO_DATA` when the user cancels the dialog to connect. The connector, however, returns `SQL_ERROR`. (Bug#36293)
- Assigning a string longer than 67 characters to the `TableType` parameter resulted in a buffer overrun when the `SQLTables()` function was called. (Bug#36275)
- The ODBC connector randomly uses logon information stored in `odbc-profile`, or prompts the user for connection information and ignores any settings stored in `odbc-profile`. (Bug#36203)
- After having successfully established a connection, a crash occurs when calling `SQLProcedures()` followed by `SQLFreeStmt()`, using the ODBC C API. (Bug#36069)

A.3. Changes in MySQL Connector/ODBC 5.1.4 (15 April 2008)

Bugs fixed:

- Wrong result obtained when using `sum()` on a `decimal(8,2)` field type. (Bug#35920)
- The driver installer could not create a new DSN if many other drivers were already installed. (Bug#35776)
- The `SQLColAttribute()` function returned `SQL_TRUE` when querying the `SQL_DESC_FIXED_PREC_SCALE (SQL_COLUMN_MONEY)` attribute of a `DECIMAL` column. Previously, the correct value of `SQL_FALSE` was returned; this is now again the case. (Bug#35581)
- On Linux, `SQLGetDiagRec()` returned `SQL_SUCCESS` in cases when it should have returned `SQL_NO_DATA`. (Bug#33910)
- The driver crashes ODBC Administrator on attempting to add a new DSN. (Bug#32057)

A.4. Changes in MySQL Connector/ODBC 5.1.3 (26 March 2008)

Platform specific notes:

- **Important Change**: You must uninstall previous 5.1.x editions of Connector/ODBC before installing the new version.

- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- The installer for 64-bit Windows installs both the 32-bit and 64-bit driver. Please note that Microsoft does not yet supply a 64-bit bridge from ADO to ODBC.

Bugs fixed:

- **Important Change**: In previous versions, the SSL certificate would automatically be verified when used as part of the Connector/ODBC connection. The default mode is now to ignore the verificate of certificates. To enforce verification of the SSL certificate during connection, use the `SSLVERIFY` DSN parameter, setting the value to 1. (Bug#29955, Bug#34648)
- Inserting characters to a UTF8 table using surrogate pairs would fail and insert invalid data. (Bug#34672)
- Installation of Connector/ODBC would fail because it was unable to uninstall a previous installed version. The file being requested would match an older release version than any installed version of the connector. (Bug#34522)
- Using `SqlGetData` in combination with `SQL_C_WCHAR` would return overlapping data. (Bug#34429)
- Descriptor records were not cleared correctly when calling `SQLFreeStmt(SQL_UNBIND)`. (Bug#34271)
- The dropdown selection for databases on a server when creating a DSN was too small. The list size now automatically adjusts up to a maximum size of 20 potential databases. (Bug#33918)
- Microsoft Access would be unable to use `DBEngine.RegisterDatabase` to create a DSN using the Connector/ODBC driver. (Bug#33825)
- Connector/ODBC erroneously reported that it supported the `CAST()` and `CONVERT()` ODBC functions for parsing values in SQL statements, which could lead to bad SQL generation during a query. (Bug#33808)
- Using a linked table in Access 2003 where the table has a `BIGINT` column as the first column in the table, and is configured as the primary key, shows `#DELETED` for all rows of the table. (Bug#24535)
- Updating a `RecordSet` when the query involves a `BLOB` field would fail. (Bug#19065)

A.5. Changes in MySQL Connector/ODBC 5.1.2 (13 February 2008)

MySQL Connector/ODBC 5.1.2-beta, a new version of the ODBC driver for the MySQL database management system, has been released. This release is the second beta (feature-complete) release of the new 5.1 series and is suitable for use with any MySQL server version since MySQL 4.1, including MySQL 5.0, 5.1, and 6.0. (It will not work with 4.0 or earlier releases.)

Keep in mind that this is a beta release, and as with any other pre-production release, caution should be taken when installing on production level systems or systems with critical data.

Platform specific notes:

- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- The installer for 64-bit Windows installs both the 32-bit and 64-bit driver. Please note that Microsoft does not yet supply a 64-bit bridge from ADO to ODBC.
- Due to differences with the installation process used on Windows and potential registry corruption, it is recommended that uninstall any existing versions of Connector/ODBC 5.1.x before upgrading.

  See also Bug#34571.

Functionality added or changed:

- Explicit descriptors are implemented. (Bug#32064)

- A full implementation of SQLForeignKeys based on the information available from INFORMATION_SCHEMA in 5.0 and later versions of the server has been implemented.
- Changed `SQL_ATTR_PARAMSET_SIZE` to return an error until support for it is implemented.
- Disabled `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` when using an SSL connection.
- `SQLForeignKeys` uses `INFORMATION_SCHEMA` when it is available on the server, which allows more complete information to be returned.

Bugs fixed:

- The `SSLCIPHER` option would be incorrectly recorded within the SSL configuration on Windows. (Bug#33897)
- Within the GUI interface, when connecting to a MySQL server on a non-standard port, the connection test within the GUI would fail. The issue was related to incorrect parsing of numeric values within the DSN when the option was not configured as the last parameter within the DSN. (Bug#33822)
- Specifying a non-existent database name within the GUI dialog would result in an empty list, not an error. (Bug#33615)
- When deleting rows from a static cursor, the cursor position would be incorrectly reported. (Bug#33388)
- `SQLGetInfo()` reported characters for `SQL_SPECIAL_CHARACTERS` that were not encoded correctly. (Bug#33130)
- Retrieving data from a `BLOB` column would fail within `SQLGetData`when the target data type was `SQL_C_WCHAR` due to incorrect handling of the character buffer. (Bug#32684)
- Renaming an existing DSN entry would create a new entry with the new name without deleting the old entry. (Bug#31165)
- Reading a `TEXT` column that had been used to store UTF8 data would result in the wrong information being returned during a query. (Bug#28617)
- `SQLForeignKeys` would return an empty string for the schema columns instead of `NULL`. (Bug#19923)
- When accessing column data, `FLAG_COLUMN_SIZE_S32` did not limit the octet length or display size reported for fields, causing problems with Microsoft Visual FoxPro.

  The list of ODBC functions that could have caused failures in Microsoft software when retrieving the length of `LONGBLOB` or `LONGTEXT` columns includes:

  - `SQLColumns`
  - `SQLColAttribute`
  - `SQLColAttributes`
  - `SQLDescribeCol`
  - `SQLSpecialColumns` (theoretically can have the same problem)

  (Bug#12805, Bug#30890)
- Dynamic cursors on statements with parameters were not supported. (Bug#11846)
- Evaluating a simple numeric expression when using the OLEDB for ODBC provider and ADO would return an error, instead of the result. (Bug#10128)
- Adding or updating a row using `SQLSetPos()` on a result set with aliased columns would fail. (Bug#6157)

A.6. Changes in MySQL Connector/ODBC 5.1.1 (13 December 2007)

MySQL Connector/ODBC 5.1.1-beta, a new version of the ODBC driver for the MySQL database management system, has been released. This release is the first beta (feature-complete) release of the new 5.1 series and is suitable for use with any MySQL server version since MySQL 4.1, including MySQL 5.0, 5.1, and 6.0. (It will not work with 4.0 or earlier releases.)

Keep in mind that this is a beta release, and as with any other pre-production release, caution should be taken when installing on production level systems or systems with critical data.

Includes changes from Connector/ODBC 3.51.21 and 3.51.22.

Built using MySQL 5.0.52.

Platform specific notes:

- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- The installer for 64-bit Windows installs both the 32-bit and 64-bit driver. Please note that Microsoft does not yet supply a 64-bit bridge from ADO to ODBC.
- Due to differences with the installation process used on Windows and potential registry corruption, it is recommended that uninstall any existing versions of Connector/ODBC 5.1.x before upgrading.

  See also Bug#34571.

Functionality added or changed:

- **Incompatible Change**: Replaced myodbc3i (now myodbc-installer) with Connector/ODBC 5.0 version.
- **Incompatible Change**: Removed monitor (myodbc3m) and dsn-editor (myodbc3c).
- **Incompatible Change**: Disallow `SET NAMES` in initial statement and in executed statements.
- A wrapper for the `SQLGetPrivateProfileStringW()` function, which is required for Unicode support, has been created. This function is missing from the unixODBC driver manager. (Bug#32685)
- Added MSI installer for Windows 64-bit. (Bug#31510)
- Implemented support for `SQLCancel()`. (Bug#15601)
- Added support for `SQL_NUMERIC_STRUCT`. (Bug#3028, Bug#24920)
- Removed non-threadsafe configuration of the driver. The driver is now always built against the threadsafe version of libmysql.
- Implemented native Windows setup library
- Replaced the internal library which handles creation and loading of DSN information. The new library, which was originally a part of Connector/ODBC 5.0, supports Unicode option values.
- The Windows installer now places files in a subdirectory of the `Program Files` directory instead of the Windows system directory.

Bugs fixed:

- The `SET NAMES` statement has been disabled because it causes problems in the ODBC driver when determining the current client character set. (Bug#32596)
- `SQLDescribeColW` returned UTF-8 column as `SQL_VARCHAR` instead of `SQL_WVARCHAR`. (Bug#32161)
- ADO was unable to open record set using dynamic cursor. (Bug#32014)
- ADO applications would not open a `RecordSet` that contained a `DECIMAL` field. (Bug#31720)
- Memory usage would increase considerably. (Bug#31115)
- SQL statements are limited to 64KB. (Bug#30983, Bug#30984)
- `SQLSetPos` with `SQL_DELETE` advances dynamic cursor incorrectly. (Bug#29765)
- Using an ODBC prepared statement with bound columns would produce an empty result set when called immediately after inserting a row into a table. (Bug#29239)
- ADO Not possible to update a client side cursor. (Bug#27961)
- Recordset `Update()` fails when using `adUseClient` cursor. (Bug#26985)

- Connector/ODBC would fail to connect to the server if the password contained certain characters, including the semicolon and other punctuation marks. (Bug#16178)
- Fixed `SQL_ATTR_PARAM_BIND_OFFSET`, and fixed row offsets to work with updatable cursors.
- `SQLSetConnectAttr()` did not clear previous errors, possibly confusing `SQLError()`.
- `SQLError()` incorrectly cleared the error information, making it unavailable from subsequent calls to `SQLGetDiagRec()`.
- NULL pointers passed to `SQLGetInfo()` could result in a crash.
- `SQL_ODBC_SQL_CONFORMANCE` was not handled by `SQLGetInfo()`.
- `SQLCopyDesc()` did not correctly copy all records.
- Diagnostics were not correctly cleared on connection and environment handles.

A.7. Changes in MySQL Connector/ODBC 5.1.0 (10 September 2007)

This release is the first of the new 5.1 series and is suitable for use with any MySQL server version since MySQL 4.1, including MySQL 5.0, 5.1, and 6.0. (It will not work with 4.0 or earlier releases.)

Keep in mind that this is a alpha release, and as with any other pre-production release, caution should be taken when installing on production level systems or systems with critical data. Not all of the features planned for the final Connector/ODBC 5.1 release are implemented.

Functionality is based on Connector/ODBC 3.51.20.

Platform specific notes:

- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- There are no installer packages for Microsoft Windows x64 Edition.
- Due to differences with the installation process used on Windows and potential registry corruption, it is recommended that uninstall any existing versions of Connector/ODBC 5.1.x before upgrading.

  See also Bug#34571.

Functionality added or changed:

- Added support for Unicode functions (`SQLConnectW`, etc).
- Added descriptor support (`SQLGetDescField`, `SQLGetDescRec`, etc).
- Added support for `SQL_C_WCHAR`.

A.8. Changes in MySQL Connector/ODBC 5.0.12 (Never released)

Note

Development on Connector/ODBC 5.0.x has ceased. New features and functionality will be incorporated into Connector/ODBC 5.1. See Section 1.2.1, “Connector/ODBC Roadmap”.

Bugs fixed:

- Inserting `NULL` values into a `DATETIME` column from Access reports an error. (Bug#27896)
- Tables with `TEXT` columns would be incorrectly identified, returning an `Unknown SQL type - 65535` error. (Bug#20127)

A.9. Changes in MySQL Connector/ODBC 5.0.11 (31 January 2007)

Functionality added or changed:

- Added support for ODBC v2 statement options using attributes.
- Driver now builds and is partially tested under Linux with the iODBC driver manager.

Bugs fixed:

- Connection string parsing for DSN-less connections could fail to identify some parameters. (Bug#25316)
- Updates of `MEMO` or `TEXT` columns from within Microsoft Access would fail. (Bug#25263)
- Transaction support has been added and tested. (Bug#25045)
- Internal function, `my_setpos_delete_ignore()` could cause a crash. (Bug#22796)
- Fixed occasional mis-handling of the `SQL_NUMERIC_C` type.
- Fixed the binding of certain integer types.

A.10. Changes in MySQL Connector/ODBC 5.0.10 (14 December 2006)

Connector/ODBC 5.0.10 is the sixth BETA release.

Functionality added or changed:

- Significant performance improvement when retrieving large text fields in pieces using `SQLGetData()` with a buffer smaller than the whole data. Mainly used in Access when fetching very large text fields. (Bug#24876)
- Added initial unicode support in data and metadata. (Bug#24837)
- Added initial support for removing braces when calling stored procedures and retrieving result sets from procedure calls. (Bug#24485)
- Added loose handling of retrieving some diagnostic data. (Bug#15782)
- Added wide-string type info for `SQLGetTypeInfo()`.

Bugs fixed:

- Editing DSN no longer crashes ODBC data source administrator. (Bug#24675)
- String query parameters are new escaped correctly. (Bug#19078)

A.11. Changes in MySQL Connector/ODBC 5.0.9 (22 November 2006)

Connector/ODBC 5.0.9 is the fifth BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Added support for column binding as SQL_NUMBERIC_STRUCT.
- Added recognition of `SQL_C_SHORT` and `SQL_C_TINYINT` as C types.

Bugs fixed:

- Fixed wildcard handling of and listing of catalogs and tables in `SQLTables`.
- Added limit of display size when requested via `SQLColAttribute`/`SQL_DESC_DISPLAY_SIZE`.
- Fixed buffer length return for SQLDriverConnect.
- ODBC v2 behaviour in driver now supports ODBC v3 date/time types (since DriverManager maps them).
- Catch use of `SQL_ATTR_PARAMSET_SIZE` and report error until we fully support.
- Fixed statistics to fail if it couldn't be completed.
- Corrected retrieval multiple field types bit and blob/text.
- Fixed SQLGetData to clear the NULL indicator correctly during multiple calls.

A.12. Changes in MySQL Connector/ODBC 5.0.8 (17 November 2006)

Connector/ODBC 5.0.8 is the fourth BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Also made `SQL_DESC_NAME` only fill in the name if there was a data pointer given, otherwise just the length.
- Fixed display size to be length if max length isn’t available.
- Made distinction between `CHAR`/`BINARY` (and VAR versions).
- Wildcards now support escaped chars and underscore matching (needed to link tables with underscores in access).

Bugs fixed:

- Fixed binding using `SQL_C_LONG`.
- Fixed using wrong pointer for `SQL_MAX_DRIVER_CONNECTIONS` in `SQLGetInfo`.
- Set default return to `SQL_SUCCESS` if nothing is done for `SQLSpecialColumns`.
- Fixed MDiagnostic to use correct v2/v3 error codes.
- Allow SQLDescribeCol to be called to retrieve the length of the column name, but not the name itself.
- Length now used when handling bind parameter (needed in particular for `SQL_WCHAR`) - this enables updating char data in MS Access.
- Updated retrieval of descriptor fields to use the right pointer types.
- Fixed hanlding of numeric pointers in SQLColAttribute.
- Fixed type returned for `MYSQL_TYPE_LONG` to `SQL_INTEGER` instead of `SQL_TINYINT`.
- Fix size return from `SQLDescribeCol`.
- Fixed string length to chars, not bytes, returned by SQLGetDiagRec.

A.13. Changes in MySQL Connector/ODBC 5.0.7 (08 November 2006)

Connector/ODBC 5.0.7 is the third BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Functionality added or changed:

- Added support for `SQLStatistics` to `MYODBCShell`.
- Improved trace/log.

Bugs fixed:

- SQLBindParameter now handles `SQL_C_DEFAULT`.
- Corrected incorrect column index within `SQLStatistics`. Many more tables can now be linked into MS Access.
- Fixed `SQLDescribeCol` returning column name length in bytes rather than chars.

A.14. Changes in MySQL Connector/ODBC 5.0.6 (03 November 2006)

Connector/ODBC 5.0.6 is the second BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release

- Connector/ODBC supports both `User` and `System` DSNs.
- Installation is provided in the form of a standard Microsoft System Installer (MSI).
- You no longer have to have Connector/ODBC 3.51 installed before installing this version.

Bugs fixed:

- You no longer have to have Connector/ODBC 3.51 installed before installing this version.
- Connector/ODBC supports both `User` and `System` DSNs.
- Installation is provided in the form of a standard Microsoft System Installer (MSI).

A.15. Changes in MySQL Connector/ODBC 5.0.5 (17 October 2006)

Connector/ODBC 5.0.5 is the first BETA release.

This is an implementation and testing release, and is not designed for use within a production environment.

You no longer have to have Connector/ODBC 3.51 installed before installing this version.

Bugs fixed:

- You no longer have to have Connector/ODBC 3.51 installed before installing this version.

A.16. Changes in Connector/ODBC 5.0.3 (Connector/ODBC 5.0 Alpha 3) (20 June 2006)

This is an implementation and testing release, and is not designed for use within a production environment.

Features, limitations and notes on this release:

- The following ODBC API functions have been added in this release:
  - `SQLBindParameter`
  - `SQLBindCol`

A.17. Changes in Connector/ODBC 5.0.2 (Never released)

Connector/ODBC 5.0.2 was an internal implementation and testing release.

A.18. Changes in Connector/ODBC 5.0.1 (Connector/ODBC 5.0 Alpha 2) (05 June 2006)

Features, limitations and notes on this release:

- Connector/ODBC 5.0 is Unicode aware.
- Connector/ODBC is currently limited to basic applications. ADO applications and Microsoft Office are not supported.
- Connector/ODBC must be used with a Driver Manager.
- The following ODBC API functions are implemented:
  - `SQLAllocHandle`
  - `SQLCloseCursor`
  - `SQLColAttribute`
  - `SQLColumns`
  - `SQLConnect`
  - `SQLCopyDesc`
  - `SQLDisconnect`
  - `SQLExecDirect`
  - `SQLExecute`
  - `SQLFetch`
  - `SQLFreeHandle`
  - `SQLFreeStmt`
  - `SQLGetConnectAttr`
  - `SQLGetData`
  - `SQLGetDescField`
  - `SQLGetDescRec`
  - `SQLGetDiagField`
  - `SQLGetDiagRec`
  - `SQLGetEnvAttr`
  - `SQLGetFunctions`
  - `SQLGetStmtAttr`
  - `SQLGetTypeInfo`
  - `SQLNumResultCols`
  - `SQLPrepare`
  - `SQLRowcount`
  - `SQLTables`

  The following ODBC API function are implemented, but not yet support all the available attributes/options:

- SQLSetConnectAttr
- SQLSetDescField
- SQLSetDescRec
- SQLSetEnvAttr
- SQLSetStmtAttr

A.19. Changes in MySQL Connector/ODBC 3.51.27 (20 November 2008)

Bugs fixed:

- The client program hung when the network connection to the server was interrupted. (Bug#40407)
- The connection string option `Enable Auto-reconnect` did not work. When the connection failed, it could not be restored, and the errors generated were the same as if the option had not been selected. (Bug#37179)
- It was not possible to use Connector/ODBC to connect to a server using SSL. The following error was generated:

  ```
  Runtime error '-2147467259 (80004005)':
  [MySQL][ODBC 3.51 Driver]SSL connection error.
  ```

  (Bug#29955)

A.20. Changes in MySQL Connector/ODBC 3.51.26 (07 July 2008)

Functionality added or changed:

- There is a new connection option, `FLAG_NO_BINARY_RESULT`. When set this option disables charset 63 for columns with an empty `org_table`. (Bug#29402)

Bugs fixed:

- When an `ADOConnection` is created and attempts to open a schema with `ADOConnection.OpenSchema` an access violation occurs in `myodbc3.dll`. (Bug#30770)
- When `SHOW CREATE TABLE` was invoked and then the field values read, the result was truncated and unusable if the table had many rows and indexes. (Bug#24131)

A.21. Changes in MySQL Connector/ODBC 3.51.25 (11 April 2008)

Bugs fixed:

- The `SQLColAttribute()` function returned `SQL_TRUE` when querying the `SQL_DESC_FIXED_PREC_SCALE (SQL_COLUMN_MONEY)` attribute of a `DECIMAL` column. Previously, the correct value of `SQL_FALSE` was returned; this is now again the case. (Bug#35581)
- The driver crashes ODBC Administrator on attempting to add a new DSN. (Bug#32057)
- When accessing column data, `FLAG_COLUMN_SIZE_S32` did not limit the octet length or display size reported for fields, causing problems with Microsoft Visual FoxPro.

  The list of ODBC functions that could have caused failures in Microsoft software when retrieving the length of `LONGBLOB` or `LONGTEXT` columns includes:

  - SQLColumns

- `SQLColAttribute`
- `SQLColAttributes`
- `SQLDescribeCol`
- `SQLSpecialColumns` (theoretically can have the same problem)

(Bug#12805, Bug#30890)

A.22. Changes in MySQL Connector/ODBC 3.51.24 (14 March 2008)

Bugs fixed:

- **Security Enhancement**: Accessing a parameer with the type of `SQL_C_CHAR`, but with a numeric type and a length of zero, the parameter marker would get stropped from the query. In addition, an SQL injection was possible if the parameter value had a nonzero length and was not numeric, the text would be inserted verbatim. (Bug#34575)
- **Important Change**: In previous versions, the SSL certificate would automatically be verified when used as part of the Connector/ODBC connection. The default mode is now to ignore the verificate of certificates. To enforce verification of the SSL certificate during connection, use the `SSLVERIFY` DSN parameter, setting the value to 1. (Bug#29955, Bug#34648)
- When using ADO, the count of parameters in a query would always return zero. (Bug#33298)
- Using tables with a single quote or other non-standard characters in the table or column names through ODBC would fail. (Bug#32989)
- When using Crystal Reports, table and column names would be truncated to 21 characters, and truncated columns in tables where the truncated name was the duplicated would lead to only a single column being displayed. (Bug#32864)
- `SQLExtendedFetch()` and `SQLFetchScroll()` ignored the rowset size if the `Don't cache result` DSN option was set. (Bug#32420)
- When using the ODBC `SQL_TXN_READ_COMMITTED` option, 'dirty' records would be read from tables as if the option had not been applied. (Bug#31959)
- When creating a System DSN using the ODBC Administrator on Mac OS X, a User DSN would be created instead. The root cause is a problem with the iODBC driver manager used on Mac OS X. The fix works around this issue.

  Note

  ODBC Administrator may still be unable to register a System DSN unless the `/Library/ODBC/odbc.ini` file has the correct permissions. You should ensure that the file is writable by the `admin` group.

  (Bug#31495)
- Calling `SQLFetch` or `SQLFetchScroll` would return negative data lengths when using `SQL_C_WCHAR`. (Bug#31220)
- `SQLSetParam()` caused memory allocation errors due to driver manager's mapping of deprecated functions (buffer length -1). (Bug#29871)
- Static cursor was unable to be used through ADO when dynamic cursors were enabled. (Bug#27351)
- Using `connection.Execute` to create a record set based on a table without declaring the cmd option as `adCmdTable` will fail when communicating with versions of MySQL 5.0.37 and higher. The issue is related to the way that `SQLSTATE` is returned when ADO tries to confirm the existence of the target object. (Bug#27158)
- Updating a `RecordSet` when the query involves a `BLOB` field would fail. (Bug#19065)
- With some connections to MySQL databases using Connector/ODBC, the connection would mistakenly report 'user cancelled' for accesses to the database information. (Bug#16653)

A.23. Changes in MySQL Connector/ODBC 3.51.23 (09 January 2008)

Platform specific notes:

- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- There are no installer packages for Microsoft Windows x64 Edition.

Bugs fixed:

- Connector/ODBC would incorrectly return `SQL_SUCCESS` when checking for distributed transaction support. (Bug#32727)
- When using unixODBC or directly linked applications where the thread level is set to less than 3 (within `odbcinst.ini`), a thread synchronization issue would lead to an application crash. This was because `SQLAllocStmt()` and `SQLFreeStmt()` did not synchronize access to the list of statements associated with a connection. (Bug#32587)
- Cleaning up environment handles in multithread environments could result in a five (or more) second delay. (Bug#32366)
- Renaming an existing DSN entry would create a new entry with the new name without deleting the old entry. (Bug#31165)
- Setting the default database using the `DefaultDatabase` property of an ADO `Connection` object would fail with the error `Provider does not support this property`. The `SQLGetInfo()` returned the wrong value for `SQL_DATABASE_NAME` when no database was selected. (Bug#3780)

A.24. Changes in MySQL Connector/ODBC 3.51.22 (13 November 2007)

Functionality added or changed:

- The workaround for this bug was removed due to the fixes in MySQL Server 5.0.48 and 5.1.21.

  This regression was introduced by Bug#10491.

Bugs fixed:

- The `English` locale would be used when formatting floating point values. The `C` locale is now used for these values. (Bug#32294)
- When accessing information about supported operations, the driver would return incorrect information about the support for `UNION`. (Bug#32253)
- Unsigned integer values greater than the maximum value of a signed integer would be handled incorrectly. (Bug#32171)
- The wrong result was returned by `SQLGetData()` when the data was an empty string and a zero-sized buffer was specified. (Bug#30958)
- Added the `FLAG_COLUMN_SIZE_S32` option to limit the reported column size to a signed 32-bit integer. This option is automatically enabled for ADO applications to provide a work around for a bug in ADO. (Bug#13776)

A.25. Changes in MySQL Connector/ODBC 3.51.21 (08 October 2007)

Bugs fixed:

- When using a rowset/cursor and add a new row with a number of fields, subsequent rows with fewer fields will include the original fields from the previous row in the final `INSERT` statement. (Bug#31246)
- Uninitiated memory could be used when C/ODBC internally calls `SQLGetFunctions()`. (Bug#31055)
- The wrong `SQL_DESC_LITERAL_PREFIX` would be returned for date/time types. (Bug#31009)
- The wrong `COLUMN_SIZE` would be returned by `SQLGetTypeInfo` for the TIME columns (`SQL_TYPE_TIME`). (Bug#30939)

- Clicking outside the character set selection box when configuring a new DSN could cause the wrong character set to be selected. (Bug#30568)
- Not specifying a user in the DSN dialog would raise a warning even though the parameter is optional. (Bug#30499)
- `SQLSetParam()` caused memory allocation errors due to driver manager's mapping of deprecated functions (buffer length -1). (Bug#29871)
- When using ADO, a column marked as `AUTO_INCREMENT` could incorrectly report that the column allowed `NULL` values. This was dur to an issue with `NULLABLE` and `IS_NULLABLE` return values from the call to `SQLColumns()`. (Bug#26108)
- Connector/ODBC would return the wrong the error code when the server disconnects the active connection because the configured `wait_timeout` has expired. Previously it would return `HY000`. Connector/ODBC now correctly returns an `SQLSTATE` of `08S01`. (Bug#3456)

A.26. Changes in MySQL Connector/ODBC 3.51.20 (10 September 2007)

Bugs fixed:

- Using `FLAG_NO_PROMPT` doesn't suppress the dialogs normally handled by `SQLDriverConnect`. (Bug#30840)
- The specified length of the user name and authentication parameters to `SQLConnect()` were not being honored. (Bug#30774)
- The wrong column size was returned for binary data. (Bug#30547)
- `SQLGetData()` will now always return `SQL_NO_DATA_FOUND` on second call when no data left, even if requested size is 0. (Bug#30520)
- `SQLGetConnectAttr()` did not reflect the connection state correctly. (Bug#14639)
- Removed checkbox in setup dialog for `FLAG_FIELD_LENGTH` (identified as `Don't Optimize Column Width` within the GUI dialog), which was removed from the driver in 3.51.18.

A.27. Changes in MySQL Connector/ODBC 3.51.19 (10 August 2007)

Connector/ODBC 3.51.19 fixes a specific issue with the 3.51.18 release. For a list of changes in the 3.51.18 release, see Section A.28, "Changes in MySQL Connector/ODBC 3.51.18 (08 August 2007)".

Functionality added or changed:

- Because of Bug#10491 in the server, character string results were sometimes incorrectly identified as `SQL_VARBINARY`. Until this server bug is corrected, the driver will identify all variable-length strings as `SQL_VARCHAR`.

A.28. Changes in MySQL Connector/ODBC 3.51.18 (08 August 2007)

Platform specific notes:

- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- Binary packages for Sun Solaris are now available as `PKG` packages.
- Binary packages as disk images with installers are now available for Mac OS X.
- A binary package without an installer is available for Microsoft Windows x64 Edition. There are no installer packages for Microsoft Windows x64 Edition.

Functionality added or changed:

- **Incompatible Change**: The `FLAG_DEBUG` option was removed.
- When connecting to a specific database when using a DSN, the system tables from the `mysql` database are no longer also available. Previously, tables from the mysql database (catalog) were listed as `SYSTEM TABLES` by `SQLTables()` even when a different catalog was being queried. (Bug#28662)
- Installed for Mac OS X has been re-instated. The installer registers the driver at a system (not user) level and makes it possible to create both user and system DSNs using the Connector/ODBC driver. The installer also fixes the situation where the necessary drivers would bge installed local to the user, not globally. (Bug#15326, Bug#10444)
- Connector/ODBC now supports batched statements. In order to enable cached statement support you must switch enable the batched statement option (`FLAG_MULTI_STATEMENTS`, 67108864, or **Allow multiple statements** within a GUI configuration). Be aware that batched statements create an increased chance of SQL injection attacks and you must ensure that your application protects against this scenario. (Bug#7445)
- The `SQL_ATTR_ROW_BIND_OFFSET_PTR` is now supported for row bind offsets. (Bug#6741)
- The `TRACE` and `TRACEFILE` DSN options have been removed. Use the ODBC driver manager trace options instead.

Bugs fixed:

- When using a table with multiple `TIMESTAMP` columns, the final `TIMESTAMP` column within the table definition would not be updateable. Note that there is still a limitation in MySQL server regarding multiple `TIMESTAMP` columns . (Bug#9927) (Bug#30081)
- Fixed an issue where the `myodbc3i` would update the user ODBC configuration file (`~/Library/ODBC/odbcinst.ini`) instead of the system `/Library/ODBC/odbcinst.ini`. This was caused because `myodbc3i` was not honoring the `s` and `u` modifiers for the `-d` command-line option. (Bug#29964)
- Getting table metadata (through the `SQLColumns()` would fail, returning a bad table definition to calling applications. (Bug#29888)
- `DATETIME` column types would return `FALSE` in place of `SQL_SUCCESS` when requesting the column type information. (Bug#28657)
- The `SQL_COLUMN_TYPE`, `SQL_COLUMN_DISPLAY` and `SQL_COLUMN_PRECISION` values would be returned incorrectly by `SQLColumns()`, `SQLDescribeCol()` and `SQLColAttribute()` when accessing character columns, especially those generated through `concat()`. The lengths returned should now conform to the ODBC specification. The `FLAG_FIELD_LENGTH` option no longer has any affect on the results returned. (Bug#27862)
- Obtaining the length of a column when using a character set for the connection of `utf8` would result in the length being returned incorrectly. (Bug#19345)
- The `SQLColumns()` function could return incorrect information about `TIMESTAMP` columns, indicating that the field was not nullable. (Bug#14414)
- The `SQLColumns()` function could return incorrect information about `AUTO_INCREMENT` columns, indicating that the field was not nullable. (Bug#14407)
- A binary package without an installer is available for Microsoft Windows x64 Edition. There are no installer packages for Microsoft Windows x64 Edition.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- `BIT(n)` columns are now treated as `SQL_BIT` data where `n = 1` and binary data where `n > 1`.
- The wrong value from `SQL_DESC_LITERAL_SUFFIX` was returned for binary fields.
- The `SQL_DATETIME_SUB` column in SQLColumns() was not correctly set for date and time types.
- The value for `SQL_DESC_FIXED_PREC_SCALE` was not returned correctly for values in MySQL 5.0 and later.
- The wrong value for `SQL_DESC_TYPE` was returned for date and time types.
- `SQLConnect()` and `SQLDriverConnect()` were rewritten to eliminate duplicate code and ensure all options were supported using both connection methods. `SQLDriverConnect()` now only requires the setup library to be present when the call requires it.

- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- Binary packages as disk images with installers are now available for Mac OS X.
- Binary packages for Sun Solaris are now available as `PKG` packages.
- The wrong value for `DECIMAL_DIGITS` in `SQLColumns()` was reported for `FLOAT` and `DOUBLE` fields, as well as the wrong value for the scale parameter to `SQLDescribeCol()`, and the `SQL_DESC_SCALE` attribute from `SQLColAttribute()`.
- The `SQL_DATA_TYPE` column in `SQLColumns()` results did not report the correct value for date and time types.

A.29. Changes in MySQL Connector/ODBC 3.51.17 (14 July 2007)

Platform specific notes:

- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- Binary packages for Sun Solaris are now available as `PKG` packages.
- Binary packages as disk images with installers are now available for Mac OS X.
- A binary package without an installer is available for Microsoft Windows x64 Edition. There are no installer packages for Microsoft Windows x64 Edition.

Functionality added or changed:

- It is now possible to specify a different character set as part of the DSN or connection string. This must be used instead of the `SET NAMES` statement. You can also configure the character set value from the GUI configuration. (Bug#9498, Bug#6667)
- Fixed calling convention ptr and wrong free in `myodbc3i`, and fixed the null terminating (was only one, not two) when writing DSN to string.
- Dis-allow NULL ptr for null indicator when calling SQLGetData() if value is null. Now returns SQL_ERROR w/state 22002.
- The setup library has been split into its own RPM package, to allow installing the driver itself with no GUI dependencies.

Bugs fixed:

- `myodbc3i` did not correctly format driver info, which could cause the installation to fail. (Bug#29709)
- Connector/ODBC crashed with Crystal Reports due to a rproblem with `SQLProcedures()`. (Bug#28316)
- Fixed a problem where the GUI would crash when configuring or removing a System or User DSN. (Bug#27315)
- Fixed error handling of out-of-memory and bad connections in catalog functions. This might raise errors in code paths that had ignored them in the past. (Bug#26934)
- For a stored procedure that returns multiple result sets, Connector/ODBC returned only the first result set. (Bug#16817)
- Calling `SQLGetDiagField` with `RecNumber 0, DiagIdentifier NOT 0` returned `SQL_ERROR`, preventing access to diagnostic header fields. (Bug#16224)
- Added a new DSN option (`FLAG_ZERO_DATE_TO_MIN`) to retrieve `XXXX-00-00` dates as the minimum allowed ODBC date (`XXXX-01-01`). Added another option (`FLAG_MIN_DATE_TO_ZERO`) to mirror this but for bound parameters. `FLAG_MIN_DATE_TO_ZERO` only changes `0000-01-01` to `0000-00-00`. (Bug#13766)
- If there was more than one unique key on a table, the correct fields were not used in handling `SQLSetPos()`. (Bug#10563)
- When inserting a large `BLOB` field, Connector/ODBC would crash due to a memory allocation error. (Bug#10562)
- The driver was using `mysql_odbc_escape_string()`, which does not handle the `NO_BACKSLASH_ESCAPES` SQL

mode. Now it uses `mysql_real_escape_string()`, which does. (Bug#9498)

- `SQLColumns()` did not handle many of its parameters correctly, which could lead to incorrect results. The table name argument was not handled as a pattern value, and most arguments were not escaped correctly when they contained non-alphanumeric characters. (Bug#8860)
- There are no binary packages for Microsoft Windows x64 Edition.
- There is no binary package for Mac OS X on 64-bit PowerPC because Apple does not currently provide a 64-bit PowerPC version of iODBC.
- Correctly return error if `SQLBindCol` is called with an invalid column.
- Fixed possible crash if `SQLBindCol()` was not called before `SQLSetPos()`.
- The Mac OS X binary packages are only provided as tarballs, there is no installer.
- The binary packages for Sun Solaris are only provided as tarballs, not the PKG format.
- The HP-UX 11.23 IA64 binary package does not include the GUI bits because of problems building Qt on that platform.

A.30. Changes in MySQL Connector/ODBC 3.51.16 (14 June 2007)

Functionality added or changed:

- Connector/ODBC now supports using SSL for communication. This is not yet exposed in the setup GUI, but must be enabled through configuration files or the DSN. (Bug#12918)

Bugs fixed:

- Calls to SQLNativeSql() could cause stack corruption due to an incorrect pointer cast. (Bug#28758)
- Using curors on results sets with multi-column keys could select the wrong value. (Bug#28255)
- `SQLForeignKeys` does not escape `_` and `%` in the table name arguments. (Bug#27723)
- When using stored procedures, making a `SELECT` or second stored procedure call after an initial stored procedure call, the second statement will fail. (Bug#27544)
- SQLTables() did not distinguish tables from views. (Bug#23031)
- Data in `TEXT` columns would fail to be read correctly. (Bug#16917)
- Specifying strings as parameters using the `adBSTR` or `adVarWChar` types, (`SQL_WVARCHAR` and `SQL_WLONGVARCHAR`) would be incorrectly quoted. (Bug#16235)
- SQL_WVARCHAR and SQL_WLONGVARCHAR parameters were not properly quoted and escaped. (Bug#16235)
- Using `BETWEEN` with date values, the wrong results could be returned. (Bug#15773)
- When using the `Don't Cache Results` (option value `1048576`) with Microsoft Access, the connection will fail using DAO/VisualBasic. (Bug#4657)
- Return values from `SQLTables()` may be truncated. (Bugs #22797)

A.31. Changes in MySQL Connector/ODBC 3.51.15 (07 May 2007)

Bugs fixed:

- Connector/ODBC would incorrectly claim to support `SQLProcedureColumns` (by returning true when queried about `SQL-PROCEDURECOLUMNS` with `SQLGetFunctions`), but this functionality is not supported. (Bug#27591)
- An incorrect transaction isolation level may not be returned when accessing the connection attributes. (Bug#27589)

- Adding a new DSN with the `myodbc3i` utility under AIX would fail. (Bug#27220)
- When inserting data using bulk statements (through `SQLBulkOperations`), the indicators for all rows within the insert would not updated correctly. (Bug#24306)
- Using `SQLProcedures` does not return the database name within the returned resultset. (Bug#23033)
- The `SQLTransact()` function did not support an empty connection handle. (Bug#21588)
- Using `SQLDriverConnect` instead of `SQLConnect` could cause later operations to fail. (Bug#7912)
- When using blobs and parameter replacement in a statement with `WHERE CURSOR OF`, the SQL is truncated. (Bug#5853)
- Connector/ODBC would return too many foreign key results when accessing tables with similar names. (Bug#4518)

A.32. Changes in MySQL Connector/ODBC 3.51.14 (08 March 2007)

Functionality added or changed:

- Use of `SQL_ATTR_CONNECTION_TIMEOUT` on the server has now been disabled. If you attempt to set this attribute on your connection the `SQL_SUCCESS_WITH_INFO` will be returned, with an error number/string of `HYC00: Optional feature not supported`. (Bug#19823)
- Added auto is null option to Connector/ODBC option parameters. (Bug#10910)
- Added auto-reconnect option to Connector/ODBC option parameters.
- Added support for the `HENV` handlers in `SQLEndTran()`.

Bugs fixed:

- On 64-bit systems, some types would be incorrectly returned. (Bug#26024)
- When retrieving `TIME` columns, C/ODBC would incorrectly interpret the type of the string and could interpret it as a `DATE` type instead. (Bug#25846)
- Connector/ODBC may insert the wrong parameter values when using prepared statements under 64-bit Linux. (Bug#22446)
- Using Connector/ODBC, with `SQLBindCol` and binding the length to the return value from `SQL_LEN_DATA_AT_EXEC` fails with a memory allocation error. (Bug#20547)
- Using `DataAdapter`, Connector/ODBC may continually consume memory when reading the same records within a loop (Windows Server 2003 SP1/SP2 only). (Bug#20459)
- When retrieving data from columns that have been compressed using `COMPRESS()`, the retrieved data would be truncated to 8KB. (Bug#20208)
- The ODBC driver name and version number were incorrectly reported by the driver. (Bug#19740)
- A string format exception would be raised when using iODBC, Connector/ODBC and the embedded MySQL server. (Bug#16535)
- The `SQLDriverConnect()` ODBC method did not work with recent Connector/ODBC releases. (Bug#12393)

A.33. Changes in MySQL Connector/ODBC 3.51.13 (Never released)

Connector/ODBC 3.51.13 was an internal implementation and testing release.

A.34. Changes in MySQL Connector/ODBC 3.51.12 (11 February 2005)

Functionality added or changed:

- N/A

Bugs fixed:

- Using stored procedures with ADO, where the `CommandType` has been set correctly to `adCmdStoredProc`, calls to stored procedures would fail. (Bug#15635)
- File DSNs could not be saved. (Bug#12019)
- `SQLColumns()` returned no information for tables that had a column named using a reserved word. (Bug#9539)

A.35. Changes in MySQL Connector/ODBC 3.51.11 (28 January 2005)

Bugs fixed:

- `mysql_list_dbcolumns()` and `insert_fields()` were retrieving all rows from a table. Fixed the queries generated by these functions to return no rows. (Bug#8198)
- `SQLGetTypoInfo()` returned `tinyblob` for `SQL_VARBINARY` and nothing for `SQL_BINARY`. Fixed to return `varbinary` for `SQL_VARBINARY`, `binary` for `SQL_BINARY`, and `longblob` for `SQL_LONGVARBINARY`. (Bug#8138)

Appendix B. MySQL Connector/NET Change History

B.1. Changes in MySQL Connector/NET 6.0.4 (Not yet released beta)

This is a new Beta development release, fixing recently discovered bugs.

Bugs fixed:

- A SQL query string containing an escaped backslash caused an exception to be generated:

  ```
  Index and length must refer to a location within the string.
  Parameter name: length
  at System.String.InternalSubStringWithChecks(Int32 startIndex, Int32 length, Boolean
  fAlwaysCopy)
  at MySql.Data.MySqlClient.MySqlTokenizer.NextParameter()
  at MySql.Data.MySqlClient.Statement.InternalBindParameters(String sql,
  MySqlParameterCollection parameters, MySqlPacket packet)
  at MySql.Data.MySqlClient.Statement.BindParameters()
  at MySql.Data.MySqlClient.PreparableStatement.Execute()
  at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior be□havior)
  at MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
  ```

  (Bug#44960)

- The Microsoft Visual Studio solution file `MySQL-VS2005.sln` was invalid. Several projects could not be loaded and thus it was not possible to build Connector/NET from source. (Bug#44822)

- The DataReader in Connector/NET 6.0.3 considered a BINARY(16) field as a GUID with a length of 16. (Bug#44507)

- When creating a new DataSet the following error was generated:

  ```
  Failed to open a connection to database.
  Cannot load type with name 'MySQL.Data.VisualStudio.StoredProcedureColumnEnumerator'
  ```

  (Bug#44460)

- The Connector/NET MySQLRoleProvider reported that there were no roles, even when roles existed. (Bug#44414)

B.2. Changes in MySQL Connector/NET 6.0.3 (28 April 2009 beta)

This is a new Beta development release, fixing recently discovered bugs.

Functionality added or changed:

- The `MySqlTokenizer` failed to split fieldnames from values if they were not separated by a space. This also happened if the string contained certain characters. As a result `MySqlCommand.ExecuteNonQuery` raised an index out of range exception.

  The resulting errors are illustrated by the following examples. Note, the example statements do not have delimiting spaces around the `=` operator.

  ```
  INSERT INTO anytable SET Text='test--test';
  ```

  The tokenizer incorrectly interpreted the value as containing a comment.

  ```
  UPDATE anytable SET Project='123-456',Text='Can you explain this ?',Duration=15 WHERE
  ID=4711;'
  ```

  A `MySqlException` was generated, as the `?` in the value was interpreted by the tokenizer as a parameter sign. The error message generated was:

  ```
  Fatal error encountered during command execution.
  EXCEPTION: MySqlException - Parameter '?'' must be defined.
  ```

  (Bug#44318)

Bugs fixed:

- `MySQL.Data` was not displayed as a Reference inside Microsoft Visual Studio 2008 Professional.

  When a new C# project was created in Microsoft Visual Studio 2008 Professional, `MySQL.Data` was not displayed when REFERENCES, ADD REFERENCE was selected. (Bug#44141)

- Column types for `SchemaProvider` and `ISSchemaProvider` did not match.

  When the source code in `SchemaProvider.cs` and `ISSchemaProvider.cs` were compared it was apparent that they were not using the same column types. The base provider used SQL such as `SHOW CREATE TABLE`, while `ISSchemaProvider` used the schema information tables. Column types used by the base class were `INT64` and the column types used by `ISSchemaProvider` were `UNSIGNED`. (Bug#44123)

B.3. Changes in MySQL Connector/NET 6.0.2 (07 April 2009 beta)

This is a new Beta development release, fixing recently discovered bugs.

Bugs fixed:

- Connector/Net 6.0.1 did not load in Microsoft Visual Studio 2008 and Visual Studio 2005 Pro.

  The following error message was generated:

  ```
  .NET Framework Data Provider for MySQL: The data provider object factory service was not
  found.
  ```

  (Bug#44064)

B.4. Changes in MySQL Connector/NET 6.0.1 (02 April 2009 beta)

This is a new Beta development release, fixing recently discovered bugs.

Bugs fixed:

- An insert and update error was generated by the decimal data type in the Entity Framework, when a German collation was used. (Bug#43574)
- Generating an Entity Data Model (EDM) schema with a table containing columns with data types `MEDIUMTEXT` and `LONGTEXT` generated a runtime error message “Max value too long or too short for Int32”. (Bug#43480)

B.5. Changes in MySQL Connector/NET 6.0.0 (02 March 2009 alpha)

This is a new Alpha development release.

Bugs fixed:

- A null reference exception was generated when `MySqlConnection.ClearPool(connection)` was called. (Bug#42801)

B.6. Changes in MySQL Connector/NET 5.3.0 (Not yet released)

Bugs fixed:

- The Web Provider did not work at all on a remote host, and did not create a database when using `autogenerateschema="true"`. (Bug#39072)
- The Connector/NET installer program ended prematurely without reporting the specific error. (Bug#39019)
- When called with an incorrect password the `MembershipProvider.GetPassword()` method threw a `MySQLException` instead of a `MembershipPasswordException` . (Bug#38939)
- Possible overflow in `MySqlPacket.ReadLong()`. (Bug#36997)

- The `TokenizeSql` method was adding query overhead and causing high CPU utilization for larger queries. (Bug#36836)

B.7. Changes in MySQL Connector/NET 5.2.7 (Not yet released)

Bugs fixed:

- The Microsoft Visual Studio solution file `MySQL-VS2005.sln` was invalid. Several projects could not be loaded and thus it was not possible to build Connector/NET from source. (Bug#44822)
- The Connector/NET MySQLRoleProvider reported that there were no roles, even when roles existed. (Bug#44414)
- When a TableAdapter was created on a DataSet, it was not possible to use a stored procedure with variables. The following error was generated:

  ```
  The method or operation is not implemented
  ```

  (Bug#39409)

B.8. Changes in MySQL Connector/NET 5.2.6 (28 April 2009)

Functionality added or changed:

- A new connection string option has been added: `use affected rows`. When `true` the connection will report changed rows instead of found rows. (Bug#44194)

Bugs fixed:

- Calling `GetSchema()` on `Indexes` or `IndexColumns` failed where index or column names were restricted.

  In `SchemaProvider.cs`, methods `GetIndexes()` and `GetIndexColumns()` passed their restrictions directly to `GetTables()`. This only worked if the restrictions were no more specific than `schemaName` and `tableName`. If `IndexName` was given, this was passed to `GetTables()` where it was treated as `TableType`. As a result no tables were returned, unless the index name happened to be `BASE TABLE` or `VIEW`. This meant that both methods failed to return any rows. (Bug#43991)
- `GetSchema("MetaDataCollections")` should have returned a table with a column named “NumberOfRestrictions” not “NumberOfRestriction”.

  This can be confirmed by referencing the Microsoft Documentation. (Bug#43990)
- Requests sent to the Connector/NET role provider to remove a user from a role failed. The query log showed the query was correctly executed within a transaction which was immediately rolled back. The rollback was caused by a missing call to the `Complete` method of the transaction. (Bug#43553)
- When using `MySqlBulkLoader.Load()`, the text file is opened by `NativeDriver.SendFileToServer`. If it encountered a problem opening the file as a stream, an exception was generated and caught. An attempt to clean up resources was then made in the `finally{}` clause by calling `fs.Close()`, but since the stream was never successfully opened, this was an attempt to execute a method of a null reference. (Bug#43332)
- A null reference exception was generated when `MySqlConnection.ClearPool(connection)` was called. (Bug#42801)
- `MySQLMembershipProvider.ValidateUser` only used the `userId` to validate. However, it should also use the `applicationId` to perform the validation correctly.

  The generated query was, for example:

  ```
  SELECT Password, PasswordKey, PasswordFormat, IsApproved, Islockedout
  FROM my_aspnet_Membership WHERE userId=13
  ```

  Note that `applicationId` is not used. (Bug#42574)
- There was an error in the `ProfileProvider` class in the `private ProfileInfoCollection GetProfiles()`

function. The column of the final table was named “lastUpdatdDate” ('e' is missing) instead of the correct “lastUpdatedDate”. (Bug#41654)

- The `GetGuid()` method of `MySqlDataReader` did not treat `BINARY(16)` column data as a GUID. When operating on such a column a `FormatException` exception was generated. (Bug#41452)

- When ASP.NET membership was configured to not require password question and answer using `requiresQuestion-AndAnswer="false"`, a `SqlNullValueException` was generated when using `Member-shipUser.ResetPassword()` to reset the user password. (Bug#41408)

- If a `Stored Procedure` contained spaces in its parameter list, and was then called from Connector/NET, an exception was generated. However, the same `Stored Procedure` called from the MySQL Query Analyzer or the MySQL Client worked correctly.

  The exception generated was:

  ```
  Parameter '0' not found in the collection.
  ```

  (Bug#41034)

- The `DATETIME` format contained an erroneous space. (Bug#41021)

- When `MySql.Web.Profile.MySQLProfileProvider` was configured, it was not possible to assign a name other than the default name `MySQLProfileProvider`.

  If the name `SCC_MySQLProfileProvider` was assigned, an exception was generated when attempting to use `Page.Context.Profile['custom prop']`.

  The exception generated was:

  ```
  The profile default provider was not found.
  ```

  Note that the exception stated: 'the profile **default provider**...', even though a different name was explicitly requested. (Bug#40871)

- When `ExecuteNonQuery` was called with a command type of `Stored Procedure` it worked for one user but resulted in a hang for another user with the same database permissions.

  However, if `CALL` was used in the command text and `ExecuteNonQuery` was used with a command type of `Text`, the call worked for both users. (Bug#40139)

B.9. Changes in MySQL Connector/NET 5.2.5 (19 November 2008)

Bugs fixed:

- Visual Studio 2008 displayed the following error three times on start-up:

  ```
  "Package Load Failure

  Package 'MySql.Data.VisualStudio.MySqlDataProviderPackage, MySql.VisualStudio,
  Version=5.2.4, Culture=neutral, PublicKeyTopen=null' has failed to load properly (GUID =
  {79A115C9-B133-4891-9E7B-242509DAD272}).  Please contact the package vendor for
  assistance.  Application restart is recommended, due to possible environment corruption.
  Would you like to disable loading the package in the future?  You may use
  'devenve/resetskippkgs' to re-enable package loading."
  ```

  (Bug#40726)

B.10. Changes in MySQL Connector/NET 5.2.4 (13 November 2008)

Bugs fixed:

- `MySqlDataReader` did not feature a `GetSByte` method. (Bug#40571)

- When working with stored procedures Connector/NET generated an exception `Unknown "table parameters" in information_schema`. (Bug#40382)

- `GetDefaultCollation` and `GetMaxLength` were not thread safe. These functions called the database to get a set of parameters and cached them in two static dictionaries in the function `InitCollections`. However, if many threads called them they would try to insert the same keys in the collections resulting in duplicate key exceptions. (Bug#40231)
- If connection pooling was not set explicitly in the connection string, Connector/NET added “;Pooling=False” to the end of the connection string when `MySqlCommand.ExecuteReader()` was called.

  If connection pooling was explicitly set in the connection string, when `MySqlConnection.Open()` was called it converted “Pooling=True” to “pooling=True”.

  If `MySqlCommand.ExecuteReader()` was subsequently called, it concatenated “;Pooling=False” to the end of the connection string. The resulting connection string was thus terminated with “pooling=True;Pooling=False”. This disabled connection pooling completely. (Bug#40091)
- The connection string option `Functions Return String` did not set the correct encoding for the result string. Even though the connection string option `Functions Return String=true;` is set, the result of `SELECT DES_DECRYPT()` contained “??” instead of the correct national character symbols. (Bug#40076)
- If, when using the `MySqlTransaction` transaction object, an exception was thrown, the transaction object was not disposed of and the transaction was not rolled back. (Bug#39817)
- After the `ConnectionString` property was initialized via the public setter of `DbConnectionStringBuilder`, the `GetConnectionString` method of `MySqlConnectionStringBuilder` incorrectly returned `null` when `true` was assigned to the `includePass` parameter. (Bug#39728)
- When using `ProfileProvider`, attempting to update a previously saved property failed. (Bug#39330)
- Reading a negative time value greater than -01:00:00 returned the absolute value of the original time value. (Bug#39294)
- Inserting a negative time value (negative `TimeSpan`) into a `Time` column through the use of `MySqlParameter` caused `MySqlException` to be thrown. (Bug#39275)
- When a data connection was created in the server explorer of Visual Studio 2008 Team, an error was generated when trying to expand stored procedures that had parameters.

  Also, if **TableAdapter** was right-clicked and then Add, Query, Use Existing Stored Procedures selected, if you then attempted to select a stored procedure, the window would close and no error message would be displayed. (Bug#39252)
- The Web Provider did not work at all on a remote host, and did not create a database when using `autogenerateschema="true"`. (Bug#39072)
- Connector/NET called hashed password methods not supported in Mono 2.0 Preview 2. (Bug#38895)

B.11. Changes in MySQL Connector/NET 5.2.3 (19 August 2008)

Functionality added or changed:

- Error string was returned after a 28000 second `wait_timeout`. This has been changed to generate a `ConnectionState.Closed` event. (Bug#38119)
- Changed how the procedure schema collection is retrieved. If the connection string contains “`use procedure bodies=true`” then a `SELECT` is performed on the `mysql.proc` table directly, as this is up to 50 times faster than the current Information Schema implementation. If the connection string contains “`use procedure bodies=false`”, then the Information Schema collection is queried. (Bug#36694)
- Changed how the procedure schema collection is retrieved. If `use procedure bodies=true` then the `mysql.proc` table is selected directly as this is up to 50 times faster than the current `information_schema` implementation. If `use procedure bodies=false`, then the `information_schema` collection is queried. (Bug#36694)
- String escaping functionality has been moved from the `MySqlString` class to the `MySqlHelper` class, where it can be accessed by the `EscapeString` method. (Bug#36205)

Bugs fixed:

- The `GetOrdinal()` method failed to return the ordinal if the column name string contained an accent. (Bug#38721)
- Connector/Net uninstaller did not clean up all installed files. (Bug#38534)

- There was a short circuit evaluation error in the `MySqlCommand.CheckState()` method. When the statement `connection == null` was true a `NullReferenceException` was thrown and not the expected `InvalidOperationException`. (Bug#38276)
- The provider did not silently create the user if the user did not exist. (Bug#38243)
- Executing a command that resulted in a fatal exception did not close the connection. (Bug#37991)
- When a prepared insert query is run that contains an `UNSIGNED TINYINT` in the parameter list, the complete query and data that should be inserted is corrupted and no error is thrown. (Bug#37968)
- In a .NET application MySQL Connector/NET modifies the connection string so that it contains several occurrences of the same option with different values. This is illustrated by the example that follows.

  The original connection string:

  ```
  host=localhost;database=test;uid=*****;pwd=*****;
  connect timeout=25; auto enlist=false;pooling=false;
  ```

  The connection string after after closing `MySqlDataReader`:

  ```
  host=localhost;database=test;uid=*****;pwd=*****;
  connect timeout=25;auto enlist=false;pooling=false;
  Allow User Variables=True;Allow User Variables=False;
  Allow User Variables=True;Allow User Variables=False;
  ```

  (Bug#37955)
- Unnecessary network traffic was generated for the normal case where the web provider schema was up to date. (Bug#37469)
- `MySqlReader.GetOrdinal()` performance enhancements break existing functionality. (Bug#37239)
- The `autogenerateschema` option produced tables with incorrect collations. (Bug#36444)
- `GetSchema` did not work correctly when querying for a collection, if using a non-English locale. (Bug#35459)
- When reading back a stored double or single value using the .NET provider, the value had less precision than the one stored. (Bug#33322)
- Using the MySQL Visual Studio plugin and a MySQL 4.1 server, certain field types (`ENUM`) would not be identified correctly. Also, when looking for tables, the plugin would list all tables matching a wildcard pattern of the database name supplied in the connection string, instead of only tables within the specified database. (Bug#30603)

B.12. Changes in MySQL Connector/NET 5.2.2 (12 May 2008)

Bugs fixed:

- Product documentation incorrectly stated '?' is the preferred parameter marker. (Bug#37349)
- An incorrect value for a bit field would returned in a multi-row query if a preceding value for the field returned `NULL`. (Bug#36313)
- Tables with `GEOMETRY` field types would return an unknown datatype exception. (Bug#36081)
- When using the `MySQLProfileProvider`, setting profile details and then reading back saved data would result in the default values being returned instead of the updated values. (Bug#36000)
- When creating a connection, setting the `ConnectionString` property of `MySqlConnection` to `NULL` would throw an exception. (Bug#35619)
- The `DbCommandBuilder.QuoteIdentifer` method was not implemented. (Bug#35492)
- When using encrypted passwords, the `GetPassword()` function would return the wrong string. (Bug#35336)
- An error would be raised when calling `GetPassword()` with a `NULL` value. (Bug#35332)
- When retreiving data where a field has been identified as containing a GUID value, the incorrect value would be returned when a previous row contained a `NULL` value for that field. (Bug#35041)

- Using the `TableAdapter Wizard` would fail when generating commands that used stored procedures due to the change in supported parameter characters. (Bug#34941)
- When creating a new stored procedured, the new parameter code which allows the use of the `@` symbol would interfere with the specification of a `DEFINER`. (Bug#34940)
- When using `SqlDataSource` to open a connection, the connection would not automatically be closed when access had completed. (Bug#34460)
- There was a high level of contention in the connection pooling code that could lead to delays when opening connections and submitting queries. The connection pooling code has been modified to try and limit the effects of the contention issue. (Bug#34001)
- Using the `TableAdaptor` wizard in combination with a suitable `SELECT` statement, only the associated `INSERT` statement would also be created, rather than the required `DELETE` and `UPDATE` statements. (Bug#31338)
- Fixed problem in datagrid code related to creating a new table. This problem may have been introduced with .NET 2.0 SP1.
- Fixed profile provider that would throw an exception if you were updating a profile that already existed.

B.13. Changes in MySQL Connector/NET 5.2.1 (27 February 2008)

Bugs fixed:

- When using the provider to generate or update users and passwords, the password checking algorithm would not validate the password strength or requirements correctly. (Bug#34792)
- When executing statements that used stored procedures and functions, the new parameter code could fail to identify the correct parameter format. (Bug#34699)
- The installer would fail to the DDEX provider binary if the Visual Studio 2005 component was not selected. The result would lead to Connector/NET not loading properly when using the interface to a MySQL server within Visual Studio. (Bug#34674)
- A number issues were identified in the case, connection and scema areas of the code for `MembershipProvider`, `RoleProvider`, `ProfileProvider`. (Bug#34495)
- When using web providers, the Connector/NET would check the schema and cache the application id, even when the connection string had been set. The effect would be to break the memvership provider list. (Bug#34451)
- Attempting to use an isolation level other than the default with a transaction scope would use the default isolation level. (Bug#34448)
- When altering a stored procedure within Visual Studio, the parameters to the procedure could be lost. (Bug#34359)
- A race condition could occur within the procedure cache resulting the cache contents overflowing beyond the configured cache size. (Bug#34338)
- Fixed problem with Visual Studio 2008 integration that caused pop-up menus on server explorer nodes to not function
- The provider code has been updated to fix a number of outstanding issues.

B.14. Changes in MySQL Connector/NET 5.2.0 (11 February 2008)

Functionality added or changed:

- Performing `GetValue()` on a field `TINYINT(1)` returned a `BOOLEAN`. While not a bug, this caused problems in software that expected an `INT` to be returned. A new connection string option `Treat Tiny As Boolean` has been added with a default value of `true`. If set to `false` the provider will treat `TINYINT(1)` as `INT`. (Bug#34052)
- Added support for `DbDataAdapter` `UpdateBatchSize`. Batching is fully supported including collapsing inserts down into the multi-value form if possible.
- DDEX provider now works under Visual Studio 2008 beta 2.
- Added ClearPool and ClearAllPools features.

Bugs fixed:

- Some speed improvements have been implemented in the `TokenizeSql` process used to identify elements of SQL statements. (Bug#34220)
- When accessing tables from different databases within the same `TransactionScope`, the same user/password combination would be used for each database connection. Connector/NET does not handle multiple connections within the same transaction scope. An error is now returned if you attempt this process, instead of using the incorrect authorization information. (Bug#34204)
- The status of connections reported through the state change handler was not being updated correctly. (Bug#34082)
- Incorporated some connection string cache optimizations sent to us by Maxim Mass. (Bug#34000)
- In an open connection where the server had disconnected unexpectedly, the status information of the connection would not be updated properly. (Bug#33909)
- Data cached from the connection string could return invalid information because the internal routines were not using case-sensitive semantics. This lead to updated connection string options not being recognized if they were of a different case than the existing cached values. (Bug#31433)
- Column name metadata was not using the character set as deifned within the connection string being used. (Bug#31185)
- Memory usage could increase and decrease significantly when updating or inserting a large number of rows. (Bug#31090)
- Commands executed from within the state change handeler would fail with a `NULL` exception. (Bug#30964)
- When running a stored procedure multiple times on the same connection, the memory usage could increase indefinitely. (Bug#30116)
- Using compression in the MySQL connection with Connector/NET would be slower than using native (uncompressed) communication. (Bug#27865)
- The `MySqlDbType.Datetime` has been replaced with `MySqlDbType.DateTime`. The old format has been obsoleted. (Bug#26344)

B.15. Changes in MySQL Connector/NET 5.1.8 (Not yet released)

Bugs fixed:

- Calling `GetSchema()` on `Indexes` or `IndexColumns` failed where index or column names were restricted.

  In `SchemaProvider.cs`, methods `GetIndexes()` and `GetIndexColumns()` passed their restrictions directly to `GetTables()`. This only worked if the restrictions were no more specific than `schemaName` and `tableName`. If `IndexName` was given, this was passed to `GetTables()` where it was treated as `TableType`. As a result no tables were returned, unless the index name happened to be `BASE TABLE` or `VIEW`. This meant that both methods failed to return any rows. (Bug#43991)
- The `DATETIME` format contained an erroneous space. (Bug#41021)
- If connection pooling was not set explicitly in the connection string, Connector/NET added “;Pooling=False” to the end of the connection string when `MySqlCommand.ExecuteReader()` was called.

  If connection pooling was explicitly set in the connection string, when `MySqlConnection.Open()` was called it converted “Pooling=True” to “pooling=True”.

  If `MySqlCommand.ExecuteReader()` was subsequently called, it concatenated “;Pooling=False” to the end of the connection string. The resulting connection string was thus terminated with “pooling=True;Pooling=False”. This disabled connection pooling completely. (Bug#40091)
- If, when using the `MySqlTransaction` transaction object, an exception was thrown, the transaction object was not disposed of and the transaction was not rolled back. (Bug#39817)
- When a prepared insert query is run that contains an `UNSIGNED TINYINT` in the parameter list, the complete query and data that should be inserted is corrupted and no error is thrown. (Bug#37968)

B.16. Changes in MySQL Connector/NET 5.1.7 (21 August 2008)

Bugs fixed:

- There was a short circuit evaluation error in the `MySqlCommand.CheckState()` method. When the statement `connection == null` was true a `NullReferenceException` was thrown and not the expected `InvalidOperationException`. (Bug#38276)
- Executing a command that resulted in a fatal exception did not close the connection. (Bug#37991)
- In a .NET application MySQL Connector/NET modifies the connection string so that it contains several occurrences of the same option with different values. This is illustrated by the example that follows.

  The original connection string:

  ```
  host=localhost;database=test;uid=*****;pwd=*****;
  connect timeout=25; auto enlist=false;pooling=false;
  ```

  The connection string after after closing `MySqlDataReader`:

  ```
  host=localhost;database=test;uid=*****;pwd=*****;
  connect timeout=25;auto enlist=false;pooling=false;
  Allow User Variables=True;Allow User Variables=False;
  Allow User Variables=True;Allow User Variables=False;
  ```

  (Bug#37955)
- As `MySqlDbType.DateTime` is not available in `VB.Net` the warning `THE DATETIME ENUM VALUE IS OBSOLETE` was always shown during compilation. (Bug#37406)
- An unknown `MySqlErrorCode` was encountered when opening a connection with an incorrect password. (Bug#37398)
- Documentation incorrectly stated that “the DataColumn class in .NET 1.0 and 1.1 does not allow columns with type of UInt16, UInt32, or UInt64 to be autoincrement columns”. (Bug#37350)
- `SemaphoreFullException` is generated when application is closed. (Bug#36688)
- `GetSchema` did not work correctly when querying for a collection, if using a non-English locale. (Bug#35459)
- When reading back a stored double or single value using the .NET provider, the value had less precision than the one stored. (Bug#33322)
- Using the MySQL Visual Studio plugin and a MySQL 4.1 server, certain field types (`ENUM`) would not be identified correctly. Also, when looking for tables, the plugin would list all tables matching a wildcard pattern of the database name supplied in the connection string, instead of only tables within the specified database. (Bug#30603)

B.17. Changes in MySQL Connector/NET 5.1.6 (12 May 2008)

Bugs fixed:

- When creating a connection pool, specifying an invalid IP address will cause the entire application to crash, instead of providing an exception. (Bug#36432)
- An incorrect value for a bit field would returned in a multi-row query if a preceding value for the field returned `NULL`. (Bug#36313)
- The `MembershipProvider` will raise an exception when the connection string is configured with `enablePasswordRetrival = true` and `RequireQuestionAndAnswer = false`. (Bug#36159)
- When calling `GetNumberOfUsersOnline` an exception is raised on the submitted query due to a missing parameter. (Bug#36157)
- Tables with `GEOMETRY` field types would return an unknown datatype exception. (Bug#36081)
- When creating a connection, setting the `ConnectionString` property of `MySqlConnection` to `NULL` would throw an exception. (Bug#35619)
- The `DbCommandBuilder.QuoteIdentifer` method was not implemented. (Bug#35492)

- When using `SqlDataSource` to open a connection, the connection would not automatically be closed when access had completed. (Bug#34460)
- Attempting to use an isolation level other than the default with a transaction scope would use the default isolation level. (Bug#34448)
- When altering a stored procedure within Visual Studio, the parameters to the procedure could be lost. (Bug#34359)
- A race condition could occur within the procedure cache resulting the cache contents overflowing beyond the configured cache size. (Bug#34338)
- Using the `TableAdaptor` wizard in combination with a suitable `SELECT` statement, only the associated `INSERT` statement would also be created, rather than the required `DELETE` and `UPDATE` statements. (Bug#31338)

B.18. Changes in MySQL Connector/NET 5.1.5 (Not yet released)

Functionality added or changed:

- Performing `GetValue()` on a field `TINYINT(1)` returned a `BOOLEAN`. While not a bug, this caused problems in software that expected an `INT` to be returned. A new connection string option `Treat Tiny As Boolean` has been added with a default value of `true`. If set to `false` the provider will treat `TINYINT(1)` as `INT`. (Bug#34052)

Bugs fixed:

- Some speed improvements have been implemented in the `TokenizeSql` process used to identify elements of SQL statements. (Bug#34220)
- When accessing tables from different databases within the same `TransactionScope`, the same user/password combination would be used for each database connection. Connector/NET does not handle multiple connections within the same transaction scope. An error is now returned if you attempt this process, instead of using the incorrect authorization information. (Bug#34204)
- The status of connections reported through the state change handler was not being updated correctly. (Bug#34082)
- Incorporated some connection string cache optimizations sent to us by Maxim Mass. (Bug#34000)
- In an open connection where the server had disconnected unexpectedly, the status information of the connection would not be updated properly. (Bug#33909)
- Connector/NET would fail to compile properly with `nant`. (Bug#33508)
- Problem with membership provider would mean that `FindUserByEmail` would fail with a `MySqlException` because it was trying to add a second parameter with the same name as the first. (Bug#33347)
- Using compression in the MySQL connection with Connector/NET would be slower than using native (uncompressed) communication. (Bug#27865)

B.19. Changes in MySQL Connector/NET 5.1.4 (20 November 2007)

Bugs fixed:

- Setting the size of a string parameter after the value could cause an exception. (Bug#32094)
- Creation of parameter objects with non-input direction using a constructor would fail. This was cause by some old legacy code preventing their use. (Bug#32093)
- A date string could be returned incorrectly by `MySqlDataTime.ToString()` when the date returned by MySQL was `0000-00-00 00:00:00`. (Bug#32010)
- A syntax error in a set of batch statements could leave the data adapter in a state that appears hung. (Bug#31930)
- Installing over a failed uninstall of a previous version could result in multiple clients being registered in the `machine.config`. This would prevent certain aspects of the MySQL connection within Visual Studio to work properly. (Bug#31731)

- Connector/NET would incorrectly report success when enlisting in a distributed transaction, although distributed transactions are not supported. (Bug#31703)
- Data cached from the connection string could return invalid information because the internal routines were not using case-sensitive semantics. This lead to updated connection string options not being recognized if they were of a different case than the existing cached values. (Bug#31433)
- Trying to use a connection that was not open could return an ambiguous and misleading error message. (Bug#31262)
- Column name metadata was not using the character set as deifned within the connection string being used. (Bug#31185)
- Memory usage could increase and decrease significantly when updating or inserting a large number of rows. (Bug#31090)
- Commands executed from within the state change handeler would fail with a `NULL` exception. (Bug#30964)
- Extracting data through XML functions within a query returns the data as `System.Byte[]`. This was due to Connector/NET incorrectly identifying `BLOB` fields as binary, rather than text. (Bug#30233)
- When running a stored procedure multiple times on the same connection, the memory usage could increase indefinitely. (Bug#30116)
- Column types with only 1-bit (such as `BOOLEAN` and `TINYINT(1)` were not returned as boolean fields. (Bug#27959)
- When accessing certain statements, the command would timeout before the command completed. Because this cannot always be controlled through the individual command timeout options, a `default command timeout` has been added to the connection string options. (Bug#27958)
- The server error code was not updated in the `Data[]` hash, which prevented `DbProviderFactory` users from accessing the server error code. (Bug#27436)
- The `MySqlDbType.Datetime` has been replaced with `MySqlDbType.DateTime`. The old format has been obsoleted. (Bug#26344)
- Changing the connection string of a connection to one that changes the parameter marker after the connection had been assigned to a command but before the connection is opened could cause parameters to not be found. (Bug#13991)

B.20. Changes in MySQL Connector/NET 5.1.3 (21 September 2007 beta)

This is a new Beta development release, fixing recently discovered bugs.

Bugs fixed:

- An incorrect `ConstraintException` could be raised on an `INSERT` when adding rows to a table with a multiple-column unique key index. (Bug#30204)
- A `DATE` field would be updated with a date/time value, causing a `MySqlDataAdapter.Update()` exception. (Bug#30077)
- The Saudi Hijri calendar was not supported. (Bug#29931)
- Calling `SHOW CREATE PROCEDURE` for routines with a hyphen in the catalog name produced a syntax error. (Bug#29526)
- Connecting to a MySQL server earlier than version 4.1 would raise a `NullException`. (Bug#29476)
- The availability of a MySQL server would not be reset when using pooled connections (`pooling=true`). This would lead to the server being reported as unavailable, even if the server become available while the application was still running. (Bug#29409)
- A `FormatException` error would be raised if a parameter had not been found, instead of `Resources.ParameterMustBeDefined`. (Bug#29312)
- An exception would be thrown when using the Manage Role functionality within the web administrator to assign a role to a user. (Bug#29236)
- Using the membership/role providers when `validationKey` or `decryptionKey` parameters are set to `AutoGenerate`, an exception would be raised when accessing the corresponding values. (Bug#29235)

- Certain operations would not check the `UsageAdvisor` setting, causing log messages from the Usage Advisor even when it was disabled. (Bug#29124)
- Using the same connection string multiple times would result in `Database=*dbname*` appearing multiple times in the resulting string. (Bug#29123)
- *Visual Studio Plugin*: Adding a new query based on a stored procedure that uses the `SELECT` statement would terminate the query/TableAdapter wizard. (Bug#29098)
- Using `TransactionScope` would cause an `InvalidOperationException`. (Bug#28709)

B.21. Changes in MySQL Connector/NET 5.1.2 (18 June 2007)

This is a new Beta development release, fixing recently discovered bugs.

Bugs fixed:

- Log messages would be truncated to 300 bytes. (Bug#28706)
- Creating a user would fail due to the application name being set incorrectly. (Bug#28648)
- *Visual Studio Plugin*: Adding a new query based on a stored procedure that used a `UPDATE`, `INSERT` or `DELETE` statement would terminate the query/TableAdapter wizard. (Bug#28536)
- *Visual Studio Plugin*: Query Builder would fail to show `TINYTEXT` columns, and any columns listed after a `TINYTEXT` column correctly. (Bug#28437)
- Accessing the results from a large query when using data compression in the connection would fail to return all the data. (Bug#28204)
- *Visual Studio Plugin*: Update commands would not be generated correctly when using the TableAdapter wizard. (Bug#26347)

B.22. Changes in MySQL Connector/NET 5.1.1 (23 May 2007)

Bugs fixed:

- Running the statement `SHOW PROCESSLIST` would return columns as byte arrays instead of native columns. (Bug#28448)
- Installation of the Connector/NET on Windows would fail if VisualStudio had not already been installed. (Bug#28260)
- Connector/NET would look for the wrong table when executing `User.IsRole().` (Bug#28251)
- Building a connection string within a tight loop would show slow performance. (Bug#28167)
- The `UNSIGNED` flag for parameters in a stored procedure would be ignored when using `MySqlCommandBuilder` to obtain the parameter information. (Bug#27679)
- Using `MySQLDataAdapter.FillSchema()` on a stored procedure would raise an exception: `Invalid attempt to access a field before calling Read()`. (Bug#27668)
- `DATETIME` fields from versions of MySQL bgefore 4.1 would be incorrectly parsed, resulting in a exception. (Bug#23342)
- Fixed password property on `MySqlConnectionStringBuilder` to use `PasswordPropertyText` attribute. This causes dots to show instead of actual password text.

B.23. Changes in MySQL Connector/NET 5.1.0 (01 May 2007)

Functionality added or changed:

- Now compiles for .NET CF 2.0.
- Rewrote stored procedure parsing code using a new SQL tokenizer. Really nasty procedures including nested comments are now supported.

- GetSchema will now report objects relative to the currently selected database. What this means is that passing in null as a database restriction will report objects on the currently selected database only.
- Added Membership and Role provider contributed by Sean Wright (thanks!).

B.24. Changes in MySQL Connector/NET 5.0.10 (Not yet released)

Bugs fixed:

- If, when using the `MySqlTransaction` transaction object, an exception was thrown, the transaction object was not disposed of and the transaction was not rolled back. (Bug#39817)
- Executing a command that resulted in a fatal exception did not close the connection. (Bug#37991)
- When a prepared insert query is run that contains an `UNSIGNED TINYINT` in the parameter list, the complete query and data that should be inserted is corrupted and no error is thrown. (Bug#37968)
- In a .NET application MySQL Connector/NET modifies the connection string so that it contains several occurrences of the same option with different values. This is illustrated by the example that follows.

  The original connection string:

  ```
  host=localhost;database=test;uid=*****;pwd=*****;
  connect timeout=25; auto enlist=false;pooling=false;
  ```

  The connection string after after closing `MySqlDataReader`:

  ```
  host=localhost;database=test;uid=*****;pwd=*****;
  connect timeout=25;auto enlist=false;pooling=false;
  Allow User Variables=True;Allow User Variables=False;
  Allow User Variables=True;Allow User Variables=False;
  ```

  (Bug#37955)
- When creating a connection pool, specifying an invalid IP address will cause the entire application to crash, instead of providing an exception. (Bug#36432)
- `GetSchema` did not work correctly when querying for a collection, if using a non-English locale. (Bug#35459)
- When reading back a stored double or single value using the .NET provider, the value had less precision than the one stored. (Bug#33322)

B.25. Changes in MySQL Connector/NET 5.0.9 (Not yet released)

Bugs fixed:

- The `DbCommandBuilder.QuoteIdentifer` method was not implemented. (Bug#35492)
- Setting the size of a string parameter after the value could cause an exception. (Bug#32094)
- Creation of parameter objects with non-input direction using a constructor would fail. This was cause by some old legacy code preventing their use. (Bug#32093)
- A date string could be returned incorrectly by `MySqlDataTime.ToString()` when the date returned by MySQL was `0000-00-00 00:00:00`. (Bug#32010)
- A syntax error in a set of batch statements could leave the data adapter in a state that appears hung. (Bug#31930)
- Installing over a failed uninstall of a previous version could result in multiple clients being registered in the `machine.config`. This would prevent certain aspects of the MySQL connection within Visual Studio to work properly. (Bug#31731)
- Data cached from the connection string could return invalid information because the internal routines were not using case-sensitive semantics. This lead to updated connection string options not being recognized if they were of a different case than the existing cached values. (Bug#31433)

- Column name metadata was not using the character set as deifned within the connection string being used. (Bug#31185)
- Memory usage could increase and decrease significantly when updating or inserting a large number of rows. (Bug#31090)
- Commands executed from within the state change handeler would fail with a `NULL` exception. (Bug#30964)
- When running a stored procedure multiple times on the same connection, the memory usage could increase indefinitely. (Bug#30116)
- The server error code was not updated in the `Data[]` hash, which prevented `DbProviderFactory` users from accessing the server error code. (Bug#27436)
- Changing the connection string of a connection to one that changes the parameter marker after the connection had been assigned to a command but before the connection is opened could cause parameters to not be found. (Bug#13991)

B.26. Changes in MySQL Connector/NET 5.0.8 (21 August 2007)

Note

This version introduces a new installer technology.

Bugs fixed:

- Extracting data through XML functions within a query returns the data as `System.Byte[]`. This was due to Connector/NET incorrectly identifying `BLOB` fields as binary, rather than text. (Bug#30233)
- An incorrect `ConstraintException` could be raised on an `INSERT` when adding rows to a table with a multiple-column unique key index. (Bug#30204)
- A `DATE` field would be updated with a date/time value, causing a `MySqlDataAdapter.Update()` exception. (Bug#30077)
- Fixed bug where Connector/Net was hand building some date time patterns rather than using the patterns provided under CultureInfo. This caused problems with some calendars that do not support the same ranges as Gregorian.. (Bug#29931)
- Calling `SHOW CREATE PROCEDURE` for routines with a hyphen in the catalog name produced a syntax error. (Bug#29526)
- The availability of a MySQL server would not be reset when using pooled connections (`pooling=true`). This would lead to the server being reported as unavailable, even if the server become available while the application was still running. (Bug#29409)
- A `FormatException` error would be raised if a parameter had not been found, instead of `Resources.ParameterMustBeDefined`. (Bug#29312)
- Certain operations would not check the `UsageAdvisor` setting, causing log messages from the Usage Advisor even when it was disabled. (Bug#29124)
- Using the same connection string multiple times would result in `Database=`*`dbname`* appearing multiple times in the resulting string. (Bug#29123)
- Log messages would be truncated to 300 bytes. (Bug#28706)
- Accessing the results from a large query when using data compression in the connection will fail to return all the data. (Bug#28204)
- Fixed problem where `MySqlConnection.BeginTransaction` checked the drivers status var before checking if the connection was open. The result was that the driver could report an invalid condition on a previously opened connection.
- Fixed problem where we were not closing prepared statement handles when commands are disposed. This could lead to using up all prepared statement handles on the server.
- Fixed the database schema collection so that it works on servers that are not properly respecting the `lower_case_table_names` setting.
- Fixed problem where any attempt to not read all the records returned from a select where each row of the select is greater than 1024 bytes would hang the driver.
- Fixed problem where a command timing out just after it actually finished would cause an exception to be thrown on the com-

mand timeout thread which would then be seen as an unhandled exception.

- Fixed some serious issues with command timeout and cancel that could present as exceptions about thread ownership. The issue was that not all queries cancel the same. Some produce resultsets while others don't. ExecuteReader had to be changed to check for this.

B.27. Changes in MySQL Connector/NET 5.0.7 (18 May 2007)

Bugs fixed:

- Running the statement `SHOW PROCESSLIST` would return columns as byte arrays instead of native columns. (Bug#28448)
- Building a connection string within a tight loop would show slow performance. (Bug#28167)
- Using logging (with the `logging=true` parameter to the connection string) would not generate a log file. (Bug#27765)
- The `UNSIGNED` flag for parameters in a stored procedure would be ignored when using `MySqlCommandBuilder` to obtain the parameter information. (Bug#27679)
- Using `MySQLDataAdapter.FillSchema()` on a stored procedure would raise an exception: `Invalid attempt to access a field before calling Read()`. (Bug#27668)
- If you close an open connection with an active transaction, the transaction is not automatically rolled back. (Bug#27289)
- When cloning an open `MySqlClient.MySqlConnection` with the `Persist Security Info=False` option set, the cloned connection is not usable because the security information has not been cloned. (Bug#27269)
- Enlisting a null transaction would affect the current connection object, such that further enlistment operations to the transaction are not possible. (Bug#26754)
- Attempting to change the `Connection Protocol` property within a `PropertyGrid` control would raise an exception. (Bug#26472)
- The `characterset` property would not be identified during a connection (also affected Visual Studion Plugin). (Bug#26147, Bug#27240)
- The `CreateFormat` column of the `DataTypes` collection did not contain a format specification for creating a new column type. (Bug#25947)
- `DATETIME` fields from versions of MySQL bgefore 4.1 would be incorrectly parsed, resulting in a exception. (Bug#23342)

B.28. Changes in MySQL Connector/NET 5.0.6 (22 March 2007)

Bugs fixed:

- Publisher listed in "Add/Remove Programs" is not consistent with other MySQL products. (Bug#27253)
- `DESCRIBE ....` SQL statement returns byte arrays rather than data on MySQL versions older than 4.1.15. (Bug#27221)
- `cmd.Parameters.RemoveAt("Id")` will cause an error if the last item is requested. (Bug#27187)
- `MySqlParameterCollection` and parameters added with `Insert` method can not be retrieved later using `ParameterName`. (Bug#27135)
- Exception thrown when using large values in `UInt64` parameters. (Bug#27093)
- MySQL Visual Studio Plugin 1.1.2 does not work with Connector/Net 5.0.5. (Bug#26960)

B.29. Changes in MySQL Connector/NET 5.0.5 (07 March 2007)

Functionality added or changed:

- Reverted behavior that required parameter names to start with the parameter marker. We apologize for this back and forth but we mistakenly changed the behavior to not match what SqlClient supports. We now support using either syntax for adding parameters however we also respond exactly like SqlClient in that if you ask for the index of a parameter using a syntax different from when you added the parameter, the result will be -1.

- Assembly now properly appears in the Visual Studio 2005 Add/Remove Reference dialog.
- Fixed problem that prevented use of SchemaOnly or SingleRow command behaviors with stored procedures or prepared statements.
- Added MySqlParameterCollection.AddWithValue and marked the Add(name, value) method as obsolete.
- Return parameters created with DeriveParameters now have the name RETURN_VALUE.
- Fixed problem with parameter name hashing where the hashes were not getting updated when parameters were removed from the collection.
- Fixed problem with calling stored functions when a return parameter was not given.
- Added Use Procedure Bodies connection string option to allow calling procedures without using procedure metadata.

Bugs fixed:

- MySqlConnection.GetSchema fails with NullReferenceException for Foreign Keys. (Bug#26660)
- Connector/NET would fail to install under Windows Vista. (Bug#26430)
- Opening a connection would be slow due to host name lookup. (Bug#26152)
- Incorrect values/formats would be applied when the OldSyntax connection string option was used. (Bug#25950)
- Registry would be incorrectly populated with installation locations. (Bug#25928)
- Times with negative values would be returned incorrectly. (Bug#25912)
- Returned data types of a DataTypes collection do not contain the right correctl CLR Datatype. (Bug#25907)
- GetSchema and DataTypes would throw an exception due to an incorrect table name. (Bug#25906)
- MySqlConnection throws an exception when connecting to MySQL v4.1.7. (Bug#25726)
- SELECT did not work correctly when using a WHERE clause containing a UTF-8 string. (Bug#25651)
- When closing and then re-opening a connection to a database, the character set specification is lost. (Bug#25614)
- Filling a table schema through a stored procedure triggers a runtime error. (Bug#25609)
- BINARY and VARBINARY columns would be returned as a string, not binary, datatype. (Bug#25605)
- A critical ConnectionPool error would result in repeated System.NullReferenceException. (Bug#25603)
- The UpdateRowSource.FirstReturnedRecord method does not work. (Bug#25569)
- When connecting to a MySQL Server earlier than version 4.1, the connection would hang when reading data. (Bug#25458)
- Using ExecuteScalar() with more than one query, where one query fails, will hang the connection. (Bug#25443)
- When a MySqlConversionException is raised on a remote object, the client application would receive a SerializationException instead. (Bug#24957)
- When connecting to a server, the return code from the connection could be zero, even though the host name was incorrect. (Bug#24802)
- High CPU utilization would be experienced when there is no idle connection waiting when using pooled connections through MySqlPool.GetConnection. (Bug#24373)
- Connector/NET would not compile properly when used with Mono 1.2. (Bug#24263)
- Applications would crash when calling with CommandType set to StoredProcedure.

B.30. Changes in MySQL Connector/NET 5.0.4 (Not released)

This is a new Beta development release, fixing recently discovered bugs.

B.31. Changes in MySQL Connector/NET 5.0.3 (05 January 2007)

Functionality added or changed:

- Usage Advisor has been implemented. The Usage Advisor checks your queries and will report if you are using the connection inefficiently.
- PerfMon hooks have been added to monitor the stored procedure cache hits and misses.
- The `MySqlCommand` object now supports asynchronous query methods. This is implemented useg the `BeginExecuteNonQuery` and `EndExecuteNonQuery` methods.
- Metadata from storaed procedures and stored function execution are cached.
- The `CommandBuilder.DeriveParameters` function has been updated to the procedure cache.
- The `ViewColumns` `GetSchema` collection has been updated.
- Improved speed and performance by re-architecting certain sections of the code.
- Support for the embedded server and client library have been removed from this release. Support will be added back to a later release.
- The ShapZipLib library has been replaced with the deflate support provided within .NET 2.0.
- SSL support has been updated.

Bugs fixed:

- Additional text added to error message (Bug#25178)
- An exception would be raised, or the process would hang, if `SELECT` privileges on a database were not granted and a stored procedure was used. (Bug#25033)
- When adding parameter objects to a command object, if the parameter direction is set to `ReturnValue` before the parameter is added to the command object then when the command is executed it throws an error. (Bug#25013)
- Using `Driver.IsTooOld()` would return the wrong value. (Bug#24661)
- When using a `DbNull.Value` as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the `DbNull.Value`. (Bug#24565)
- Stored procedure executions are not thread safe. (Bug#23905)
- Deleting a connection to a disconnected server when using the Visual Studio Plugin would cause an assertion failure. (Bug#23687)
- Nested transactions (which are unsupported)do not raise an error or warning. (Bug#22400)

B.32. Changes in MySQL Connector/NET 5.0.2 (06 November 2006)

Functionality added or changed:

- An `Ignore Prepare` option has been added to the connection string options. If enabled, prepared statements will be disabled application-wide. The default for this option is true.
- Implemented a stored procedure cache. By default, the connector caches the metadata for the last 25 procedures that are seen. You can change the numbver of procedures that are cacheds by using the `procedure cache` connection string.
- **Important change:** Due to a number of issues with the use of server-side prepared statements, Connector/NET 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any

way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore prepare=false
```

The default value of this property is true.

Bugs fixed:

- One system where IPv6 was enabled, Connector/NET would incorrectly resolve host names. (Bug#23758)
- Column names with accented characters were not parsed properly causing malformed column names in result sets. (Bug#23657)
- An exception would be thrown when calling `GetSchemaTable` and `fields` was null. (Bug#23538)
- A `System.FormatException` exception would be raised when invoking a stored procedure with an `ENUM` input parameter. (Bug#23268)
- During installation, an antivirus error message would be raised (indicating a malicious script problem). (Bug#23245)
- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that Connector/NET 5.0.2 must be installed. (Bug#23071)
- Using Windows Vista (RC2) as a non-privileged user would raise a `Registry key 'Global' access denied`. (Bug#22882)
- Within Mono, using the `PreparedStatement` interface could result in an error due to a `BitArray` copying error. (Bug#18186)
- Connector/NET did not work as a data source for the `SqlDataSource` object used by ASP.NET 2.0. (Bug#16126)

B.33. Changes in MySQL Connector/NET 5.0.1 (01 October 2006)

Bugs fixed:

- Connector/NET on a Tukish operating system, may fail to execute certain SQL statements correctly. (Bug#22452)
- Starting a transaction on a connection created by `MySql.Data.MySqlClient.MySqlClientFactory`, using `BeginTransaction` without specifying an isolation level, causes the SQL statement to fail with a syntax error. (Bug#22042)
- The `MySqlexception` class is now derived from the `DbException` class. (Bug#21874)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug#21521)
- You can now install the Connector/NET MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug#19994)
- Submitting an empty string to a command object through `prepare` raises an `System.IndexOutOfRangeException`, rather than a Connector/Net exception. (Bug#18391)
- Using `ExecuteScalar` with a datetime field, where the value of the field is "0000-00-00 00:00:00", a `MySqlConversionException` exception would be raised. (Bug#11991)
- An `MySql.Data.Types.MySqlConversionException` would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug#9619)
- Executing multiple queries as part of a transaction returns `There is already an openDataReader associated with this Connection which must be closed first`. (Bug#7248)
- Incorrect field/data lengths could be returned for `VARCHAR` UTF8 columns. Bug (#14592)

B.34. Changes in MySQL Connector/NET 5.0.0 (08 August 2006)

Functionality added or changed:

- Replaced use of ICSharpCode with .NET 2.0 internal deflate support.
- Refactored test suite to test all protocols in a single pass.
- Added usage advisor warnings for requesting column values by the wrong type.
- Reimplemented PacketReader/PacketWriter support into `MySqlStream` class.
- Reworked connection string classes to be simpler and faster.
- Added procedure metadata caching.
- Added internal implemention of SHA1 so we don't have to distribute the OpenNetCF on mobile devices.
- Implemented `MySqlClientFactory` class.
- Added perfmon hooks for stored procedure cache hits and misses.
- Implemented classes and interfaces for ADO.Net 2.0 support.
- Added Async query methods.
- Implemented Usage Advisor.
- Completely refactored how column values are handled to avoid boxing in some cases.
- Implemented `MySqlConnectionBuilder` class.

Bugs fixed:

- CommandText: Question mark in comment line is being parsed as a parameter. (Bug#6214)

B.35. Changes in MySQL Connector/NET 1.0.11 (Not yet released)

Bugs fixed:

- Attempting to utilize MySQL Connector .Net version 1.0.10 throws a fatal exception under Mono when pooling is enabled. (Bug#33682)
- Setting the size of a string parameter after the value could cause an exception. (Bug#32094)
- Creation of parameter objects with non-input direction using a constructor would fail. This was cause by some old legacy code preventing their use. (Bug#32093)
- Memory usage could increase and decrease significantly when updating or inserting a large number of rows. (Bug#31090)
- Commands executed from within the state change handeler would fail with a `NULL` exception. (Bug#30964)
- Extracting data through XML functions within a query returns the data as `System.Byte[]`. This was due to Connector/NET incorrectly identifying `BLOB` fields as binary, rather than text. (Bug#30233)
- Using compression in the MySQL connection with Connector/NET would be slower than using native (uncompressed) communication. (Bug#27865)
- Changing the connection string of a connection to one that changes the parameter marker after the connection had been assigned to a command but before the connection is opened could cause parameters to not be found. (Bug#13991)

B.36. Changes in MySQL Connector/NET 1.0.10 (24 August 2007)

Bugs fixed:

- An incorrect `ConstraintException` could be raised on an `INSERT` when adding rows to a table with a multiple-column unique key index. (Bug#30204)

- The availability of a MySQL server would not be reset when using pooled connections (`pooling=true`). This would lead to the server being reported as unavailable, even if the server become available while the application was still running. (Bug#29409)
- Publisher listed in "Add/Remove Programs" is not consistent with other MySQL products. (Bug#27253)
- `MySqlParameterCollection` and parameters added with `Insert` method can not be retrieved later using `ParameterName`. (Bug#27135)
- `BINARY` and `VARBINARY` columns would be returned as a string, not binary, datatype. (Bug#25605)
- A critical `ConnectionPool` error would result in repeated `System.NullReferenceException`. (Bug#25603)
- When a `MySqlConversionException` is raised on a remote object, the client application would receive a `SerializationException` instead. (Bug#24957)
- High CPU utilization would be experienced when there is no idle connection waiting when using pooled connections through `MySqlPool.GetConnection`. (Bug#24373)

B.37. Changes in MySQL Connector/NET 1.0.9 (02 February 2007)

Functionality added or changed:

- The ICSharpCode ZipLib is no longer used by the Connector, and is no longer distributed with it.
- **Important change:** Binaries for .NET 1.0 are no longer supplied with this release. If you need support for .NET 1.0, you must build from source.
- Improved `CommandBuilder.DeriveParameters` to first try and use the procedure cache before querying for the stored procedure metadata. Return parameters created with `DeriveParameters` now have the name `RETURN_VALUE`.
- An `Ignore Prepare` option has been added to the connection string options. If enabled, prepared statements will be disabled application-wide. The default for this option is true.
- Implemented a stored procedure cache. By default, the connector caches the metadata for the last 25 procedures that are seen. You can change the numbver of procedures that are cacheds by using the `procedure cache` connection string.
- **Important change:** Due to a number of issues with the use of server-side prepared statements, Connector/NET 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

  To enable server-side prepared statements you must add the following configuration property to your connector string properties:

  ```
  ignore prepare=false
  ```

  The default value of this property is true.

Bugs fixed:

- Times with negative values would be returned incorrectly. (Bug#25912)
- `MySqlConnection` throws a `NullReferenceException` and `ArgumentNullException` when connecting to MySQL v4.1.7. (Bug#25726)
- `SELECT` did not work correctly when using a `WHERE` clause containing a UTF-8 string. (Bug#25651)
- When closing and then re-opening a connection to a database, the character set specification is lost. (Bug#25614)
- Trying to fill a table schema through a stored procedure triggers a runtime error. (Bug#25609)
- Using `ExecuteScalar()` with more than one query, where one query fails, will hang the connection. (Bug#25443)
- Additional text added to error message. (Bug#25178)

- When adding parameter objects to a command object, if the parameter direction is set to `ReturnValue` before the parameter is added to the command object then when the command is executed it throws an error. (Bug#25013)
- When connecting to a server, the return code from the connection could be zero, even though the host name was incorrect. (Bug#24802)
- Using `Driver.IsTooOld()` would return the wrong value. (Bug#24661)
- When using a `DbNull.Value` as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the `DbNull.Value`. (Bug#24565)
- Stored procedure executions are not thread safe. (Bug#23905)
- The `CommandBuilder` would mistakenly add insert parameters for a table column with auto incrementation enabled. (Bug#23862)
- One system where IPv6 was enabled, Connector/NET would incorrectly resolve host names. (Bug#23758)
- Nested transactions do not raise an error or warning. (Bug#22400)
- An `System.OverflowException` would be raised when accessing a varchar field over 255 bytes. Bug (#23749)
- Within Mono, using the `PreparedStatement` interface could result in an error due to a `BitArray` copying error. (Bug 18186)

B.38. Changes in MySQL Connector/NET 1.0.8 (20 October 2006)

Functionality added or changed:

- Stored procedures are now cached.
- The method for retrieving stored procedured metadata has been changed so that users without `SELECT` privileges on the `mysql.proc` table can use a stored procedure.

Bugs fixed:

- Connector/NET on a Tukish operating system, may fail to execute certain SQL statements correctly. (Bug#22452)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug#21521)
- Calling `Close` on a connection after calling a stored procedure would trigger a `NullReferenceException`. (Bug#20581)
- You can now install the Connector/NET MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug#19994)
- The DiscoverParameters function would fail when a stored procedure used a `NUMERIC` parameter type. (Bug#19515)
- When running a query that included a date comparison, a DateReader error would be raised. (Bug#19481)
- `IDataRecord.GetString` would raise `NullPointerException` for null values in returned rows. Method now throws `SqlNullValueException`. (Bug#19294)
- Parameter substitution in queries where the order of parameters and table fields did not match would substitute incorrect values. (Bug#19261)
- Submitting an empty string to a command object through `prepare` raises an `System.IndexOutOfRangeException`, rather than a Connector/Net exception. (Bug#18391)
- An exception would be raised when using an output parameter to a `System.String` value. (Bug#17814)
- CHAR type added to MySqlDbType. (Bug#17749)
- A `SELECT` query on a table with a date with a value of `'0000-00-00'` would hang the application. (Bug#17736)
- The CommandBuilder ignored Unsigned flag at Parameter creation. (Bug#17375)
- When working with multiple threads, character set initialization would generate errors. (Bug#17106)

- When using an unsigned 64-bit integer in a stored procedure, the unsigned bit would be lost stored. (Bug#16934)
- `DataReader` would show the value of the previous row (or last row with non-null data) if the current row contained a `datetime` field with a null value. (Bug#16884)
- Unsigned data types were not properly supported. (Bug#16788)
- The connection string parser did not allow single or double quotes in the password. (Bug#16659)
- The `MySqlDateTime` class did not contain constructors. (Bug#15112)
- Called `MySqlCommandBuilder.DeriveParameters` for a stored procedure that has no paramers would cause an application crash. (Bug#15077)
- Using `ExecuteScalar` with a datetime field, where the value of the field is "0000-00-00 00:00:00", a `MySqlConversionException` exception would be raised. (Bug#11991)
- An `MySql.Data.Types.MySqlConversionException` would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug#9619)
- When using `MySqlDataAdapter`, connections to a MySQL server may remain open and active, even though the use of the connection has been completed and the data received. (Bug#8131)
- Executing multiple queries as part of a transaction returns `There is already an openDataReader associated with this Connection which must be closed first`. (Bug#7248)
- Incorrect field/data lengths could be returned for `VARCHAR` UTF8 columns. Bug (#14592)

B.39. Changes in MySQL Connector/NET 1.0.7 (21 November 2005)

Bugs fixed:

- Unsigned `tinyint` (NET byte) would lead to and incorrectly determined parameter type from the parameter value. (Bug#18570)
- A `#42000Query was empty` exception occurred when executing a query built with `MySqlCommandBuilder`, if the query string ended with a semicolon. (Bug#14631)
- The parameter collection object's `Add()` method added parameters to the list without first checking to see whether they already existed. Now it updates the value of the existing parameter object if it exists. (Bug#13927)
- Added support for the `cp932` character set. (Bug#13806)
- Calling a stored procedure where a parameter contained special characters (such as `'@'`) would produce an exception. Note that `ANSI_QUOTES` had to be enabled to make this possible. (Bug#13753)
- The `Ping()` method did not update the `State` property of the `Connection` object. (Bug#13658)
- Implemented the `MySqlCommandBuilder.DeriveParameters` method that is used to discover the parameters for a stored procedure. (Bug#13632)
- A statement that contained multiple references to the same parameter could not be prepared. (Bug#13541)

B.40. Changes in MySQL Connector/NET 1.0.6 (03 October 2005)

Bugs fixed:

- Connector/NET 1.0.5 could not connect on Mono. (Bug#13345)
- Serializing a parameter failed if the first value passed in was `NULL`. (Bug#13276)
- Field names that contained the following characters caused errors: `()%<>/` (Bug#13036)
- The `nant` build sequence had problems. (Bug#12978)
- The Connector/NET 1.0.5 installer would not install alongside Connector/NET 1.0.4. (Bug#12835)

B.41. Changes in MySQL Connector/NET 1.0.5 (29 August 2005)

Bugs fixed:

- Connector/NET could not connect to MySQL 4.1.14. (Bug#12771)
- With multiple hosts in the connection string, Connector/NET would not connect to the last host in the list. (Bug#12628)
- The `ConnectionString` property could not be set when a `MySqlConnection` object was added with the designer. (Bug#12551, Bug#8724)
- The `cp1250` character set was not supported. (Bug#11621)
- A call to a stored procedure caused an exception if the stored procedure had no parameters. (Bug#11542)
- Certain malformed queries would trigger a `Connection must be valid and open` error message. (Bug#11490)
- Trying to use a stored procedure when `Connection.Database` was not populated generated an exception. (Bug#11450)
- Connector/NET interpreted the new decimal data type as a byte array. (Bug#11294)
- Added support to call a stored function from Connector/NET. (Bug#10644)
- Connection could fail when .NET thread pool had no available worker threads. (Bug#10637)
- Calling `MySqlConnection.clone` when a connection string had not yet been set on the original connection would generate an error. (Bug#10281)
- Decimal parameters caused syntax errors. (Bug#10152, Bug#11550, Bug#10486)
- Parameters were not recognized when they were separated by linefeeds. (Bug#9722)
- The `MySqlCommandBuilder` class could not handle queries that referenced tables in a database other than the default database. (Bug#8382)
- Trying to read a `TIMESTAMP` column generated an exception. (Bug#7951)
- Connector/NET could not work properly with certain regional settings. (WL#8228)

B.42. Changes in MySQL Connector/NET 1.0.4 (20 January 2005)

Bugs fixed:

- `MySqlReader.GetInt32` throws exception if column is unsigned. (Bug#7755)
- Quote character \222 not quoted in `EscapeString`. (Bug#7724)
- GetBytes is working no more. (Bug#7704)
- `MySqlDataReader.GetString(index)` returns non-Null value when field is `Null`. (Bug#7612)
- Clone method bug in `MySqlCommand`. (Bug#7478)
- Problem with Multiple resultsets. (Bug#7436)
- `MySqlAdapter.Fill` method throws error message `Non-negative number required`. (Bug#7345)
- `MySqlCommand.Connection` returns an IDbConnection. (Bug#7258)
- Calling prepare causing exception. (Bug#7243)
- Fixed problem with shared memory connections.
- Added or filled out several more topics in the API reference documentation.
- Fixed another small problem with prepared statements.
- Fixed problem that causes named pipes to not work with some blob functionality.

B.43. Changes in MySQL Connector/NET 1.0.3 (12 October 2004 gamma)

Bugs fixed:

- Invalid query string when using inout parameters (Bug#7133)
- Inserting `DateTime` causes `System.InvalidCastException` to be thrown. (Bug#7132)
- `MySqlDateTime` in Datatables sorting by Text, not Date. (Bug#7032)
- Exception stack trace lost when re-throwing exceptions. (Bug#6983)
- Errors in parsing stored procedure parameters. (Bug#6902)
- InvalidCast when using `DATE_ADD`-function. (Bug#6879)
- Int64 Support in `MySqlCommand` Parameters. (Bug#6863)
- Test suite fails with MySQL 4.0 because of case sensitivity of table names. (Bug#6831)
- `MySqlDataReader.GetChar(int i)` throws `IndexOutOfRange` exception. (Bug#6770)
- Integer "out" parameter from stored procedure returned as string. (Bug#6668)
- An Open Connection has been Closed by the Host System. (Bug#6634)
- Fixed Invalid character set index: 200. (Bug#6547)
- Connections now do not have to give a database on the connection string.
- Installer now includes options to install into GAC and create START MENU items.
- Fixed major problem with detecting null values when using prepared statements.
- Fixed problem where multiple resultsets having different numbers of columns would cause a problem.
- Added `ServerThread` property to `MySqlConnection` to expose server thread id.
- Added Ping method to `MySqlConnection`.
- Changed the name of the test suite to `MySql.Data.Tests.dll`.
- Now `SHOW COLLATION` is used upon connection to retrieve the full list of charset ids.
- Made MySQL the default named pipe name.

B.44. Changes in MySQL Connector/NET 1.0.2 (15 November 2004 gamma)

Bugs fixed:

- Fixed Objects not being disposed (Bug#6649)
- Fixed Charset-map for UCS-2 (Bug#6541)
- Fixed Zero date "0000-00-00" is returned wrong when filling Dataset (Bug#6429)
- Fixed double type handling in MySqlParameter(string parameterName, object value) (Bug#6428)
- Fixed Installation directory ignored using custom installation (Bug#6329)
- Fixed #HY000 Illegal mix of collations (latin1_swedish_ci,IMPLICIT) and (utf8_general_ (Bug#6322)
- Added the TableEditor CS and VB sample
- Added charset connection string option

- Fixed problem with MySqlBinary where string values could not be used to update extended text columns
- Provider is now using character set specified by server as default
- Updated the installer to include the new samples
- Fixed problem where setting command text leaves the command in a prepared state
- Fixed Long inserts take very long time (Bu #5453)
- Fixed problem where calling stored procedures might cause an "Illegal mix of collations" problem.

B.45. Changes in MySQL Connector/NET 1.0.1 (27 October 2004 beta)

Bugs fixed:

- Fixed IndexOutOfBounds when reading BLOB with DataReader with GetString(index) (Bug#6230)
- Fixed GetBoolean returns wrong values (Bug#6227)
- Fixed Method TokenizeSql() uses only a limited set of valid characters for parameters (Bug#6217)
- Fixed NET Connector source missing resx files (Bug#6216)
- Fixed System.OverflowException when using YEAR datatype (Bug#6036)
- Fixed MySqlDateTime sets IsZero property on all subseq.records after first zero found (Bug#6006)
- Fixed serializing of floating point parameters (double, numeric, single, decimal) (Bug#5900)
- Fixed missing Reference in DbType setter (Bug#5897)
- Fixed Parsing the ';' char (Bug#5876)
- Fixed DBNull Values causing problems with retrieving/updating queries. (Bug#5798)
- IsNullable error (Bug#5796)
- Fixed problem where MySqlParameterCollection.Add() would throw unclear exception when given a null value (Bug#5621)
- Fixed construtor initialize problems in MySqlCommand() (Bug#5613)
- Fixed Yet Another "object reference not set to an instance of an object" (Bug#5496)
- Fixed Can't display Chinese correctly (Bug#5288)
- Fixed MySqlDataReader and 'show tables from ...' behavior (Bug#5256)
- Fixed problem in PacketReader where it could try to allocate the wrong buffer size in EnsureCapacity
- Fixed problem where using old syntax while using the interfaces caused problems
- Fixed Bug#5458 Calling GetChars on a longtext column throws an exception
- Added test case for resetting the command text on a prepared command
- Fixed Bug#5388 DataReader reports all rows as NULL if one row is NULL
- Fixed problem where connection lifetime on the connect string was not being respected
- Fixed Bug#5602 Possible bug in MySqlParameter(string, object) constructor
- Field buffers being reused to decrease memory allocations and increase speed
- Fixed Bug#5392 MySqlCommand sees "?" as parameters in string literals
- Added Aggregate function test (wasn't really a bug)

- Using PacketWriter instead of Packet for writing to streams
- Implemented SequentialAccess
- Fixed problem with ConnectionInternal where a key might be added more than once
- Fixed Russian character support as well
- Fixed Bug#5474 cannot run a stored procedure populating mysqlcommand.parameters
- Fixed problem where connector was not issuing a CMD_QUIT before closing the socket
- Fixed problem where Min Pool Size was not being respected
- Refactored compression code into CompressedStream to clean up NativeDriver
- CP1252 is now used for Latin1 only when the server is 4.1.2 and later
- Fixed Bug#5469 Setting DbType throws NullReferenceException
- Virtualized driver subsystem so future releases could easily support client or embedded server support

B.46. Changes in MySQL Connector/NET 1.0.0 (01 September 2004)

Bugs fixed:

- Thai encoding not correctly supported. (Bug#3889)
- Bumped version number to 1.0.0 for beta 1 release.
- Removed all of the XML comment warnings.
- Added `COPYING.rtf` file for use in installer.
- Updated many of the test cases.
- Fixed problem with using compression.
- Removed some last references to ByteFX.

B.47. Changes in MySQL Connector/NET Version 0.9.0 (30 August 2004)

- Added test fixture for prepared statements.
- All type classes now implement a `SerializeBinary` method for sending their data to a `PacketWriter`.
- Added `PacketWriter` class that will enable future low-memory large object handling.
- Fixed many small bugs in running prepared statements and stored procedures.
- Changed command so that an exception will not be thrown in executing a stored procedure with parameters in old syntax mode.
- `SingleRow` behavior now working right even with limit.
- `GetBytes` now only works on binary columns.
- Logger now truncates long sql commands so blob columns do not blow out our log.
- host and database now have a default value of "" unless otherwise set.
- Connection Timeout seems to be ignored. (Bug#5214)
- Added test case for bug# 5051: GetSchema not working correctly.
- Fixed problem where `GetSchema` would return false for `IsUnique` when the column is key.

- MySqlDataReader GetXXX methods now using the field level MySqlValue object and not performing conversions.
- DataReader returning NULL for time column. (Bug#5097)
- Added test case for LOAD DATA LOCAL INFILE.
- Added replacetext custom nant task.
- Added CommandBuilderTest fixture.
- Added Last One Wins feature to CommandBuilder.
- Fixed persist security info case problem.
- Fixed GetBool so that 1, true, "true", and "yes" all count as true.
- Make parameter mark configurable.
- Added the "old syntax" connection string parameter to allow use of @ parameter marker.
- MySqlCommandBuilder. (Bug#4658)
- ByteFX.MySqlClient caches passwords if Persist Security Info is false. (Bug#4864)
- Updated license banner in all source files to include FLOSS exception.
- Added new .Types namespace and implementations for most current MySql types.
- Added MySqlField41 as a subclass of MySqlField.
- Changed many classes to now use the new .Types types.
- Changed type enum int to Int32, short to Int16, and bigint to Int64.
- Added dummy types UInt16, UInt32, and UInt64 to allow an unsigned parameter to be made.
- Connections are now reset when they are pulled from the connection pool.
- Refactored auth code in driver so it can be used for both auth and reset.
- Added UserReset test in PoolingTests.cs.
- Connections are now reset using COM_CHANGE_USER when pulled from the pool.
- Implemented SingleResultSet behavior.
- Implemented support of unicode.
- Added char set mappings for utf-8 and ucs-2.
- Time fields overflow using bytefx .net mysql driver (Bug#4520)
- Modified time test in data type test fixture to check for time spans where hours > 24.
- Wrong string with backslash escaping in ByteFx.Data.MySqlClient.MySqlParameter. (Bug#4505)
- Added code to Parameter test case TestQuoting to test for backslashes.
- MySqlCommandBuilder fails with multi-word column names. (Bug#4486)
- Fixed bug in TokenizeSql where underscore would terminate character capture in parameter name.
- Added test case for spaces in column names.
- MySqlDataReader.GetBytes do not work correctly. (Bug#4324)
- Added GetBytes() test case to DataReader test fixture.
- Now reading all server variables in InternalConnection.Configure into Hashtable.
- Now using string[] for index map in CharSetMap.

- Added CRInSQL test case for carriage returns in SQL.
- Setting maxPacketSize to default value in `Driver.ctor`.
- Setting `MySqlDbType` on a parameter doesn't set generic type. (Bug#4442)
- Removed obsolete data types `Long` and `LongLong`.
- Overflow exception thrown when using "use pipe" on connection string. (Bug#4071)
- Changed "use pipe" keyword to "pipe name" or just "pipe".
- Allow reading multiple resultsets from a single query.
- Added flags attribute to `ServerStatusFlags` enum.
- Changed name of `ServerStatus` enum to `ServerStatusFlags`.
- Inserted data row doesn't update properly.
- Error processing show create table. (Bug#4074)
- Change `Packet.ReadLenInteger` to `ReadPackedLong` and added `packet.ReadPackedInteger` that always reads integers packed with 2,3,4.
- Added `syntax.cs` test fixture to test various SQL syntax bugs.
- Improper handling of time values. Now time value of 00:00:00 is not treated as null. (Bug#4149)
- Moved all test suite files into `TestSuite` folder.
- Fixed bug where null column would move the result packet pointer backward.
- Added new nant build script.
- Clear tablename so it will be regen'ed properly during the next `GenerateSchema`. (Bug#3917)
- `GetValues` was always returning zero and was also always trying to copy all fields rather than respecting the size of the array passed in. (Bug#3915)
- Implemented shared memory access protocol.
- Implemented prepared statements for MySQL 4.1.
- Implemented stored procedures for MySQL 5.0.
- Renamed `MySqlInternalConnection` to `InternalConnection`.
- SQL is now parsed as chars, fixes problems with other languages.
- Added logging and allow batch connection string options.
- `RowUpdating` event not set when setting the `DataAdapter` property. (Bug#3888)
- Fixed bug in char set mapping.
- Implemented 4.1 authentication.
- Improved open/auth code in driver.
- Improved how connection bits are set during connection.
- Database name is now passed to server during initial handshake.
- Changed namespace for client to `MySql.Data.MySqlClient`.
- Changed assembly name of client to `MySql.Data.dll`.
- Changed license text in all source files to GPL.
- Added the `MySqlClient.build` Nant file.

- Removed the mono batch files.
- Moved some of the unused files into notused folder so nant build file can use wildcards.
- Implemented shared memory access.
- Major revamp in code structure.
- Prepared statements now working for MySql 4.1.1 and later.
- Finished implementing auth for 4.0, 4.1.0, and 4.1.1.
- Changed namespace from `MySQL.Data.MySQLClient` back to `MySql.Data.MySqlClient`.
- Fixed bug in `CharSetMapping` where it was trying to use text names as ints.
- Changed namespace to `MySQL.Data.MySQLClient`.
- Integrated auth changes from UC2004.
- Fixed bug where calling any of the GetXXX methods on a datareader before or after reading data would not throw the appropriate exception (thanks Luca Morelli).
- Added `TimeSpan` code in parameter.cs to properly serialize a timespan object to mysql time format (thanks Gianluca Colombo).
- Added `TimeStamp` to parameter serialization code. Prevented `DataAdatper` updates from working right (thanks Michael King).
- Fixed a misspelling in `MySqlHelper.cs` (thanks Patrick Kristiansen).

B.48. Changes in MySQL Connector/NET Version 0.76

- Driver now using charset number given in handshake to create encoding.
- Changed command editor to point to `MySqlClient.Design`.
- Fixed bug in `Version.isAtLeast`.
- Changed `DBConnectionString` to support changes done to `MySqlConnectionString`.
- Removed `SqlCommandEditor` and `DataAdapterPreviewDialog`.
- Using new long return values in many places.
- Integrated new `CompressedStream` class.
- Changed `ConnectionString` and added attributes to allow it to be used in `MySqlClient.Design`.
- Changed `packet.cs` to support newer lengths in `ReadLenInteger`.
- Changed other classes to use new properties and fields of `MySqlConnectionString`.
- `ConnectionInternal` is now using PING to see whether the server is alive.
- Moved toolbox bitmaps into resource folder.
- Changed `field.cs` to allow values to come directly from row buffer.
- Changed to use the new driver.Send syntax.
- Using a new packet queueing system.
- Started work handling the "broken" compression packet handling.
- Fixed bug in `StreamCreator` where failure to connect to a host would continue to loop infinitly (thanks Kevin Casella).
- Improved connectstring handling.

- Moved designers into Pro product.
- Removed some old commented out code from `command.cs`.
- Fixed a problem with compression.
- Fixed connection object where an exception throw prior to the connection opening would not leave the connection in the connecting state (thanks Chris Cline).
- Added GUID support.
- Fixed sequence out of order bug (thanks Mark Reay).

B.49. Changes in MySQL Connector/NET Version 0.75

- Enum values now supported as parameter values (thanks Philipp Sumi).
- Year datatype now supported.
- Fixed compression.
- Fixed bug where a parameter with a `TimeSpan` as the value would not serialize properly.
- Fixed bug where default constructor would not set default connection string values.
- Added some XML comments to some members.
- Work to fix/improve compression handling.
- Improved `ConnectionString` handling so that it better matches the standard set by `SqlClient`.
- A `MySqlException` is now thrown if a user name is not included in the connection string.
- Localhost is now used as the default if not specified on the connection string.
- An exception is now thrown if an attempt is made to set the connection string while the connection is open.
- Small changes to `ConnectionString` docs.
- Removed `MultiHostStream` and `MySqlStream`. Replaced it with `Common/StreamCreator`.
- Added support for Use Pipe connection string value.
- Added Platform class for easier access to platform utility functions.
- Fixed small pooling bug where new connection was not getting created after `IsAlive` fails.
- Added `Platform.cs` and `StreamCreator.cs`.
- Fixed `Field.cs` to properly handle 4.1 style timestamps.
- Changed `Common.Version` to `Common.DBVersion` to avoid name conflict.
- Fixed `field.cs` so that text columns return the right field type.
- Added `MySqlError` class to provide some reference for error codes (thanks Geert Veenstra).

B.50. Changes in MySQL Connector/NET Version 0.74

- Added Unix socket support (thanks Mohammad DAMT).
- Only calling `Thread.Sleep` when no data is available.
- Improved escaping of quote characters in parameter data.
- Removed misleading comments from `parameter.cs`.

- Fixed pooling bug.
- Fixed `ConnectionString` editor dialog (thanks marco p (pomarc)).
- `UserId` now supported in connection strings (thanks Jeff Neeley).
- Attempting to create a parameter that is not input throws an exception (thanks Ryan Gregg).
- Added much documentation.
- Checked in new `MultiHostStream` capability. Big thanks to Dan Guisinger for this. he originally submitted the code and idea of supporting multiple machines on the connect string.
- Added a lot of documentation.
- Fixed speed issue with 0.73.
- Changed to Thread.Sleep(0) in MySqlDataStream to help optimize the case where it doesn't need to wait (thanks Todd German).
- Prepopulating the idlepools to `MinPoolSize`.
- Fixed `MySqlPool` deadlock condition as well as stupid bug where CreateNewPooledConnection was not ever adding new connections to the pool. Also fixed `MySqlStream.ReadBytes` and `ReadByte` to not use `TicksPerSecond` which does not appear to always be right. (thanks Matthew J. Peddlesden)
- Fix for precision and scale (thanks Matthew J. Peddlesden).
- Added `Thread.Sleep(1)` to stream reading methods to be more cpu friendly (thanks Sean McGinnis).
- Fixed problem where `ExecuteReader` would sometime return null (thanks Lloyd Dupont).
- Fixed major bug with null field handling (thanks Naucki).
- Enclosed queries for `max_allowed_packet` and `characterset` inside try catch (and set defaults).
- Fixed problem where socket was not getting closed properly (thanks Steve!).
- Fixed problem where `ExecuteNonQuery` was not always returning the right value.
- Fixed `InternalConnection` to not use `@@session.max_allowed_packet` but use `@@max_allowed_packet`. (Thanks Miguel)
- Added many new XML doc lines.
- Fixed sql parsing to not send empty queries (thanks Rory).
- Fixed problem where the reader was not unpeeking the packet on close.
- Fixed problem where user variables were not being handled (thanks Sami Vaaraniemi).
- Fixed loop checking in the MySqlPool (thanks Steve M. Brown)
- Fixed `ParameterCollection.Add` method to match `SqlClient` (thanks Joshua Mouch).
- Fixed `ConnectionString` parsing to handle no and yes for boolean and not lowercase values (thanks Naucki).
- Added `InternalConnection` class, changes to pooling.
- Implemented Persist Security Info.
- Added `security.cs` and `version.cs` to project
- Fixed `DateTime` handling in `Parameter.cs` (thanks Burkhard Perkens-Golomb).
- Fixed parameter serialization where some types would throw a cast exception.
- Fixed `DataReader` to convert all returned values to prevent casting errors (thanks Keith Murray).
- Added code to `Command.ExecuteReader` to return null if the initial SQL statement throws an exception (thanks Burkhard Perkens-Golomb).

- Fixed `ExecuteScalar` bug introduced with restructure.
- Restructure to allow for `LOCAL DATA INFILE` and better sequencing of packets.
- Fixed several bugs related to restructure.
- Early work done to support more secure passwords in Mysql 4.1. Old passwords in 4.1 not supported yet.
- Parameters appearing after system parameters are now handled correctly (Adam M. (adammil)).
- Strings can now be assigned directly to blob fields (Adam M.).
- Fixed float parameters (thanks Pent).
- Improved Parameter constructor and `ParameterCollection.Add` methods to better match SqlClient (thanks Joshua Mouch).
- Corrected `Connection.CreateCommand` to return a `MySqlCommand` type.
- Fixed connection string designer dialog box problem (thanks Abraham Guyt).
- Fixed problem with sending commands not always reading the response packet (thanks Joshua Mouch).
- Fixed parameter serialization where some blobs types were not being handled (thanks Sean McGinnis).
- Removed spurious `MessageBox.show` from `DataReader` code (thanks Joshua Mouch).
- Fixed a nasty bug in the split sql code (thanks everyone!).

B.51. Changes in MySQL Connector/NET Version 0.71

- Fixed bug in `MySqlStream` where too much data could attempt to be read (thanks Peter Belbin)
- Implemented `HasRows` (thanks Nash Pherson).
- Fixed bug where tables with more than 252 columns cause an exception (thanks Joshua Kessler).
- Fixed bug where SQL statements ending in ; would cause a problem (thanks Shane Krueger).
- Fixed bug in driver where error messages were getting truncated by 1 character (thanks Shane Krueger).
- Made `MySqlException` serializable (thanks Mathias Hasselmann).

B.52. Changes in MySQL Connector/NET Version 0.70

- Updated some of the character code pages to be more accurate.
- Fixed problem where readers could be opened on connections that had readers open.
- Moved test to separate assembly `MySqlClientTests`.
- Fixed stupid problem in driver with sequence out of order (Thanks Peter Belbin).
- Added some pipe tests.
- Increased default max pool size to 50.
- Compiles with Mono 0-24.
- Fixed connection and data reader dispose problems.
- Added `String` datatype handling to parameter serialization.
- Fixed sequence problem in driver that occurred after thrown exception (thanks Burkhard Perkens-Golomb).
- Added support for `CommandBehavior.SingleRow` to `DataReader`.

- Fixed command sql processing so quotes are better handled (thanks Theo Spears).
- Fixed parsing of double, single, and decimal values to account for non-English separators. You still have to use the right syntax if you using hard coded sql, but if you use parameters the code will convert floating point types to use '.' appropriately internal both into the server and out.
- Added `MySqlStream` class to simplify timeouts and driver coding.
- Fixed `DataReader` so that it is closed properly when the associated connection is closed. [thanks smishra]
- Made client more SqlClient compliant so that DataReaders have to be closed before the connection can be used to run another command.
- Improved `DBNull.Value` handling in the fields.
- Added several unit tests.
- Fixed `MySqlException` base class.
- Improved driver coding
- Fixed bug where NextResult was returning false on the last resultset.
- Added more tests for MySQL.
- Improved casting problems by equating unsigned 32bit values to Int64 and unsigned 16bit values to Int32, and so forth.
- Added new constructor for `MySqlParameter` for (name, type, size, srccol)
- Fixed bug in `MySqlDataReader` where it didn't check for null fieldlist before returning field count.
- Started adding `MySqlClient` unit tests (added `MySqlClient/Tests` folder and some test cases).
- Fixed some things in Connection String handling.
- Moved `INIT_DB` to `MySqlPool`. I may move it again, this is in preparation of the conference.
- Fixed bug inside `CommandBuilder` that prevented inserts from happening properly.
- Reworked some of the internals so that all three execute methods of Command worked properly.
- Fixed many small bugs found during benchmarking.
- The first cut of `CoonectionPooling` is working. "min pool size" and "max pool size" are respected.
- Work to enable multiple resultsets to be returned.
- Character sets are handled much more intelligently now. The driver queries MySQL at startup for the default character set. That character set is then used for conversions if that code page can be loaded. If not, then the default code page for the current OS is used.
- Added code to save the inferred type in the name,value constructor of `Parameter`.
- Also, inferred type if value of null parameter is changed using `Value` property.
- Converted all files to use proper Camel case. MySQL is now MySql in all files. PgSQL is now PgSql.
- Added attribute to PgSql code to prevent designer from trying to show.
- Added `MySQLDbType` property to Parameter object and added proper conversion code to convert from `DbType` to `MySQLDbType`).
- Removed unused `ObjectToString` method from `MySQLParameter.cs`.
- Fixed `Add(..)` method in `ParameterCollection` so that it doesn't use `Add(name, value)` instead.
- Fixed `IndexOf` and `Contains` in `ParameterCollection` to be aware that parameter names are now stored without @.
- Fixed `Command.ConvertSQLToBytes` so it only allows characters that can be in MySQL variable names.
- Fixed `DataReader` and `Field` so that blob fields read their data from `Field.cs` and `GetBytes` works right.

- Added simple query builder editor to `CommandText` property of `MySQLCommand`.
- Fixed `CommandBuilder` and `Parameter` serialization to account for Parameters not storing @ in their names.
- Removed `MySQLFieldType` enum from Field.cs. Now using `MySQLDbType` enum.
- Added `Designer` attribute to several classes to prevent designer view when using VS.Net.
- Fixed Initial catalog typo in `ConnectionString` designer.
- Removed 3 parameter constructor for `MySQLParameter` that conflicted with (name, type, value).
- Changed `MySQLParameter` so `paramName` is now stored without leading @ (this fixed null inserts when using designer).
- Changed `TypeConverter` for `MySQLParameter` to use the constructor with all properties.

B.53. Changes in MySQL Connector/NET Version 0.68

- Fixed sequence issue in driver.
- Added `DbParametersEditor` to make parameter editing more like `SqlClient`.
- Fixed `Command` class so that parameters can be edited using the designer
- Update connection string designer to support `Use Compression` flag.
- Fixed string encoding so that European characters will work correctly.
- Creating base classes to aid in building new data providers.
- Added support for UID key in connection string.
- Field, parameter, command now using DBNull.Value instead of null.
- `CommandBuilder` using `DBNull.Value`.
- `CommandBuilder` now builds insert command correctly when an auto_insert field is not present.
- Field now uses typeof keyword to return `System.Types` (performance).

B.54. Changes in MySQL Connector/NET Version 0.65

- `MySQLCommandBuilder` now implemented.
- Transaction support now implemented (not all table types support this).
- `GetSchemaTable` fixed to not use xsd (for Mono).
- Driver is now Mono-compatible.
- TIME data type now supported.
- More work to improve Timestamp data type handling.
- Changed signatures of all classes to match corresponding `SqlClient` classes.

B.55. Changes in MySQL Connector/NET Version 0.60

- Protocol compression using SharpZipLib (www.icsharpcode.net).
- Named pipes on Windows now working properly.
- Work done to improve `Timestamp` data type handling.

- Implemented `IEnumerable` on `DataReader` so `DataGrid` would work.

B.56. Changes in MySQL Connector/NET Version 0.50

- Speed increased dramatically by removing bugging network sync code.
- Driver no longer buffers rows of data (more ADO.Net compliant).
- Conversion bugs related to `TIMESTAMP` and `DATETIME` fields fixed.

Appendix C. MySQL Visual Studio Plugin Change History

Note

As of Connector/NET 5.1.2 (14 June 2007), the Visual Studion Plugin is part of the main Connector/NET package. For the change history for the Visual Studio Plugin, see Appendix B, *MySQL Connector/NET Change History*.

C.1. Changes in MySQL Visual Studio Plugin 1.0.3 (Not yet released)

Bugs fixed:

- Running queries based on a stored procedure would cause the data set designer to terminate. (Bugs #26364)
- DataSet wizard would show all tables instead of only the tables available within the selected database. (Bugs #26348)

C.2. Changes in MySQL Visual Studio Plugin 1.0.2 (Not yet released)

Bugs fixed:

- The Add Connection dialog of the Server Explorer would freeze when accessing databases with capitalized characters in their name. (Bug#24875)
- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that Connector/NET 5.0.2 must be installed. (Bug#23071)

C.3. Changes in MySQL Visual Studio Plugin 1.0.1 (4 October 2006)

This is a bug fix release to resolve an incompatibility issue with Connector/NET 5.0.1.

It is critical that this release only be used with Connector/NET 5.0.1. After installing Connector/NET 5.0.1, you will need to make a small change in your machine.config file. This file should be located at `%win%\Microsoft.Net\Framework\v2.0.50727\CONFIG\machine.config` (`%win%` should be the location of your Windows folder). Near the bottom of the file you will see a line like this:

```
<add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data"/>
```

It needs to be changed to be like this:

```
<add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data,
Version=5.0.1.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
```

C.4. Changes in MySQL Visual Studio Plugin 1.0.0 (4 October 2006)

Bugs fixed:

- Ability to work with MySQL objects (tables, views, stored procedures, etc) from within Server Explorer.
- DDEX (Data Designer Extensibility) compatibility.

Appendix D. MySQL Connector/J Change History

D.1. Changes in MySQL Connector/J 5.1.x

D.1.1. Changes in MySQL Connector/J 5.1.8 (Not yet released)

Bugs fixed:

- The result set returned by `getIndexInfo()` did not have the format defined in the JDBC API specifications. The fourth column, `DATA_TYPE`, of the result set should be of type `BOOLEAN`. Connector/J however returns `CHAR`. (Bug#44869)
- The result set returned by `getTypeInfo()` did not have the format defined in the JDBC API specifications. The second column, `DATA_TYPE`, of the result set should be of type `INTEGER`. Connector/J however returns `SMALLINT`. (Bug#44868)
- The `DEFERRABILITY` column in database metadata result sets was expected to be of type `SHORT`. However, Connector/J returned it as `INTEGER`.

  This affected the following methods: `getImportedKeys()`, `getExportedKeys()`, `getCrossReference()`. (Bug#44867)
- The result set returned by `getColumns()` did not have the format defined in the JDBC API specifications. The fifth column, `DATA_TYPE`, of the result set should be of type `INTEGER`. Connector/J however returns `SMALLINT`. (Bug#44865)
- The result set returned by `getVersionColumns()` did not have the format defined in the JDBC API specifications. The third column, `DATA_TYPE`, of the result set should be of type `INTEGER`. Connector/J however returns `SMALLINT`. (Bug#44863)
- The result set returned by `getBestRowIdentifier()` did not have the format defined in the JDBC API specifications. The third column, `DATA_TYPE`, of the result set should be of type `INTEGER`. Connector/J however returns `SMALLINT`. (Bug#44862)
- Connector/J contains logic to generate a message text specifically for streaming result sets when there are `CommunicationsException` exceptions generated. However, this code was never reached.

  In the `CommunicationsException` code:

  ```
  private boolean streamingResultSetInPlay = false;

  public CommunicationsException(ConnectionImpl conn, long lastPacketSentTimeMs,
  long lastPacketReceivedTimeMs, Exception underlyingException) {

  this.exceptionMessage = SQLError.createLinkFailureMessageBasedOnHeuristics(conn,
        lastPacketSentTimeMs, lastPacketReceivedTimeMs, underlyingException,
        this.streamingResultSetInPlay);
  ```

  `streamingResultSetInPlay` was always false, which in the following code in `SQLError.createLinkFailureMessageBasedOnHeuristics()` never being executed:

  ```
  if (streamingResultSetInPlay) {
      exceptionMessageBuf.append(
      Messages.getString("CommunicationsException.ClientWasStreaming")); //$NON-NLS-1$
  } else {
  ...
  ```

  (Bug#44588)
- `Statement.getGeneratedKeys()` retained result set instances until the statement was closed. This caused memory leaks for long-lived statements, or statements used in tight loops. (Bug#44056)
- `LoadBalancingConnectionProxy.doPing()` did not have blacklist awareness.

  `LoadBalancingConnectionProxy` implemented `doPing()` to ping all underlying connections, but it threw any exceptions it encountered during this process.

  With the global blacklist enabled, it catches these exceptions, adds the host to the global blacklist, and only throws an exception if all hosts are down. (Bug#43421)
- When the MySQL Server was upgraded from 4.0 to 5.0, the Connector/J application then failed to connect to the server. This was because authentication failed when the application ran from EBCDIC platforms such as z/OS. (Bug#43071)

- When connecting with `traceProtocol=true`, no trace data was generated for the server greeting or login request. (Bug#43070)
- A `ConcurrentModificationException` was generated in `LoadBalancingConnectionProxy`:

  ```
  java.util.ConcurrentModificationException
   at java.util.HashMap$HashIterator.nextEntry(Unknown Source)
   at java.util.HashMap$KeyIterator.next(Unknown Source)
   at
  com.mysql.jdbc.LoadBalancingConnectionProxy.getGlobalBlacklist(LoadBalancingConnectionProxy.java:520)
   at com.mysql.jdbc.RandomBalanceStrategy.pickConnection(RandomBalanceStrategy.java:55)
   at
  com.mysql.jdbc.LoadBalancingConnectionProxy.pickNewConnection(LoadBalancingConnectionProxy.java:414)
   at
  com.mysql.jdbc.LoadBalancingConnectionProxy.invoke(LoadBalancingConnectionProxy.java:390)
  ```

  (Bug#42055)
- SQL injection was possible when using a string containing U+00A5 in a client-side prepared statement, and the character set being used was SJIS/Windows-31J. (Bug#41730)
- MySQL Connector/J 5.1.7 was slower than previous versions when the `rewriteBatchedStatements` option was set to `true`.

  Note

  The performance regression in `indexOfIgnoreCaseRespectMarker()`has been fixed. It has also been made possible for the driver to rewrite `INSERT` statements with `ON DUPLICATE KEY UPDATE` clauses in them, as long as the `UPDATE` clause contains no reference to `LAST_INSERT_ID()`, as that would cause the driver to return bogus values for `getGeneratedKeys()` invocations. This has resulted in improved performance over version 5.1.7.

  (Bug#41532)
- `PreparedStatement.addBatch()` did not check for all parameters being set, which led to inconsistent behavior in `executeBatch()`, especially when rewriting batched statements into multi-value `INSERT`s. (Bug#41161)

D.1.2. Changes in MySQL Connector/J 5.1.7 (21 October 2008)

Functionality added or changed:

- When statements include `ON DUPLICATE UPDATE`, and `rewriteBatchedStatements` is set to true, batched statements are not rewritten into the form `INSERT INTO table VALUES (), (), ()`, instead the statements are executed sequentially.

Bugs fixed:

- `Statement.getGeneratedKeys()` returned two keys when using `ON DUPLICATE KEY UPDATE` and the row was updated, not inserted. (Bug#42309)
- When using the replication driver with `autoReconnect=true`, Connector/J checks in `PreparedStatement.execute` (also called by `CallableStatement.execute`) to determine if the first character of the statement is an “S”, in an attempt to block all statements that are not read-only-safe, for example non-`SELECT` statements. However, this also blocked `CALL`s to stored procedures, even if the stored procedures were defined as `SQL READ DATA` or `NO SQL`. (Bug#40031)
- With large result sets `ResultSet.findColumn` became a performance bottleneck. (Bug#39962)
- Connector/J ignored the value of the MySQL Server variable `auto_increment_increment`. (Bug#39956)
- Connector/J failed to parse `TIMESTAMP` strings for nanos correctly. (Bug#39911)
- When the `LoadBalancingConnectionProxy` handles a `SQLException` with SQL state starting with “08”, it calls `invalidateCurrentConnection`, which in turn removes that `Connection` from `liveConnections` and the `connectionsToHostsMap`, but it did not add the host to the new global blacklist, if the global blacklist was enabled.

  There was also the possibility of a `NullPointerException` when trying to update stats, where `connectionsToHostsMap.get(this.currentConn)` was called:

  ```
  int hostIndex = ((Integer) this.hostsToListIndexMap.get(this.connectionsToHostsMap.get(this.currentConn))).intValue
  ```

This could happen if a client tried to issue a rollback after catching a `SQLException` caused by a connection failure. (Bug#39784)

- When configuring the Java Replication Driver the last slave specified was never used. (Bug#39611)
- When an `INSERT ON DUPLICATE KEY UPDATE` was performed, and the key already existed, the `affected-rows` value was returned as 1 instead of 0. (Bug#39352)
- When using the random load balancing strategy and starting with two servers that were both unavailable, an `IndexOutOfBoundsException` was generated when removing a server from the `whiteList`. (Bug#38782)
- Connector/J threw the following exception when using a read-only connection:

  ```
  java.sql.SQLException: Connection is read-only. Queries leading to data
  modification are not allowed.
  ```

  (Bug#38747)
- Connector/J was unable to connect when using a non-latin1 password. (Bug#37570)
- Incorrect result is returned from `isAfterLast()` in streaming `ResultSet` when using `setFetchSize(Integer.MIN_VALUE)`. (Bug#35170)
- When `getGeneratedKeys()` was called on a statement that had not been created with `RETURN_GENERATED_KEYS`, no exception was thrown, and batched executions then returned erroneous values. (Bug#34185)
- The `loadBalance` `bestResponseTime` blacklists did not have a global state. (Bug#33861)

D.1.3. Changes in MySQL Connector/J 5.1.6 (07 March 2008)

Functionality added or changed:

- Multiple result sets were not supported when using streaming mode to return data. Both normal statements and the resul sets from stored procedures now return multiple results sets, with the exception of result sets using registered `OUTPUT` paramaters. (Bug#33678)
- XAConnections and datasources have been updated to the JDBC-4.0 standard.
- The profiler event handling has been made extensible via the `profilerEventHandler` connection property.
- Add the `verifyServerCertificate` propery. If set to "false" the driver will not verify the server's certificate when `useSSL` is set to "true"

  When using this feature, the keystore parameters should be specified by the `clientCertificateKeyStore*` properties, rather than system properties, as the JSSE doesn't it straightforward to have a non-verifying trust store and the "default" key store.

Bugs fixed:

- `DatabaseMetaData.getColumns()` returns incorrect `COLUMN_SIZE` value for `SET` column. (Bug#36830)
- When trying to read `Time` values like “00:00:00” with `ResultSet.getTime(int)` an exception is thrown. (Bug#36051)
- JDBC connection URL parameters is ignored when using `MysqlConnectionPoolDataSource`. (Bug#35810)
- When `useServerPrepStmts=true` and slow query logging is enabled, the connector throws a `NullPointerException` when it encounters a slow query. (Bug#35666)
- When using the keyword “loadbalance” in the connection string and trying to perform load balancing between two databases, the driver appears to hang. (Bug#35660)
- JDBC data type getter method was changed to accept only column name, whereas previously it accepted column label. (Bug#35610)
- Prepared statements from pooled connections caused a `NullPointerException` when `closed()` under JDBC-4.0.

Bug#35489)

- In calling a stored function returning a `bigint`, an exception is encountered beginning:

  ```
  java.sql.SQLException: java.lang.NumberFormatException: For input string:
  ```

  followed by the text of the stored function starting after the argument list. (Bug#35199)

- The JDBC driver uses a different method for evaluating column names in `resultsetmetadata.getColumnName()` and when looking for a column in `resultset.getObject(columnName)`. This causes Hibernate to fail in queries where the two methods yield different results, for example in queries that use alias names:

  ```
  SELECT column AS aliasName from table
  ```

  (Bug#35150)

- `MysqlConnectionPoolDataSource` does not support `ReplicationConnection`. Notice that we implemented `com.mysql.jdbc.Connection` for `ReplicationConnection`, however, only accessors from ConnectionProperties are implemented (not the mutators), and they return values from the currently active connection. All other methods from `com.mysql.jdbc.Connection` are implemented, and operate on the currently active connection, with the exception of `resetServerState()` and `changeUser()`. (Bug#34937)

- `ResultSet.getTimestamp()` returns incorrect values for month/day of `TIMESTAMP`s when using server-side prepared statements (not enabled by default). (Bug#34913)

- `RowDataStatic` does't always set the metadata in `ResultSetRow`, which can lead to failures when unpacking `DATE`, `TIME`, `DATETIME` and `TIMESTAMP` types when using absolute, relative, and previous result set navigation methods. (Bug#34762)

- When calling `isValid()` on an active connection, if the timeout is nonzero then the `Connection` is invalidated even if the `Connection` is valid. (Bug#34703)

- It was not possible to truncate a `BLOB` using `Blog.truncate()` when using 0 as an argument. (Bug#34677)

- When using a cursor fetch for a statement, the internal prepared statement could cause a memory leak until the connection was closed. The internal prepared statement is now deleted when the corresponding result set is closed. (Bug#34518)

- When retrieving the column type name of a geometry field, the driver would return `UNKNOWN` instead of `GEOMETRY`. (Bug#34194)

- Statements with batched values do not return correct values for `getGeneratedKeys()` when `rewriteBatchedStatements` is set to `true`, and the statement has an `ON DUPLICATE KEY UPDATE` clause. (Bug#34093)

- The internal class `ResultSetInternalMethods` referenced the non-public class `com.mysql.jdbc.CachedResultSetMetaData`. (Bug#33823)

- A `NullPointerException` could be raised when using client-side prepared statements and enabled the prepared statement cache using the `cachePrepStmts`. (Bug#33734)

- Using server side cursors and cursor fetch, the table metadata information would return the data type name instead of the column name. (Bug#33594)

- `ResultSet.getTimestamp()` would throw a `NullPointerException` instead of a `SQLException` when called on an empty `ResultSet`. (Bug#33162)

- Load balancing connection using best response time would incorrectly "stick" to hosts that were down when the connection was first created.

  We solve this problem with a black list that is used during the picking of new hosts. If the black list ends up including all configured hosts, the driver will retry for a configurable number of times (the `retriesAllDown` configuration property, with a default of 120 times), sleeping 250ms between attempts to pick a new connection.

  We've also went ahead and made the balancing strategy extensible. To create a new strategy, implement the interface `com.mysql.jdbc.BalanceStrategy` (which also includes our standard "extension" interface), and tell the driver to use it by passing in the class name via the `loadBalanceStrategy` configuration property. (Bug#32877)

- During a Daylight Savings Time (DST) switchover, there was no way to store two timestamp/datetime values , as the hours end up being the same when sent as the literal that MySQL requires.

  Note that to get this scenario to work with MySQL (since it doesn't support per-value timezones), you need to configure your server (or session) to be in UTC, and tell the driver not to use the legacy date/time code by setting `useLegacyDatetime-`

`Code` to "false". This will cause the driver to always convert to/from the server and client timezone consistently.

This bug fix also fixes Bug#15604, by adding entirely new date/time handling code that can be switched on by `useLegacy-DatetimeCode` being set to "false" as a JDBC configuration property. For Connector/J 5.1.x, the default is "true", in trunk and beyond it will be "false" (i.e. the old date/time handling code will be deprecated) (Bug#32577, Bug#15604)

- When unpacking rows directly, we don't hand off error message packets to the internal method which decodes them correctly, so no exception is raised, and the driver than hangs trying to read rows that aren't there. This tends to happen when calling stored procedures, as normal SELECTs won't have an error in this spot in the protocol unless an I/O error occurs. (Bug#32246)
- When using a connection from `ConnectionPoolDataSource`, some `Connection.prepareStatement()` methods would return null instead of the prepared statement. (Bug#32101)
- Using `CallableStatement.setNull()` on a stored function would throw an `ArrayIndexOutOfBounds` exception when setting the last parameter to null. (Bug#31823)
- `MysqlValidConnectionChecker` doesn't properly handle connections created using `ReplicationConnection`. (Bug#31790)
- Retrieving the server version information for an active connection could return invalid information if the default character encoding on the host was not ASCII compatible. (Bug#31192)
- Further fixes have been made to this bug in the event that a node is non-responsive. Connector/J will now try a different random node instead of waiting for the node to recover before continuing. (Bug#31053)
- `ResultSet` returned by `Statement.getGeneratedKeys()` is not closed automatically when statement that created it is closed. (Bug#30508)
- `DatabaseMetadata.getColumns()` doesn't return the correct column names if the connection character isn't UTF-8. A bug in MySQL server compounded the issue, but was fixed within the MySQL 5.0 release cycle. The fix includes changes to all the sections of the code that access the server metadata. (Bug#20491)
- Fixed `ResultSetMetadata.getColumnName()` for result sets returned from `Statement.getGeneratedKeys()` - it was returning null instead of "GENERATED_KEY" as in 5.0.x.

D.1.4. Changes in MySQL Connector/J 5.1.5 (09 October 2007)

The following features are new, compared to the 5.0 series of Connector/J

- Support for JDBC-4.0 `NCHAR`, `NVARCHAR` and `NCLOB` types.
- JDBC-4.0 support for setting per-connection client information (which can be viewed in the comments section of a query via `SHOW PROCESSLIST` on a MySQL server, or can be extended to support custom persistence of the information via a public interface).
- Support for JDBC-4.0 XML processing via JAXP interfaces to DOM, SAX and StAX.
- JDBC-4.0 standardized unwrapping to interfaces that include vendor extensions.

Functionality added or changed:

- Added `autoSlowLog` configuration property, overrides `slowQueryThreshold*` properties, driver determines slow queries by those that are slower than 5 * stddev of the mean query time (outside the 96% percentile).

Bugs fixed:

- When a connection is in read-only mode, queries that are wrapped in parentheses were incorrectly identified DML statements. (Bug#28256)

D.1.5. Changes in MySQL Connector/J 5.1.4 (Not Released)

Only released internally.

D.1.6. Changes in MySQL Connector/J 5.1.3 (10 September 2007)

The following features are new, compared to the 5.0 series of Connector/J

- Support for JDBC-4.0 `NCHAR`, `NVARCHAR` and `NCLOB` types.
- JDBC-4.0 support for setting per-connection client information (which can be viewed in the comments section of a query via `SHOW PROCESSLIST` on a MySQL server, or can be extended to support custom persistence of the information via a public interface).
- Support for JDBC-4.0 XML processing via JAXP interfaces to DOM, SAX and StAX.
- JDBC-4.0 standardized unwrapping to interfaces that include vendor extensions.

Functionality added or changed:

- Connector/J now connects using an initial character set of `utf-8` solely for the purpose of authentication to allow user names or database names in any character set to be used in the JDBC connection URL. (Bug#29853)
- Added two configuration parameters:
  - `blobsAreStrings` — Should the driver always treat BLOBs as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
  - `functionsNeverReturnBlobs` — Should the driver always treat data from functions returning `BLOBs` as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
- Setting `rewriteBatchedStatements` to `true` now causes CallableStatements with batched arguments to be re-written in the form "CALL (...); CALL (...); ..." to send the batch in as few client-server round trips as possible.
- The driver now picks appropriate internal row representation (whole row in one buffer, or individual byte[]s for each column value) depending on heuristics, including whether or not the row has `BLOB` or `TEXT` types and the overall row-size. The threshold for row size that will cause the driver to use a buffer rather than individual byte[]s is configured by the configuration property `largeRowSizeThreshold`, which has a default value of 2KB.
- The data (and how it is stored) for `ResultSet` rows are now behind an interface which allows us (in some cases) to allocate less memory per row, in that for "streaming" result sets, we re-use the packet used to read rows, since only one row at a time is ever active.
- Added experimental support for statement "interceptors" via the `com.mysql.jdbc.StatementInterceptor` interface, examples are in `com/mysql/jdbc/interceptors`. Implement this interface to be placed "in between" query execution, so that it can be influenced (currently experimental).
- The driver will automatically adjust the server session variable `net_write_timeout` when it determines its been asked for a "streaming" result, and resets it to the previous value when the result set has been consumed. (The configuration property is named `netTimeoutForStreamingResults`, with a unit of seconds, the value '0' means the driver will not try and adjust this value).
- JDBC-4.0 ease-of-development features including auto-registration with the `DriverManager` via the service provider mechanism, standardized Connection validity checks and categorized `SQLExceptions` based on recoverability/retry-ability and class of the underlying error.
- `Statement.setQueryTimeout()`s now affect the entire batch for batched statements, rather than the individual statements that make up the batch.
- Errors encountered during `Statement`/`PreparedStatement`/`CallableStatement.executeBatch()` when `rewriteBatchStatements` has been set to `true` now return `BatchUpdateExceptions` according to the setting of `continueBatchOnError`.

  If `continueBatchOnError` is set to `true`, the update counts for the "chunk" that were sent as one unit will all be set to `EXECUTE_FAILED`, but the driver will attempt to process the remainder of the batch. You can determine which "chunk" failed by looking at the update counts returned in the `BatchUpdateException`.

  If `continueBatchOnError` is set to "false", the update counts returned will contain all updates up-to and including the failed "chunk", with all counts for the failed "chunk" set to `EXECUTE_FAILED`.

  Since MySQL doesn't return multiple error codes for multiple-statements, or for multi-value `INSERT`/`REPLACE`, it is the application's responsibility to handle determining which item(s) in the "chunk" actually failed.

- New methods on com.mysql.jdbc.Statement: setLocalInfileInputStream() and getLocalInfileInputStream():

  - 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 via setLocalInfileInputStream().

- Setting useBlobToStoreUTF8OutsideBMP to true 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.

  Set utf8OutsideBmpExcludedColumnNamePattern to a regex so that column names matching the given regex will still be treated as BLOBs The regex must follow the patterns used for the java.util.regexpackage. The default is to exclude no columns, and include all columns.

  Set utf8OutsideBmpIncludedColumnNamePattern to specify exclusion rules to utf8OutsideBmpExcludedColumnNamePattern". The regex must follow the patterns used for the java.util.regex package.

Bugs fixed:

- setObject(int, Object, int, int) delegate in PreparedStatmentWrapper delegates to wrong method. (Bug#30892)
- NPE with null column values when padCharsWithSpace is set to true. (Bug#30851)
- Collation on VARBINARY column types would be misidentified. A fix has been added, but this fix only works for MySQL server versions 5.0.25 and newer, since earlier versions didn't consistently return correct metadata for functions, and thus results from subqueries and functions were indistinguishable from each other, leading to type-related bugs. (Bug#30664)
- An ArithmeticException or NullPointerException would be raised when the batch had zero members and rewriteBatchedStatements=true when addBatch() was never called, or executeBatch() was called immediately after clearBatch(). (Bug#30550)
- Closing a load-balanced connection would cause a ClassCastException. (Bug#29852)
- Connection checker for JBoss didn't use same method parameters via reflection, causing connections to always seem "bad". (Bug#29106)
- DatabaseMetaData.getTypeInfo() for the types DECIMAL and NUMERIC will return a precision of 254 for server versions older than 5.0.3, 64 for versions 5.0.3-5.0.5 and 65 for versions newer than 5.0.5. (Bug#28972)
- CallableStatement.executeBatch() doesn't work when connection property noAccessToProcedureBodies has been set to true.

  The fix involves changing the behavior of noAccessToProcedureBodies,in that the driver will now report all paramters as "IN" paramters but allow callers to call registerOutParameter() on them without throwing an exception. (Bug#28689)
- DatabaseMetaData.getColumns() doesn't contain SCOPE_* or IS_AUTOINCREMENT columns. (Bug#27915)
- Schema objects with identifiers other than the connection character aren't retrieved correctly in ResultSetMetadata. (Bug#27867)
- Connection.getServerCharacterEncoding() doesn't work for servers with version >= 4.1. (Bug#27182)
- The automated SVN revisions in DBMD.getDriverVersion(). The SVN revision of the directory is now inserted into the version information during the build. (Bug#21116)

- Specifying a "validation query" in your connection pool that starts with "/* ping */" _exactly_ will cause the driver to instead send a ping to the server and return a fake result set (much lighter weight), and when using a ReplicationConnection or a LoadBalancedConnection, will send the ping across all active connections.

D.1.7. Changes in MySQL Connector/J 5.1.2 (29 June 2007)

This is a new Beta development release, fixing recently discovered bugs.

Functionality added or changed:

- Setting the configuration property `rewriteBatchedStatements` to `true` will now cause the driver to rewrite batched prepared statements with more than 3 parameter sets in a batch into multi-statements (separated by ";") if they are not plain (that is, without `SELECT` or `ON DUPLICATE KEY UPDATE` clauses) `INSERT` or `REPLACE` statements.

D.1.8. Changes in MySQL Connector/J 5.1.1 (22 June 2007)

This is a new Alpha development release, adding new features and fixing recently discovered bugs.

Functionality added or changed:

- **Incompatible Change**: Pulled vendor-extension methods of `Connection` implementation out into an interface to support `java.sql.Wrapper` functionality from `ConnectionPoolDataSource`. The vendor extensions are javadoc'd in the `com.mysql.jdbc.Connection` interface.

  For those looking further into the driver implementation, it is not an API that is used for plugability of implementations inside our driver (which is why there are still references to `ConnectionImpl` throughout the code).

  We've also added server and client `prepareStatement()` methods that cover all of the variants in the JDBC API.

  `Connection.serverPrepare(String)` has been re-named to `Connection.serverPrepareStatement()` for consistency with `Connection.clientPrepareStatement()`.

- Row navigation now causes any streams/readers open on the result set to be closed, as in some cases we're reading directly from a shared network packet and it will be overwritten by the "next" row.

- Made it possible to retrieve prepared statement parameter bindings (to be used in `StatementInterceptors`, primarily).

- Externalized the descriptions of connection properties.

- The data (and how it is stored) for `ResultSet` rows are now behind an interface which allows us (in some cases) to allocate less memory per row, in that for "streaming" result sets, we re-use the packet used to read rows, since only one row at a time is ever active.

- Similar to `Connection`, we pulled out vendor extensions to `Statement` into an interface named `com.mysql.Statement`, and moved the `Statement` class into `com.mysql.StatementImpl`. The two methods (javadoc'd in `com.mysql.Statement` are `enableStreamingResults()`, which already existed, and `disableStreamingResults()` which sets the statement instance back to the fetch size and result set type it had before `enableStreamingResults()` was called.

- Driver now picks appropriate internal row representation (whole row in one buffer, or individual byte[]s for each column value) depending on heuristics, including whether or not the row has `BLOB` or `TEXT` types and the overall row-size. The threshold for row size that will cause the driver to use a buffer rather than individual byte[]s is configured by the configuration property `largeRowSizeThreshold`, which has a default value of 2KB.

- Added experimental support for statement "interceptors" via the `com.mysql.jdbc.StatementInterceptor` interface, examples are in `com/mysql/jdbc/interceptors`.

  Implement this interface to be placed "in between" query execution, so that you can influence it. (currently experimental).

  `StatementInterceptors` are "chainable" when configured by the user, the results returned by the "current" interceptor will be passed on to the next on in the chain, from left-to-right order, as specified by the user in the JDBC configuration property `statementInterceptors`.

- See the sources (fully javadoc'd) for `com.mysql.jdbc.StatementInterceptor` for more details until we iron out the API and get it documented in the manual.

- Setting `rewriteBatchedStatements` to `true` now causes `CallableStatements` with batched arguments to be re-written in the form `CALL (...); CALL (...); ...` to send the batch in as few client-server round trips as possible.

D.1.9. Changes in MySQL Connector/J 5.1.0 (11 April 2007)

This is the first public alpha release of the current Connector/J 5.1 development branch, providing an insight to upcoming features. Although some of these are still under development, this release includes the following new features and changes (in comparison to the current Connector/J 5.0 production release):

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is `false` (that is, Connector/J does not use server-side prepared statements).

> **Note**
>
> The disabling of server-side prepared statements does not affect the operation of the connector. However, if you use the `useTimezone=true` connection option and use client-side prepared statements (instead of server-side prepared statements) you should also set `useSSPSCompatibleTimezoneShift=true`.

Functionality added or changed:

- Refactored `CommunicationsException` into a JDBC-3.0 version, and a JDBC-4.0 version (which extends `SQLRecoverableException`, now that it exists).

  > **Note**
  >
  > This change means that if you were catching `com.mysql.jdbc.CommunicationsException` in your applications instead of looking at the SQLState class of `08`, and are moving to Java 6 (or newer), you need to change your imports to that exception to be `com.mysql.jdbc.exceptions.jdbc4.CommunicationsException`, as the old class will not be instantiated for communications link-related errors under Java 6.

- Added support for JDBC-4.0 categorized `SQLExceptions`.
- Added support for JDBC-4.0's `NCLOB`, and `NCHAR`/`NVARCHAR` types.
- `com.mysql.jdbc.java6.javac` — full path to your Java-6 `javac` executable
- Added support for JDBC-4.0's SQLXML interfaces.
- Re-worked Ant buildfile to build JDBC-4.0 classes separately, as well as support building under Eclipse (since Eclipse can't mix/match JDKs).

  To build, you must set `JAVA_HOME` to J2SDK-1.4.2 or Java-5, and set the following properties on your Ant command line:

  - `com.mysql.jdbc.java6.javac` — full path to your Java-6 `javac` executable
  - `com.mysql.jdbc.java6.rtjar` — full path to your Java-6 `rt.jar` file

- New feature — driver will automatically adjust session variable `net_write_timeout` when it determines it has been asked for a "streaming" result, and resets it to the previous value when the result set has been consumed. (configuration property is named `netTimeoutForStreamingResults` value and has a unit of seconds, the value `0` means the driver will not try and adjust this value).
- Added support for JDBC-4.0's client information. The backend storage of information provided via `Connection.setClientInfo()` and retrieved by `Connection.getClientInfo()` is pluggable by any class that implements the `com.mysql.jdbc.JDBC4ClientInfoProvider` interface and has a no-args constructor.

  The implementation used by the driver is configured using the `clientInfoProvider` configuration property (with a default of value of `com.mysql.jdbc.JDBC4CommentClientInfoProvider`, an implementation which lists the client information as a comment prepended to every query sent to the server).

  This functionality is only available when using Java-6 or newer.

- `com.mysql.jdbc.java6.rtjar` — full path to your Java-6 `rt.jar` file
- Added support for JDBC-4.0's `Wrapper` interface.

D.2. Changes in MySQL Connector/J 5.0.x

D.2.1. Changes in MySQL Connector/J 5.0.8 (09 October 2007)

Functionality added or changed:

- `blobsAreStrings` — Should the driver always treat BLOBs as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
- Added two configuration parameters:
  - `blobsAreStrings` — Should the driver always treat BLOBs as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
  - `functionsNeverReturnBlobs` — Should the driver always treat data from functions returning `BLOBs` as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
- `functionsNeverReturnBlobs` — Should the driver always treat data from functions returning `BLOBs` as Strings. Added specifically to work around dubious metadata returned by the server for `GROUP BY` clauses. Defaults to false.
- XAConnections now start in auto-commit mode (as per JDBC-4.0 specification clarification).
- Driver will now fall back to sane defaults for `max_allowed_packet` and `net_buffer_length` if the server reports them incorrectly (and will log this situation at `WARN` level, since it is actually an error condition).

Bugs fixed:

- Connections established using URLs of the form `jdbc:mysql:loadbalance://` weren't doing failover if they tried to connect to a MySQL server that was down. The driver now attempts connections to the next "best" (depending on the load balance strategy in use) server, and continues to attempt connecting to the next "best" server every 250 milliseconds until one is found that is up and running or 5 minutes has passed.

  If the driver gives up, it will throw the last-received `SQLException`. (Bug#31053)
- `setObject(int, Object, int, int)` delegate in PreparedStatmentWrapper delegates to wrong method. (Bug#30892)
- NPE with null column values when `padCharsWithSpace` is set to true. (Bug#30851)
- Collation on `VARBINARY` column types would be misidentified. A fix has been added, but this fix only works for MySQL server versions 5.0.25 and newer, since earlier versions didn't consistently return correct metadata for functions, and thus results from subqueries and functions were indistinguishable from each other, leading to type-related bugs. (Bug#30664)
- An `ArithmeticException` or `NullPointerException` would be raised when the batch had zero members and `rewriteBatchedStatements=true` when `addBatch()` was never called, or `executeBatch()` was called immediately after `clearBatch()`. (Bug#30550)
- Closing a load-balanced connection would cause a `ClassCastException`. (Bug#29852)
- Connection checker for JBoss didn't use same method parameters via reflection, causing connections to always seem "bad". (Bug#29106)
- `DatabaseMetaData.getTypeInfo()` for the types `DECIMAL` and `NUMERIC` will return a precision of 254 for server versions older than 5.0.3, 64 for versions 5.0.3-5.0.5 and 65 for versions newer than 5.0.5. (Bug#28972)
- `CallableStatement.executeBatch()` doesn't work when connection property `noAccessToProcedureBodies` has been set to `true`.

  The fix involves changing the behavior of `noAccessToProcedureBodies`,in that the driver will now report all paramters as "IN" paramters but allow callers to call registerOutParameter() on them without throwing an exception. (Bug#28689)
- When a connection is in read-only mode, queries that are wrapped in parentheses were incorrectly identified DML statements.

Bug#28256)

- `UNSIGNED` types not reported via `DBMD.getTypeInfo()`, and capitalization of type names is not consistent between `DBMD.getColumns()`, `RSMD.getColumnTypeName()` and `DBMD.getTypeInfo()`.

  This fix also ensures that the precision of `UNSIGNED MEDIUMINT` and `UNSIGNED BIGINT` is reported correctly via `DBMD.getColumns()`. (Bug#27916)

- `DatabaseMetaData.getColumns()` doesn't contain `SCOPE_*` or `IS_AUTOINCREMENT` columns. (Bug#27915)

- Schema objects with identifiers other than the connection character aren't retrieved correctly in `ResultSetMetadata`. (Bug#27867)

- Cached metadata with `PreparedStatement.execute()` throws `NullPointerException`. (Bug#27412)

- `Connection.getServerCharacterEncoding()` doesn't work for servers with version >= 4.1. (Bug#27182)

- The automated SVN revisions in `DBMD.getDriverVersion()`. The SVN revision of the directory is now inserted into the version information during the build. (Bug#21116)

- Specifying a "validation query" in your connection pool that starts with "/* ping */" _exactly_ will cause the driver to instead send a ping to the server and return a fake result set (much lighter weight), and when using a ReplicationConnection or a LoadBalancedConnection, will send the ping across all active connections.

D.2.2. Changes in MySQL Connector/J 5.0.7 (20 July 2007)

Functionality added or changed:

- The driver will now automatically set `useServerPrepStmts` to `true` when `useCursorFetch` has been set to `true`, since the feature requires server-side prepared statements in order to function.

- `tcpKeepAlive` - Should the driver set SO_KEEPALIVE (default `true`)?

- Give more information in EOFExceptions thrown out of MysqlIO (how many bytes the driver expected to read, how many it actually read, say that communications with the server were unexpectedly lost).

- Driver detects when it is running in a ColdFusion MX server (tested with version 7), and uses the configuration bundle `coldFusion`, which sets `useDynamicCharsetInfo` to `false` (see previous entry), and sets `useLocalSessionState` and autoReconnect to `true`.

- `tcpNoDelay` - Should the driver set SO_TCP_NODELAY (disabling the Nagle Algorithm, default `true`)?

- Added configuration property `slowQueryThresholdNanos` - if `useNanosForElapsedTime` is set to `true`, and this property is set to a nonzero value the driver will use this threshold (in nanosecond units) to determine if a query was slow, instead of using millisecond units.

- `tcpRcvBuf` - 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.

- Setting `useDynamicCharsetInfo` to `false` now causes driver to use static lookups for collations as well (makes ResultSetMetadata.isCaseSensitive() much more efficient, which leads to performance increase for ColdFusion, which calls this method for every column on every table it sees, it appears).

- Added configuration properties to allow tuning of TCP/IP socket parameters:

  - `tcpNoDelay` - Should the driver set SO_TCP_NODELAY (disabling the Nagle Algorithm, default `true`)?

  - `tcpKeepAlive` - Should the driver set SO_KEEPALIVE (default `true`)?

  - `tcpRcvBuf` - 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.

  - `tcpSndBuf` - 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.

  - `tcpTrafficClass` - Should the driver set traffic class or type-of-service fields? See the documentation for java.net.Socket.setTrafficClass() for more information.

- Setting the configuration parameter `useCursorFetch` to `true` for MySQL-5.0+ enables the use of cursors that allow Connector/J to save memory by fetching result set rows in chunks (where the chunk size is set by calling setFetchSize() on a Statement or ResultSet) by using fully-materialized cursors on the server.

- `tcpSndBuf` - 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.
- `tcpTrafficClass` - Should the driver set traffic class or type-of-service fields? See the documentation for java.net.Socket.setTrafficClass() for more information.
- Added new debugging functionality - Setting configuration property `includeInnodbStatusInDeadlockExceptions` to `true` will cause the driver to append the output of `SHOW ENGINE INNODB STATUS` to deadlock-related exceptions, which will enumerate the current locks held inside InnoDB.
- Added configuration property `useNanosForElapsedTime` - for profiling/debugging functionality that measures elapsed time, should the driver try to use nanoseconds resolution if available (requires JDK >= 1.5)?

  Note

  If `useNanosForElapsedTime` is set to `true`, and this property is set to "0" (or left default), then elapsed times will still be measured in nanoseconds (if possible), but the slow query threshold will be converted from milliseconds to nanoseconds, and thus have an upper bound of approximately 2000 milliseconds (as that threshold is represented as an integer, not a long).

Bugs fixed:

- Don't send any file data in response to LOAD DATA LOCAL INFILE if the feature is disabled at the client side. This is to prevent a malicious server or man-in-the-middle from asking the client for data that the client is not expecting. Thanks to Jan Kneschke for discovering the exploit and Andrey "Poohie" Hristov, Konstantin Osipov and Sergei Golubchik for discussions about implications and possible fixes. (Bug#29605)
- Parser in client-side prepared statements runs to end of statement, rather than end-of-line for '#' comments. Also added support for '--' single-line comments. (Bug#28956)
- Parser in client-side prepared statements eats character following '/' if it is not a multi-line comment. (Bug#28851)
- PreparedStatement.getMetaData() for statements containing leading one-line comments is not returned correctly.

  As part of this fix, we also overhauled detection of DML for `executeQuery()` and `SELECT`s for `executeUpdate()` in plain and prepared statements to be aware of the same types of comments. (Bug#28469)

D.2.3. Changes in MySQL Connector/J 5.0.6 (15 May 2007)

Functionality added or changed:

- Added an experimental load-balanced connection designed for use with SQL nodes in a MySQL Cluster/NDB environment (This is not for master-slave replication. For that, we suggest you look at `ReplicationConnection` or `lbpool`).

  If the JDBC URL starts with `jdbc:mysql:loadbalance://host-1,host-2,...host-n`, the driver will create an implementation of `java.sql.Connection` that load balances requests across a series of MySQL JDBC connections to the given hosts, where the balancing takes place after transaction commit.

  Therefore, for this to work (at all), you must use transactions, even if only reading data.

  Physical connections to the given hosts will not be created until needed.

  The driver will invalidate connections that it detects have had communication errors when processing a request. A new connection to the problematic host will be attempted the next time it is selected by the load balancing algorithm.

  There are two choices for load balancing algorithms, which may be specified by the `loadBalanceStrategy` JDBC URL configuration property:

  - `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.
  - `bestResponseTime` — the driver will route the request to the host that had the best response time for the previous transaction.

- `bestResponseTime` — the driver will route the request to the host that had the best response time for the previous transaction.

- Added configuration property `padCharsWithSpace` (defaults to `false`). If set to `true`, and 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, the driver will pad the remaining characters with space (for ANSI compliance).

- When `useLocalSessionState` is set to `true` and connected to a MySQL-5.0 or later server, the JDBC driver will now determine whether an actual `commit` or `rollback` statement needs to be sent to the database when `Connection.commit()` or `Connection.rollback()` is called.

  This is especially helpful for high-load situations with connection pools that always call `Connection.rollback()` on connection check-in/check-out because it avoids a round-trip to the server.

- Added configuration property `useDynamicCharsetInfo`. If set to `false` (the default), the driver will use a per-connection cache of character set information queried from the server when necessary, or when set to `true`, 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.

  Note

  This only affects the `padCharsWithSpace` configuration property and the `ResultSetMetaData.getColumnDisplayWidth()` method.

- New configuration property, `enableQueryTimeouts` (default `true`).

  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. (this configuration property is part of the `maxPerformance` configuration bundle).

- Give better error message when "streaming" result sets, and the connection gets clobbered because of exceeding `net_write_timeout` on the server.

- `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.

- `com.mysql.jdbc.[NonRegistering]Driver` now understands URLs of the format `jdbc:mysql:replication://` and `jdbc:mysql:loadbalance://` which will create a ReplicationConnection (exactly like when using `[NonRegistering]ReplicationDriver`) and an experimental load-balanced connection designed for use with SQL nodes in a MySQL Cluster/NDB environment, respectively.

  In an effort to simplify things, we're working on deprecating multiple drivers, and instead specifying different core behavior based upon JDBC URL prefixes, so watch for `[NonRegistering]ReplicationDriver` to eventually disappear, to be replaced with `com.mysql.jdbc[NonRegistering]Driver` with the new URL prefix.

- Fixed issue where a failed-over connection would let an application call `setReadOnly(false)`, when that call should be ignored until the connection is reconnected to a writable master unless `failoverReadOnly` had been set to `false`.

- Driver will now use `INSERT INTO ... VALUES (DEFAULT)`form of statement for updatable result sets for `ResultSet.insertRow()`, rather than pre-populating the insert row with values from `DatabaseMetaData.getColumns()`(which results in a `SHOW FULL COLUMNS` on the server for every result set). If an application requires access to the default values before `insertRow()` has been called, the JDBC URL should be configured with `populateInsertRowWithDefaultValues` set to `true`.

  This fix specifically targets performance issues with ColdFusion and the fact that it seems to ask for updatable result sets no matter what the application does with them.

- More intelligent initial packet sizes for the "shared" packets are used (512 bytes, rather than 16K), and initial packets used during handshake are now sized appropriately as to not require reallocation.

Bugs fixed:

- More useful error messages are generated when the driver thinks a result set is not updatable. (Thanks to Ashley Martens for the patch). (Bug#28085)

- `Connection.getTransactionIsolation()` uses "`SHOW VARIABLES LIKE`" which is very inefficient on MySQL-

5.0+ servers. (Bug#27655)

- Fixed issue where calling `getGeneratedKeys()` on a prepared statement after calling `execute()` didn't always return the generated keys (`executeUpdate()` worked fine however). (Bug#27655)

- `CALL /* ... */ some_proc()` doesn't work. As a side effect of this fix, you can now use `/* */` and `#` comments when preparing statements using client-side prepared statement emulation.

  If the comments happen to contain parameter markers (`?`), they will be treated as belonging to the comment (that is, not recognized) rather than being a parameter of the statement.

  > **Note**
  >
  > The statement when sent to the server will contain the comments as-is, they're not stripped during the process of preparing the `PreparedStatement` or `CallableStatement`.

  (Bug#27400)

- `ResultSet.get*()` with a column index < 1 returns misleading error message. (Bug#27317)

- Using `ResultSet.get*()` with a column index less than 1 returns a misleading error message. (Bug#27317)

- Comments in DDL of stored procedures/functions confuse procedure parser, and thus metadata about them can not be created, leading to inability to retrieve said metadata, or execute procedures that have certain comments in them. (Bug#26959)

- Fast date/time parsing doesn't take into account `00:00:00` as a legal value. (Bug#26789)

- `PreparedStatement` is not closed in `BlobFromLocator.getBytes()`. (Bug#26592)

- When the configuration property `useCursorFetch` was set to `true`, sometimes server would return new, more exact metadata during the execution of the server-side prepared statement that enables this functionality, which the driver ignored (using the original metadata returned during `prepare()`), causing corrupt reading of data due to type mismatch when the actual rows were returned. (Bug#26173)

- `CallableStatements` with `OUT/INOUT` parameters that are "binary" (`BLOB`, `BIT`, `(VAR)BINARY`, `JAVA_OBJECT`) have extra 7 bytes. (Bug#25715)

- Whitespace surrounding storage/size specifiers in stored procedure parameters declaration causes `NumberFormatException` to be thrown when calling stored procedure on JDK-1.5 or newer, as the Number classes in JDK-1.5+ are whitespace intolerant. (Bug#25624)

- Client options not sent correctly when using SSL, leading to stored procedures not being able to return results. Thanks to Don Cohen for the bug report, testcase and patch. (Bug#25545)

- `Statement.setMaxRows()` is not effective on result sets materialized from cursors. (Bug#25517)

- `BIT(> 1)` is returned as `java.lang.String` from `ResultSet.getObject()` rather than `byte[]`. (Bug#25328)

D.2.4. Changes in MySQL Connector/J 5.0.5 (02 March 2007)

Functionality added or changed:

- Usage Advisor will now issue warnings for result sets with large numbers of rows. You can configure the trigger value by using the `resultSetSizeThreshold` parameter, which has a default value of 100.

- The `rewriteBatchedStatements` feature can now be used with server-side prepared statements.

- **Important change:** Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

  To enable server-side prepared statements you must add the following configuration property to your connector string:

  ```
  useServerPrepStmts=true
  ```

  The default value of this property is `false` (that is, Connector/J does not use server-side prepared statements).

- Improved speed of `datetime` parsing for ResultSets that come from plain or non-server-side prepared statements. You can enable old implementation with `useFastDateParsing=false` as a configuration parameter.

- Usage Advisor now detects empty results sets and does not report on columns not referenced in those empty sets.
- Fixed logging of XA commands sent to server, it is now configurable via `logXaCommands` property (defaults to `false`).
- Added configuration property `localSocketAddress`, which is the host name 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.
- We've added a new configuration option `treatUtilDateAsTimestamp`, which is `false` by default, as (1) We already had specific behavior to treat java.util.Date as a java.sql.Timestamp because it is useful to many folks, and (2) that behavior will very likely be required for drivers JDBC-post-4.0.

Bugs fixed:

- Connection property `socketFactory` wasn't exposed via correctly named mutator/accessor, causing data source implementations that use JavaBean naming conventions to set properties to fail to set the property (and in the case of SJAS, fail silently when trying to set this parameter). (Bug#26326)
- A query execution which timed out did not always throw a `MySQLTimeoutException`. (Bug#25836)
- Storing a `java.util.Date` object in a `BLOB` column would not be serialized correctly during `setObject`. (Bug#25787)
- Timer instance used for `Statement.setQueryTimeout()` created per-connection, rather than per-VM, causing memory leak. (Bug#25514)
- `EscapeProcessor` gets confused by multiple backslashes. We now push the responsibility of syntax errors back on to the server for most escape sequences. (Bug#25399)
- `INOUT` parameters in `CallableStatements` get doubly-escaped. (Bug#25379)
- When using the `rewriteBatchedStatements` connection option with `PreparedState.executeBatch()` an internal memory leak would occur. (Bug#25073)
- Fixed issue where field-level for metadata from `DatabaseMetaData` when using `INFORMATION_SCHEMA` didn't have references to current connections, sometimes leading to Null Pointer Exceptions (NPEs) when introspecting them via `ResultSetMetaData`. (Bug#25073)
- `StringUtils.indexOfIgnoreCaseRespectQuotes()` isn't case-insensitive on the first character of the target. This bug also affected `rewriteBatchedStatements` functionality when prepared statements did not use uppercase for the `VALUES` clause. (Bug#25047)
- Client-side prepared statement parser gets confused by in-line comments `/*...*/` and therefore cannot rewrite batch statements or reliably detect the type of statements when they are used. (Bug#25025)
- Results sets from `UPDATE` statements that are part of multi-statement queries would cause an `SQLException` error, "Result is from UPDATE". (Bug#25009)
- Specifying `US-ASCII` as the character set in a connection to a MySQL 4.1 or newer server does not map correctly. (Bug#24840)
- Using `DatabaseMetaData.getSQLKeywords()` does not return a all of the of the reserved keywords for the current MySQL version. Current implementation returns the list of reserved words for MySQL 5.1, and does not distinguish between versions. (Bug#24794)
- Calling `Statement.cancel()` could result in a Null Pointer Exception (NPE). (Bug#24721)
- Using `setFetchSize()` breaks prepared `SHOW` and other commands. (Bug#24360)
- Calendars and timezones are now lazily instantiated when required. (Bug#24351)
- Using `DATETIME` columns would result in time shifts when `useServerPrepStmts` was true. The reason was due to different behavior when using client-side compared to server-side prepared statements and the `useJDBCCompliantTimezoneShift` option. This is now fixed if moving from server-side prepared statements to client-side prepared statements by setting `useSSPSCompatibleTimezoneShift` to `true`, as the driver can't tell if this is a new deployment that never used server-side prepared statements, or if it is an existing deployment that is switching to client-side prepared statements from server-side prepared statements. (Bug#24344)
- Connector/J now returns a better error message when server doesn't return enough information to determine stored procedure/function parameter types. (Bug#24065)

- A connection error would occur when connecting to a MySQL server with certain character sets. Some collations/character sets reported as "unknown" (specifically `cias` variants of existing character sets), and inability to override the detected server character set. (Bug#23645)
- Inconsistency between `getSchemas` and `INFORMATION_SCHEMA`. (Bug#23304)
- `DatabaseMetaData.getSchemas()` doesn't return a `TABLE_CATALOG` column. (Bug#23303)
- When using a JDBC connection URL that is malformed, the `NonRegisteringDriver.getPropertyInfo` method will throw a Null Pointer Exception (NPE). (Bug#22628)
- Some exceptions thrown out of `StandardSocketFactory` were needlessly wrapped, obscuring their true cause, especially when using socket timeouts. (Bug#21480)
- When using a server-side prepared statement the driver would send timestamps to the server using nanoseconds instead of milliseconds. (Bug#21438)
- When using server-side prepared statements and timestamp columns, value would be incorrectly populated (with nanoseconds, not microseconds). (Bug#21438)
- `ParameterMetaData` throws `NullPointerException` when prepared SQL has a syntax error. Added `generateSimpleParameterMetadata` configuration property, which when set to `true` will generate metadata reflecting `VARCHAR` for every parameter (the default is `false`, which will cause an exception to be thrown if no parameter metadata for the statement is actually available). (Bug#21267)
- Fixed an issue where `XADataSources` couldn't be bound into JNDI, as the `DataSourceFactory` didn't know how to create instances of them.

Other changes:

- Avoid static synchronized code in JVM class libraries for dealing with default timezones.
- Performance enhancement of initial character set configuration, driver will only send commands required to configure connection character set session variables if the current values on the server do not match what is required.
- Re-worked stored procedure parameter parser to be more robust. Driver no longer requires `BEGIN` in stored procedure definition, but does have requirement that if a stored function begins with a label directly after the "returns" clause, that the label is not a quoted identifier.
- Throw exceptions encountered during timeout to thread calling `Statement.execute*()`, rather than `RuntimeException`.
- Changed cached result set metadata (when using `cacheResultSetMetadata=true`) to be cached per-connection rather than per-statement as previously implemented.
- Reverted back to internal character conversion routines for single-byte character sets, as the ones internal to the JVM are using much more CPU time than our internal implementation.
- When extracting foreign key information from `SHOW CREATE TABLE` in `DatabaseMetaData`, ignore exceptions relating to tables being missing (which could happen for cross-reference or imported-key requests, as the list of tables is generated first, then iterated).
- Fixed some Null Pointer Exceptions (NPEs) when cached metadata was used with `UpdatableResultSets`.
- Take `localSocketAddress` property into account when creating instances of `CommunicationsException` when the underyling exception is a `java.net.BindException`, so that a friendlier error message is given with a little internal diagnostics.
- Fixed cases where `ServerPreparedStatements` weren't using cached metadata when `cacheResultSetMetadata=true` was used.
- Use a `java.util.TreeMap` to map column names to ordinal indexes for `ResultSet.findColumn()` instead of a HashMap. This allows us to have case-insensitive lookups (required by the JDBC specification) without resorting to the many transient object instances needed to support this requirement with a normal `HashMap` with either case-adjusted keys, or case-insensitive keys. (In the worst case scenario for lookups of a 1000 column result set, TreeMaps are about half as fast wall-clock time as a HashMap, however in normal applications their use gives many orders of magnitude reduction in transient object instance creation which pays off later for CPU usage in garbage collection).
- When using cached metadata, skip field-level metadata packets coming from the server, rather than reading them and discard-

ing them without creating `com.mysql.jdbc.Field` instances.

D.2.5. Changes in MySQL Connector/J 5.0.4 (20 October 2006)

Bugs fixed:

- DBMD.getColumns() does not return expected COLUMN_SIZE for the SET type, now returns length of largest possible set disregarding whitespace or the "," delimitters to be consistent with the ODBC driver. (Bug#22613)
- Added new _ci collations to CharsetMapping - utf8_unicode_ci not working. (Bug#22456)
- Driver was using milliseconds for Statement.setQueryTimeout() when specification says argument is to be in seconds. (Bug#22359)
- Workaround for server crash when calling stored procedures via a server-side prepared statement (driver now detects prepare(stored procedure) and substitutes client-side prepared statement). (Bug#22297)
- Driver issues truncation on write exception when it shouldn't (due to sending big decimal incorrectly to server with server-side prepared statement). (Bug#22290)
- Newlines causing whitespace to span confuse procedure parser when getting parameter metadata for stored procedures. (Bug#22024)
- When using information_schema for metadata, COLUMN_SIZE for getColumns() is not clamped to range of java.lang.Integer as is the case when not using information_schema, thus leading to a truncation exception that isn't present when not using information_schema. (Bug#21544)
- Column names don't match metadata in cases where server doesn't return original column names (column functions) thus breaking compatibility with applications that expect 1-1 mappings between findColumn() and rsmd.getColumnName(), usually manifests itself as "Can't find column ('')" exceptions. (Bug#21379)
- Driver now sends numeric 1 or 0 for client-prepared statement `setBoolean()` calls instead of '1' or '0'.
- Fixed configuration property `jdbcCompliantTruncation` was not being used for reads of result set values.
- DatabaseMetaData correctly reports `true` for `supportsCatalog*()` methods.
- Driver now supports `{call sp}` (without "()" if procedure has no arguments).

D.2.6. Changes in MySQL Connector/J 5.0.3 (26 July 2006 beta)

Functionality added or changed:

- Added configuration option `noAccessToProcedureBodies` which will cause the driver to create basic parameter metadata for `CallableStatements` when the user does not have access to procedure bodies via `SHOW CREATE PROCEDURE` or selecting from `mysql.proc` instead of throwing an exception. The default value for this option is `false`

Bugs fixed:

- Fixed `Statement.cancel()` causes `NullPointerException` if underlying connection has been closed due to server failure. (Bug#20650)
- If the connection to the server has been closed due to a server failure, then the cleanup process will call `Statement.cancel()`, triggering a `NullPointerException`, even though there is no active connection. (Bug#20650)

D.2.7. Changes in MySQL Connector/J 5.0.2 (11 July 2006)

Bugs fixed:

- `MysqlXaConnection.recover(int flags)` now allows combinations of `XAResource.TMSTARTRSCAN` and `TMENDRSCAN`. To simulate the “scanning” nature of the interface, we return all prepared XIDs for `TMSTARTRSCAN`, and no new XIDs for calls with `TMNOFLAGS`, or `TMENDRSCAN` when not in combination with `TMSTARTRSCAN`. This change was made for API compliance, as well as integration with IBM WebSphere's transaction manager. (Bug#20242)

- Fixed `MysqlValidConnectionChecker` for JBoss doesn't work with `MySQLXADataSources`. (Bug#20242)
- Added connection/datasource property `pinGlobalTxToPhysicalConnection` (defaults to `false`). When set to `true`, when using `XAConnections`, the driver ensures 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, and is also a workaround for transaction managers that don't maintain thread affinity for a global transaction (most either always maintain thread affinity, or have it as a configuration option). (Bug#20242)
- Better caching of character set converters (per-connection) to remove a bottleneck for multibyte character sets. (Bug#20242)
- Fixed `ConnectionProperties` (and thus some subclasses) are not serializable, even though some J2EE containers expect them to be. (Bug#19169)
- Fixed driver fails on non-ASCII platforms. The driver was assuming that the platform character set would be a superset of MySQL's `latin1` when doing the handshake for authentication, and when reading error messages. We now use Cp1252 for all strings sent to the server during the handshake phase, and a hard-coded mapping of the `language` systtem variable to the character set that is used for error messages. (Bug#18086)
- Fixed can't use `XAConnection` for local transactions when no global transaction is in progress. (Bug#17401)

D.2.8. Changes in MySQL Connector/J 5.0.1 (Not Released)

Not released due to a packaging error

D.2.9. Changes in MySQL Connector/J 5.0.0 (22 December 2005)

Bugs fixed:

- Added support for Connector/MXJ integration via url subprotocol `jdbc:mysql:mxj://....` (Bug#14729)
- Idle timeouts cause `XAConnections` to whine about rolling themselves back. (Bug#14729)
- When fix for Bug#14562 was merged from 3.1.12, added functionality for `CallableStatement`'s parameter metadata to return correct information for `.getParameterClassName()`. (Bug#14729)
- Added service-provider entry to `META-INF/services/java.sql.Driver` for JDBC-4.0 support. (Bug#14729)
- Fuller synchronization of `Connection` to avoid deadlocks when using multithreaded frameworks that multithread a single connection (usually not recommended, but the JDBC spec allows it anyways), part of fix to Bug#14972). (Bug#14729)
- Moved all `SQLException` constructor usage to a factory in `SQLError` (ground-work for JDBC-4.0 `SQLState`-based exception classes). (Bug#14729)
- Removed Java5-specific calls to `BigDecimal` constructor (when result set value is `''`, `(int)0` was being used as an argument indirectly via method return value. This signature doesn't exist prior to Java5.) (Bug#14729)
- Implementation of `Statement.cancel()` and `Statement.setQueryTimeout()`. Both require MySQL-5.0.0 or newer server, require a separate connection to issue the `KILL QUERY` statement, and in the case of `setQueryTimeout()` creates an additional thread to handle the timeout functionality.

  Note: Failures to cancel the statement for `setQueryTimeout()` may manifest themselves as `RuntimeExceptions` 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. (Bug#14729)
- Return "[VAR]BINARY" for `RSMD.getColumnTypeName()` when that is actually the type, and it can be distinguished (MySQL-4.1 and newer). (Bug#14729)
- Attempt detection of the MySQL type `BINARY` (it is an alias, so this isn't always reliable), and use the `java.sql.Types.BINARY` type mapping for it.
- Added unit tests for `XADatasource`, as well as friendlier exceptions for XA failures compared to the "stock" `XAException` (which has no messages).
- If the connection `useTimezone` is set to `true`, then also respect time zone conversions in escape-processed string literals (for example, `"{ts ...}"` and `"{t ...}"`).

- Don't allow `.setAutoCommit(true)`, or `.commit()` or `.rollback()` on an XA-managed connection as per the JDBC specification.
- `XADataSource` implemented (ported from 3.2 branch which won't be released as a product). Use `com.mysql.jdbc.jdbc2.optional.MysqlXADataSource` as your datasource class name in your application server to utilize XA transactions in MySQL-5.0.10 and newer.
- Moved `-bin-g.jar` file into separate `debug` subdirectory to avoid confusion.
- Return original column name for `RSMD.getColumnName()` if the column was aliased, alias name for `.getColumnLabel()` (if aliased), and original table name for `.getTableName()`. Note this only works for MySQL-4.1 and newer, as older servers don't make this information available to clients.
- Setting `useJDBCCompliantTimezoneShift=true` (it is not the default) causes the driver to use GMT for *all* `TIMESTAMP`/`DATETIME` time zones, and the current VM time zone for any other type that refers to time zones. This feature can not be used when `useTimezone=true` to convert between server and client time zones.
- `PreparedStatement.setString()` didn't work correctly when `sql_mode` on server contained `NO_BACKSLASH_ESCAPES` and no characters that needed escaping were present in the string.
- Add one level of indirection of internal representation of `CallableStatement` parameter metadata to avoid class not found issues on JDK-1.3 for `ParameterMetadata` interface (which doesn't exist prior to JDBC-3.0).

D.3. Changes in MySQL Connector/J 3.1.x

D.3.1. Changes in MySQL Connector/J 3.1.15 (Not yet released)

Important change: Due to a number of issues with the use of server-side prepared statements, Connector/J 5.0.5 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string:

```
useServerPrepStmts=true
```

The default value of this property is `false` (that is, Connector/J does not use server-side prepared statements).

Bugs fixed:

- Specifying `US-ASCII` as the character set in a connection to a MySQL 4.1 or newer server does not map correctly. (Bug#24840)

D.3.2. Changes in MySQL Connector/J 3.1.14 (10-19-2006)

Bugs fixed:

- Check and store value for continueBatchOnError property in constructor of Statements, rather than when executing batches, so that Connections closed out from underneath statements don't cause NullPointerExceptions when it is required to check this property. (Bug#22290)
- Fixed Bug#18258 - DatabaseMetaData.getTables(), columns() with bad catalog parameter threw exception rather than return empty result set (as required by spec). (Bug#22290)
- Driver now sends numeric 1 or 0 for client-prepared statement setBoolean() calls instead of '1' or '0'. (Bug#22290)
- Fixed bug where driver would not advance to next host if roundRobinLoadBalance=true and the last host in the list is down. (Bug#22290)
- Driver issues truncation on write exception when it shouldn't (due to sending big decimal incorrectly to server with server-side prepared statement). (Bug#22290)
- Fixed bug when calling stored functions, where parameters weren't numbered correctly (first parameter is now the return value, subsequent parameters if specified start at index "2"). (Bug#22290)
- Removed logger autodetection altogether, must now specify logger explicitly if you want to use a logger other than one that logs to STDERR. (Bug#21207)

- DDriver throws NPE when tracing prepared statements that have been closed (in asSQL()). (Bug#21207)
- ResultSet.getSomeInteger() doesn't work for BIT(>1). (Bug#21062)
- Escape of quotes in client-side prepared statements parsing not respected. Patch covers more than bug report, including NO_BACKSLASH_ESCAPES being set, and stacked quote characters forms of escaping (that is, " or ""). (Bug#20888)
- Fixed can't pool server-side prepared statements, exception raised when re-using them. (Bug#20687)
- Fixed Updatable result set that contains a BIT column fails when server-side prepared statements are used. (Bug#20485)
- Fixed updatable result set throws ClassCastException when there is row data and moveToInsertRow() is called. (Bug#20479)
- Fixed ResultSet.getShort() for UNSIGNED TINYINT returns incorrect values when using server-side prepared statements. (Bug#20306)
- ReplicationDriver does not always round-robin load balance depending on URL used for slaves list. (Bug#19993)
- Fixed calling toString() on ResultSetMetaData for driver-generated (that is, from DatabaseMetaData method calls, or from getGeneratedKeys()) result sets would raise a NullPointerException. (Bug#19993)
- Connection fails to localhost when using timeout and IPv6 is configured. (Bug#19726)
- ResultSet.getFloatFromString() can't retrieve values near Float.MIN/MAX_VALUE. (Bug#18880)
- Fixed memory leak with profileSQL=true. (Bug#16987)
- Fixed NullPointerException in MysqlDataSourceFactory due to Reference containing RefAddrs with null content. (Bug#16791)

D.3.3. Changes in MySQL Connector/J 3.1.13 (26 May 2006)

Bugs fixed:

- Fixed `PreparedStatement.setObject(int, Object, int)` doesn't respect scale of BigDecimals. (Bug#19615)
- Fixed `ResultSet.wasNull()` returns incorrect value when extracting native string from server-side prepared statement generated result set. (Bug#19282)
- Fixed invalid classname returned for `ResultSetMetaData.getColumnClassName()` for `BIGINT type`. (Bug#19282)
- Fixed case where driver wasn't reading server status correctly when fetching server-side prepared statement rows, which in some cases could cause warning counts to be off, or multiple result sets to not be read off the wire. (Bug#19282)
- Fixed data truncation and `getWarnings()` only returns last warning in set. (Bug#18740)
- Fixed aliased column names where length of name > 251 are corrupted. (Bug#18554)
- Improved performance of retrieving `BigDecimal`, `Time`, `Timestamp` and `Date` values from server-side prepared statements by creating fewer short-lived instances of `Strings` when the native type is not an exact match for the requested type. (Bug#18496)
- Added performance feature, re-writing of batched executes for `Statement.executeBatch()` (for all DML statements) and `PreparedStatement.executeBatch()` (for INSERTs with VALUE clauses only). Enable by using "rewriteBatchedStatements=true" in your JDBC URL. (Bug#18041)
- Fixed issue where server-side prepared statements don't cause truncation exceptions to be thrown when truncation happens. (Bug#18041)
- Fixed `CallableStatement.registerOutParameter()` not working when some parameters pre-populated. Still waiting for feedback from JDBC experts group to determine what correct parameter count from `getMetaData()` should be, however. (Bug#17898)
- Fixed calling `clearParameters()` on a closed prepared statement causes NPE. (Bug#17587)
- Map "latin1" on MySQL server to CP1252 for MySQL > 4.1.0. (Bug#17587)
- Added additional accessor and mutator methods on ConnectionProperties so that DataSource users can use same naming as reg-

ular URL properties. (Bug#17587)

- Fixed `ResultSet.wasNull()` not always reset correctly for booleans when done via conversion for server-side prepared statements. (Bug#17450)
- Fixed `Statement.getGeneratedKeys()` throws `NullPointerException` when no query has been processed. (Bug#17099)
- Fixed updatable result set doesn't return `AUTO_INCREMENT` values for `insertRow()` when multiple column primary keys are used. (the driver was checking for the existence of single-column primary keys and an autoincrement value > 0 instead of a straightforward `isAutoIncrement()` check). (Bug#16841)
- `DBMD.getColumns()` returns wrong type for `BIT`. (Bug#15854)
- `lib-nodist` directory missing from package breaks out-of-box build. (Bug#15676)
- Fixed issue with `ReplicationConnection` incorrectly copying state, doesn't transfer connection context correctly when transitioning between the same read-only states. (Bug#15570)
- No "dos" character set in MySQL > 4.1.0. (Bug#15544)
- `INOUT` parameter does not store `IN` value. (Bug#15464)
- `PreparedStatement.setObject()` serializes `BigInteger` as object, rather than sending as numeric value (and is thus not complementary to `.getObject()` on an `UNSIGNED LONG` type). (Bug#15383)
- Fixed issue where driver was unable to initialize character set mapping tables. Removed reliance on `.properties` files to hold this information, as it turns out to be too problematic to code around class loader hierarchies that change depending on how an application is deployed. Moved information back into the `CharsetMapping` class. (Bug#14938)
- Exception thrown for new decimal type when using updatable result sets. (Bug#14609)
- Driver now aware of fix for `BIT` type metadata that went into MySQL-5.0.21 for server not reporting length consistently . (Bug#13601)
- Added support for Apache Commons logging, use "com.mysql.jdbc.log.CommonsLogger" as the value for the "logger" configuration property. (Bug#13469)
- Fixed driver trying to call methods that don't exist on older and newer versions of Log4j. The fix is not trying to auto-detect presence of log4j, too many different incompatible versions out there in the wild to do this reliably.

  If you relied on autodetection before, you will need to add "logger=com.mysql.jdbc.log.Log4JLogger" to your JDBC URL to enable Log4J usage, or alternatively use the new "CommonsLogger" class to take care of this. (Bug#13469)
- LogFactory now prepends "com.mysql.jdbc.log" to log class name if it can't be found as-specified. This allows you to use "short names" for the built-in log factories, for example "logger=CommonsLogger" instead of "logger=com.mysql.jdbc.log.CommonsLogger". (Bug#13469)
- `ResultSet.getShort()` for `UNSIGNED TINYINT` returned wrong values. (Bug#11874)

D.3.4. Changes in MySQL Connector/J 3.1.12 (30 November 2005)

Bugs fixed:

- Process escape tokens in `Connection.prepareStatement(...)`. You can disable this behavior by setting the JDBC URL configuration property `processEscapeCodesForPrepStmts` to `false`. (Bug#15141)
- Usage advisor complains about unreferenced columns, even though they've been referenced. (Bug#15065)
- Driver incorrectly closes streams passed as arguments to `PreparedStatements`. Reverts to legacy behavior by setting the JDBC configuration property `autoClosePStmtStreams` to `true` (also included in the 3-0-Compat configuration “bundle”). (Bug#15024)
- Deadlock while closing server-side prepared statements from multiple threads sharing one connection. (Bug#14972)
- Unable to initialize character set mapping tables (due to J2EE classloader differences). (Bug#14938)
- Escape processor replaces quote character in quoted string with string delimiter. (Bug#14909)

- `DatabaseMetaData.getColumns()` doesn't return `TABLE_NAME` correctly. (Bug#14815)
- `storesMixedCaseIdentifiers()` returns `false` (Bug#14562)
- `storesLowerCaseIdentifiers()` returns `true` (Bug#14562)
- `storesMixedCaseQuotedIdentifiers()` returns `false` (Bug#14562)
- `storesMixedCaseQuotedIdentifiers()` returns `true` (Bug#14562)
- If `lower_case_table_names=0` (on server):
  - `storesLowerCaseIdentifiers()` returns `false`
  - `storesLowerCaseQuotedIdentifiers()` returns `false`
  - `storesMixedCaseIdentifiers()` returns `true`
  - `storesMixedCaseQuotedIdentifiers()` returns `true`
  - `storesUpperCaseIdentifiers()` returns `false`
  - `storesUpperCaseQuotedIdentifiers()` returns `true`

  (Bug#14562)
- `storesUpperCaseIdentifiers()` returns `false` (Bug#14562)
- `storesUpperCaseQuotedIdentifiers()` returns `true` (Bug#14562)
- If `lower_case_table_names=1` (on server):
  - `storesLowerCaseIdentifiers()` returns `true`
  - `storesLowerCaseQuotedIdentifiers()` returns `true`
  - `storesMixedCaseIdentifiers()` returns `false`
  - `storesMixedCaseQuotedIdentifiers()` returns `false`
  - `storesUpperCaseIdentifiers()` returns `false`
  - `storesUpperCaseQuotedIdentifiers()` returns `true`

  (Bug#14562)
- `storesLowerCaseQuotedIdentifiers()` returns `true` (Bug#14562)
- Fixed `DatabaseMetaData.stores*Identifiers()`:
  - If `lower_case_table_names=0` (on server):
    - `storesLowerCaseIdentifiers()` returns `false`
    - `storesLowerCaseQuotedIdentifiers()` returns `false`
    - `storesMixedCaseIdentifiers()` returns `true`
    - `storesMixedCaseQuotedIdentifiers()` returns `true`
    - `storesUpperCaseIdentifiers()` returns `false`
    - `storesUpperCaseQuotedIdentifiers()` returns `true`
  - If `lower_case_table_names=1` (on server):
    - `storesLowerCaseIdentifiers()` returns `true`
    - `storesLowerCaseQuotedIdentifiers()` returns `true`
    - `storesMixedCaseIdentifiers()` returns `false`

  - `storesMixedCaseQuotedIdentifiers()` returns `false`
  - `storesUpperCaseIdentifiers()` returns `false`
  - `storesUpperCaseQuotedIdentifiers()` returns `true`

  (Bug#14562)

- `storesMixedCaseIdentifiers()` returns `true` (Bug#14562)
- `storesLowerCaseQuotedIdentifiers()` returns `false` (Bug#14562)
- Java type conversion may be incorrect for `MEDIUMINT`. (Bug#14562)
- `storesLowerCaseIdentifiers()` returns `false` (Bug#14562)
- Added configuration property `useGmtMillisForDatetimes` which when set to `true` causes `ResultSet.getDate()`, `.getTimestamp()` to return correct millis-since GMT when `.getTime()` is called on the return value (currently default is `false` for legacy behavior). (Bug#14562)
- Extraneous sleep on `autoReconnect`. (Bug#13775)
- Reconnect during middle of `executeBatch()` should not occur if `autoReconnect` is enabled. (Bug#13255)
- `maxQuerySizeToLog` is not respected. Added logging of bound values for `execute()` phase of server-side prepared statements when `profileSQL=true` as well. (Bug#13048)
- OpenOffice expects `DBMD.supportsIntegrityEnhancementFacility()` to return `true` if foreign keys are supported by the datasource, even though this method also covers support for check constraints, which MySQL *doesn't* have. Setting the configuration property `overrideSupportsIntegrityEnhancementFacility` to `true` causes the driver to return `true` for this method. (Bug#12975)
- Added `com.mysql.jdbc.testsuite.url.default` system property to set default JDBC url for testsuite (to speed up bug resolution when I'm working in Eclipse). (Bug#12975)
- `logSlowQueries` should give better info. (Bug#12230)
- Don't increase timeout for failover/reconnect. (Bug#6577)
- Fixed client-side prepared statement bug with embedded `?` characters inside quoted identifiers (it was recognized as a placeholder, when it was not).
- Don't allow `executeBatch()` for `CallableStatements` with registered `OUT`/`INOUT` parameters (JDBC compliance).
- Fall back to platform-encoding for `URLDecoder.decode()` when parsing driver URL properties if the platform doesn't have a two-argument version of this method.

D.3.5. Changes in MySQL Connector/J 3.1.11 (07 October 2005)

Bugs fixed:

- The configuration property `sessionVariables` now allows you to specify variables that start with the “`@`” sign. (Bug#13453)
- URL configuration parameters don't allow “`&`” or “`=`” in their values. The JDBC driver now parses configuration parameters as if they are encoded using the application/x-www-form-urlencoded format as specified by `java.net.URLDecoder` (http://java.sun.com/j2se/1.5.0/docs/api/java/net/URLDecoder.html).

  If the “`%`” character is present in a configuration property, it must now be represented as `%25`, which is the encoded form of “`%`” when using application/x-www-form-urlencoded encoding. (Bug#13453)
- Workaround for Bug#13374: `ResultSet.getStatement()` on closed result set returns `NULL` (as per JDBC 4.0 spec, but not backward-compatible). Set the connection property `retainStatementAfterResultSetClose` to `true` to be able to retrieve a `ResultSet`'s statement after the `ResultSet` has been closed via `.getStatement()` (the default is `false`, to be JDBC-compliant and to reduce the chance that code using JDBC leaks `Statement` instances). (Bug#13277)
- `ResultSetMetaData` from `Statement.getGeneratedKeys()` caused a `NullPointerException` to be thrown

whenever a method that required a connection reference was called. (Bug#13277)

- Backport of `VAR[BINARY|CHAR] [BINARY]` types detection from 5.0 branch. (Bug#13277)
- Fixed `NullPointerException` when converting `catalog` parameter in many `DatabaseMetaDataMethods` to `byte[]`s (for the result set) when the parameter is `null`. (`null` isn't technically allowed by the JDBC specification, but we've historically allowed it). (Bug#13277)
- Backport of `Field` class, `ResultSetMetaData.getColumnClassName()`, and `ResultSet.getObject(int)` changes from 5.0 branch to fix behavior surrounding `VARCHAR BINARY/VARBINARY` and related types. (Bug#13277)
- Read response in `MysqlIO.sendFileToServer()`, even if the local file can't be opened, otherwise next query issued will fail, because it is reading the response to the empty `LOAD DATA INFILE` packet sent to the server. (Bug#13277)
- When `gatherPerfMetrics` is enabled for servers older than 4.1.0, a `NullPointerException` is thrown from the constructor of `ResultSet` if the query doesn't use any tables. (Bug#13043)
- `java.sql.Types.OTHER` returned for `BINARY` and `VARBINARY` columns when using `DatabaseMetaData.getColumns()`. (Bug#12970)
- `ServerPreparedStatement.getBinding()` now checks if the statement is closed before attempting to reference the list of parameter bindings, to avoid throwing a `NullPointerException`. (Bug#12970)
- Tokenizer for `=` in URL properties was causing `sessionVariables=....` to be parameterized incorrectly. (Bug#12753)
- `cp1251` incorrectly mapped to `win1251` for servers newer than 4.0.x. (Bug#12752)
- `getExportedKeys()` (Bug#12541)
- Specifying a catalog works as stated in the API docs. (Bug#12541)
- Specifying `NULL` means that catalog will not be used to filter the results (thus all databases will be searched), unless you've set `nullCatalogMeansCurrent=true` in your JDBC URL properties. (Bug#12541)
- `getIndexInfo()` (Bug#12541)
- `getProcedures()` (and thus indirectly `getProcedureColumns()`) (Bug#12541)
- `getImportedKeys()` (Bug#12541)
- Specifying `""` means “current” catalog, even though this isn't quite JDBC spec compliant, it is there for legacy users. (Bug#12541)
- `getCrossReference()` (Bug#12541)
- Added `Connection.isMasterConnection()` for clients to be able to determine if a multi-host master/slave connection is connected to the first host in the list. (Bug#12541)
- `getColumns()` (Bug#12541)
- Handling of catalog argument in `DatabaseMetaData.getIndexInfo()`, which also means changes to the following methods in `DatabaseMetaData`:
  - `getBestRowIdentifier()`
  - `getColumns()`
  - `getCrossReference()`
  - `getExportedKeys()`
  - `getImportedKeys()`
  - `getIndexInfo()`
  - `getPrimaryKeys()`
  - `getProcedures()` (and thus indirectly `getProcedureColumns()`)
  - `getTables()`

  The `catalog` argument in all of these methods now behaves in the following way:

  - Specifying NULL means that catalog will not be used to filter the results (thus all databases will be searched), unless you've set nullCatalogMeansCurrent=true in your JDBC URL properties.
  - Specifying "" means "current" catalog, even though this isn't quite JDBC spec compliant, it is there for legacy users.
  - Specifying a catalog works as stated in the API docs.
  - Made Connection.clientPrepare() available from "wrapped" connections in the jdbc2.optional package (connections built by ConnectionPoolDataSource instances).

  (Bug#12541)

- getBestRowIdentifier() (Bug#12541)
- Made Connection.clientPrepare() available from "wrapped" connections in the jdbc2.optional package (connections built by ConnectionPoolDataSource instances). (Bug#12541)
- getTables() (Bug#12541)
- getPrimaryKeys() (Bug#12541)
- Connection.prepareCall() is database name case-sensitive (on Windows systems). (Bug#12417)
- explainSlowQueries hangs with server-side prepared statements. (Bug#12229)
- Properties shared between master and slave with replication connection. (Bug#12218)
- Geometry types not handled with server-side prepared statements. (Bug#12104)
- maxPerformance.properties mis-spells "elideSetAutoCommits". (Bug#11976)
- ReplicationConnection won't switch to slave, throws "Catalog can't be null" exception. (Bug#11879)
- Pstmt.setObject(...., Types.BOOLEAN) throws exception. (Bug#11798)
- Escape tokenizer doesn't respect stacked single quotes for escapes. (Bug#11797)
- GEOMETRY type not recognized when using server-side prepared statements. (Bug#11797)
- Foreign key information that is quoted is parsed incorrectly when DatabaseMetaData methods use that information. (Bug#11781)
- The sendBlobChunkSize property is now clamped to max_allowed_packet with consideration of stream buffer size and packet headers to avoid PacketTooBigExceptions when max_allowed_packet is similar in size to the default sendBlobChunkSize which is 1M. (Bug#11781)
- CallableStatement.clearParameters() now clears resources associated with INOUT/OUTPUT parameters as well as INPUT parameters. (Bug#11781)
- Fixed regression caused by fix for Bug#11552 that caused driver to return incorrect values for unsigned integers when those integers where within the range of the positive signed type. (Bug#11663)
- Moved source code to Subversion repository. (Bug#11663)
- Incorrect generation of testcase scripts for server-side prepared statements. (Bug#11663)
- Fixed statements generated for testcases missing ; for "plain" statements. (Bug#11629)
- Spurious ! on console when character encoding is utf8. (Bug#11629)
- StringUtils.getBytes() doesn't work when using multi-byte character encodings and a length in *characters* is specified. (Bug#11614)
- DBMD.storesLower/Mixed/UpperIdentifiers() reports incorrect values for servers deployed on Windows. (Bug#11575)
- Reworked Field class, *Buffer, and MysqlIO to be aware of field lengths > Integer.MAX_VALUE. (Bug#11498)
- Escape processor didn't honor strings demarcated with double quotes. (Bug#11498)
- Updated DBMD.supportsCorrelatedQueries() to return true for versions > 4.1, supportsGroupByUnre-

`lated()` to return `true` and `getResultSetHoldability()` to return `HOLD_CURSORS_OVER_COMMIT`. (Bug#11498)

- Lifted restriction of changing streaming parameters with server-side prepared statements. As long as `all` streaming parameters were set before execution, `.clearParameters()` does not have to be called. (due to limitation of client/server protocol, prepared statements can not reset *individual* stream data on the server side). (Bug#11498)
- `ResultSet.moveToCurrentRow()` fails to work when preceded by a call to `ResultSet.moveToInsertRow()`. (Bug#11190)
- `VARBINARY` data corrupted when using server-side prepared statements and `.setBytes()`. (Bug#11115)
- `Statement.getWarnings()` fails with NPE if statement has been closed. (Bug#10630)
- Only get `char[]` from SQL in `PreparedStatement.ParseInfo()` when needed. (Bug#10630)

D.3.6. Changes in MySQL Connector/J 3.1.10 (23 June 2005)

Bugs fixed:

- Initial implemention of `ParameterMetadata` for `PreparedStatement.getParameterMetadata()`. Only works fully for `CallableStatements`, as current server-side prepared statements return every parameter as a `VARCHAR` type.
- Fixed connecting without a database specified raised an exception in `MysqlIO.changeDatabaseTo()`.

D.3.7. Changes in MySQL Connector/J 3.1.9 (22 June 2005)

Bugs fixed:

- Production package doesn't include JBoss integration classes. (Bug#11411)
- Removed nonsensical “costly type conversion” warnings when using usage advisor. (Bug#11411)
- Fixed `PreparedStatement.setClob()` not accepting `null` as a parameter. (Bug#11360)
- Connector/J dumping query into `SQLException` twice. (Bug#11360)
- `autoReconnect` ping causes exception on connection startup. (Bug#11259)
- `Connection.setCatalog()` is now aware of the `useLocalSessionState` configuration property, which when set to `true` will prevent the driver from sending `USE ...` to the server if the requested catalog is the same as the current catalog. (Bug#11115)
- `3-0-Compat` — Compatibility with Connector/J 3.0.x functionality (Bug#11115)
- `maxPerformance` — maximum performance without being reckless (Bug#11115)
- `solarisMaxPerformance` — maximum performance for Solaris, avoids syscalls where it can (Bug#11115)
- Added `maintainTimeStats` configuration property (defaults to `true`), which tells the driver whether or not to keep track of the last query time and the last successful packet sent to the server's time. If set to `false`, removes two syscalls per query. (Bug#11115)
- `VARBINARY` data corrupted when using server-side prepared statements and `ResultSet.getBytes()`. (Bug#11115)
- Added the following configuration bundles, use one or many via the `useConfigs` configuration property:
  - `maxPerformance` — maximum performance without being reckless
  - `solarisMaxPerformance` — maximum performance for Solaris, avoids syscalls where it can
  - `3-0-Compat` — Compatibility with Connector/J 3.0.x functionality

  (Bug#11115)
- Try to handle `OutOfMemoryErrors` more gracefully. Although not much can be done, they will in most cases close the connection they happened on so that further operations don't run into a connection in some unknown state. When an OOM has

happened, any further operations on the connection will fail with a “Connection closed” exception that will also list the OOM exception as the reason for the implicit connection close event. (Bug#10850)

- Setting cachePrepStmts=true now causes the Connection to also cache the check the driver performs to determine if a prepared statement can be server-side or not, as well as caches server-side prepared statements for the lifetime of a connection. As before, the prepStmtCacheSize parameter controls the size of these caches. (Bug#10850)
- Don't send COM_RESET_STMT for each execution of a server-side prepared statement if it isn't required. (Bug#10850)
- 0-length streams not sent to server when using server-side prepared statements. (Bug#10850)
- Driver detects if you're running MySQL-5.0.7 or later, and does not scan for LIMIT ?[,?] in statements being prepared, as the server supports those types of queries now. (Bug#10850)
- Reorganized directory layout. Sources now are in src folder. Don't pollute parent directory when building, now output goes to ./build, distribution goes to ./dist. (Bug#10496)
- Added support/bug hunting feature that generates .sql test scripts to STDERR when autoGenerateTestcaseScript is set to true. (Bug#10496)
- SQLException is thrown when using property characterSetResults with cp932 or eucjpms. (Bug#10496)
- The datatype returned for TINYINT(1) columns when tinyInt1isBit=true (the default) can be switched between Types.BOOLEAN and Types.BIT using the new configuration property transformedBitIsBoolean, which defaults to false. If set to false (the default), DatabaseMetaData.getColumns() and ResultSetMetaData.getColumnType() will return Types.BOOLEAN for TINYINT(1) columns. If true, Types.BOOLEAN will be returned instead. Regardless of this configuration property, if tinyInt1isBit is enabled, columns with the type TINYINT(1) will be returned as java.lang.Boolean instances from ResultSet.getObject(...), and ResultSetMetaData.getColumnClassName() will return java.lang.Boolean. (Bug#10485)
- SQLException thrown when retrieving YEAR(2) with ResultSet.getString(). The driver will now always treat YEAR types as java.sql.Dates and return the correct values for getString(). Alternatively, the yearIsDateType connection property can be set to false and the values will be treated as SHORTs. (Bug#10485)
- Driver doesn't support {?=CALL(...)} for calling stored functions. This involved adding support for function retrieval to DatabaseMetaData.getProcedures() and getProcedureColumns() as well. (Bug#10310)
- Unsigned SMALLINT treated as signed for ResultSet.getInt(), fixed all cases for UNSIGNED integer values and server-side prepared statements, as well as ResultSet.getObject() for UNSIGNED TINYINT. (Bug#10156)
- Made ServerPreparedStatement.asSql() work correctly so auto-explain functionality would work with server-side prepared statements. (Bug#10155)
- Double quotes not recognized when parsing client-side prepared statements. (Bug#10155)
- Made JDBC2-compliant wrappers public in order to allow access to vendor extensions. (Bug#10155)
- DatabaseMetaData.supportsMultipleOpenResults() now returns true. The driver has supported this for some time, DBMD just missed that fact. (Bug#10155)
- Cleaned up logging of profiler events, moved code to dump a profiler event as a string to com.mysql.jdbc.log.LogUtils so that third parties can use it. (Bug#10155)
- Made enableStreamingResults() visible on com.mysql.jdbc.jdbc2.optional.StatementWrapper. (Bug#10155)
- Actually write manifest file to correct place so it ends up in the binary jar file. (Bug#10144)
- Added createDatabaseIfNotExist property (default is false), which will cause the driver to ask the server to create the database specified in the URL if it doesn't exist. You must have the appropriate privileges for database creation for this to work. (Bug#10144)
- Memory leak in ServerPreparedStatement if serverPrepare() fails. (Bug#10144)
- com.mysql.jdbc.PreparedStatement.ParseInfo does unnecessary call to toCharArray(). (Bug#9064)
- Driver now correctly uses CP932 if available on the server for Windows-31J, CP932 and MS932 java encoding names, otherwise it resorts to SJIS, which is only a close approximation. Currently only MySQL-5.0.3 and newer (and MySQL-4.1.12 or .13, depending on when the character set gets backported) can reliably support any variant of CP932.
- Overhaul of character set configuration, everything now lives in a properties file.

D.3.8. Changes in MySQL Connector/J 3.1.8 (14 April 2005)

Bugs fixed:

- Should accept `null` for catalog (meaning use current) in DBMD methods, even though it is not JDBC-compliant for legacy's sake. Disable by setting connection property `nullCatalogMeansCurrent` to `false` (which will be the default value in C/J 3.2.x). (Bug#9917)
- Fixed driver not returning `true` for `-1` when `ResultSet.getBoolean()` was called on result sets returned from server-side prepared statements. (Bug#9778)
- Added a `Manifest.MF` file with implementation information to the `.jar` file. (Bug#9778)
- More tests in `Field.isOpaqueBinary()` to distinguish opaque binary (that is, fields with type `CHAR(n)` and `CHARACTER SET BINARY`) from output of various scalar and aggregate functions that return strings. (Bug#9778)
- `DBMD.getTables()` shouldn't return tables if views are asked for, even if the database version doesn't support views. (Bug#9778)
- Should accept `null` for name patterns in DBMD (meaning “`%`”), even though it isn't JDBC compliant, for legacy's sake. Disable by setting connection property `nullNamePatternMatchesAll` to `false` (which will be the default value in C/J 3.2.x). (Bug#9769)
- The performance metrics feature now gathers information about number of tables referenced in a SELECT. (Bug#9704)
- The logging system is now automatically configured. If the value has been set by the user, via the URL property `logger` or the system property `com.mysql.jdbc.logger`, then use that, otherwise, autodetect it using the following steps:

  1. Log4j, if it is available,
  2. Then JDK1.4 logging,
  3. Then fallback to our `STDERR` logging.

  (Bug#9704)
- `Statement.getMoreResults()` could throw NPE when existing result set was `.close()`d. (Bug#9704)
- Stored procedures with `DECIMAL` parameters with storage specifications that contained “`,`” in them would fail. (Bug#9682)
- `PreparedStatement.setObject(int, Object, int type, int scale)` now uses scale value for `BigDecimal` instances. (Bug#9682)
- Added support for the c3p0 connection pool's (http://c3p0.sf.net/) validation/connection checker interface which uses the lightweight `COM_PING` call to the server if available. To use it, configure your c3p0 connection pool's `connectionTesterClassName` property to use `com.mysql.jdbc.integration.c3p0.MysqlConnectionTester`. (Bug#9320)
- `PreparedStatement.getMetaData()` inserts blank row in database under certain conditions when not using server-side prepared statements. (Bug#9320)
- Better detection of `LIMIT` inside/outside of quoted strings so that the driver can more correctly determine whether a prepared statement can be prepared on the server or not. (Bug#9320)
- `Connection.canHandleAsPreparedStatement()` now makes “best effort” to distinguish `LIMIT` clauses with placeholders in them from ones without in order to have fewer false positives when generating work-arounds for statements the server cannot currently handle as server-side prepared statements. (Bug#9320)
- Fixed `build.xml` to not compile `log4j` logging if `log4j` not available. (Bug#9320)
- Added finalizers to `ResultSet` and `Statement` implementations to be JDBC spec-compliant, which requires that if not explicitly closed, these resources should be closed upon garbage collection. (Bug#9319)
- Stored procedures with same name in different databases confuse the driver when it tries to determine parameter counts/types. (Bug#9319)
- A continuation of Bug#8868, where functions used in queries that should return non-string types when resolved by temporary tables suddenly become opaque binary strings (work-around for server limitation). Also fixed fields with type of `CHAR(n) CHARACTER SET BINARY` to return correct/matching classes for `RSMD.getColumnClassName()` and `ResultSet.getObject()`. (Bug#9236)
- Cannot use `UTF-8` for characterSetResults configuration property. (Bug#9206)

- `PreparedStatement.addBatch()` doesn't work with server-side prepared statements and streaming `BINARY` data. (Bug#9040)
- `ServerPreparedStatements` now correctly “stream” `BLOB`/`CLOB` data to the server. You can configure the threshold chunk size using the JDBC URL property `blobSendChunkSize` (the default is 1MB). (Bug#8868)
- `DATE_FORMAT()` queries returned as `BLOB`s from `getObject()`. (Bug#8868)
- Server-side session variables can be preset at connection time by passing them as a comma-delimited list for the connection property `sessionVariables`. (Bug#8868)
- `BlobFromLocator` now uses correct identifier quoting when generating prepared statements. (Bug#8868)
- Fixed regression in `ping()` for users using `autoReconnect=true`. (Bug#8868)
- Check for empty strings (`''`) when converting `CHAR`/`VARCHAR` column data to numbers, throw exception if `emptyStringsConvertToZero` configuration property is set to `false` (for backward-compatibility with 3.0, it is now set to `true` by default, but will most likely default to `false` in 3.2). (Bug#8803)
- `DATA_TYPE` column from `DBMD.getBestRowIdentifier()` causes `ArrayIndexOutOfBoundsException` when accessed (and in fact, didn't return any value). (Bug#8803)
- `DBMD.supportsMixedCase*Identifiers()` returns wrong value on servers running on case-sensitive file systems. (Bug#8800)
- `DBMD.supportsResultSetConcurrency()` not returning `true` for forward-only/read-only result sets (we obviously support this). (Bug#8792)
- Fixed `ResultSet.getTime()` on a `NULL` value for server-side prepared statements throws NPE.
- Made `Connection.ping()` a public method.
- Added support for new precision-math `DECIMAL` type in MySQL 5.0.3 and up.
- Fixed `DatabaseMetaData.getTables()` returning views when they were not asked for as one of the requested table types.

D.3.9. Changes in MySQL Connector/J 3.1.7 (18 February 2005)

Bugs fixed:

- `PreparedStatements` not creating streaming result sets. (Bug#8487)
- Don't pass `NULL` to `String.valueOf()` in `ResultSet.getNativeConvertToString()`, as it stringifies it (that is, returns `null`), which is not correct for the method in question. (Bug#8487)
- Fixed NPE in `ResultSet.realClose()` when using usage advisor and result set was already closed. (Bug#8428)
- `ResultSet.getString()` doesn't maintain format stored on server, bug fix only enabled when `noDatetimeStringSync` property is set to `true` (the default is `false`). (Bug#8428)
- Added support for `BIT` type in MySQL-5.0.3. The driver will treat `BIT(1-8)` as the JDBC standard `BIT` type (which maps to `java.lang.Boolean`), as the server does not currently send enough information to determine the size of a bitfield when < 9 bits are declared. `BIT(>9)` will be treated as `VARBINARY`, and will return `byte[]` when `getObject()` is called. (Bug#8424)
- Added `useLocalSessionState` configuration property, when set to `true` the JDBC driver trusts that the application is well-behaved and only sets autocommit and transaction isolation levels using the methods provided on `java.sql.Connection`, and therefore can manipulate these values in many cases without incurring round-trips to the database server. (Bug#8424)
- Added `enableStreamingResults()` to `Statement` for connection pool implementations that check `Statement.setFetchSize()` for specification-compliant values. Call `Statement.setFetchSize(>=0)` to disable the streaming results for that statement. (Bug#8424)
- `ResultSet.getBigDecimal()` throws exception when rounding would need to occur to set scale. The driver now chooses a rounding mode of “half up” if non-rounding `BigDecimal.setScale()` fails. (Bug#8424)
- Fixed synchronization issue with `ServerPreparedStatement.serverPrepare()` that could cause deadlocks/crashes

if connection was shared between threads. (Bug#8096)

- Emulated locators corrupt binary data when using server-side prepared statements. (Bug#8096)
- Infinite recursion when “falling back” to master in failover configuration. (Bug#7952)
- Disable multi-statements (if enabled) for MySQL-4.1 versions prior to version 4.1.10 if the query cache is enabled, as the server returns wrong results in this configuration. (Bug#7952)
- Removed `dontUnpackBinaryResults` functionality, the driver now always stores results from server-side prepared statements as is from the server and unpacks them on demand. (Bug#7952)
- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly. (Bug#7952)
- Added `holdResultsOpenOverStatementClose` property (default is `false`), that keeps result sets open over statement.close() or new execution on same statement (suggested by Kevin Burton). (Bug#7715)
- Detect new `sql_mode` variable in string form (it used to be integer) and adjust quoting method for strings appropriately. (Bug#7715)
- Timestamps converted incorrectly to strings with server-side prepared statements and updatable result sets. (Bug#7715)
- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug#7686)
- Choose correct “direction” to apply time adjustments when both client and server are in GMT time zone when using `ResultSet.get(..., cal)` and `PreparedStatement.set(...., cal)`. (Bug#4718)
- Remove `_binary` introducer from parameters used as in/out parameters in `CallableStatement`. (Bug#4718)
- Always return `byte[]`s for output parameters registered as `*BINARY`. (Bug#4718)
- By default, the driver now scans SQL you are preparing via 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 by passing `emulateUnsupportedPstmts=false` in your JDBC URL. (Bug#4718)
- Added `dontTrackOpenResources` option (default is `false`, to be JDBC compliant), which helps with memory use for non-well-behaved apps (that is, applications that don't close `Statement` objects when they should). (Bug#4718)
- Send correct value for “boolean” `true` to server for `PreparedStatement.setObject(n, "true", Types.BIT)`. (Bug#4718)
- Fixed bug with Connection not caching statements from `prepareStatement()` when the statement wasn't a server-side prepared statement. (Bug#4718)

D.3.10. Changes in MySQL Connector/J 3.1.6 (23 December 2004)

Bugs fixed:

- `DBMD.getProcedures()` doesn't respect catalog parameter. (Bug#7026)
- Fixed hang on `SocketInputStream.read()` with `Statement.setMaxRows()` and multiple result sets when driver has to truncate result set directly, rather than tacking a `LIMIT n` on the end of it.

D.3.11. Changes in MySQL Connector/J 3.1.5 (02 December 2004)

Bugs fixed:

- Use 1MB packet for sending file for `LOAD DATA LOCAL INFILE` if that is < `max_allowed_packet` on server. (Bug#6537)
- `SUM()` on `DECIMAL` with server-side prepared statement ignores scale if zero-padding is needed (this ends up being due to conversion to `DOUBLE` by server, which when converted to a string to parse into `BigDecimal`, loses all “padding” zeros). (Bug#6537)

- Use `DatabaseMetaData.getIdentifierQuoteString()` when building DBMD queries. (Bug#6537)
- Use our own implementation of buffered input streams to get around blocking behavior of `java.io.BufferedInputStream`. Disable this with `useReadAheadInput=false`. (Bug#6399)
- Make auto-deserialization of `java.lang.Objects` stored in `BLOB` columns configurable via `autoDeserialize` property (defaults to `false`). (Bug#6399)
- `ResultSetMetaData.getColumnDisplaySize()` returns incorrect values for multi-byte charsets. (Bug#6399)
- Re-work `Field.isOpaqueBinary()` to detect `CHAR(n) CHARACTER SET BINARY` to support fixed-length binary fields for `ResultSet.getObject()`. (Bug#6399)
- Failing to connect to the server when one of the addresses for the given host name is IPV6 (which the server does not yet bind on). The driver now loops through *all* IP addresses for a given host, and stops on the first one that `accepts()` a `socket.connect()`. (Bug#6348)
- Removed unwanted new `Throwable()` in `ResultSet` constructor due to bad merge (caused a new object instance that was never used for every result set created). Found while profiling for Bug#6359. (Bug#6225)
- `ServerSidePreparedStatement` allocating short-lived objects unnecessarily. (Bug#6225)
- Use null-safe-equals for key comparisons in updatable result sets. (Bug#6225)
- Fixed too-early creation of `StringBuffer` in `EscapeProcessor.escapeSQL()`, also return `String` when escaping not needed (to avoid unnecessary object allocations). Found while profiling for Bug#6359. (Bug#6225)
- `UNSIGNED BIGINT` unpacked incorrectly from server-side prepared statement result sets. (Bug#5729)
- Added experimental configuration property `dontUnpackBinaryResults`, which delays unpacking binary result set values until they're asked for, and only creates object instances for non-numerical values (it is set to `false` by default). For some usecase/jvm combinations, this is friendlier on the garbage collector. (Bug#5706)
- Don't throw exceptions for `Connection.releaseSavepoint()`. (Bug#5706)
- Inefficient detection of pre-existing string instances in `ResultSet.getNativeString()`. (Bug#5706)
- Use a per-session `Calendar` instance by default when decoding dates from `ServerPreparedStatements` (set to old, less performant behavior by setting property `dynamicCalendars=true`). (Bug#5706)
- Fixed batched updates with server prepared statements weren't looking if the types had changed for a given batched set of parameters compared to the previous set, causing the server to return the error “Wrong arguments to mysql_stmt_execute()”. (Bug#5235)
- Handle case when string representation of timestamp contains trailing “`.`” with no numbers following it. (Bug#5235)
- Server-side prepared statements did not honor `zeroDateTimeBehavior` property, and would cause class-cast exceptions when using `ResultSet.getObject()`, as the all-zero string was always returned. (Bug#5235)
- Fix comparisons made between string constants and dynamic strings that are converted with either `toUpperCase()` or `toLowerCase()` to use `Locale.ENGLISH`, as some locales “override” case rules for English. Also use `StringUtils.indexOfIgnoreCase()` instead of `.toUpperCase().indexOf()`, avoids creating a very short-lived transient `String` instance.

D.3.12. Changes in MySQL Connector/J 3.1.4 (04 September 2004)

Bugs fixed:

- Fixed `ServerPreparedStatement` to read prepared statement metadata off the wire, even though it is currently a placeholder instead of using `MysqlIO.clearInputStream()` which didn't work at various times because data wasn't available to read from the server yet. This fixes sporadic errors users were having with `ServerPreparedStatements` throwing `ArrayIndexOutOfBoundExceptions`. (Bug#5032)
- Added three ways to deal with all-zero datetimes when reading them from a `ResultSet`: `exception` (the default), which throws an `SQLException` with an SQLState of `S1009`; `convertToNull`, which returns `NULL` instead of the date; and `round`, which rounds the date to the nearest closest value which is `'0001-01-01'`. (Bug#5032)
- The driver is more strict about truncation of numerics on `ResultSet.get*()`, and will throw an `SQLException` when truncation is detected. You can disable this by setting `jdbcCompliantTruncation` to `false` (it is enabled by default, as

this functionality is required for JDBC compliance). (Bug#5032)

- You can now use URLs in `LOAD DATA LOCAL INFILE` statements, and the driver will use Java's built-in handlers for retreiving the data and sending it to the server. This feature is not enabled by default, you must set the `allowUrlInLocalInfile` connection property to `true`. (Bug#5032)
- `ResultSet.getObject()` doesn't return type `Boolean` for pseudo-bit types from prepared statements on 4.1.x (shortcut for avoiding extra type conversion when using binary-encoded result sets obscured test in `getObject()` for "pseudo" bit type). (Bug#5032)
- Use `com.mysql.jdbc.Message`'s classloader when loading resource bundle, should fix sporadic issues when the caller's classloader can't locate the resource bundle. (Bug#5032)
- `ServerPreparedStatements` dealing with return of `DECIMAL` type don't work. (Bug#5012)
- Track packet sequence numbers if `enablePacketDebug=true`, and throw an exception if packets received out-of-order. (Bug#4689)
- `ResultSet.wasNull()` does not work for primatives if a previous `null` was returned. (Bug#4689)
- Optimized integer number parsing, enable "old" slower integer parsing using JDK classes via `useFastIntParsing=false` property. (Bug#4642)
- Added `useOnlyServerErrorMessages` property, which causes message text in exceptions generated by the server to only contain the text sent by the server (as opposed to the SQLState's "standard" description, followed by the server's error message). This property is set to `true` by default. (Bug#4642)
- `ServerPreparedStatement.execute*()` sometimes threw `ArrayIndexOutOfBoundsException` when unpacking field metadata. (Bug#4642)
- Connector/J 3.1.3 beta does not handle integers correctly (caused by changes to support unsigned reads in `Buffer.readInt()` -> `Buffer.readShort()`). (Bug#4510)
- Added support in `DatabaseMetaData.getTables()` and `getTableTypes()` for views, which are now available in MySQL server 5.0.x. (Bug#4510)
- `ResultSet.getObject()` returns wrong type for strings when using prepared statements. (Bug#4482)
- Calling `MysqlPooledConnection.close()` twice (even though an application error), caused NPE. Fixed. (Bug#4482)

D.3.13. Changes in MySQL Connector/J 3.1.3 (07 July 2004)

Bugs fixed:

- Support new time zone variables in MySQL-4.1.3 when `useTimezone=true`. (Bug#4311)
- Error in retrieval of `mediumint` column with prepared statements and binary protocol. (Bug#4311)
- Support for unsigned numerics as return types from prepared statements. This also causes a change in `ResultSet.getObject()` for the `bigint unsigned` type, which used to return `BigDecimal` instances, it now returns instances of `java.lang.BigInteger`. (Bug#4311)
- Externalized more messages (on-going effort). (Bug#4119)
- Null bitmask sent for server-side prepared statements was incorrect. (Bug#4119)
- Added constants for MySQL error numbers (publicly accessible, see `com.mysql.jdbc.MysqlErrorNumbers`), and the ability to generate the mappings of vendor error codes to SQLStates that the driver uses (for documentation purposes). (Bug#4119)
- Added packet debuging code (see the `enablePacketDebug` property documentation). (Bug#4119)
- Use SQL Standard SQL states by default, unless `useSqlStateCodes` property is set to `false`. (Bug#4119)
- Mangle output parameter names for `CallableStatements` so they will not clash with user variable names.
- Added support for `INOUT` parameters in `CallableStatements`.

D.3.14. Changes in MySQL Connector/J 3.1.2 (09 June 2004)

Bugs fixed:

- Don't enable server-side prepared statements for server version 5.0.0 or 5.0.1, as they aren't compatible with the '4.1.2+' style that the driver uses (the driver expects information to come back that isn't there, so it hangs). (Bug#3804)
- `getWarnings()` returns `SQLWarning` instead of `DataTruncation`. (Bug#3804)
- `getProcedureColumns()` doesn't work with wildcards for procedure name. (Bug#3540)
- `getProcedures()` does not return any procedures in result set. (Bug#3539)
- Fixed `DatabaseMetaData.getProcedures()` when run on MySQL-5.0.0 (output of `SHOW PROCEDURE STATUS` changed between 5.0.0 and 5.0.1. (Bug#3520)
- Added `connectionCollation` property to cause driver to issue `set collation_connection=...` query on connection init if default collation for given charset is not appropriate. (Bug#3520)
- `DBMD.getSQLStateType()` returns incorrect value. (Bug#3520)
- Correctly map output parameters to position given in `prepareCall()` versus. order implied during `registerOutParameter()`. (Bug#3146)
- Cleaned up detection of server properties. (Bug#3146)
- Correctly detect initial character set for servers >= 4.1.0. (Bug#3146)
- Support placeholder for parameter metadata for server >= 4.1.2. (Bug#3146)
- Added `gatherPerformanceMetrics` property, along with properties to control when/where this info gets logged (see docs for more info).
- Fixed case when no parameters could cause a `NullPointerException` in `CallableStatement.setOutputParameters()`.
- Enabled callable statement caching via `cacheCallableStmts` property.
- Fixed sending of split packets for large queries, enabled nio ability to send large packets as well.
- Added `.toString()` functionality to `ServerPreparedStatement`, which should help if you're trying to debug a query that is a prepared statement (it shows SQL as the server would process).
- Added `logSlowQueries` property, along with `slowQueriesThresholdMillis` property to control when a query should be considered “slow.”
- Removed wrapping of exceptions in `MysqlIO.changeUser()`.
- Fixed stored procedure parameter parsing info when size was specified for a parameter (for example, `char()`, `varchar()`).
- `ServerPreparedStatements` weren't actually de-allocating server-side resources when `.close()` was called.
- Fixed case when no output parameters specified for a stored procedure caused a bogus query to be issued to retrieve out parameters, leading to a syntax error from the server.

D.3.15. Changes in MySQL Connector/J 3.1.1 (14 February 2004 alpha)

Bugs fixed:

- Use DocBook version of docs for shipped versions of drivers. (Bug#2671)
- `NULL` fields were not being encoded correctly in all cases in server-side prepared statements. (Bug#2671)
- Fixed rare buffer underflow when writing numbers into buffers for sending prepared statement execution requests. (Bug#2671)
- Fixed `ConnectionProperties` that weren't properly exposed via accessors, cleaned up `ConnectionProperties` code. (Bug#2623)
- Class-cast exception when using scrolling result sets and server-side prepared statements. (Bug#2623)

- Merged unbuffered input code from 3.0. (Bug#2623)
- Enabled streaming of result sets from server-side prepared statements. (Bug#2606)
- Server-side prepared statements were not returning datatype `YEAR` correctly. (Bug#2606)
- Fixed charset conversion issue in `getTables()`. (Bug#2502)
- Implemented multiple result sets returned from a statement or stored procedure. (Bug#2502)
- Implemented `Connection.prepareCall()`, and `DatabaseMetaData`. `getProcedures()` and `getProcedure-Columns()`. (Bug#2359)
- Merged prepared statement caching, and `.getMetaData()` support from 3.0 branch. (Bug#2359)
- Fixed off-by-1900 error in some cases for years in `TimeUtil.fastDate`/`TimeCreate()` when unpacking results from server-side prepared statements. (Bug#2359)
- Reset `long binary` parameters in `ServerPreparedStatement` when `clearParameters()` is called, by sending `COM_RESET_STMT` to the server. (Bug#2359)
- `NULL` values for numeric types in binary encoded result sets causing `NullPointerExceptions`. (Bug#2359)
- Display where/why a connection was implicitly closed (to aid debugging). (Bug#1673)
- `DatabaseMetaData.getColumns()` is not returning correct column ordinal info for non-`'%'` column name patterns. (Bug#1673)
- Fixed `NullPointerException` in `ServerPreparedStatement.setTimestamp()`, as well as year and month descrepencies in `ServerPreparedStatement.setTimestamp()`, `setDate()`. (Bug#1673)
- Added ability to have multiple database/JVM targets for compliance and regression/unit tests in `build.xml`. (Bug#1673)
- Fixed sending of queries larger than 16M. (Bug#1673)
- Merged fix of datatype mapping from MySQL type `FLOAT` to `java.sql.Types.REAL` from 3.0 branch. (Bug#1673)
- Fixed NPE and year/month bad conversions when accessing some datetime functionality in `ServerPreparedStatements` and their resultant result sets. (Bug#1673)
- Added named and indexed input/output parameter support to `CallableStatement`. MySQL-5.0.x or newer. (Bug#1673)
- `CommunicationsException` implemented, that tries to determine why communications was lost with a server, and displays possible reasons when `.getMessage()` is called. (Bug#1673)
- Detect collation of column for `RSMD.isCaseSensitive()`. (Bug#1673)
- Optimized `Buffer.readLenByteArray()` to return shared empty byte array when length is 0.
- Fix support for table aliases when checking for all primary keys in `UpdatableResultSet`.
- Unpack “unknown” data types from server prepared statements as `Strings`.
- Implemented `Statement.getWarnings()` for MySQL-4.1 and newer (using `SHOW WARNINGS`).
- Ensure that warnings are cleared before executing queries on prepared statements, as-per JDBC spec (now that we support warnings).
- Correctly initialize datasource properties from JNDI Refs, including explicitly specified URLs.
- Implemented long data (Blobs, Clobs, InputStreams, Readers) for server prepared statements.
- Deal with 0-length tokens in `EscapeProcessor` (caused by callable statement escape syntax).
- `DatabaseMetaData` now reports `supportsStoredProcedures()` for MySQL versions >= 5.0.0
- Support for `mysql_change_user()`. See the `changeUser()` method in `com.mysql.jdbc.Connection`.
- Removed `useFastDates` connection property.
- Support for NIO. Use `useNIO=true` on platforms that support NIO.

- Check for closed connection on delete/update/insert row operations in `UpdatableResultSet`.
- Support for transaction savepoints (MySQL >= 4.0.14 or 4.1.1).
- Support “old” `profileSql` capitalization in `ConnectionProperties`. This property is deprecated, you should use `profileSQL` if possible.
- Fixed character encoding issues when converting bytes to ASCII when MySQL doesn't provide the character set, and the JVM is set to a multi-byte encoding (usually affecting retrieval of numeric values).
- Centralized setting of result set type and concurrency.
- Fixed bug with `UpdatableResultSets` not using client-side prepared statements.
- Default result set type changed to `TYPE_FORWARD_ONLY` (JDBC compliance).
- Fixed `IllegalAccessError` to `Calendar.getTimeInMillis()` in `DateTimeValue` (for JDK < 1.4).
- Allow contents of `PreparedStatement.setBlob()` to be retained between calls to `.execute*()`.
- Fixed stack overflow in `Connection.prepareCall()` (bad merge).
- Refactored how connection properties are set and exposed as `DriverPropertyInfo` as well as `Connection` and `DataSource` properties.
- Reduced number of methods called in average query to be more efficient.
- Prepared `Statements` will be re-prepared on auto-reconnect. Any errors encountered are postponed until first attempt to re-execute the re-prepared statement.

D.3.16. Changes in MySQL Connector/J 3.1.0 (18 February 2003 alpha)

Bugs fixed:

- Added `useServerPrepStmts` property (default `false`). The driver will use server-side prepared statements when the server version supports them (4.1 and newer) when this property is set to `true`. It is currently set to `false` by default until all bind/fetch functionality has been implemented. Currently only DML prepared statements are implemented for 4.1 server-side prepared statements.
- Added `requireSSL` property.
- Track open `Statements`, close all when `Connection.close()` is called (JDBC compliance).

D.4. Changes in MySQL Connector/J 3.0.x

D.4.1. Changes in MySQL Connector/J 3.0.17 (23 June 2005)

Bugs fixed:

- Workaround for server Bug#9098: Default values of `CURRENT_*` for `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` columns can't be distinguished from `string` values, so `UpdatableResultSet.moveToInsertRow()` generates bad SQL for inserting default values. (Bug#8812)
- `NON_UNIQUE` column from `DBMD.getIndexInfo()` returned inverted value. (Bug#8812)
- `EUCKR` charset is sent as `SET NAMES euc_kr` which MySQL-4.1 and newer doesn't understand. (Bug#8629)
- Added support for the `EUC_JP_Solaris` character encoding, which maps to a MySQL encoding of `eucjpms` (backported from 3.1 branch). This only works on servers that support `eucjpms`, namely 5.0.3 or later. (Bug#8629)
- Use hex escapes for `PreparedStatement.setBytes()` for double-byte charsets including “aliases” `Windows-31J`, `CP934`, `MS932`. (Bug#8629)
- `DatabaseMetaData.supportsSelectForUpdate()` returns correct value based on server version. (Bug#8629)
- Which requires hex escaping of binary data when using multi-byte charsets with prepared statements. (Bug#8064)

- Fixed duplicated code in `configureClientCharset()` that prevented `useOldUTF8Behavior=true` from working properly. (Bug#7952)
- Backported SQLState codes mapping from Connector/J 3.1, enable with `useSqlStateCodes=true` as a connection property, it defaults to `false` in this release, so that we don't break legacy applications (it defaults to `true` starting with Connector/J 3.1). (Bug#7686)
- Timestamp key column data needed `_binary` stripped for `UpdatableResultSet.refreshRow()`. (Bug#7686)
- `MS932`, `SHIFT_JIS`, and `Windows_31J` not recognized as aliases for `sjis`. (Bug#7607)
- Handle streaming result sets with more than 2 billion rows properly by fixing wraparound of row number counter. (Bug#7601)
- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug#7601)
- Escape sequence {fn convert(..., type)} now supports ODBC-style types that are prepended by `SQL_`. (Bug#7601)
- Statements created from a pooled connection were returning physical connection instead of logical connection when `getConnection()` was called. (Bug#7316)
- Support new protocol type `MYSQL_TYPE_VARCHAR`. (Bug#7081)
- Added `useOldUTF8Behavior`' configuration property, which causes JDBC driver to act like it did with MySQL-4.0.x and earlier when the character encoding is `utf-8` when connected to MySQL-4.1 or newer. (Bug#7081)
- `DatabaseMetaData.getIndexInfo()` ignored `unique` parameter. (Bug#7081)
- `PreparedStatement.fixDecimalExponent()` adding extra `+`, making number unparseable by MySQL server. (Bug#7061)
- `PreparedStatements` don't encode Big5 (and other multi-byte) character sets correctly in static SQL strings. (Bug#7033)
- Connections starting up failed-over (due to down master) never retry master. (Bug#6966)
- Adding `CP943` to aliases for `sjis`. (Bug#6549, Bug#7607)
- `Timestamp`/`Time` conversion goes in the wrong “direction” when `useTimeZone=true` and server time zone differs from client time zone. (Bug#5874)

D.4.2. Changes in MySQL Connector/J 3.0.16 (15 November 2004)

Bugs fixed:

- Made `TINYINT(1)` -> `BIT`/`Boolean` conversion configurable via `tinyInt1isBit` property (default `true` to be JDBC compliant out of the box). (Bug#5664)
- Off-by-one bug in `Buffer.readString(`*`string`*`)`. (Bug#5664)
- `ResultSet.updateByte()` when on insert row throws `ArrayOutOfBoundsException`. (Bug#5664)
- Fixed regression where `useUnbufferedInput` was defaulting to `false`. (Bug#5664)
- `ResultSet.getTimestamp()` on a column with `TIME` in it fails. (Bug#5664)
- Fixed `DatabaseMetaData.getTypes()` returning incorrect (this is, non-negative) scale for the `NUMERIC` type. (Bug#5664)
- Only set `character_set_results` during connection establishment if server version >= 4.1.1. (Bug#5664)
- Fixed `ResultSetMetaData.isReadOnly()` to detect non-writable columns when connected to MySQL-4.1 or newer, based on existence of “original” table and column names.
- Re-issue character set configuration commands when re-using pooled connections and/or `Connection.changeUser()` when connected to MySQL-4.1 or newer.

D.4.3. Changes in MySQL Connector/J 3.0.15 (04 September 2004)

Bugs fixed:

- `ResultSet.getMetaData()` should not return incorrectly initialized metadata if the result set has been closed, but should instead throw an `SQLException`. Also fixed for `getRow()` and `getWarnings()` and traversal methods by calling `checkClosed()` before operating on instance-level fields that are nullified during `.close()`. (Bug#5069)
- Use `_binary` introducer for `PreparedStatement.setBytes()` and `set*Stream()` when connected to MySQL-4.1.x or newer to avoid misinterpretation during character conversion. (Bug#5069)
- Parse new time zone variables from 4.1.x servers. (Bug#5069)
- `ResultSet` should release `Field[]` instance in `.close()`. (Bug#5022)
- `RSMD.getPrecision()` returning 0 for non-numeric types (should return max length in chars for nonbinary types, max length in bytes for binary types). This fix also fixes mapping of `RSMD.getColumnType()` and `RSMD.getColumnTypeName()` for the `BLOB` types based on the length sent from the server (the server doesn't distinguish between `TINYBLOB`, `BLOB`, `MEDIUMBLOB` or `LONGBLOB` at the network protocol level). (Bug#4880)
- "Production" is now "GA" (General Availability) in naming scheme of distributions. (Bug#4860, Bug#4138)
- `DBMD.getColumns()` returns incorrect JDBC type for unsigned columns. This affects type mappings for all numeric types in the `RSMD.getColumnType()` and `RSMD.getColumnTypeNames()` methods as well, to ensure that "like" types from `DBMD.getColumns()` match up with what `RSMD.getColumnType()` and `getColumnTypeNames()` return. (Bug#4860, Bug#4138)
- Calling `.close()` twice on a `PooledConnection` causes NPE. (Bug#4808)
- `DOUBLE` mapped twice in `DBMD.getTypeInfo()`. (Bug#4742)
- Added FLOSS license exemption. (Bug#4742)
- Removed redundant calls to `checkRowPos()` in `ResultSet`. (Bug#4334)
- Failover for `autoReconnect` not using port numbers for any hosts, and not retrying all hosts.

  Warning

  This required a change to the `SocketFactory` `connect()` method signature, which is now `public Socket connect(String host, int portNumber, Properties props)`; therefore, any third-party socket factories will have to be changed to support this signature.

  (Bug#4334)
- Logical connections created by `MysqlConnectionPoolDataSource` will now issue a `rollback()` when they are closed and sent back to the pool. If your application server/connection pool already does this for you, you can set the `rollbackOnPooledClose` property to `false` to avoid the overhead of an extra `rollback()`. (Bug#4334)
- `StringUtils.escapeEasternUnicodeByteStream` was still broken for GBK. (Bug#4010)

D.4.4. Changes in MySQL Connector/J 3.0.14 (28 May 2004)

Bugs fixed:

- Fixed URL parsing error.

D.4.5. Changes in MySQL Connector/J 3.0.13 (27 May 2004)

Bugs fixed:

- `No Database Selected` when using `MysqlConnectionPoolDataSource`. (Bug#3920)
- `PreparedStatement.getGeneratedKeys()` method returns only 1 result for batched insertions. (Bug#3873)
- Using a `MySQLDatasource` without server name fails. (Bug#3848)

D.4.6. Changes in MySQL Connector/J 3.0.12 (18 May 2004)

Bugs fixed:

- Inconsistent reporting of data type. The server still doesn't return all types for *BLOBs *TEXT correctly, so the driver won't return those correctly. (Bug#3570)
- `UpdatableResultSet` not picking up default values for `moveToInsertRow()`. (Bug#3557)
- Not specifying database in URL caused `MalformedURL` exception. (Bug#3554)
- Auto-convert MySQL encoding names to Java encoding names if used for `characterEncoding` property. (Bug#3554)
- Use `junit.textui.TestRunner` for all unit tests (to allow them to be run from the command line outside of Ant or Eclipse). (Bug#3554)
- Added encoding names that are recognized on some JVMs to fix case where they were reverse-mapped to MySQL encoding names incorrectly. (Bug#3554)
- Made `StringRegressionTest` 4.1-unicode aware. (Bug#3520)
- Fixed regression in `PreparedStatement.setString()` and eastern character encodings. (Bug#3520)
- `DBMD.getSQLStateType()` returns incorrect value. (Bug#3520)
- Renamed `StringUtils.escapeSJISByteStream()` to more appropriate `escapeEasternUnicodeByteStream()`. (Bug#3511)
- `StringUtils.escapeSJISByteStream()` not covering all eastern double-byte charsets correctly. (Bug#3511)
- Return creating statement for `ResultSets` created by `getGeneratedKeys()`. (Bug#2957)
- Use `SET character_set_results` during initialization to allow any charset to be returned to the driver for result sets. (Bug#2670)
- Don't truncate `BLOB` or `CLOB` values when using `setBytes()` and/or `setBinary/CharacterStream()`. . (Bug#2670)
- Dynamically configure character set mappings for field-level character sets on MySQL-4.1.0 and newer using `SHOW COLLATION` when connecting. (Bug#2670)
- Map `binary` character set to `US-ASCII` to support `DATETIME` charset recognition for servers >= 4.1.2. (Bug#2670)
- Use `charsetnr` returned during connect to encode queries before issuing `SET NAMES` on MySQL >= 4.1.0. (Bug#2670)
- Add helper methods to `ResultSetMetaData` (`getColumnCharacterEncoding()` and `getColumnCharacterSet()`) to allow end-users to see what charset the driver thinks it should be using for the column. (Bug#2670)
- Only set `character_set_results` for MySQL >= 4.1.0. (Bug#2670)
- Allow `url` parameter for `MysqlDataSource` and `MysqlConnectionPool DataSource` so that passing of other properties is possible from inside appservers.
- Don't escape SJIS/GBK/BIG5 when using MySQL-4.1 or newer.
- Backport documentation tooling from 3.1 branch.
- Added `failOverReadOnly` property, to allow end-user to configure state of connection (read-only/writable) when failed over.
- Allow `java.util.Date` to be sent in as parameter to `PreparedStatement.setObject()`, converting it to a `Timestamp` to maintain full precision. . (Bug#103)
- Add unsigned attribute to `DatabaseMetaData.getColumns()` output in the `TYPE_NAME` column.
- Map duplicate key and foreign key errors to SQLState of `23000`.
- Backported “change user” and “reset server state” functionality from 3.1 branch, to allow clients of `MysqlConnectionPoolDataSource` to reset server state on `getConnection()` on a pooled connection.

D.4.7. Changes in MySQL Connector/J 3.0.11 (19 February 2004)

Bugs fixed:

- Return `java.lang.Double` for `FLOAT` type from `ResultSetMetaData.getColumnClassName()`. (Bug#2855)
- Return `[B` instead of `java.lang.Object` for `BINARY`, `VARBINARY` and `LONGVARBINARY` types from `ResultSet-MetaData.getColumnClassName()` (JDBC compliance). (Bug#2855)
- Issue connection events on all instances created from a `ConnectionPoolDataSource`. (Bug#2855)
- Return `java.lang.Integer` for `TINYINT` and `SMALLINT` types from `ResultSet-MetaData.getColumnClassName()`. (Bug#2852)
- Added `useUnbufferedInput` parameter, and now use it by default (due to JVM issue http://developer.java.sun.com/developer/bugParade/bugs/4401235.html) (Bug#2578)
- Fixed failover always going to last host in list. (Bug#2578)
- Detect `on`/`off` or `1`, `2`, `3` form of `lower_case_table_names` value on server. (Bug#2578)
- `AutoReconnect` time was growing faster than exponentially. (Bug#2447)
- Trigger a `SET NAMES utf8` when encoding is forced to `utf8` *or* `utf-8` via the `characterEncoding` property. Previously, only the Java-style encoding name of `utf-8` would trigger this.

D.4.8. Changes in MySQL Connector/J 3.0.10 (13 January 2004)

Bugs fixed:

- Enable caching of the parsing stage of prepared statements via the `cachePrepStmts`, `prepStmtCacheSize`, and `prep-StmtCacheSqlLimit` properties (disabled by default). (Bug#2006)
- Fixed security exception when used in Applets (applets can't read the system property `file.encoding` which is needed for `LOAD DATA LOCAL INFILE`). (Bug#2006)
- Speed up parsing of `PreparedStatements`, try to use one-pass whenever possible. (Bug#2006)
- Fixed exception `Unknown character set 'danish'` on connect with JDK-1.4.0 (Bug#2006)
- Fixed mappings in SQLError to report deadlocks with SQLStates of `41000`. (Bug#2006)
- Removed static synchronization bottleneck from instance factory method of `SingleByteCharsetConverter`. (Bug#2006)
- Removed static synchronization bottleneck from `PreparedStatement.setTimestamp()`. (Bug#2006)
- `ResultSet.findColumn()` should use first matching column name when there are duplicate column names in `SELECT` query (JDBC-compliance). (Bug#2006)
- `maxRows` property would affect internal statements, so check it for all statement creation internal to the driver, and set to 0 when it is not. (Bug#2006)
- Use constants for SQLStates. (Bug#2006)
- Map charset `ko18_ru` to `ko18r` when connected to MySQL-4.1.0 or newer. (Bug#2006)
- Ensure that `Buffer.writeString()` saves room for the `\0`. (Bug#2006)
- `ArrayIndexOutOfBounds` when parameter number == number of parameters + 1. (Bug#1958)
- Connection property `maxRows` not honored. (Bug#1933)
- Statements being created too many times in `DBMD.extractForeignKeyFromCreateTable()`. (Bug#1925)
- Support escape sequence {fn convert ... }. (Bug#1914)
- Implement `ResultSet.updateClob()`. (Bug#1913)
- Autoreconnect code didn't set catalog upon reconnect if it had been changed. (Bug#1913)

- `ResultSet.getObject()` on `TINYINT` and `SMALLINT` columns should return Java type `Integer`. (Bug#1913)
- Added more descriptive error message `Server Configuration Denies Access to DataSource`, as well as retrieval of message from server. (Bug#1913)
- `ResultSetMetaData.isCaseSensitive()` returned wrong value for `CHAR`/`VARCHAR` columns. (Bug#1913)
- Added `alwaysClearStream` connection property, which causes the driver to always empty any remaining data on the input stream before each query. (Bug#1913)
- `DatabaseMetaData.getSystemFunction()` returning bad function `VResultsSion`. (Bug#1775)
- Foreign Keys column sequence is not consistent in `DatabaseMetaData.getImported/Exported/CrossReference()`. (Bug#1731)
- Fix for `ArrayIndexOutOfBounds` exception when using `Statement.setMaxRows()`. (Bug#1695)
- Subsequent call to `ResultSet.updateFoo()` causes NPE if result set is not updatable. (Bug#1630)
- Fix for 4.1.1-style authentication with no password. (Bug#1630)
- Cross-database updatable result sets are not checked for updatability correctly. (Bug#1592)
- `DatabaseMetaData.getColumns()` should return `Types.LONGVARCHAR` for MySQL `LONGTEXT` type. (Bug#1592)
- Fixed regression of `Statement.getGeneratedKeys()` and `REPLACE` statements. (Bug#1576)
- Barge blobs and split packets not being read correctly. (Bug#1576)
- Backported fix for aliased tables and `UpdatableResultSets` in `checkUpdatability()` method from 3.1 branch. (Bug#1534)
- “Friendlier” exception message for `PacketTooLargeException`. (Bug#1534)
- Don't count quoted IDs when inside a 'string' in `PreparedStatement` parsing. (Bug#1511)

D.4.9. Changes in MySQL Connector/J 3.0.9 (07 October 2003)

Bugs fixed:

- `ResultSet.get/setString` mashing char 127. (Bug#1247)
- Added property to “clobber” streaming results, by setting the `clobberStreamingResults` property to `true` (the default is `false`). This will cause a “streaming” `ResultSet` to be automatically closed, and any oustanding data still streaming from the server to be discarded if another query is executed before all the data has been read from the server. (Bug#1247)
- Added `com.mysql.jdbc.util.BaseBugReport` to help creation of testcases for bug reports. (Bug#1247)
- Backported authentication changes for 4.1.1 and newer from 3.1 branch. (Bug#1247)
- Made `databaseName`, `portNumber`, and `serverName` optional parameters for `MysqlDataSourceFactory`. (Bug#1246)
- Optimized `CLOB.setChracterStream()`. (Bug#1131)
- Fixed `CLOB.truncate()`. (Bug#1130)
- Fixed deadlock issue with `Statement.setMaxRows()`. (Bug#1099)
- `DatabaseMetaData.getColumns()` getting confused about the keyword “set” in character columns. (Bug#1099)
- Clip +/- INF (to smallest and largest representative values for the type in MySQL) and NaN (to 0) for `setDouble`/`setFloat()`, and issue a warning on the statement when the server does not support +/- INF or NaN. (Bug#884)
- Don't fire connection closed events when closing pooled connections, or on `PooledConnection.getConnection()` with already open connections. (Bug#884)
- Double-escaping of `'\'` when charset is SJIS or GBK and `'\'` appears in non-escaped input. (Bug#879)

- When emptying input stream of unused rows for “streaming” result sets, have the current thread `yield()` every 100 rows in order to not monopolize CPU time. (Bug#879)
- Issue exception on `ResultSet.get*XXX*()` on empty result set (wasn't caught in some cases). (Bug#848)
- Don't hide messages from exceptions thrown in I/O layers. (Bug#848)
- Fixed regression in large split-packet handling. (Bug#848)
- Better diagnostic error messages in exceptions for “streaming” result sets. (Bug#848)
- Don't change timestamp TZ twice if `useTimezone==true`. (Bug#774)
- Don't wrap `SQLExceptions` in `RowDataDynamic`. (Bug#688)
- Don't try and reset isolation level on reconnect if MySQL doesn't support them. (Bug#688)
- The `insertRow` in an `UpdatableResultSet` is now loaded with the default column values when `moveToInser-tRow()` is called. (Bug#688)
- `DatabaseMetaData.getColumns()` wasn't returning `NULL` for default values that are specified as `NULL`. (Bug#688)
- Change default statement type/concurrency to `TYPE_FORWARD_ONLY` and `CONCUR_READ_ONLY` (spec compliance). (Bug#688)
- Fix `UpdatableResultSet` to return values for `get*XXX*()` when on insert row. (Bug#675)
- Support `InnoDB` contraint names when extracting foreign key information in `DatabaseMetaData` (implementing ideas from Parwinder Sekhon). (Bug#664, Bug#517)
- Backported 4.1 protocol changes from 3.1 branch (server-side SQL states, new field information, larger client capability flags, connect-with-database, and so forth). (Bug#664, Bug#517)
- `refreshRow` didn't work when primary key values contained values that needed to be escaped (they ended up being doubly escaped). (Bug#661)
- Fixed `ResultSet.previous()` behavior to move current position to before result set when on first row of result set. (Bug#496)
- Fixed `Statement` and `PreparedStatement` issuing bogus queries when `setMaxRows()` had been used and a `LIMIT` clause was present in the query. (Bug#496)
- Faster date handling code in `ResultSet` and `PreparedStatement` (no longer uses `Date` methods that synchronize on static calendars).
- Fixed test for end of buffer in `Buffer.readString()`.

D.4.10. Changes in MySQL Connector/J 3.0.8 (23 May 2003)

Bugs fixed:

- Fixed SJIS encoding bug, thanks to Naoto Sato. (Bug#378)
- Fix problem detecting server character set in some cases. (Bug#378)
- Allow multiple calls to `Statement.close()`. (Bug#378)
- Return correct number of generated keys when using `REPLACE` statements. (Bug#378)
- Unicode character 0xFFFF in a string would cause the driver to throw an `ArrayOutOfBoundsException`. . (Bug#378)
- Fix row data decoding error when using *very* large packets. (Bug#378)
- Optimized row data decoding. (Bug#378)
- Issue exception when operating on an already closed prepared statement. (Bug#378)
- Optimized usage of `EscapeProcessor`. (Bug#378)

- Use JVM charset with file names and `LOAD DATA [LOCAL] INFILE`.
- Fix infinite loop with `Connection.cleanup()`.
- Changed Ant target `compile-core` to `compile-driver`, and made testsuite compilation a separate target.
- Fixed result set not getting set for `Statement.executeUpdate()`, which affected `getGeneratedKeys()` and `getUpdateCount()` in some cases.
- Return list of generated keys when using multi-value `INSERTS` with `Statement.getGeneratedKeys()`.
- Allow bogus URLs in `Driver.getPropertyInfo()`.

D.4.11. Changes in MySQL Connector/J 3.0.7 (08 April 2003)

Bugs fixed:

- Fixed charset issues with database metadata (charset was not getting set correctly).
- You can now toggle profiling on/off using `Connection.setProfileSql(boolean)`.
- 4.1 Column Metadata fixes.
- Fixed `MysqlPooledConnection.close()` calling wrong event type.
- Fixed `StringIndexOutOfBoundsException` in `PreparedStatement.setClob()`.
- `IOExceptions` during a transaction now cause the `Connection` to be closed.
- Remove synchronization from `Driver.connect()` and `Driver.acceptsUrl()`.
- Fixed missing conversion for `YEAR` type in `ResultSetMetaData.getColumnTypeName()`.
- Updatable `ResultSets` can now be created for aliased tables/columns when connected to MySQL-4.1 or newer.
- Fixed `LOAD DATA LOCAL INFILE` bug when file > `max_allowed_packet`.
- Don't pick up indexes that start with `pri` as primary keys for `DBMD.getPrimaryKeys()`.
- Ensure that packet size from `alignPacketSize()` does not exceed `max_allowed_packet` (JVM bug)
- Don't reset `Connection.isReadOnly()` when autoReconnecting.
- Fixed escaping of 0x5c (`'\'`) character for GBK and Big5 charsets.
- Fixed `ResultSet.getTimestamp()` when underlying field is of type `DATE`.
- Throw `SQLExceptions` when trying to do operations on a forcefully closed `Connection` (that is, when a communication link failure occurs).

D.4.12. Changes in MySQL Connector/J 3.0.6 (18 February 2003)

Bugs fixed:

- Backported 4.1 charset field info changes from Connector/J 3.1.
- Fixed `Statement.setMaxRows()` to stop sending `LIMIT` type queries when not needed (performance).
- Fixed `DBMD.getTypeInfo()` and `DBMD.getColumns()` returning different value for precision in `TEXT` and `BLOB` types.
- Fixed `SQLExceptions` getting swallowed on initial connect.
- Fixed `ResultSetMetaData` to return `""` when catalog not known. Fixes `NullPointerExceptions` with Sun's `CachedRowSet`.
- Allow ignoring of warning for “non transactional tables” during rollback (compliance/usability) by setting `ig-`

noreNonTxTables property to true.

- Clean up Statement query/method mismatch tests (that is, INSERT not allowed with .executeQuery()).
- Fixed ResultSetMetaData.isWritable() to return correct value.
- More checks added in ResultSet traversal method to catch when in closed state.
- Implemented Blob.setBytes(). You still need to pass the resultant Blob back into an updatable ResultSet or PreparedStatement to persist the changes, because MySQL does not support “locators”.
- Add “window” of different NULL sorting behavior to DBMD.nullsAreSortedAtStart (4.0.2 to 4.0.10, true; otherwise, no).

D.4.13. Changes in MySQL Connector/J 3.0.5 (22 January 2003)

Bugs fixed:

- Fixed ResultSet.isBeforeFirst() for empty result sets.
- Added missing LONGTEXT type to DBMD.getColumns().
- Implemented an empty TypeMap for Connection.getTypeMap() so that some third-party apps work with MySQL (IBM WebSphere 5.0 Connection pool).
- Added update options for foreign key metadata.
- Fixed Buffer.fastSkipLenString() causing ArrayIndexOutOfBounds exceptions with some queries when unpacking fields.
- Quote table names in DatabaseMetaData.getColumns(), getPrimaryKeys(), getIndexInfo(), getBestRowIdentifier().
- Retrieve TX_ISOLATION from database for Connection.getTransactionIsolation() when the MySQL version supports it, instead of an instance variable.
- Greatly reduce memory required for setBinaryStream() in PreparedStatements.

D.4.14. Changes in MySQL Connector/J 3.0.4 (06 January 2003)

Bugs fixed:

- Streamlined character conversion and byte[] handling in PreparedStatements for setByte().
- Fixed PreparedStatement.executeBatch() parameter overwriting.
- Added quoted identifiers to database names for Connection.setCatalog.
- Added support for 4.0.8-style large packets.
- Reduce memory footprint of PreparedStatements by sharing outbound packet with MysqlIO.
- Added strictUpdates property to allow control of amount of checking for “correctness” of updatable result sets. Set this to false if you want faster updatable result sets and you know that you create them from SELECT statements on tables with primary keys and that you have selected all primary keys in your query.
- Added support for quoted identifiers in PreparedStatement parser.

D.4.15. Changes in MySQL Connector/J 3.0.3 (17 December 2002)

Bugs fixed:

- Allow user to alter behavior of Statement/ PreparedStatement.executeBatch() via continueBatchOnError property (defaults to true).

- More robust escape tokenizer: Recognize `--` comments, and allow nested escape sequences (see `testsuite.EscapeProcessingTest`).
- Fixed `Buffer.isLastDataPacket()` for 4.1 and newer servers.
- `NamedPipeSocketFactory` now works (only intended for Windows), see `README` for instructions.
- Changed `charsToByte` in `SingleByteCharConverter` to be non-static.
- Use non-aliased table/column names and database names to fully qualify tables and columns in `UpdatableResultSet` (requires MySQL-4.1 or newer).
- `LOAD DATA LOCAL INFILE ...` now works, if your server is configured to allow it. Can be turned off with the `allowLoadLocalInfile` property (see the `README`).
- Implemented `Connection.nativeSQL()`.
- Fixed `ResultSetMetaData.getColumnTypeName()` returning `BLOB` for `TEXT` and `TEXT` for `BLOB` types.
- Fixed charset handling in `Fields.java`.
- Because of above, implemented `ResultSetMetaData.isAutoIncrement()` to use `Field.isAutoIncrement()`.
- Substitute `'?'` for unknown character conversions in single-byte character sets instead of `'\0'`.
- Added `CLIENT_LONG_FLAG` to be able to get more column flags (`isAutoIncrement()` being the most important).
- Honor `lower_case_table_names` when enabled in the server when doing table name comparisons in `DatabaseMetaData` methods.
- `DBMD.getImported/ExportedKeys()` now handles multiple foreign keys per table.
- More robust implementation of updatable result sets. Checks that *all* primary keys of the table have been selected.
- Some MySQL-4.1 protocol support (extended field info from selects).
- Check for connection closed in more `Connection` methods (`createStatement`, `prepareStatement`, `setTransactionIsolation`, `setAutoCommit`).
- Fixed `ResultSetMetaData.getPrecision()` returning incorrect values for some floating-point types.
- Changed `SingleByteCharConverter` to use lazy initialization of each converter.

D.4.16. Changes in MySQL Connector/J 3.0.2 (08 November 2002)

Bugs fixed:

- Implemented `Clob.setString()`.
- Added `com.mysql.jdbc.MiniAdmin` class, which allows you to send `shutdown` command to MySQL server. This is intended to be used when “embedding” Java and MySQL server together in an end-user application.
- Added SSL support. See `README` for information on how to use it.
- All `DBMD` result set columns describing schemas now return `NULL` to be more compliant with the behavior of other JDBC drivers for other database systems (MySQL does not support schemas).
- Use `SHOW CREATE TABLE` when possible for determining foreign key information for `DatabaseMetaData`. Also allows cascade options for `DELETE` information to be returned.
- Implemented `Clob.setCharacterStream()`.
- Failover and `autoReconnect` work only when the connection is in an `autoCommit(false)` state, in order to stay transaction-safe.
- Fixed `DBMD.supportsResultSetConcurrency()` so that it returns `true` for `ResultSet.TYPE_SCROLL_INSENSITIVE` and `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`.
- Implemented `Clob.setAsciiStream()`.

- Removed duplicate code from `UpdatableResultSet` (it can be inherited from `ResultSet`, the extra code for each method to handle updatability I thought might someday be necessary has not been needed).
- Fixed `UnsupportedEncodingException` thrown when “forcing” a character encoding via properties.
- Fixed incorrect conversion in `ResultSet.getLong()`.
- Implemented `ResultSet.updateBlob()`.
- Removed some not-needed temporary object creation by smarter use of `Strings` in `EscapeProcessor`, `Connection` and `DatabaseMetaData` classes.
- Escape `0x5c` character in strings for the SJIS charset.
- `PreparedStatement` now honors stream lengths in setBinary/Ascii/Character Stream() unless you set the connection property `useStreamLengthsInPrepStmts` to `false`.
- Fixed issue with updatable result sets and `PreparedStatements` not working.
- Fixed start position off-by-1 error in `Clob.getSubString()`.
- Added `connectTimeout` parameter that allows users of JDK-1.4 and newer to specify a maximum time to wait to establish a connection.
- Fixed various non-ASCII character encoding issues.
- Fixed `ResultSet.isLast()` for empty result sets (should return `false`).
- Added driver property `useHostsInPrivileges`. Defaults to `true`. Affects whether or not `@hostname` will be used in `DBMD.getColumn/TablePrivileges`.
- Fixed `ResultSet.setFetchDirection(FETCH_UNKNOWN)`.
- Added `queriesBeforeRetryMaster` property that specifies how many queries to issue when failed over before attempting to reconnect to the master (defaults to 50).
- Fixed issue when calling `Statement.setFetchSize()` when using arbitrary values.
- Properly restore connection properties when autoReconnecting or failing-over, including `autoCommit` state, and isolation level.
- Implemented `Clob.truncate()`.

D.4.17. Changes in MySQL Connector/J 3.0.1 (21 September 2002)

Bugs fixed:

- Charsets now automatically detected. Optimized code for single-byte character set conversion.
- Fixed `ResultSetMetaData.isSigned()` for `TINYINT` and `BIGINT`.
- Fixed `RowDataStatic.getAt()` off-by-one bug.
- Fixed `ResultSet.getRow()` off-by-one bug.
- Massive code clean-up to follow Java coding conventions (the time had come).
- Implemented `ResultSet.getCharacterStream()`.
- Added limited `Clob` functionality (`ResultSet.getClob()`, `PreparedStatemtent.setClob()`, `PreparedStatement.setObject(Clob)`.
- `Connection.isClosed()` no longer “pings” the server.
- `Connection.close()` issues `rollback()` when `getAutoCommit()` is `false`.
- Added `socketTimeout` parameter to URL.
- Added `LOCAL TEMPORARY` to table types in `DatabaseMetaData.getTableTypes()`.

- Added `paranoid` parameter, which sanitizes error messages by removing “sensitive” information from them (such as host names, ports, or user names), as well as clearing “sensitive” data structures when possible.

D.4.18. Changes in MySQL Connector/J 3.0.0 (31 July 2002)

Bugs fixed:

- General source-code cleanup.
- The driver now only works with JDK-1.2 or newer.
- Fix and sort primary key names in `DBMetaData` (SF bugs 582086 and 582086).
- `ResultSet.getTimestamp()` now works for `DATE` types (SF bug 559134).
- Float types now reported as `java.sql.Types.FLOAT` (SF bug 579573).
- Support for streaming (row-by-row) result sets (see `README`) Thanks to Doron.
- Testsuite now uses Junit (which you can get from http://www.junit.org.
- JDBC Compliance: Passes all tests besides stored procedure tests.
- `ResultSet.getDate/Time/Timestamp` now recognizes all forms of invalid values that have been set to all zeros by MySQL (SF bug 586058).
- Added multi-host failover support (see `README`).
- Repackaging: New driver name is `com.mysql.jdbc.Driver`, old name still works, though (the driver is now provided by MySQL-AB).
- Support for large packets (new addition to MySQL-4.0 protocol), see `README` for more information.
- Better checking for closed connections in `Statement` and `PreparedStatement`.
- Performance improvements in string handling and field metadata creation (lazily instantiated) contributed by Alex Twisleton-Wykeham-Fiennes.
- JDBC-3.0 functionality including `Statement/PreparedStatement.getGeneratedKeys()` and `ResultSet.getURL()`.
- Overall speed improvements via controlling transient object creation in `MysqlIO` class when reading packets.
- **!!! LICENSE CHANGE !!!** The driver is now GPL. If you need non-GPL licenses, please contact me `<mark@mysql.com>`.
- Performance enchancements: Driver is now 50–100% faster in most situations, and creates fewer temporary objects.

D.5. Changes in MySQL Connector/J 2.0.x

D.5.1. Changes in MySQL Connector/J 2.0.14 (16 May 2002)

Bugs fixed:

- `ResultSet.getDouble()` now uses code built into JDK to be more precise (but slower).
- Fixed typo for `relaxAutoCommit` parameter.
- `LogicalHandle.isClosed()` calls through to physical connection.
- Added SQL profiling (to `STDERR`). Set `profileSql=true` in your JDBC URL. See `README` for more information.
- `PreparedStatement` now releases resources on `.close()`. (SF bug 553268)
- More code cleanup.
- Quoted identifiers not used if server version does not support them. Also, if server started with `--ansi` or `-`

`-sql-mode=ANSI_QUOTES`, “`"`” will be used as an identifier quote character, otherwise “`'`” will be used.

D.5.2. Changes in MySQL Connector/J 2.0.13 (24 April 2002)

Bugs fixed:

- Fixed unicode chars being read incorrectly. (SF bug 541088)
- Faster blob escaping for `PrepStmt`.
- Added `setURL()` to `MySQLXADataSource`. (SF bug 546019)
- Added `set`/`getPortNumber()` to `DataSource(s)`. (SF bug 548167)
- `PreparedStatement.toString()` fixed. (SF bug 534026)
- More code cleanup.
- Rudimentary version of `Statement.getGeneratedKeys()` from JDBC-3.0 now implemented (you need to be using JDK-1.4 for this to work, I believe).
- `DBMetaData.getIndexInfo()` - bad PAGES fixed. (SF BUG 542201)
- `ResultSetMetaData.getColumnClassName()` now implemented.

D.5.3. Changes in MySQL Connector/J 2.0.12 (07 April 2002)

Bugs fixed:

- Fixed `testsuite.Traversal` `afterLast()` bug, thanks to Igor Lastric.
- Added new types to `getTypeInfo()`, fixed existing types thanks to Al Davis and Kid Kalanon.
- Fixed time zone off-by-1-hour bug in `PreparedStatement` (538286, 528785).
- Added identifier quoting to all `DatabaseMetaData` methods that need them (should fix 518108).
- Added support for `BIT` types (51870) to `PreparedStatement`.
- `ResultSet.insertRow()` should now detect auto_increment fields in most cases and use that value in the new row. This detection will not work in multi-valued keys, however, due to the fact that the MySQL protocol does not return this information.
- Relaxed synchronization in all classes, should fix 520615 and 520393.
- `DataSources` - fixed `setUrl` bug (511614, 525565), wrong datasource class name (532816, 528767).
- Added support for `YEAR` type (533556).
- Fixes for `ResultSet` updatability in `PreparedStatement`.
- `ResultSet`: Fixed updatability (values being set to `null` if not updated).
- Added `getTable/ColumnPrivileges()` to DBMD (fixes 484502).
- Added `getIdleFor()` method to `Connection` and `MysqlLogicalHandle`.
- `ResultSet.refreshRow()` implemented.
- Fixed `getRow()` bug (527165) in `ResultSet`.
- General code cleanup.

D.5.4. Changes in MySQL Connector/J 2.0.11 (27 January 2002)

Bugs fixed:

- Full synchronization of `Statement.java`.
- Fixed missing `DELETE_RULE` value in `DBMD.getImported/ExportedKeys()` and `getCrossReference()`.
- More changes to fix `Unexpected end of input stream` errors when reading `BLOB` values. This should be the last fix.

D.5.5. Changes in MySQL Connector/J 2.0.10 (24 January 2002)

Bugs fixed:

- Fixed null-pointer-exceptions when using `MysqlConnectionPoolDataSource` with Websphere 4 (bug 505839).
- Fixed spurious `Unexpected end of input stream` errors in `MysqlIO` (bug 507456).

D.5.6. Changes in MySQL Connector/J 2.0.9 (13 January 2002)

Bugs fixed:

- Fixed extra memory allocation in `MysqlIO.readPacket()` (bug 488663).
- Added detection of network connection being closed when reading packets (thanks to Todd Lizambri).
- Fixed casting bug in `PreparedStatement` (bug 488663).
- `DataSource` implementations moved to `org.gjt.mm.mysql.jdbc2.optional` package, and (initial) implementations of `PooledConnectionDataSource` and `XADataSource` are in place (thanks to Todd Wolff for the implementation and testing of `PooledConnectionDataSource` with IBM WebSphere 4).
- Fixed quoting error with escape processor (bug 486265).
- Removed concatenation support from driver (the `||` operator), as older versions of VisualAge seem to be the only thing that use it, and it conflicts with the logical `||` operator. You will need to start `mysqld` with the `--ansi` flag to use the `||` operator as concatenation (bug 491680).
- `Ant` build was corrupting included `jar` files, fixed (bug 487669).
- Report batch update support through `DatabaseMetaData` (bug 495101).
- Implementation of `DatabaseMetaData.getExported/ImportedKeys()` and `getCrossReference()`.
- Fixed off-by-one-hour error in `PreparedStatement.setTimestamp()` (bug 491577).
- Full synchronization on methods modifying instance and class-shared references, driver should be entirely thread-safe now (please let me know if you have problems).

D.5.7. Changes in MySQL Connector/J 2.0.8 (25 November 2001)

Bugs fixed:

- `XADataSource`/`ConnectionPoolDataSource` code (experimental)
- `DatabaseMetaData.getPrimaryKeys()` and `getBestRowIdentifier()` are now more robust in identifying primary keys (matches regardless of case or abbreviation/full spelling of `Primary Key` in `Key_type` column).
- Batch updates now supported (thanks to some inspiration from Daniel Rall).
- `PreparedStatement.setAnyNumericType()` now handles positive exponents correctly (adds `+` so MySQL can understand it).

D.5.8. Changes in MySQL Connector/J 2.0.7 (24 October 2001)

Bugs fixed:

- Character sets read from database if `useUnicode=true` and `characterEncoding` is not set. (thanks to Dmitry Vereshchagin)
- Initial transaction isolation level read from database (if available). (thanks to Dmitry Vereshchagin)
- Fixed `PreparedStatement` generating SQL that would end up with syntax errors for some queries.
- `PreparedStatement.setCharacterStream()` now implemented
- Captialize type names when `captializeTypeNames=true` is passed in URL or properties (for WebObjects. (thanks to Anjo Krank)
- `ResultSet.getBlob()` now returns `null` if column value was `null`.
- Fixed `ResultSetMetaData.getPrecision()` returning one less than actual on newer versions of MySQL.
- Fixed dangling socket problem when in high availability (`autoReconnect=true`) mode, and finalizer for `Connection` will close any dangling sockets on GC.
- Fixed time zone issue in `PreparedStatement.setTimestamp()`. (thanks to Erik Olofsson)
- PreparedStatement.setDouble() now uses full-precision doubles (reverting a fix made earlier to truncate them).
- Fixed `DatabaseMetaData.supportsTransactions()`, and `supportsTransactionIsolationLevel()` and `getTypeInfo()` `SQL_DATETIME_SUB` and `SQL_DATA_TYPE` fields not being readable.
- Updatable result sets now correctly handle `NULL` values in fields.
- PreparedStatement.setBoolean() will use 1/0 for values if your MySQL version is 3.21.23 or higher.
- Fixed `ResultSet.isAfterLast()` always returning `false`.

D.5.9. Changes in MySQL Connector/J 2.0.6 (16 June 2001)

Bugs fixed:

- Fixed `PreparedStatement` parameter checking.
- Fixed case-sensitive column names in `ResultSet.java`.

D.5.10. Changes in MySQL Connector/J 2.0.5 (13 June 2001)

Bugs fixed:

- `ResultSet.insertRow()` works now, even if not all columns are set (they will be set to `NULL`).
- Added `Byte` to `PreparedStatement.setObject()`.
- Fixed data parsing of `TIMESTAMP` values with 2-digit years.
- Added `ISOLATION` level support to `Connection.setIsolationLevel()`
- `DataBaseMetaData.getCrossReference()` no longer `ArrayIndexOOB`.
- `ResultSet.getBoolean()` now recognizes `-1` as `true`.
- `ResultSet` has +/-Inf/inf support.
- `getObject()` on `ResultSet` correctly does `TINYINT`->`Byte` and `SMALLINT`->`Short`.
- Fixed `ResultSetMetaData.getColumnTypeName` for `TEXT`/`BLOB`.
- Fixed `ArrayIndexOutOfBounds` when sending large `BLOB` queries. (Max size packet was not being set)

- Fixed NPE on `PreparedStatement.executeUpdate()` when all columns have not been set.
- Fixed `ResultSet.getBlob()` `ArrayIndex` out-of-bounds.

D.5.11. Changes in MySQL Connector/J 2.0.3 (03 December 2000)

Bugs fixed:

- Fixed composite key problem with updatable result sets.
- Faster ASCII string operations.
- Fixed off-by-one error in `java.sql.Blob` implementation code.
- Fixed incorrect detection of `MAX_ALLOWED_PACKET`, so sending large blobs should work now.
- Added detection of -/+INF for doubles.
- Added `ultraDevHack` URL parameter, set to `true` to allow (broken) Macromedia UltraDev to use the driver.
- Implemented `getBigDecimal()` without scale component for JDBC2.

D.5.12. Changes in MySQL Connector/J 2.0.1 (06 April 2000)

Bugs fixed:

- Columns that are of type `TEXT` now return as `Strings` when you use `getObject()`.
- Cleaned up exception handling when driver connects.
- Fixed `RSMD.isWritable()` returning wrong value. Thanks to Moritz Maass.
- `DatabaseMetaData.getPrimaryKeys()` now works correctly with respect to `key_seq`. Thanks to Brian Slesinsky.
- Fixed many JDBC-2.0 traversal, positioning bugs, especially with respect to empty result sets. Thanks to Ron Smits, Nick Brook, Cessar Garcia and Carlos Martinez.
- No escape processing is done on `PreparedStatements` anymore per JDBC spec.
- Fixed some issues with updatability support in `ResultSet` when using multiple primary keys.

D.5.13. Changes in MySQL Connector/J 2.0.0pre5 (21 February 2000)

- Fixed Bad Handshake problem.

D.5.14. Changes in MySQL Connector/J 2.0.0pre4 (10 January 2000)

- Fixes to ResultSet for insertRow() - Thanks to Cesar Garcia
- Fix to Driver to recognize JDBC-2.0 by loading a JDBC-2.0 class, instead of relying on JDK version numbers. Thanks to John Baker.
- Fixed ResultSet to return correct row numbers
- Statement.getUpdateCount() now returns rows matched, instead of rows actually updated, which is more SQL-92 like.

10-29-99

- Statement/PreparedStatement.getMoreResults() bug fixed. Thanks to Noel J. Bergman.

- Added Short as a type to PreparedStatement.setObject(). Thanks to Jeff Crowder
- Driver now automagically configures maximum/preferred packet sizes by querying server.
- Autoreconnect code uses fast ping command if server supports it.
- Fixed various bugs with respect to packet sizing when reading from the server and when alloc'ing to write to the server.

D.5.15. Changes in MySQL Connector/J 2.0.0pre (17 August 1999)

- Now compiles under JDK-1.2. The driver supports both JDK-1.1 and JDK-1.2 at the same time through a core set of classes. The driver will load the appropriate interface classes at runtime by figuring out which JVM version you are using.
- Fixes for result sets with all nulls in the first row. (Pointed out by Tim Endres)
- Fixes to column numbers in SQLExceptions in ResultSet (Thanks to Blas Rodriguez Somoza)
- The database no longer needs to specified to connect. (Thanks to Christian Motschke)

D.6. Changes in MySQL Connector/J 1.2b (04 July 1999)

- Better Documentation (in progress), in doc/mm.doc/book1.html
- DBMD now allows null for a column name pattern (not in spec), which it changes to '%'.
- DBMD now has correct types/lengths for getXXX().
- ResultSet.getDate(), getTime(), and getTimestamp() fixes. (contributed by Alan Wilken)
- EscapeProcessor now handles \{ \} and { or } inside quotes correctly. (thanks to Alik for some ideas on how to fix it)
- Fixes to properties handling in Connection. (contributed by Juho Tikkala)
- ResultSet.getObject() now returns null for NULL columns in the table, rather than bombing out. (thanks to Ben Grosman)
- ResultSet.getObject() now returns Strings for types from MySQL that it doesn't know about. (Suggested by Chris Perdue)
- Removed DataInput/Output streams, not needed, 1/2 number of method calls per IO operation.
- Use default character encoding if one is not specified. This is a work-around for broken JVMs, because according to spec, EVERY JVM must support "ISO8859_1", but they do not.
- Fixed Connection to use the platform character encoding instead of "ISO8859_1" if one isn't explicitly set. This fixes problems people were having loading the character- converter classes that didn't always exist (JVM bug). (thanks to Fritz Elfert for pointing out this problem)
- Changed MysqlIO to re-use packets where possible to reduce memory usage.
- Fixed escape-processor bugs pertaining to {} inside quotes.

D.7. Changes in MySQL Connector/J 1.2.x and lower

D.7.1. Changes in MySQL Connector/J 1.2a (14 April 1999)

- Fixed character-set support for non-Javasoft JVMs (thanks to many people for pointing it out)
- Fixed ResultSet.getBoolean() to recognize 'y' & 'n' as well as '1' & '0' as boolean flags. (thanks to Tim Pizey)
- Fixed ResultSet.getTimestamp() to give better performance. (thanks to Richard Swift)
- Fixed getByte() for numeric types. (thanks to Ray Bellis)
- Fixed DatabaseMetaData.getTypeInfo() for DATE type. (thanks to Paul Johnston)

- Fixed EscapeProcessor for "fn" calls. (thanks to Piyush Shah at locomotive.org)
- Fixed EscapeProcessor to not do extraneous work if there are no escape codes. (thanks to Ryan Gustafson)
- Fixed Driver to parse URLs of the form "jdbc:mysql://host:port" (thanks to Richard Lobb)

D.7.2. Changes in MySQL Connector/J 1.1i (24 March 1999)

- Fixed Timestamps for PreparedStatements
- Fixed null pointer exceptions in RSMD and RS
- Re-compiled with jikes for valid class files (thanks ms!)

D.7.3. Changes in MySQL Connector/J 1.1h (08 March 1999)

- Fixed escape processor to deal with unmatched { and } (thanks to Craig Coles)
- Fixed escape processor to create more portable (between DATETIME and TIMESTAMP types) representations so that it will work with BETWEEN clauses. (thanks to Craig Longman)
- MysqlIO.quit() now closes the socket connection. Before, after many failed connections some OS's would run out of file descriptors. (thanks to Michael Brinkman)
- Fixed NullPointerException in Driver.getPropertyInfo. (thanks to Dave Potts)
- Fixes to MysqlDefs to allow all *text fields to be retrieved as Strings. (thanks to Chris at Leverage)
- Fixed setDouble in PreparedStatement for large numbers to avoid sending scientific notation to the database. (thanks to J.S. Ferguson)
- Fixed getScale() and getPrecision() in RSMD. (contrib'd by James Klicman)
- Fixed getObject() when field was DECIMAL or NUMERIC (thanks to Bert Hobbs)
- DBMD.getTables() bombed when passed a null table-name pattern. Fixed. (thanks to Richard Lobb)
- Added check for "client not authorized" errors during connect. (thanks to Hannes Wallnoefer)

D.7.4. Changes in MySQL Connector/J 1.1g (19 February 1999)

- Result set rows are now byte arrays. Blobs and Unicode work bidriectonally now. The useUnicode and encoding options are implemented now.
- Fixes to PreparedStatement to send binary set by setXXXStream to be sent untouched to the MySQL server.
- Fixes to getDriverPropertyInfo().

D.7.5. Changes in MySQL Connector/J 1.1f (31 December 1998)

- Changed all ResultSet fields to Strings, this should allow Unicode to work, but your JVM must be able to convert between the character sets. This should also make reading data from the server be a bit quicker, because there is now no conversion from StringBuffer to String.
- Changed PreparedStatement.streamToString() to be more efficient (code from Uwe Schaefer).
- URL parsing is more robust (throws SQL exceptions on errors rather than NullPointerExceptions)
- PreparedStatement now can convert Strings to Time/Date values via setObject() (code from Robert Currey).
- IO no longer hangs in Buffer.readInt(), that bug was introduced in 1.1d when changing to all byte-arrays for result sets. (Pointed out by Samo Login)

D.7.6. Changes in MySQL Connector/J 1.1b (03 November 1998)

- Fixes to DatabaseMetaData to allow both IBM VA and J-Builder to work. Let me know how it goes. (thanks to Jac Kersing)
- Fix to ResultSet.getBoolean() for NULL strings (thanks to Barry Lagerweij)
- Beginning of code cleanup, and formatting. Getting ready to branch this off to a parallel JDBC-2.0 source tree.
- Added "final" modifier to critical sections in MysqlIO and Buffer to allow compiler to inline methods for speed.

9-29-98

- If object references passed to setXXX() in PreparedStatement are null, setNull() is automatically called for you. (Thanks for the suggestion goes to Erik Ostrom)
- setObject() in PreparedStatement will now attempt to write a serialized representation of the object to the database for objects of Types.OTHER and objects of unknown type.
- Util now has a static method readObject() which given a ResultSet and a column index will re-instantiate an object serialized in the above manner.

D.7.7. Changes in MySQL Connector/J 1.1 (02 September 1998)

- Got rid of "ugly hack" in MysqlIO.nextRow(). Rather than catch an exception, Buffer.isLastDataPacket() was fixed.
- Connection.getCatalog() and Connection.setCatalog() should work now.
- Statement.setMaxRows() works, as well as setting by property maxRows. Statement.setMaxRows() overrides maxRows set via properties or url parameters.
- Automatic re-connection is available. Because it has to "ping" the database before each query, it is turned off by default. To use it, pass in "autoReconnect=true" in the connection URL. You may also change the number of reconnect tries, and the initial timeout value via "maxReconnects=n" (default 3) and "initialTimeout=n" (seconds, default 2) parameters. The timeout is an exponential backoff type of timeout; for example, if you have initial timeout of 2 seconds, and maxReconnects of 3, then the driver will timeout 2 seconds, 4 seconds, then 16 seconds between each re-connection attempt.

D.7.8. Changes in MySQL Connector/J 1.0 (24 August 1998)

- Fixed handling of blob data in Buffer.java
- Fixed bug with authentication packet being sized too small.
- The JDBC Driver is now under the LPGL

8-14-98

- Fixed Buffer.readLenString() to correctly read data for BLOBS.
- Fixed PreparedStatement.stringToStream to correctly read data for BLOBS.
- Fixed PreparedStatement.setDate() to not add a day. (above fixes thanks to Vincent Partington)
- Added URL parameter parsing (?user=... and so forth).

D.7.9. Changes in MySQL Connector/J 0.9d (04 August 1998)

- Big news! New package name. Tim Endres from ICE Engineering is starting a new source tree for GNU GPL'd Java software. He's graciously given me the org.gjt.mm package directory to use, so now the driver is in the org.gjt.mm.mysql package scheme. I'm "legal" now. Look for more information on Tim's project soon.

- Now using dynamically sized packets to reduce memory usage when sending commands to the DB.
- Small fixes to getTypeInfo() for parameters, and so forth.
- DatabaseMetaData is now fully implemented. Let me know if these drivers work with the various IDEs out there. I've heard that they're working with JBuilder right now.
- Added JavaDoc documentation to the package.
- Package now available in .zip or .tar.gz.

D.7.10. Changes in MySQL Connector/J 0.9 (28 July 1998)

- Implemented getTypeInfo(). Connection.rollback() now throws an SQLException per the JDBC spec.
- Added PreparedStatement that supports all JDBC API methods for PreparedStatement including InputStreams. Please check this out and let me know if anything is broken.
- Fixed a bug in ResultSet that would break some queries that only returned 1 row.
- Fixed bugs in DatabaseMetaData.getTables(), DatabaseMetaData.getColumns() and DatabaseMetaData.getCatalogs().
- Added functionality to Statement that allows executeUpdate() to store values for IDs that are automatically generated for AUTO_INCREMENT fields. Basically, after an executeUpdate(), look at the SQLWarnings for warnings like "LAST_INSERTED_ID = 'some number', COMMAND = 'your SQL query'". If you are using AUTO_INCREMENT fields in your tables and are executing a lot of executeUpdate()s on one Statement, be sure to clearWarnings() every so often to save memory.

D.7.11. Changes in MySQL Connector/J 0.8 (06 July 1998)

- Split MysqlIO and Buffer to separate classes. Some ClassLoaders gave an IllegalAccess error for some fields in those two classes. Now mm.mysql works in applets and all classloaders. Thanks to Joe Ennis <jce@mail.boone.com> for pointing out the problem and working on a fix with me.

D.7.12. Changes in MySQL Connector/J 0.7 (01 July 1998)

- Fixed DatabaseMetadata problems in getColumns() and bug in switch statement in the Field constructor. Thanks to Costin Manolache <costin@tdiinc.com> for pointing these out.

D.7.13. Changes in MySQL Connector/J 0.6 (21 May 1998)

- Incorporated efficiency changes from Richard Swift <Richard.Swift@kanatek.ca> in `MysqlIO.java` and `ResultSet.java`:
- We're now 15% faster than gwe's driver.
- Started working on `DatabaseMetaData`.
- The following methods are implemented:
  - `getTables()`
  - `getTableTypes()`
  - `getColumns()`
  - `getCatalogs()`

Appendix E. MySQL Connector/MXJ Change History

E.1. Changes in MySQL Connector/MXJ 5.0.6 (04 May 2007)

Functionality added or changed:

- Updated internal jar file names to include version information and be more consistent with Connector/J jar naming. For example, `connector-mxj.jar` is now `mysql-connector-mxj-${mxj-version}.jar`.
- Updated commercial license files.
- Added copyright notices to some classes which were missing them.
- Added `InitializeUser` and `QueryUtil` classes to support new feature.
- Added new tests for initial-user & expanded some existing tests.
- `ConnectorMXJUrlTestExample` and `ConnectorMXJObjectTestExample` now demonstrate the initialization of user/password and creating the initial database (rather than using "test").
- Added new connection property `initialize-user` which, if set to `true` will remove the default, un-passworded anonymous and root users, and create the user/password from the connection url.
- Removed obsolete field `SimpleMysqldDynamicMBean.lastInvocation`.
- Clarified code in `DefaultsMap.entrySet()`.
- Removed obsolete `PatchedStandardSocketFactory` java file.
- Added `main(String[])` to `com/mysql/management/AllTestsSuite.java`.
- Errors reading `portFile` are now reported using `stacktrace(err)`, previously `System.err` was used.
- `portFile` now contains a new-line to be consistent with `pidFile`.
- Fixed where `versionString.trim()` was ignored.
- Removed references to `File.deleteOnExit`, a warning is printed instead.

Bugs fixed:

- Changed tests to shutdown mysqld prior to deleting files.
- Fixed port file to always be writen to datadir.
- Added os.name-os.arch to resource directory mapping properties file.
- Swapped out commercial binaries for v5.0.40.
- Delete `portFile` on shutdown.
- Moved `platform-map.properties` into `db-files.jar`.
- Clarified the startup max wait numbers.
- Updated `build.xml` in preperation for next beta build.
- Removed `use-default-architecture` property replaced.
- Added null-check to deal with C/MXJ being loaded by the bootstrap classloaders with JVMs for which `getClassLoader()` returns null.
- Added robustness around reading portfile.
- Removed `PatchedStandardSocketFactory` (fixed in Connetor/J 5.0.6).
- Refactored duplication from tests and examples to `QueryUtil`.

- Removed obsolete `InitializePasswordExample`

E.2. Changes in MySQL Connector/MXJ 5.0.5 (14 March 2007)

Bugs fixed:

- Moved `MysqldFactory` to main package.
- Reformatting: Added newlines some files which did not end in them.
- Swapped out commercial binaries for v5.0.36.
- Found and removed dynamic linking in mysql_kill; updated solution.
- Changed protected constructor of `SimpleMysqldDynamicMBean` from taking a `MysqldResource` to taking a `Mysqld-Factory`, in order to lay groundwork for addressing BUG discovered by Andrew Rubinger. See: MySQL Forums (Actual testing with JBoss, and filing a bug, is still required.)
- `build.xml`: `usage` now slightly more verbose; some reformatting.
- Now incoporates Reggie Bernett's `SafeTerminateProcess` and only calls the unsafe TerminateProcess as a final last resort.
- New windows `kill.exe` fixes bug where mysqld was being force terminated. Issue reported by bruno haleblian and others, see: MySQL Forums.
- Replaced `Boolean.parseBoolean` with JDK 1.4 compliant `valueOf`.
- Changed `connector-mxj.properties` default mysql version to 5.0.37.
- In testing so far mysqld reliably shuts down cleanly much faster.
- Added testcase to `com.mysql.management.jmx.AcceptanceTest` which demonstrats that `dataDir` is a mutable MBean property.
- Updated `build.xml` in prep for next release.
- Changed `SimpleMysqldDynamicMBean` to create `MysqldResource` on demand in order to allow setting of `datadir`. (Rubinger bug groundwork).
- Clarified the synchronization of `MysqldResource` methods.
- `SIGHUP` is replaced with `MySQLShutdown<PID>` event.
- Clarified the immutability of `baseDir`, `dataDir`, `pidFile`, `portFile`.
- Added 5.1.15 binaries to the repository.
- Removed 5.1.14 binaries from the repository.
- Added `getDataDir()` to interface `MysqldResourceI`.
- Added 5.1.14 binaries to repository.
- Replaced windows `kill.exe` resource with re-written version specific to mysqld.
- Added Patched `StandardSocketFactory` from Connector/J 5-0 HEAD.
- Ensured 5.1.14 compatibility.
- Swapped out gpl binaries for v5.0.37.
- Removed 5.0.22 binaries from the repository.

E.3. Changes in MySQL Connector/MXJ 5.0.4 (28 January 2007)

Bugs fixed:

- Allow multiple calls to start server from URL connection on non-3306 port. (Bug#24004)
- Updated `build.xml` to build to handle with different gpl and commercial mysld version numbers.
- Only populate the options map from the help text if specifically requested or in the MBean case.
- Introduced property for Linux & WinXX to default to 32bit versions.
- Swapped out gpl binaries for v5.0.27.
- Swapped out commercial binaries for v5.0.32.
- Moved mysqld binary resourced into separate jar file NOTICE: `CLASSPATH` will now need to `connector-mxj-db-files.jar`.
- Minor test robustness improvements.
- Moved default version string out of java class into a text editable properties file (`connector-mxj.properties`) in the resources directory.
- Fixed test to be tollerant of `/tmp` being a symlink to `/foo/tmp`.

E.4. Changes in MySQL Connector/MXJ 5.0.3 (24 June 2006)

Bugs fixed:

- Removed unused imports, formatted code, made minor edits to tests.
- Removed "TeeOutputStream" - no longer needed.
- Swapped out the mysqld binaries for MySQL v5.0.22.

E.5. Changes in MySQL Connector/MXJ 5.0.2 (15 June 2006)

Bugs fixed:

- Replaced string parsing with JDBC connection attempt for determining if a mysqld is "ready for connections" `CLASSPATH` will now need to include Connector/J jar.
- "platform" directories replace spaces with underscores
- extracted array and list printing to ListToString utility class
- Swapped out the mysqld binaries for MySQL v5.0.21
- Added trace level logging with Aspect/J. `CLASSPATH` will now need to include `lib/aspectjrt.jar`
- reformatted code
- altered to be "basedir" rather than "port" oriented.
- help parsing test reflects current help options
- insulated users from problems with "." in basedir
- swapped out the mysqld binaries for MySQL v5.0.18
- Made tests more robust be deleting the /tmp/test-c.mxj directory before running tests.
- ServerLauncherSocketFactory.shutdown API change: now takes File parameter (basedir) instead of port.
- socket is now "mysql.sock" in datadir
- added ability to specify "mysql-version" as an url parameter
- Extended timeout for help string parsing, to avoid cases where the help text was getting prematurely flushed, and thus truncated.

- swapped out the mysqld binaries for MySQL v5.0.19
- MysqldResource now tied to dataDir as well as basedir (API CHANGE)
- moved PID file into datadir
- ServerLauncherSocketFactory.shutdown now works across JVMs.
- extracted splitLines(String) to Str utility class
- ServerLauncherSocketFactory.shutdown(port) no longer throws, only reports to System.err
- ServerLauncherSocketFactory now treats URL parameters in the form of `&server.foo=null` as `serverOptionMap.put("foo", null)`
- ServerLauncherSocketFactory.shutdown API change: now takes 2 File parameters (basedir, datadir)

E.6. Changes in MySQL Connector/MXJ 5.0.1 (Never released)

This was an internal only release.

E.7. Changes in MySQL Connector/MXJ 5.0.0 (09 December 2005)

Bugs fixed:

- Removed HelpOptionsParser's need to reference a MysqldResource.
- Reorganized utils into a single "Utils" collaborator.
- Minor test tweaks
- Altered examples and tests to use new Connector/J 5.0 URL syntax for launching Connector/MXJ ("jdbc:mysql:mxj://")
- Swapped out the mysqld binaries for MySQL v5.0.16.
- Ditched "ClassUtil" (merged with Str).
- Minor refactorings for type casting and exception handling.

Appendix F. MySQL Connector/OpenOffice.org Change History

F.1. Changes in MySQL Connector/OpenOffice.org 1.0.2 (09 April 09 alpha)

This is a new Alpha development release, fixing recently discovered bugs.

F.2. Changes in MySQL Connector/OpenOffice.org 1.0.1 (02 December 08 alpha)

This is a new Alpha development release, fixing recently discovered bugs.

F.3. Changes in MySQL Connector/OpenOffice.org 1.0.0 (05 August 08 preview)

Bugs fixed:

- Test

Appendix G. MySQL Connector/C++ Change History

G.1. Changes in MySQL Connector/C++ 1.0.x

G.1.1. Changes in MySQL Connector/CPP 1.0.5 (21 April 2009)

This is the first Generally Available (GA) release.

Functionality added or changed:

- The interface of `sql::ConnectionMetaData`, `sql::ResultSetMetaData` and `sql::ParameterMetaData` was modified to have a protected destructor. As a result the client code has no need to destruct the metadata objects returned by the connector. MySQL Connector/C++ handles the required destruction. This enables statements such as:

  ```
  connection->getMetaData->getSchema();
  ```

  This avoids potential memory leaks that could occur as a result of losing the pointer returned by `getMetaData()`.
- Improved memory management. Potential memory leak situations are handled more robustly.
- Changed the interface of `sql::Driver` and `sql::Connection` so they accept the options map by alias instead of by value.
- Changed the return type of `sql::SQLException::getSQLState()` from `std::string` to `const char *` to be consistent with `std::exception::what()`.
- Implemented `getResultSetType()` and `setResultSetType()` for `Statement`. Uses `TYPE_FORWARD_ONLY`, which means unbuffered result set and `TYPE_SCROLL_INSENSITIVE`, which means buffered result set.
- Implemented `getResultSetType()` for `PreparedStatement`. The setter is not implemented because currently `Pre-paredStatement` cannot do refetching. Storing the result means the bind buffers will be correct.
- Added the option `defaultStatementResultType` to `MySQL_Connection::setClientOption()`. Also, the method now returns `sql::Connection *`.
- Added `Result::getType()`. Implemented for the three result set classes.
- Enabled tracing functionality when building with Microsoft Visual C++ 8 and later, which corresponds to Microsoft Visual Stu-dio 2005 and later.
- Added better support for named pipes, on Windows. Use `pipe://` and add the path to the pipe. Shared memory connections are currently not supported.

Bugs fixed:

- A bug was fixed in `MySQL_Connection::setSessionVariable()`, which had been causing exceptions to be thrown.

G.1.2. Changes in MySQL Connector/CPP 1.0.4 (31 March 2009 beta)

Functionality added or changed:

- An installer was added for the Windows operating system.
- Minimum CMake version required was changed from 2.4.2 to 2.6.2. The latest version is required for building on Windows.
- `metadataUseInfoSchema` was added to the connection property map, which allows control of the `INFORMA-TION_SCHEMA` for meta data.
- Implemented `MySQL_ConnectionMetaData::supportsConvert(from, to)`.
- Added support for MySQL Connector/C.

Bugs fixed:

- A bug was fixed in all implementations of `ResultSet::relative()` which was giving a wrong return value although positioning was working correctly.
- A leak was fixed in `MySQL_PreparedResultSet`, which occurred when the result contained a `BLOB` column.

G.1.3. Changes in MySQL Connector/CPP 1.0.3 (02 March 2009 alpha)

Functionality added or changed:

- Added new tests in `test/unit/classes`. Those tests are mostly about code coverage. Most of the actual functionality of the driver is tested by the tests found in `test/CJUnitPort`.
- New data types added to the list returned by `DatabaseMetaData::getTypeInfo()` are `FLOAT UNSIGED`, `DECIMAL UNSIGNED`, `DOUBLE UNSIGNED`. Those tests may not be in the JDBC specification. However, due to the change you should be able to look up every type and type name returned by, for example, `ResultSetMetaData::getColumnTypeName()`.
- `MySQL_Driver::getPatchVersion` introduced.
- Major performance improvements due to new buffered `ResultSet` implementation.
- Addition of `test/unit/README` with instructions for writing bug and regression tests.
- Experimental support for STLPort. This feature may be removed again at any time later without prior warning! Type `cmake -L` for configuration instructions.
- Added properties enabled methods for connecting, which add many connect options. This uses a dictionary (map) of key value pairs. Methods added are `Driver::connect(map)`, and `Connection::Connection(map)`.
- New BLOB implementation. `sql::Blob` was removed in favor of `std::istream`. C++'s `IOStream` library is very powerful, similar to PHP's streams. It makes no sense to reinvent the wheel. For example, you can pass a `std::istringstream` object to `setBlob()` if the data is in memory, or just open a file `std::fstream` and let it stream to the DB, or write its own stream. This is also true for `getBlob()` where you can just copy data (if a buffered result set), or stream data (if implemented).
- Implemented `ResultSet::getBlob()` which returns `std::stream`.
- Fixed `MySQL_DatabaseMetaData::getTablePrivileges()`. Test cases were added in the first unit testing framework.
- Implemented `MySQL_Connection::setSessionVariable()` for setting variables like `sql_mode`.
- Implemented `MySQL_DatabaseMetaData::getColumnPrivileges()`.
- `cppconn/datatype.h` has changed and is now used again. Reimplemented the type subsystem to be more usable - more types for binary and nonbinary strings.
- Implementation for `MySQL_DatabaseMetaData::getImportedKeys()` for MySQL versions before 5.1.16 using `SHOW`, and above using `INFORMATION_SCHEMA`.
- Implemented `MySQL_ConnectionMetaData::getProcedureColumns()`.
- `make package_source` now packs with bzip2.
- Re-added `getTypeInfo()` with information about all types supported by MySQL and the `sql::DataType`.
- Changed the implementation of `MySQL_ConstructedResultSet` to use the more efficient O(1) access method. This should improve the speed with which the metadata result sets are used. Also, there is less copying during the construction of the result set, which means that all result sets returned from the meta data functions will be faster.
- Introduced, internally, `sql::mysql::MyVal` which has implicit constructors. Used in `mysql_metadata.cpp` to create result sets with native data instead of always string (varchar).
- Renamed `ResultSet::getLong()` to `ResultSet::getInt64()`. `resultset.h` includes typdefs for Windows to be able to use `int64_t`.

- Introduced `ResultSet::getUInt()` and `ResultSet::getUInt64()`.
- Improved the implementation for `ResultSetMetaData::isReadOnly()`. Values generated from views are read-only. These generated values don't have `db` in `MYSQL_FIELD` set, while all normal columns do have.
- Implemented `MySQL_DatabaseMetaData::getExportedKeys()`.
- Implemented `MySQL_DatabaseMetaData::getCrossReference()`.

Bugs fixed:

- Bug fixed in `MySQL_PreparedResultSet::getString()`. Returned string that had real data but the length was random. Now, the string is initialized with the correct length and thus is binary safe.
- Corrected handling of unsigned server types. Now returning correct values.
- Fixed handling of numeric columns in `ResultSetMetaData::isCaseSensitive` to return `false`.

G.1.4. Changes in MySQL Connector/CPP 1.0.2 (19 December 2008 alpha)

Functionality added or changed:

- Implemented `getScale()`, `getPrecision()` and `getColumnDisplaySize()` for `MySQL_ResultSetMetaData` and `MySQL_Prepared_ResultSetMetaData`.
- Changed `ResultSetMetaData` methods `getColumnDisplaySize()`, `getPrecision()`, `getScale()` to return `unsigned int` instead of `signed int`.
- `DATE`, `DATETIME` and `TIME` are now being handled when calling the `MySQL_PreparedResultSet` methods `getString()`, `getDouble()`, `getInt()`, `getLong()`, `getBoolean()`.
- Reverted implementation of `MySQL_DatabaseMetaData::getTypeInfo()`. Now unimplemented. In addition, removed `cppconn/datatype.h` for now, until a more robust implementation of the types can be developed.
- Implemented `MySQL_PreparedStatement::setNull()`.
- Implemented `MySQL_PreparedStatement::clearParameters()`.
- Added PHP script `examples/cpp_trace_analyzer.php` to filter the output of the debug trace. Please see the inline comments for documentation. This script is unsupported.
- Implemented `MySQL_ResultSetMetaData::getPrecision()` and `MySQL_Prepared_ResultSetMetaData::getPrecision()`, updating example.
- Added new unit test framework for JDBC compliance and regression testing.
- Added `test/unit` as a basis for general unit tests using the new test framework, see `test/unit/example` for basic usage examples.

Bugs fixed:

- Fixed `MySQL_PreparedStatementResultSet::getDouble()` to return the correct value when the underlying type is `MYSQL_TYPE_FLOAT`.
- Fixed bug in `MySQL_ConnectionMetaData::getIndexInfo()`. The method did not work because the schema name wasn't included in the query sent to the server.
- Fixed a bug in `MySQL_ConnectionMetaData::getColumns()` which was performing a cartesian product of the columns in the table times the columns matching `columnNamePattern`. The example `example/connection_meta_schemaobj.cpp` was extended to cover the function.
- Fixed bugs in `MySQL_DatabaseMetaData`. All `supportsCatalogXXXXX` methods were incorrectly returning `true` and all `supportsSchemaXXXX` methods were incorrectly returning `false`. Now `supportsCatalogXXXXX` returns `false` and `supportsSchemaXXXXX` returns `true`.

- Fixed bugs in the `MySQL_PreparedStatements` methods `setBigInt()` and `setDatetime()`. They decremented the internal column index before forwarding the request. This resulted in a double-decrement and therefore the wrong internal column index. The error message generated was:

  ```
  setString() ... invalid "parameterIndex"
  ```

- Fixed a bug in `getString()`. `getString()` is now binary safe. A new example was also added.
- Fixed bug in `FLOAT` handling.
- Fixed `MySQL_PreparedStatement::setBlob()`. In the tests there is a simple example of a class implementing `sql::Blob`.

G.1.5. Changes in MySQL Connector/CPP 1.0.1 (01 December 2008 alpha)

Functionality added or changed:

- `sql::mysql::MySQL_SQLException` was removed. The distinction between server and client (connector) errors, based on the type of the exception, has been removed. However, the error code can still be checked in order to evaluate the error type.
- Support for (n)make install was added. You can change the default installation path. Carefully read the messages displayed after executing cmake. The following are installed:
  - Static and the dynamic version of the library, `libmysqlcppconn`.
  - Generic interface, `cppconn`.
  - Two MySQL specific headers:

    `mysql_driver.h`, use this if you want to get your connections from the driver instead of instantiating a `MySQL_Connection` object. This makes your code portable when using the common interface.

    `mysql_connection.h`, use this if you intend to link directly to the `MySQL_Connection` class and use its specifics not found in `sql::Connection`.

    However, you can make your application fully abstract by using the generic interface rather than these two headers.
- Driver Manager was removed.
- Added `ConnectionMetaData::getSchemas()` and `Connection::setSchema()`.
- `ConnectionMetaData::getCatalogTerm()` returns not applicable, there is no counterpart to catalog in MySQL Connector/C++.
- Added experimental GCov support, `cmake -DMYSQLCPPCONN_GCOV_ENABLE:BOOL=1`
- All examples can be given optional connection parameters on the command line, for example:

  ```
  examples/connect tcp://host:port user pass database
  ```

  or

  ```
  examples/connect unix:///path/to/mysql.sock user pass database
  ```

- Renamed `ConnectionMetaData::getTables: TABLE_COMMENT` to `REMARKS`.
- Renamed `ConnectionMetaData::getProcedures: PROCEDURE_SCHEMA` to `PROCEDURE_SCHEM`.
- Renamed `ConnectionMetaData::getPrimaryKeys(): COLUMN` to `COLUMN_NAME`, `SEQUENCE` to `KEY_SEQ`, and `INDEX_NAME` to `PK_NAME`.
- Renamed `ConnectionMetaData::getImportedKeys(): PKTABLE_CATALOG` to `PKTABLE_CAT`, `PKTABLE_SCHEMA` to `PKTABLE_SCHEM`, `FKTABLE_CATALOG` to `FKTABLE_CAT`, `FKTABLE_SCHEMA` to `FKTABLE_SCHEM`.
- Changed metadata column name `TABLE_CATALOG` to `TABLE_CAT` and `TABLE_SCHEMA` to `TABLE_SCHEM` to ensure JDBC compliance.
- Introduced experimental CPack support, see make help.

- All tests changed to create TAP compliant output.
- Renamed `sql::DbcMethodNotImplemented` to `sql::MethodNotImplementedException`
- Renamed `sql::DbcInvalidArgument` to `sql::InvalidArgumentException`
- Changed `sql::DbcException` to implement the interface of JDBC's `SQLException`. Renamed to `sql::SQLException`.
- Converted Connector/J tests added.
- MySQL Workbench 5.1 changed to use MySQL Connector/C++ for its database connectivity.
- New directory layout.