

MySQL Connector/NET

MySQL Connector/NET

Abstract

This manual describes MySQL Connector/NET.

Document generated on: 2009-06-02 (revision: 15165)

Copyright © 1997-2008 MySQL AB, 2009 Sun Microsystems, Inc. All rights reserved. U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements. Use is subject to license terms. Sun, Sun Microsystems, the Sun logo, Java, Solaris, StarOffice, MySQL Enterprise Monitor 2.0, MySQL logo™ and MySQL™ are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Copyright © 1997-2008 MySQL AB, 2009 Sun Microsystems, Inc. Tous droits réservés. L'utilisation est soumise aux termes du contrat de licence. Sun, Sun Microsystems, le logo Sun, Java, Solaris, StarOffice, MySQL Enterprise Monitor 2.0, MySQL logo™ et MySQL™ sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

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

For more information on the terms of this license, for details on how the MySQL documentation is built and produced, or if you are interested in doing a translation, please contact the [Documentation Team](#).

For additional licensing information, including licenses for libraries used by MySQL, see [Preface, Notes, Licenses](#).

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](#).

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 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 [Chapter 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 4.5, "Connector/NET Connection String Options Reference"](#).

Chapter 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 2.3, “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.

Chapter 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>.

2.1. 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.

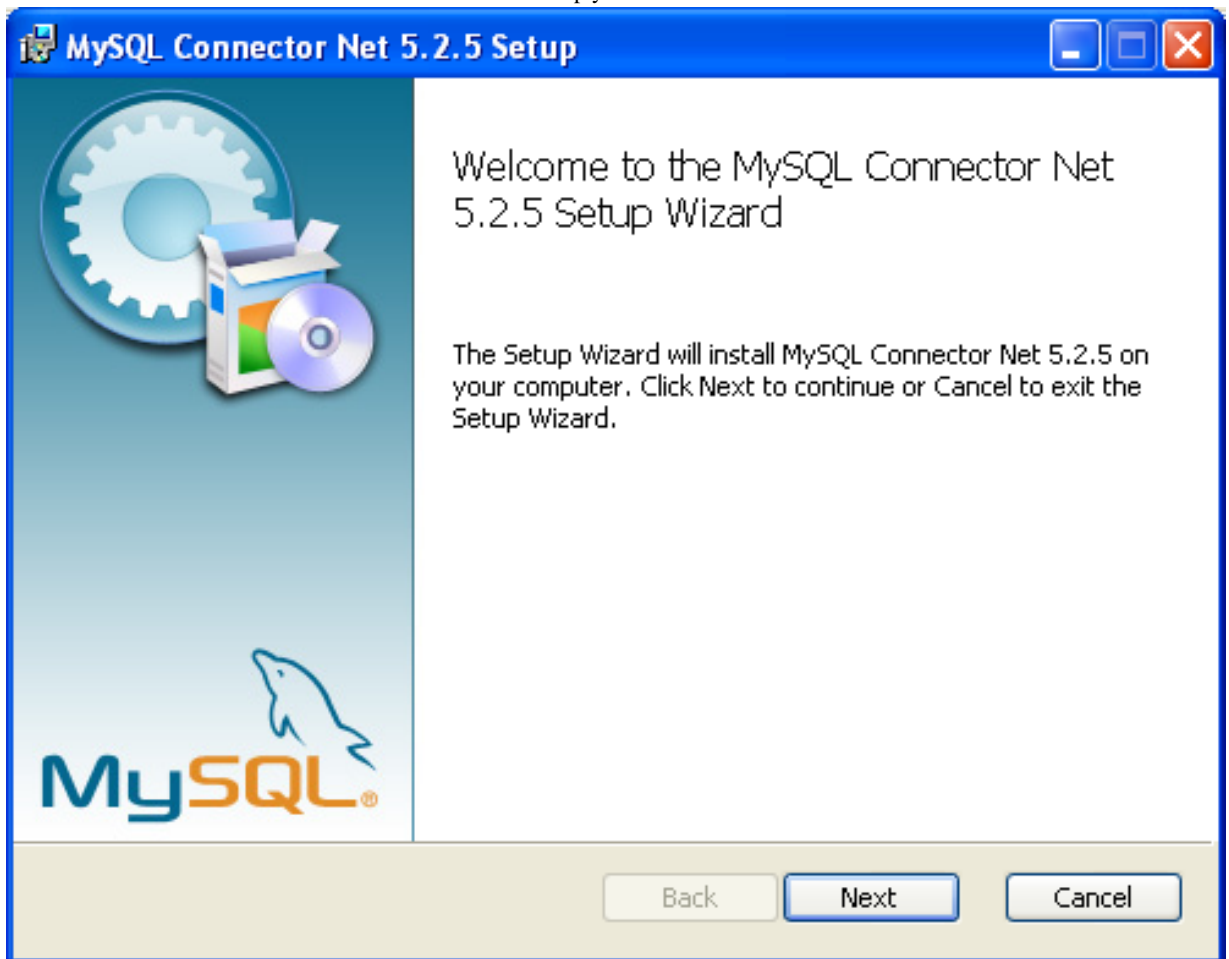
2.1.1. Installing Connector/NET using the Installer

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.

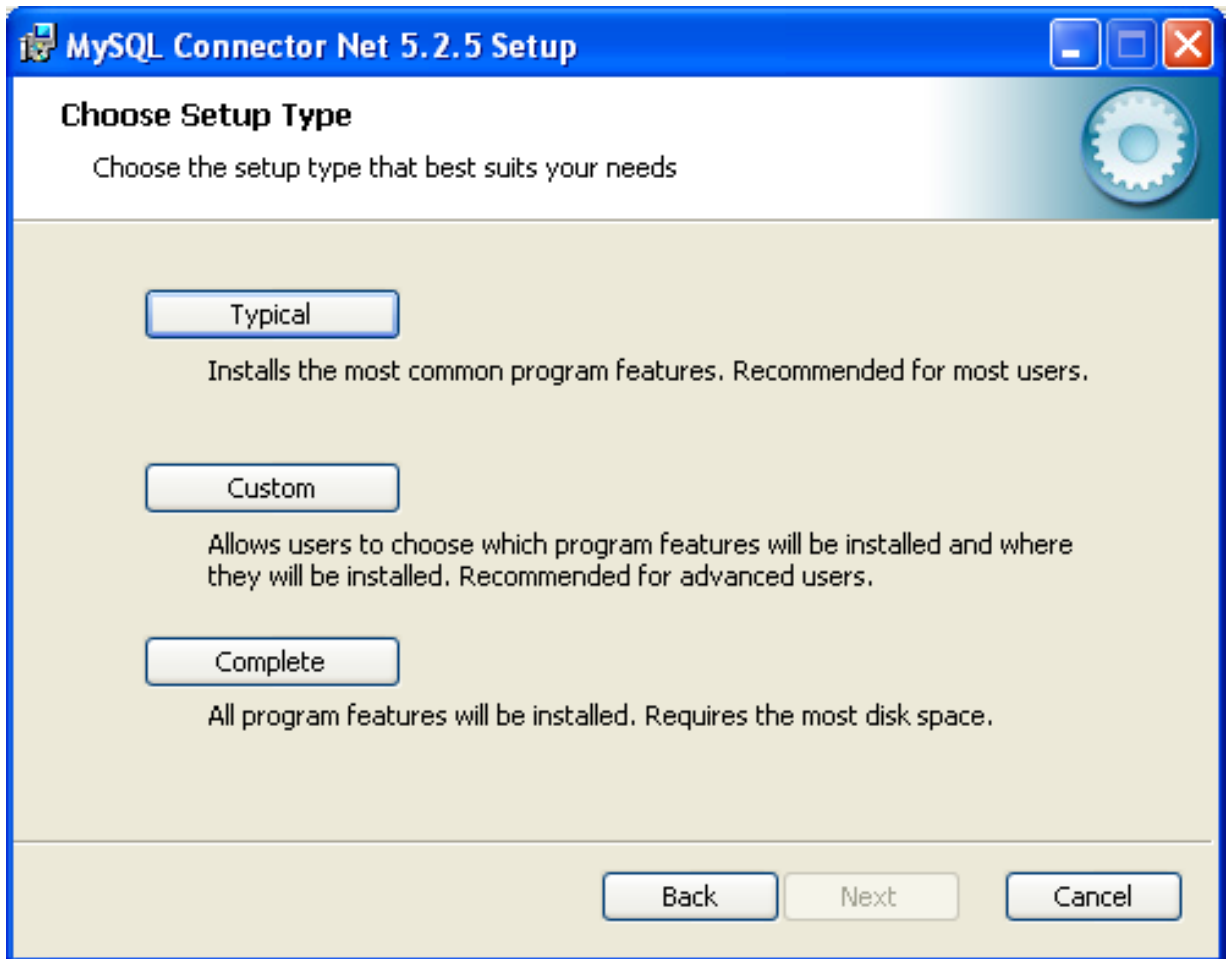
Connector/NET is installed through the use of a Windows Installer (.msi) installation package, which can be used to install Connector/NET on all Windows operating systems. The MSI package is contained within a ZIP archive named `mysql-connector-net-version.zip`, where `version` indicates the Connector/NET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



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



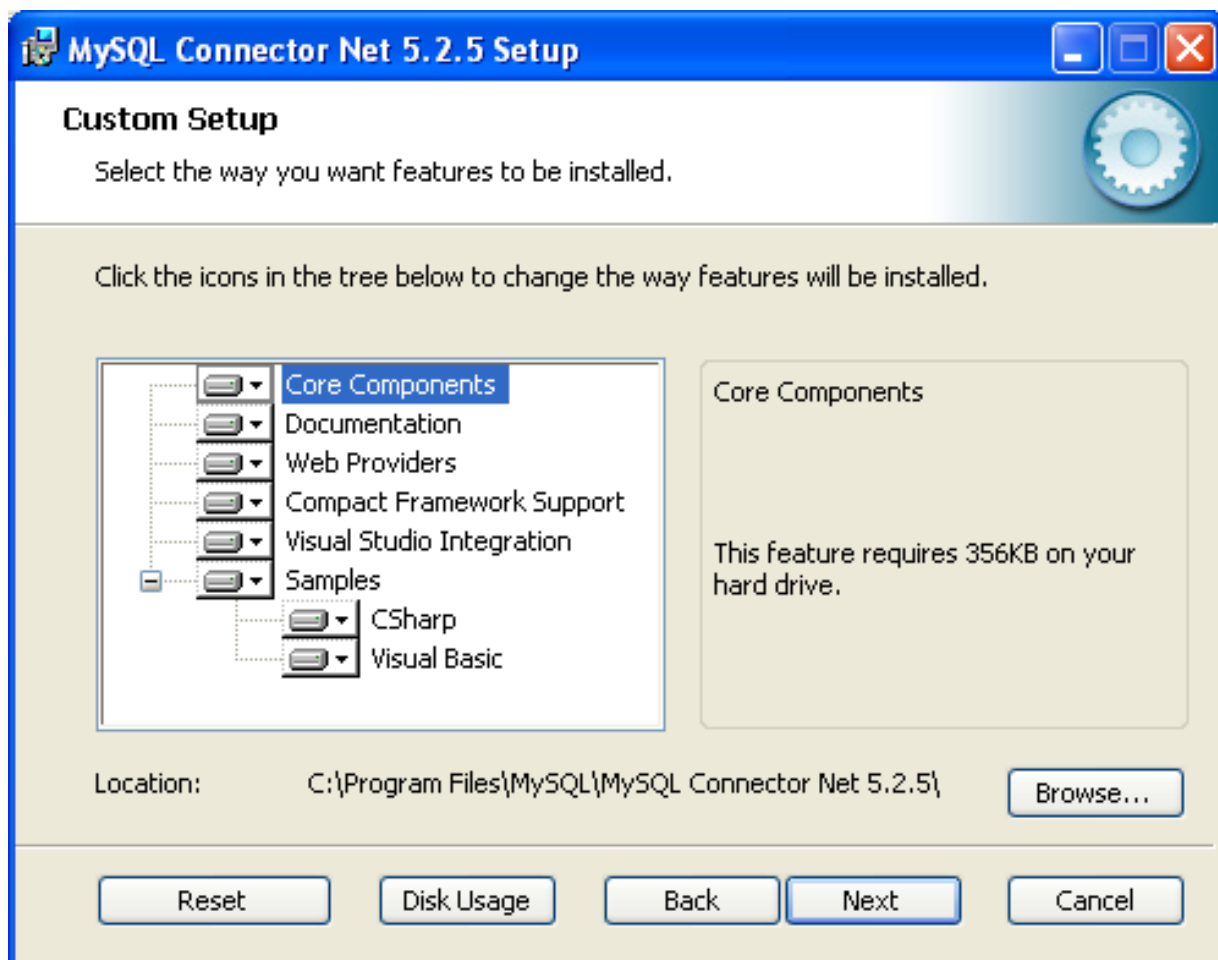
For most situations, the Typical installation will be suitable. Click the TYPICAL button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the COMPLETE button and proceed to step 5. If you want to customize your installation, including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

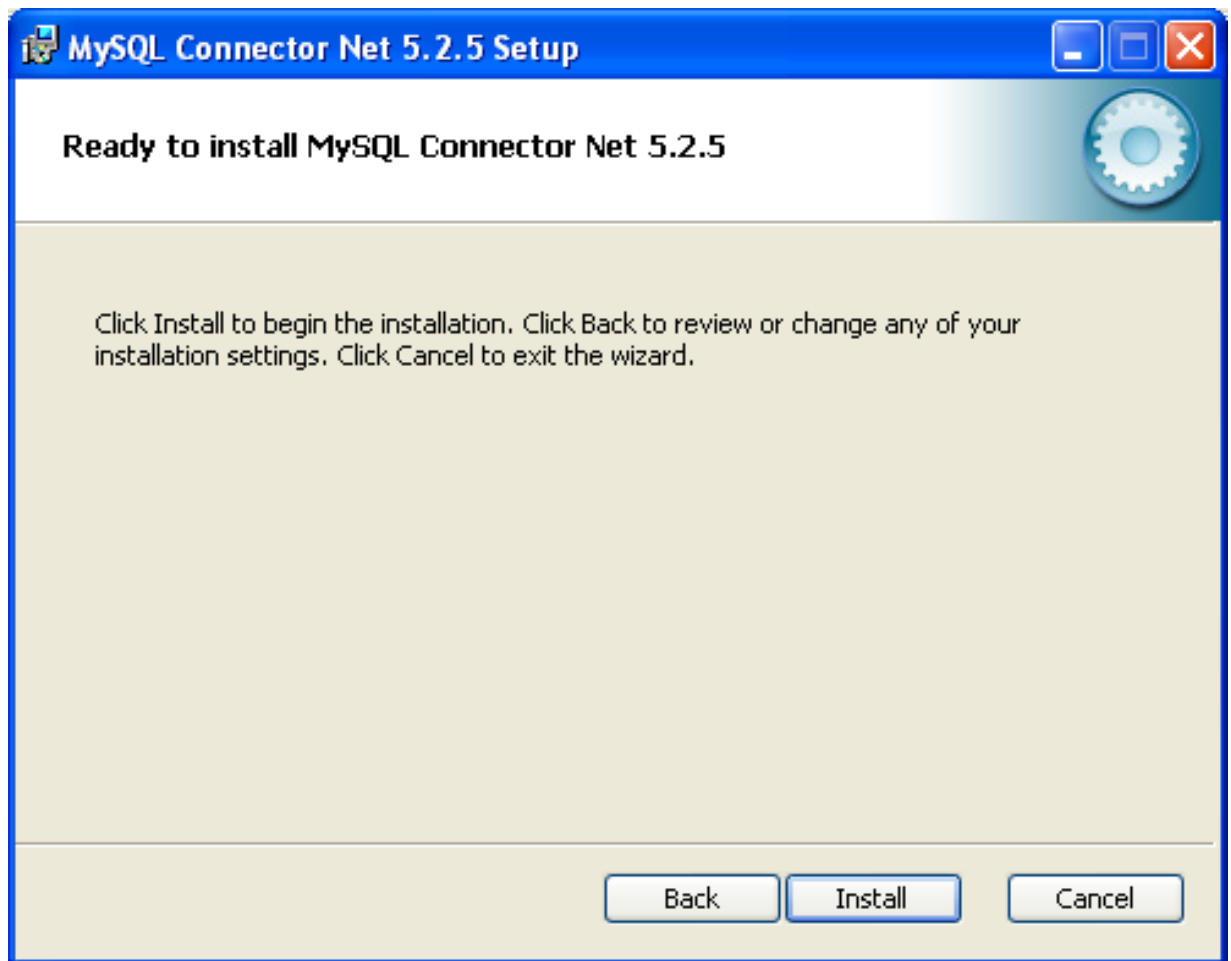
3. If you have chosen a custom installation, you can select the individual components that you want to install, including the core interface component, supporting documentation (a CHM file) samples and examples and the source code. Select the items, and their installation level, and then click NEXT to continue the installation.

Note

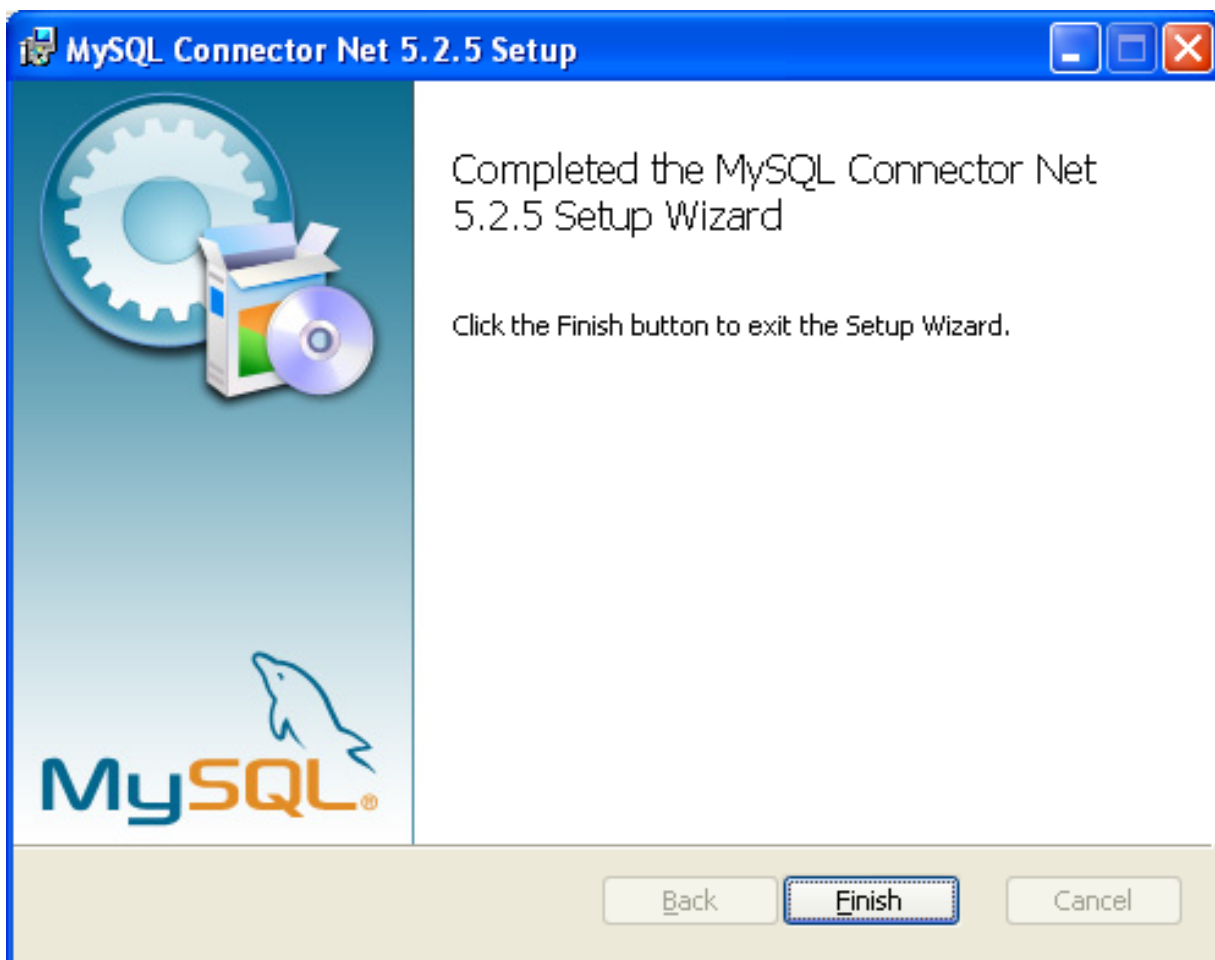
For Connector/NET 1.0.8 or lower and Connector 5.0.4 and lower the installer will attempt to install binaries for both 1.x and 2.x of the .NET Framework. If you only have one version of the framework installed, the connector installation may fail. If this happens, you can choose the framework version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



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



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



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.

2.1.2. 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](#) - This 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. 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 accesses 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
```

2.3. 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.

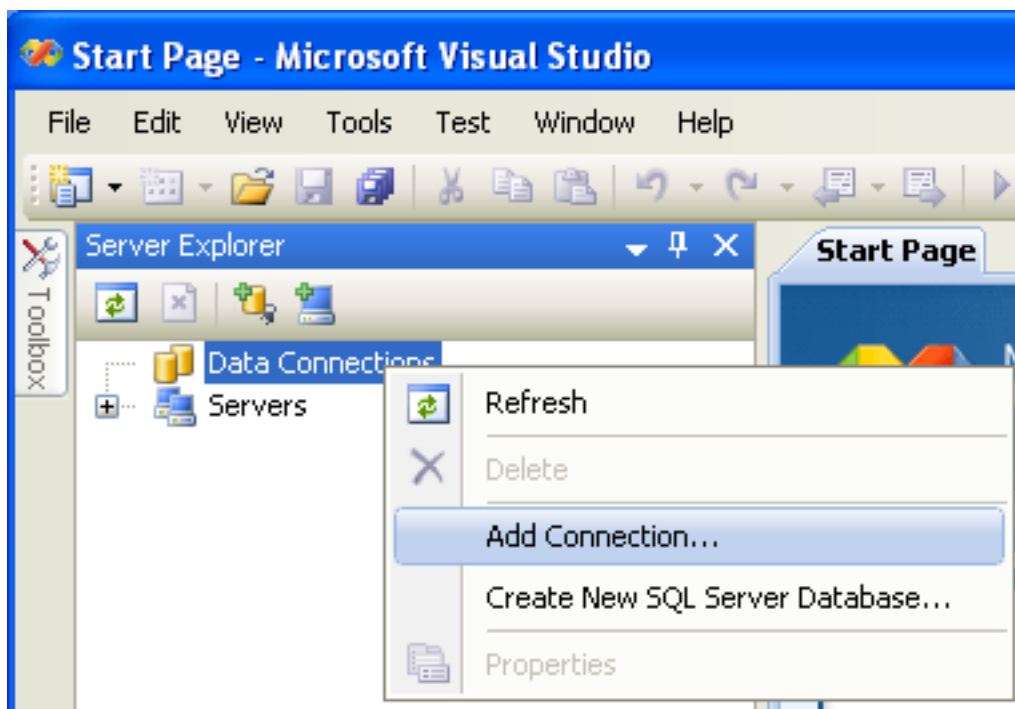
Chapter 3. Visual Studio User Guide

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:

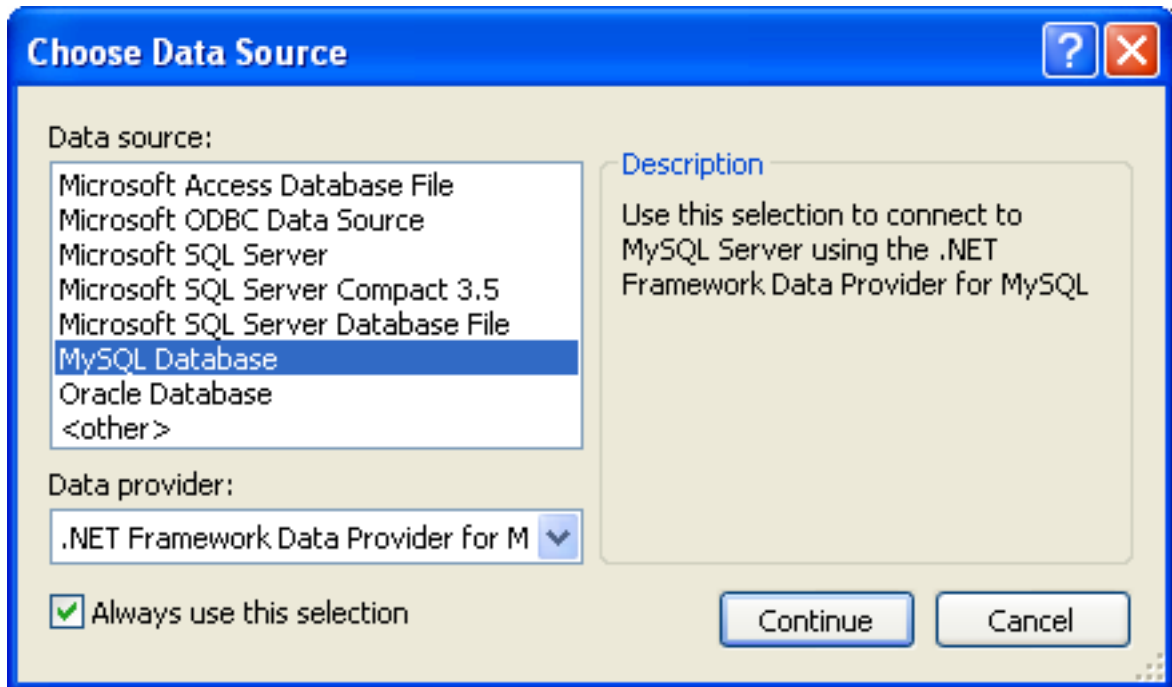
- Start Visual Studio, and open the Server Explorer window (VIEW, SERVER EXPLORER option in the main Visual Studio menu, or **Ctrl+W, L** hot keys).
- Right-click on the Data Connections node, and choose the ADD CONNECTION... menu item.
- Add Connection dialog opens. Press the **C**HANGE button to choose MySQL Database as a data source.

Figure 3.1. Add Connection Context Menu



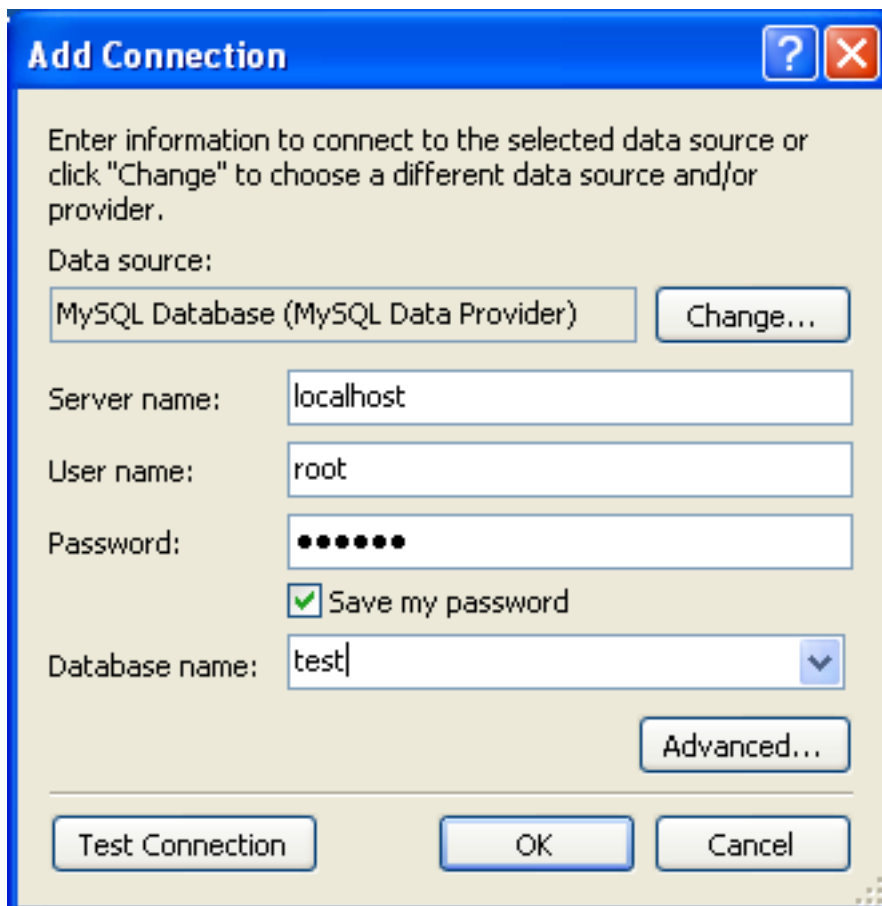
- Change Data Source dialog opens. Choose MySQL Database in the list of data sources (or the <other> option, if MySQL Database is absent), and then choose .NET Framework Data Provider for MySQL in the combo box of data providers.

Figure 3.2. Choose Data Source



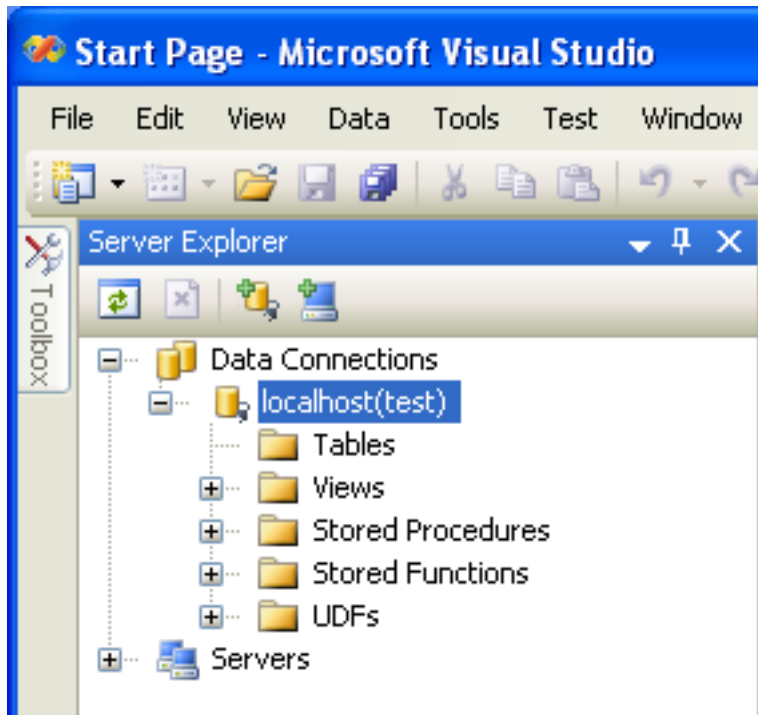
- Input the connection settings: the server host name (for example, localhost if the MySQL server is installed on the local machine), the user name, the password, and the default schema name. Note that you must specify the default schema name to open the connection.

Figure 3.3. Add Connection Dialog



- 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 3.4. 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 lose your data.

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 3.5. Editing New Table

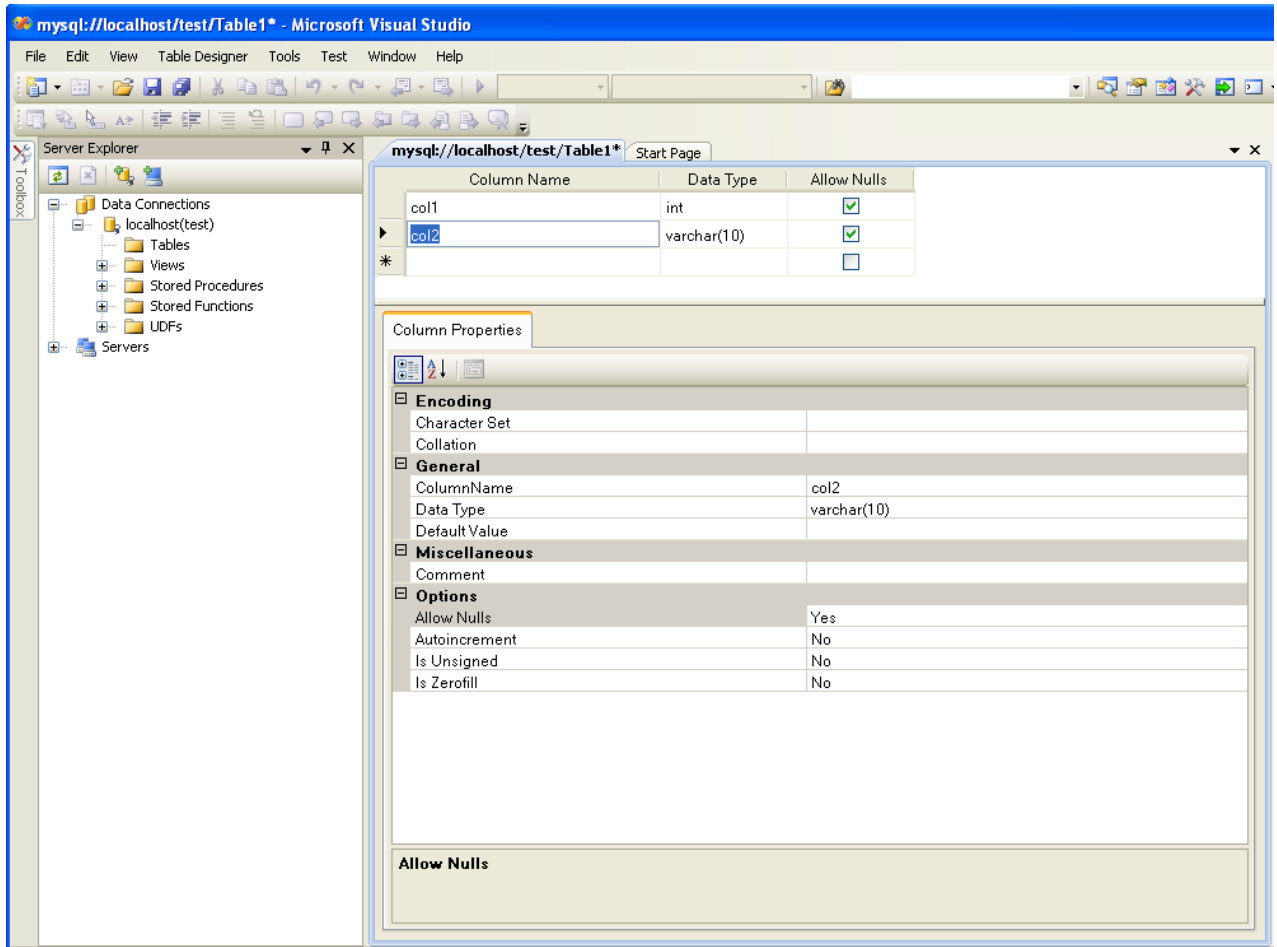


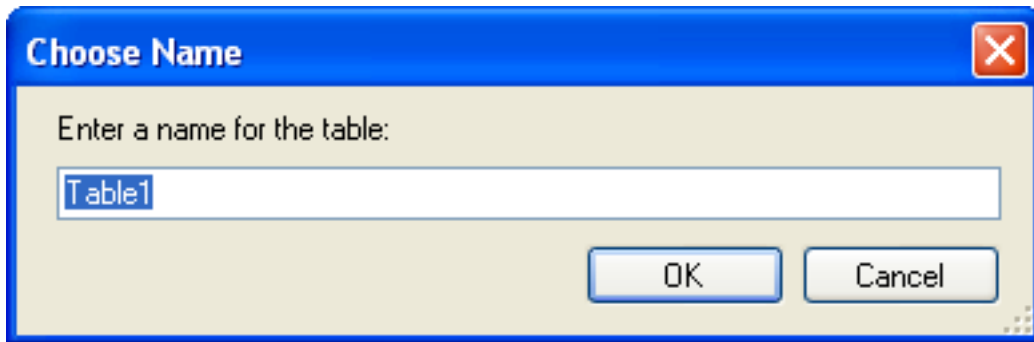
Table Designer consists of the following parts:

- 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.

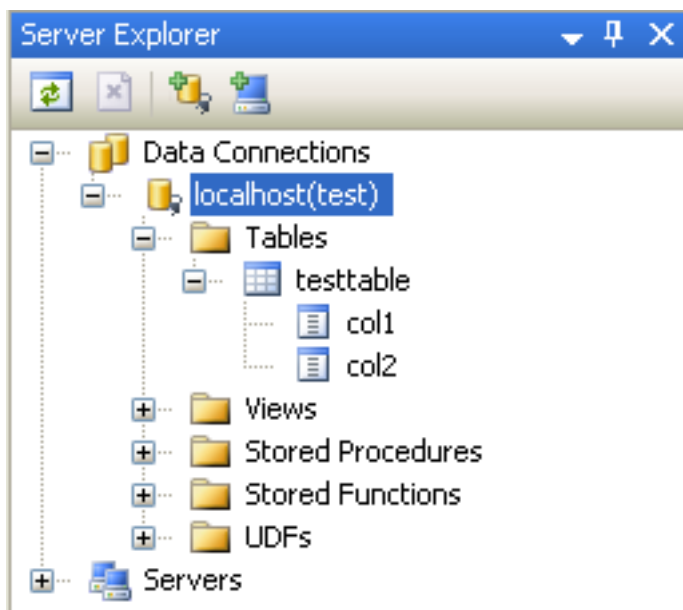
To save changes you have made in the Table Designer, use either **SAVE** or **SAVE ALL** button of the Visual Studio main toolbar, or just press **Ctrl+S**. If you have not already named the table you will be prompted to do so.

Figure 3.6. Choose Table Name



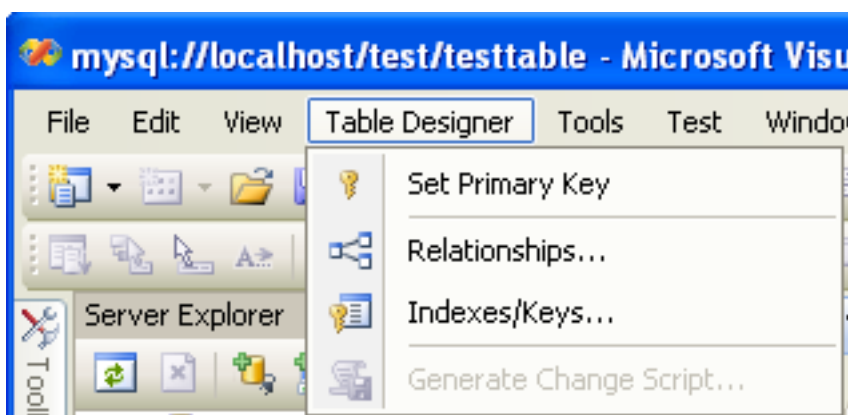
Once created you can view the table in the Server Explorer.

Figure 3.7. 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 3.8. Table Designer Main Menu



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.

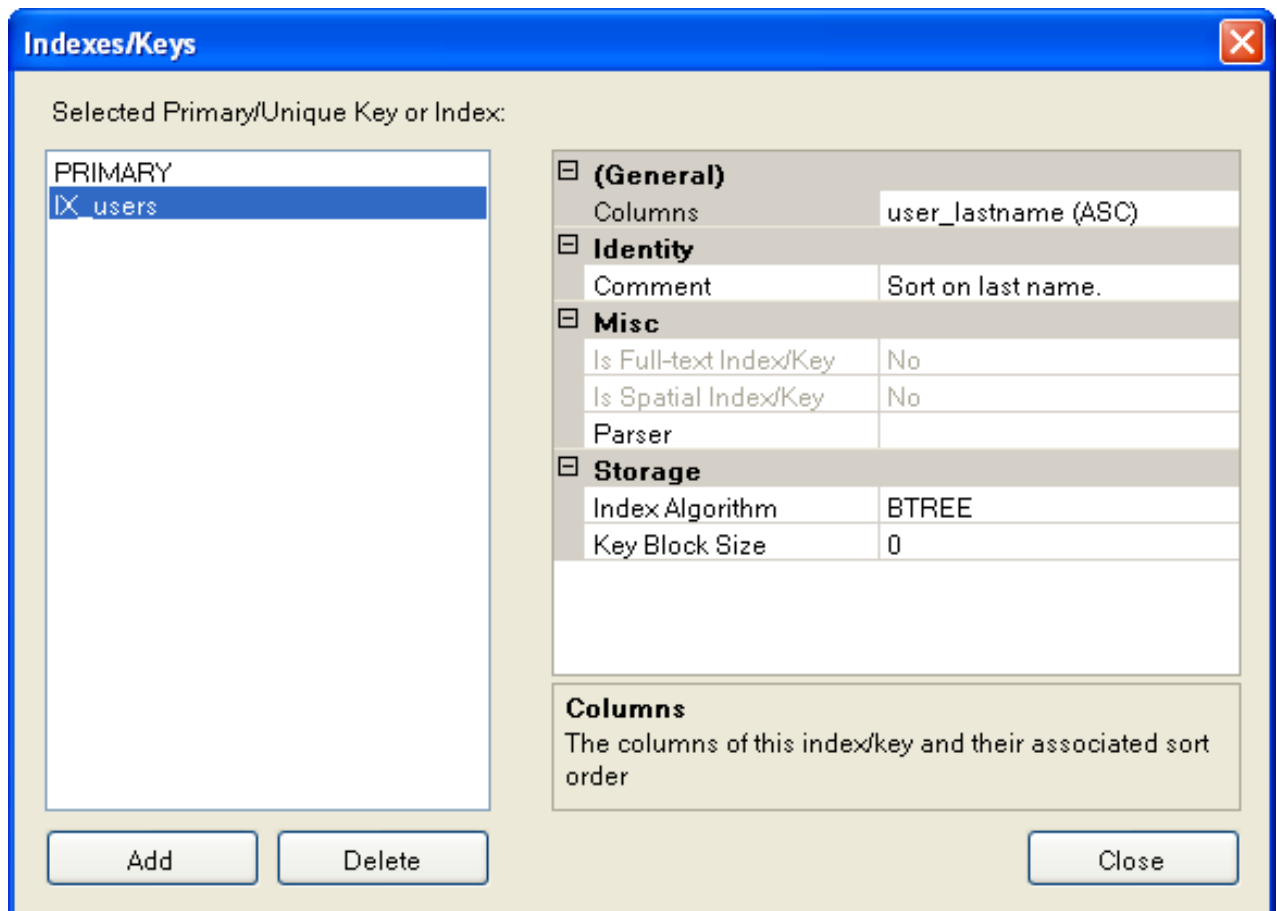
To delete a column, select it by clicking on the selector column on the left of the column grid, then press the **Delete** button on a keyboard.

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 3.9. Indexes Dialog



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.

3.2.3. Editing Foreign Keys

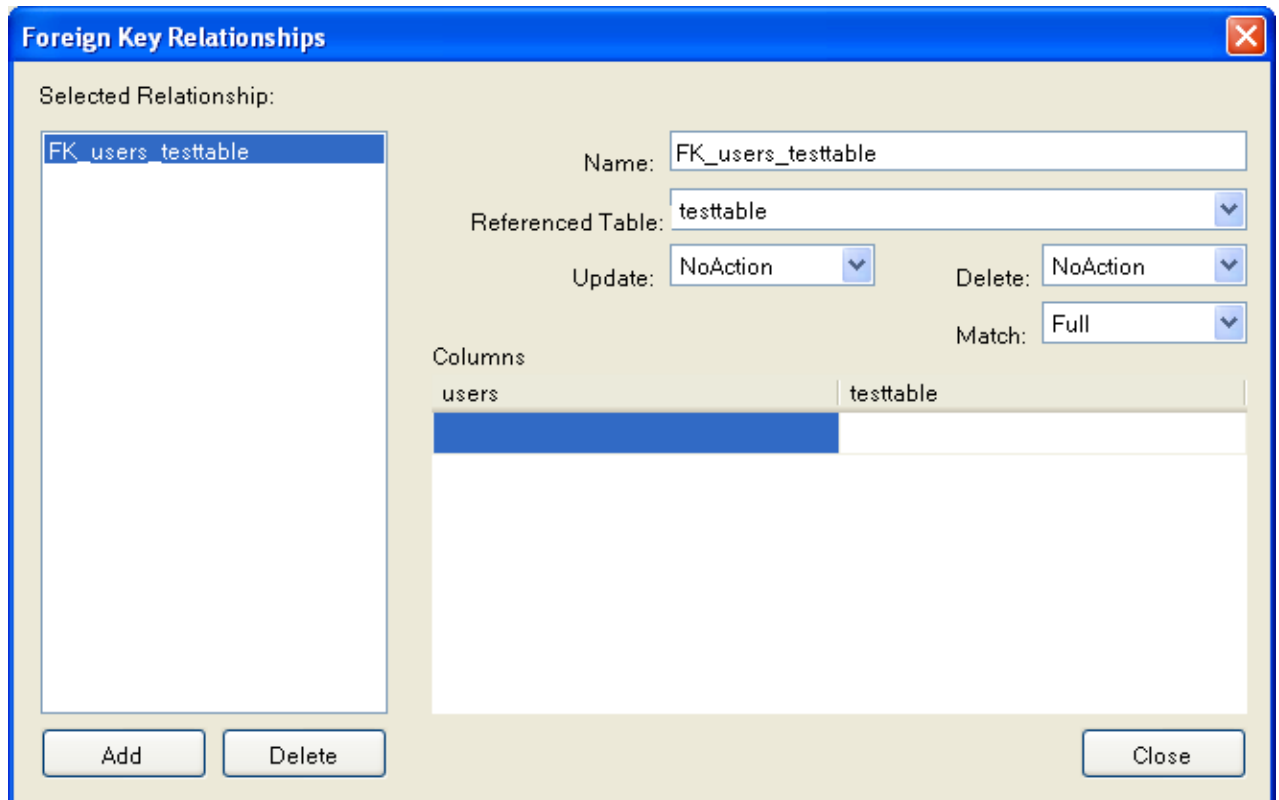
Foreign Keys management is performed via the **FOREIGN KEY RELATIONSHIPS** dialog.

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 3.10. Foreign Key Relationships Dialog



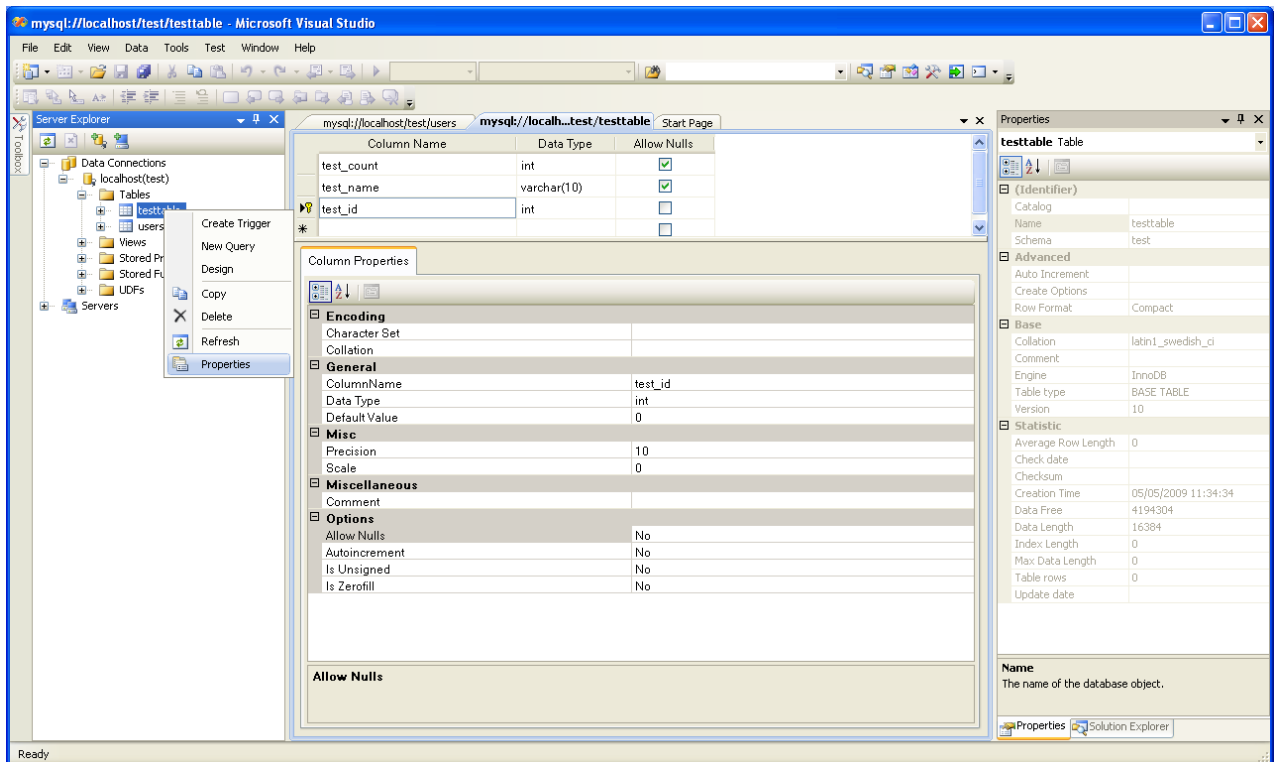
3.2.4. Column Properties

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.

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 3.11. Table Properties Menu Item

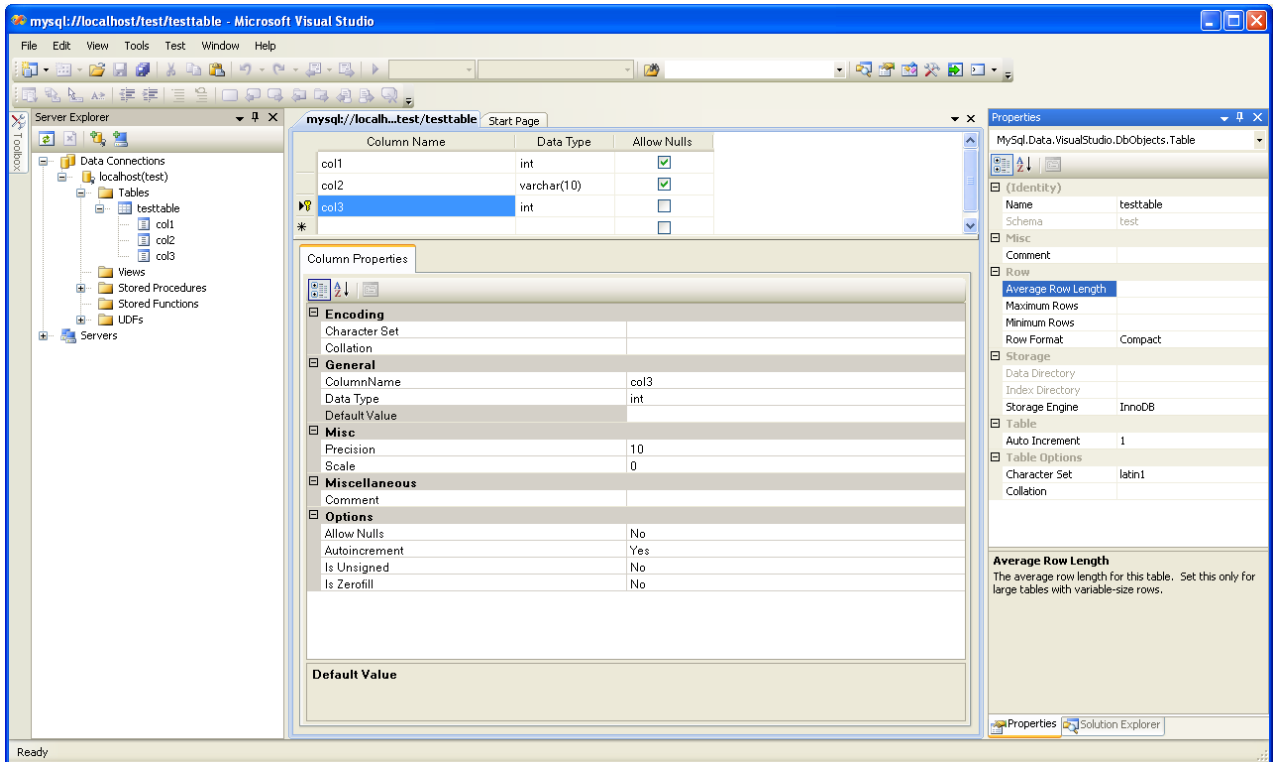


The following table properties can be set:

- 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.

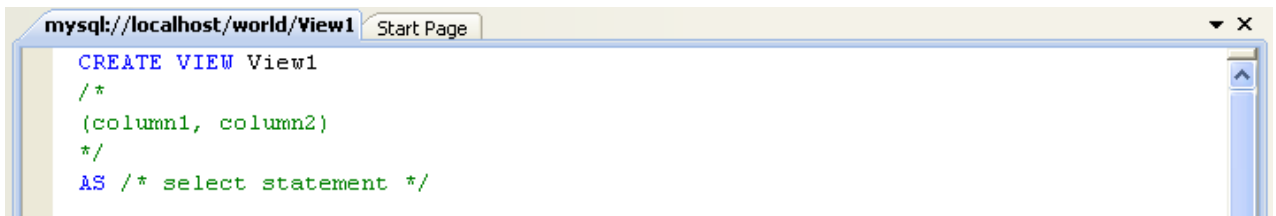
Figure 3.12. Table Properties



3.3. Editing Views

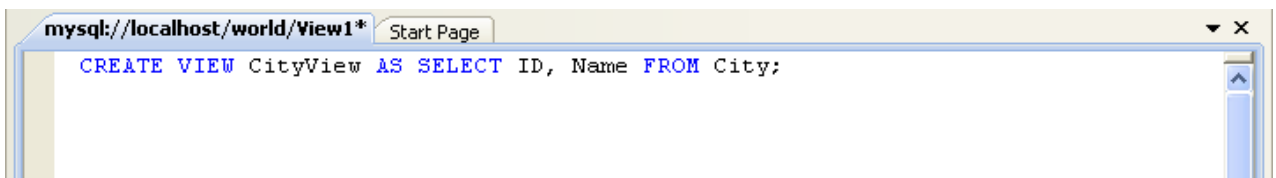
To create a new view, right click the Views node under the connection node in Server Explorer. From the node's context menu, choose the [CREATE VIEW](#) command. This command opens the SQL Editor.

Figure 3.13. Editing View SQL



You can then enter the SQL for your view.

Figure 3.14. View SQL Added



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

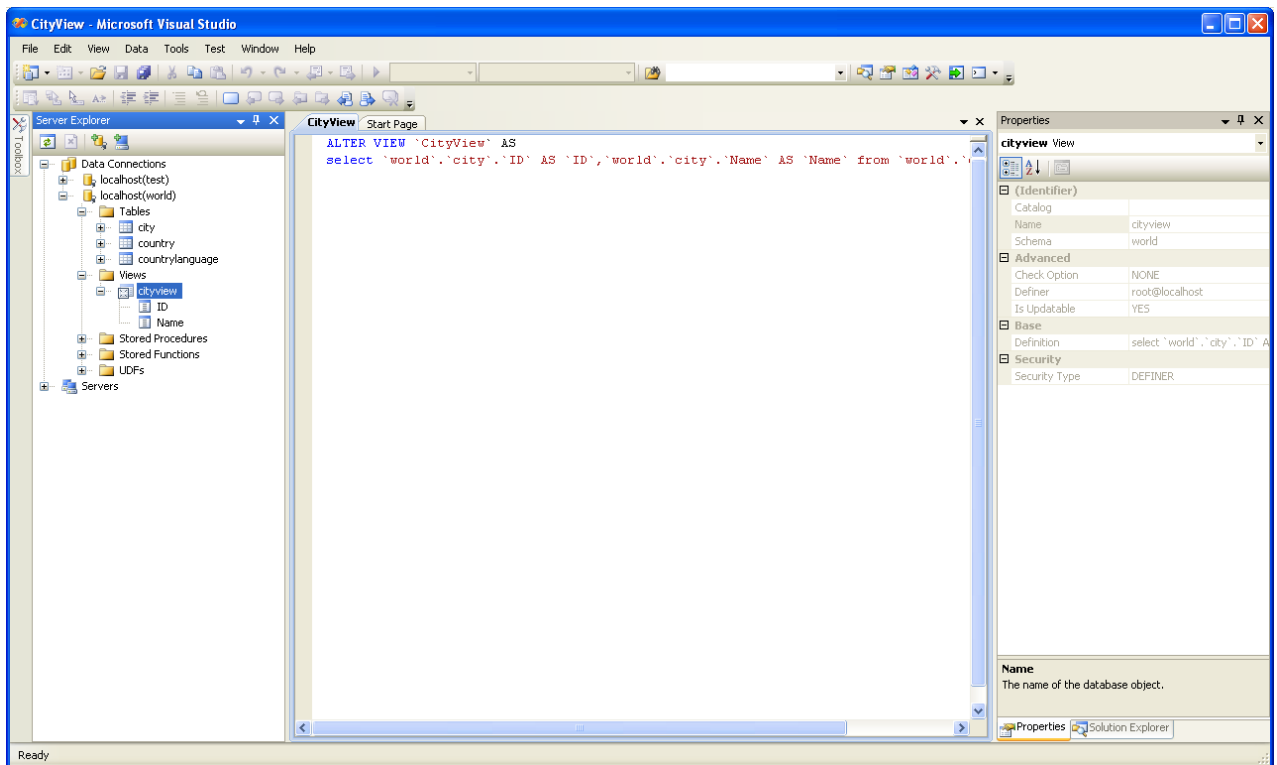
- Definer
- Definition
- Definer
- Is Updatable
- Name
- Schema
- Security Type

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`.

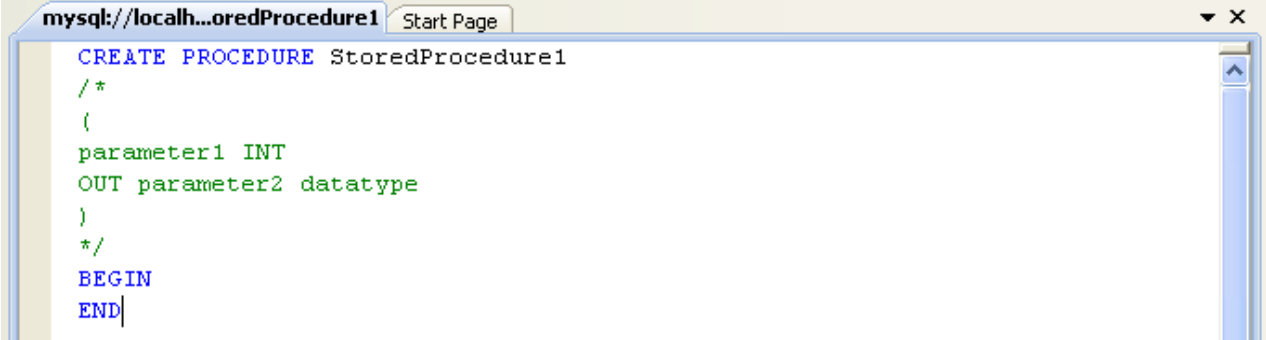
Figure 3.15. View SQL Saved



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 3.16. 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:

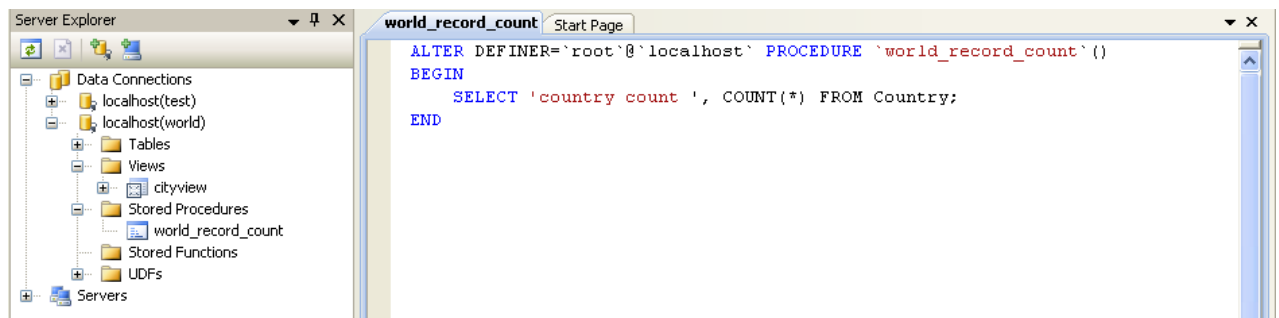
- Body
- Catalog
- Comment
- Creation Time
- Data Access
- Definer
- Definition
- External Name
- External Language
- Is Deterministic
- Last Modified
- Name
- Parameter Style
- Returns
- Schema
- Security Type
- Specific Name
- SQL Mode
- SQL Path
- Type

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 3.17. Stored Procedure SQL Saved



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.

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.

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.

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.

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](#).

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 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.

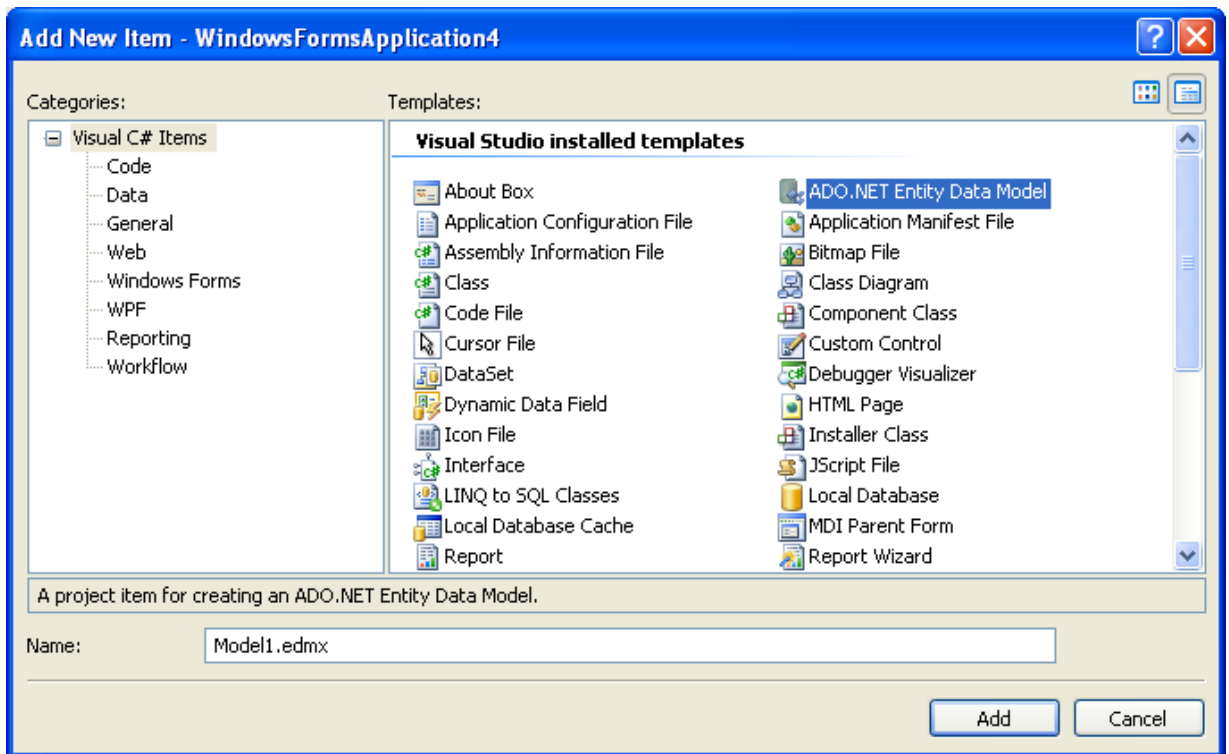
2. Choose the **WINDOWS FORMS APPLICATION** installed template. Click OK. The solution is created.

Adding an Entity Data Model

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

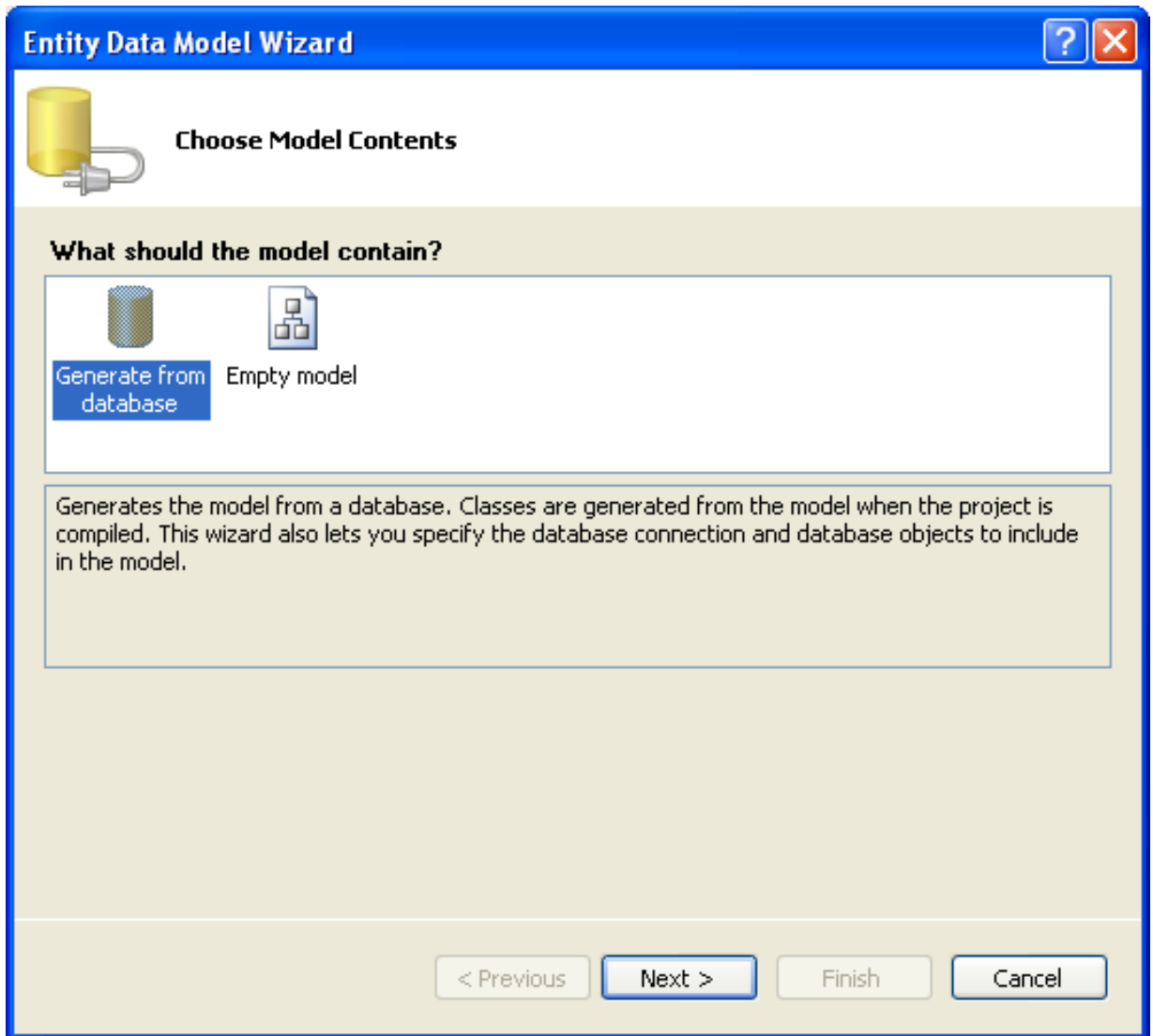
1. In the Solution Explorer, right click on your application and select **ADD, NEW ITEM...**. From **VISUAL STUDIO INSTALLED TEMPLATES** select **ADO.NET ENTITY DATA MODEL**. Click ADD.

Figure 3.18. Add Entity Data Model



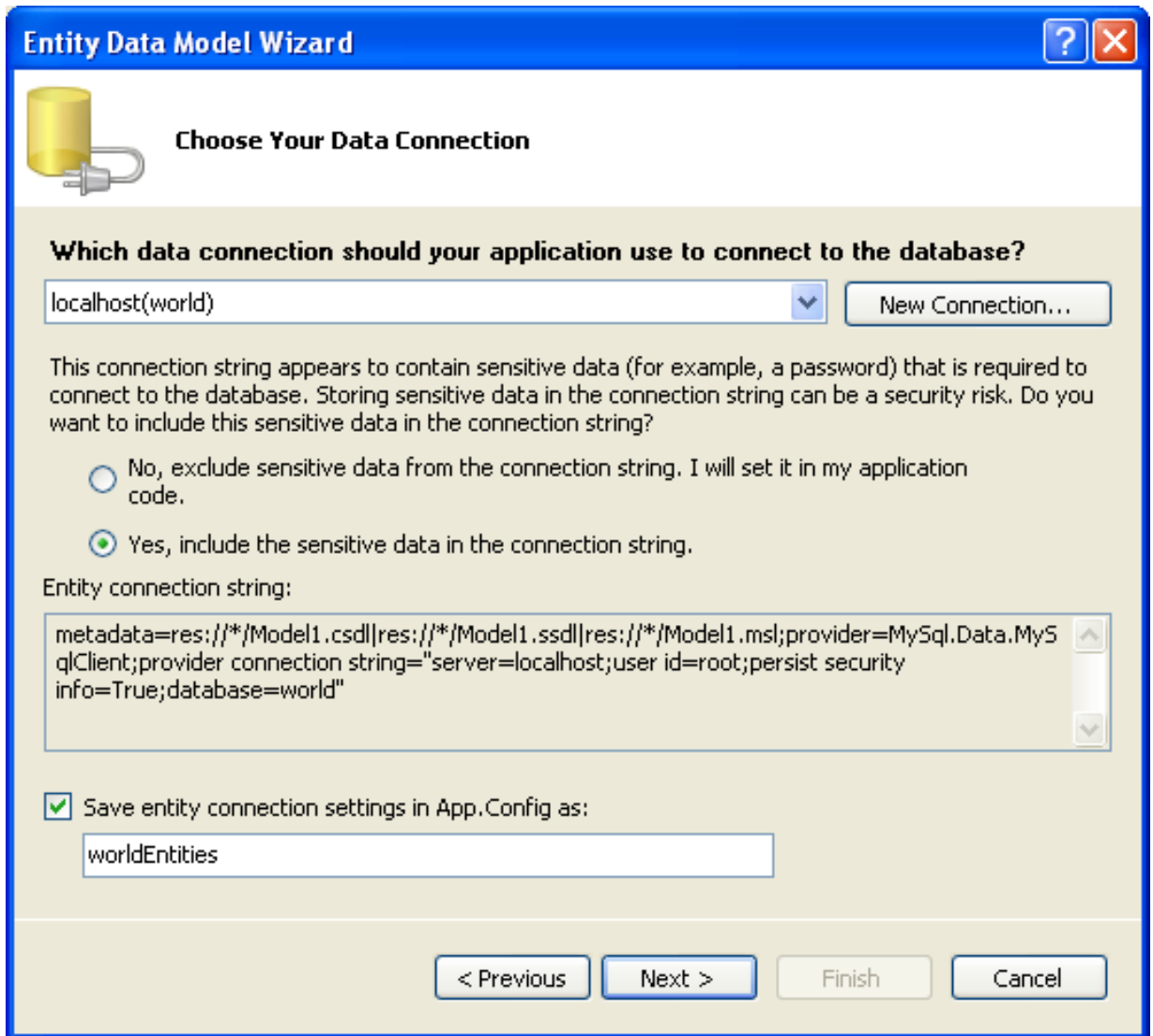
2. You will now see the Entity Data Model Wizard. You will use the wizard to generate the Entity Data Model from the world example database. Select the icon **GENERATE FROM DATABASE**. Click NEXT.

Figure 3.19. Entity Data Model Wizard Screen 1



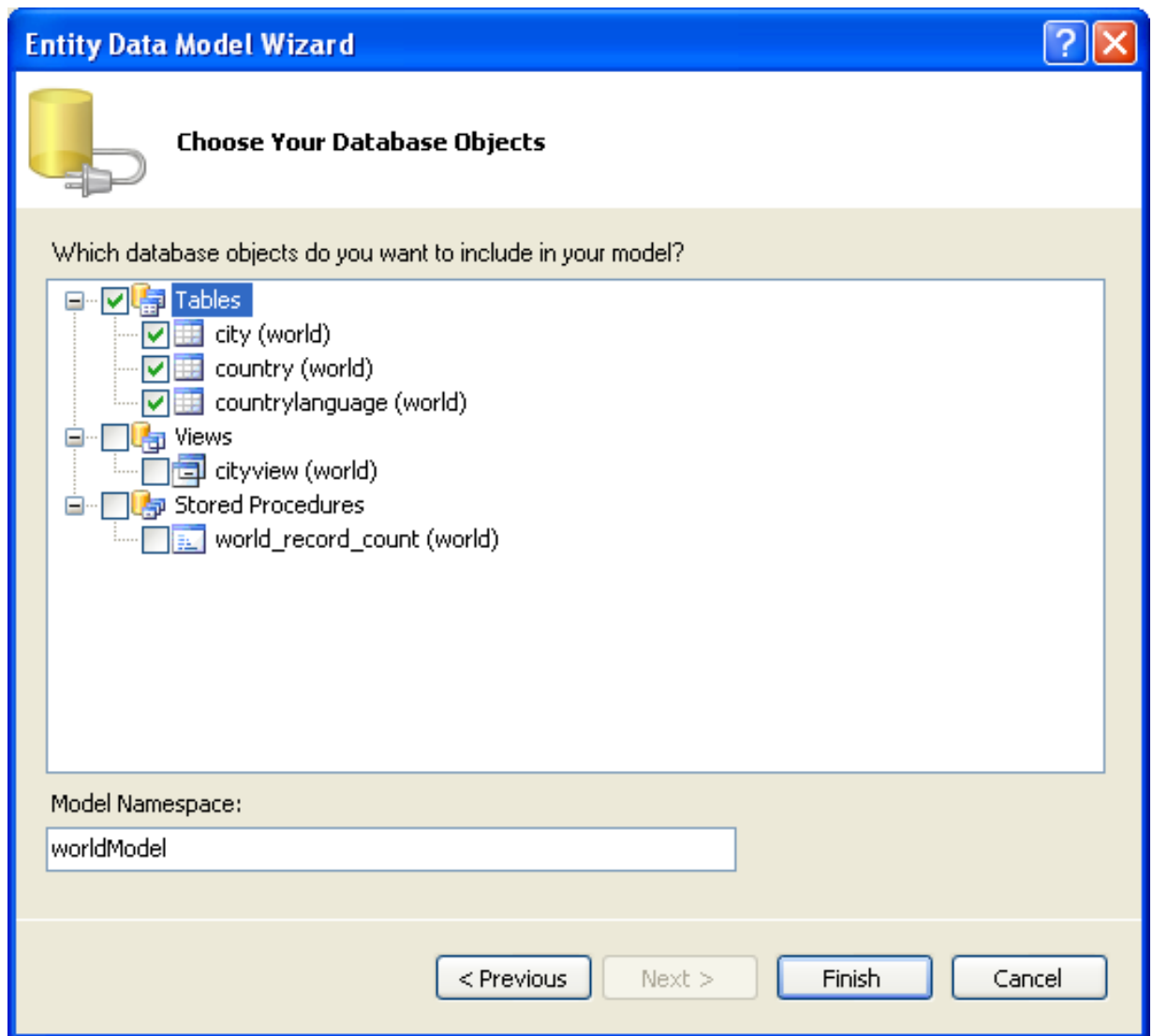
3. You can now select the connection you made earlier to the World database. If you have not already done so, you can create the new connection at this time by clicking on NEW CONNECTION.... For further instructions on creating a connection to a database see [Section 3.1, "Making a connection"](#).

Figure 3.20. Entity Data Model Wizard Screen 2



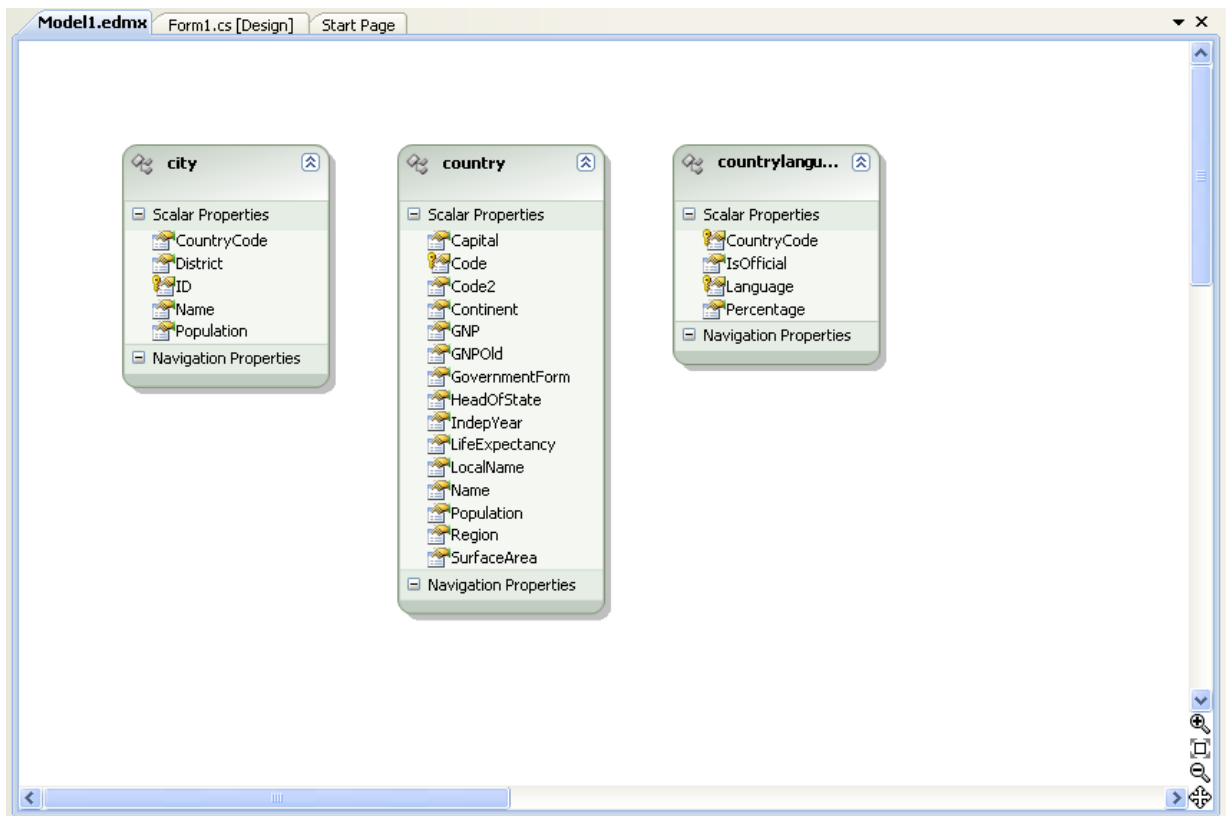
4. Make a note of the entity connection settings to be used in App.Config, as these will be used later to write the necessary control code.
5. Click NEXT.
6. The Entity Data Model Wizard connects to the database. You are then presented with a tree structure of the database. From this you can select the 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 3.21. Entity Data Model Wizard Screen 3



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

Figure 3.22. Entity Data Model Diagram



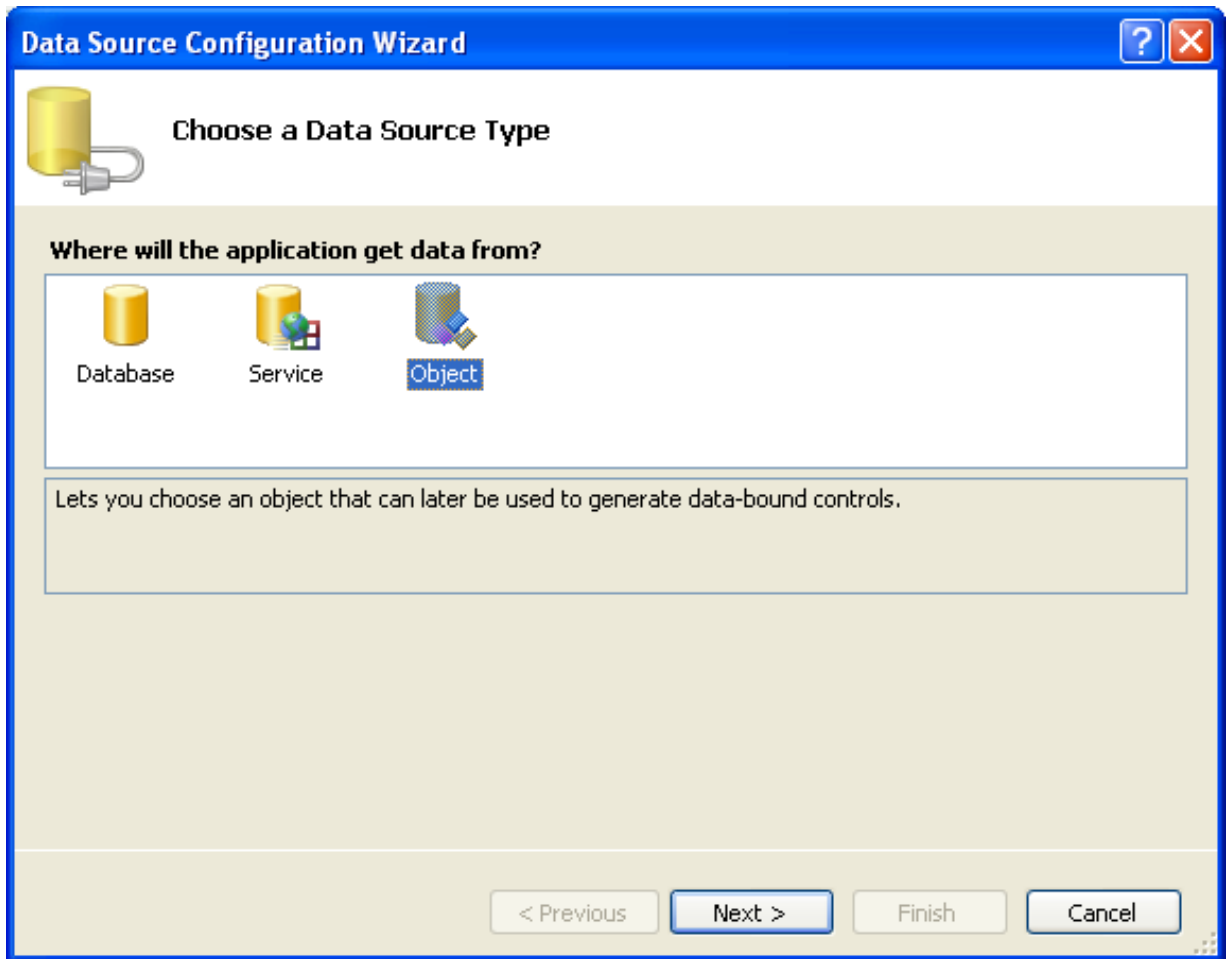
8. From the Visual Studio main menu select **BUILD, BUILD SOLUTION**, to ensure that everything compiles correctly so far.

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.

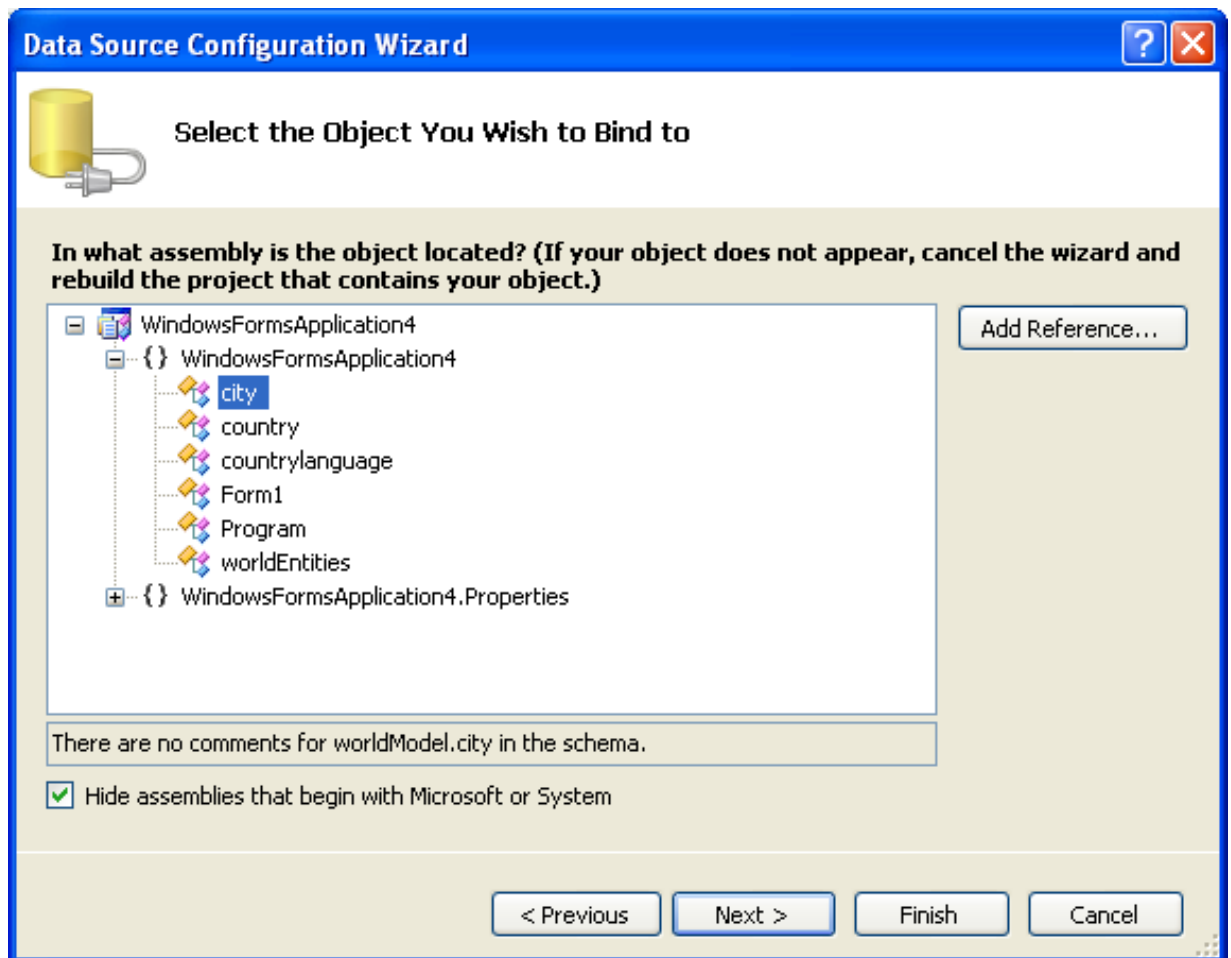
1. From the Visual Studio main menu select **DATA, ADD NEW DATA SOURCE...**. You will be presented with the Data Source Configuration Wizard.

Figure 3.23. Entity Data Source Configuration Wizard Screen 1



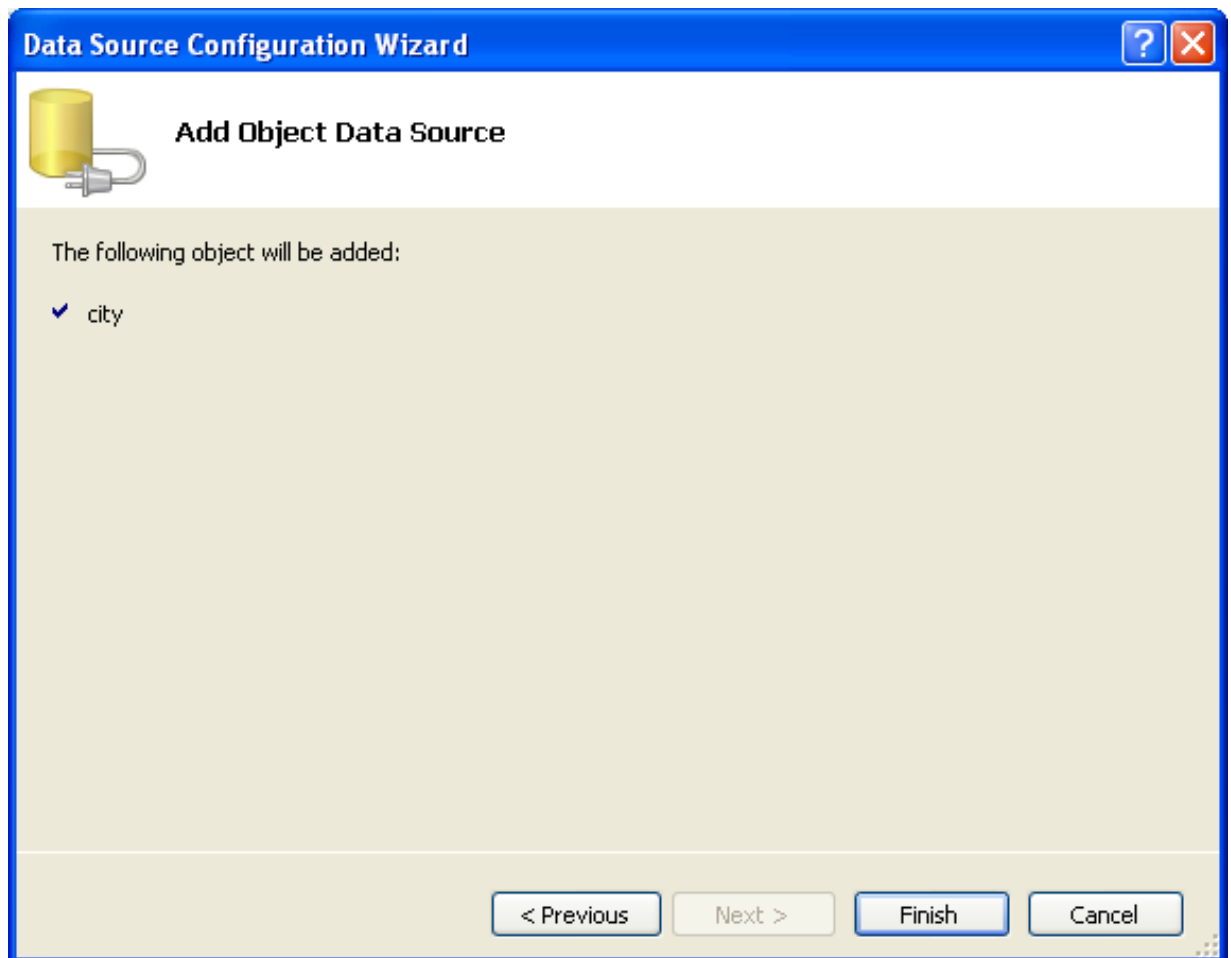
2. Select the **OBJECT** icon. Click NEXT.
3. You will now select the Object you wish to bind to. Expand the tree. In this tutorial you will select the city table. Once the city table has been selected click NEXT.

Figure 3.24. Entity Data Source Configuration Wizard Screen 2



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

Figure 3.25. Entity Data Source Configuration Wizard Screen 3



5. The city object will be display in the Data Sources panel. If the Data Sources panel is not displayed, select **DATA**, **SHOW DATA SOURCES** from the Visual Studio main menu. The docked panel will then be displayed.

Figure 3.26. Data Sources

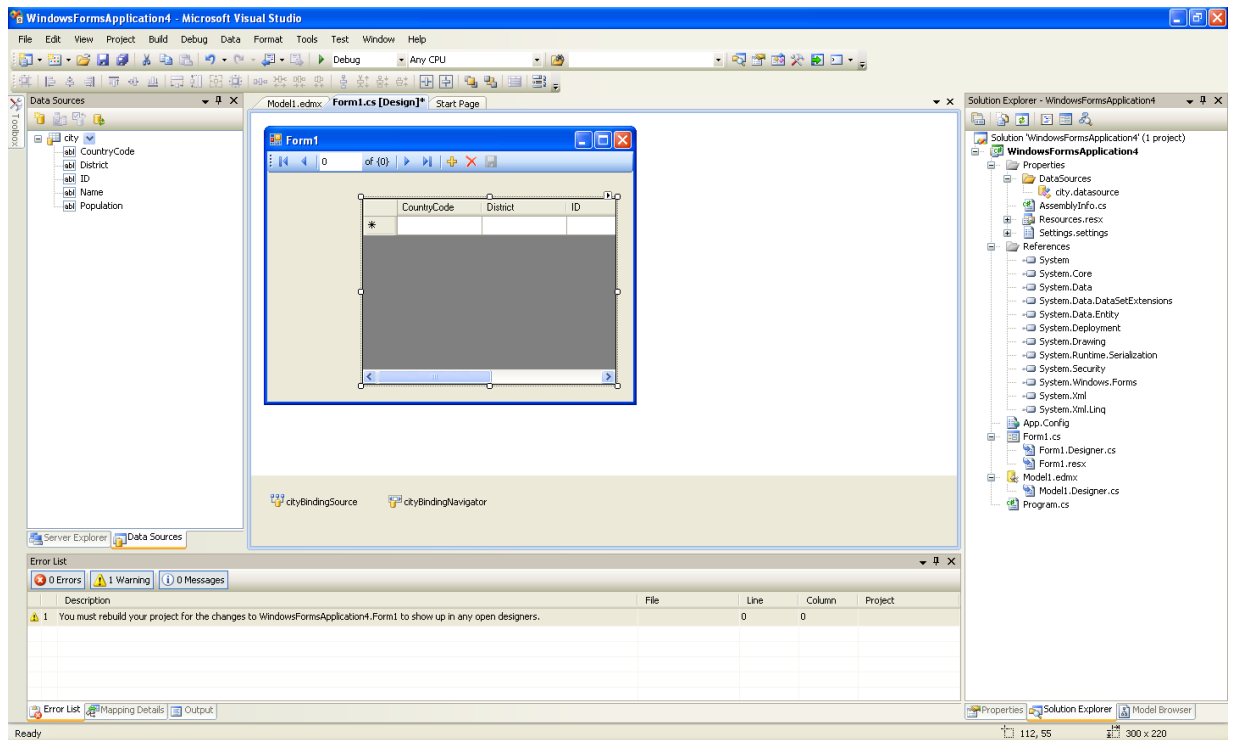


Using the Data Source in a Windows Form

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 `cityBindingSource` and the Navigator control is bound to `cityBindingNavigator`.

Figure 3.27. Data Form Designer



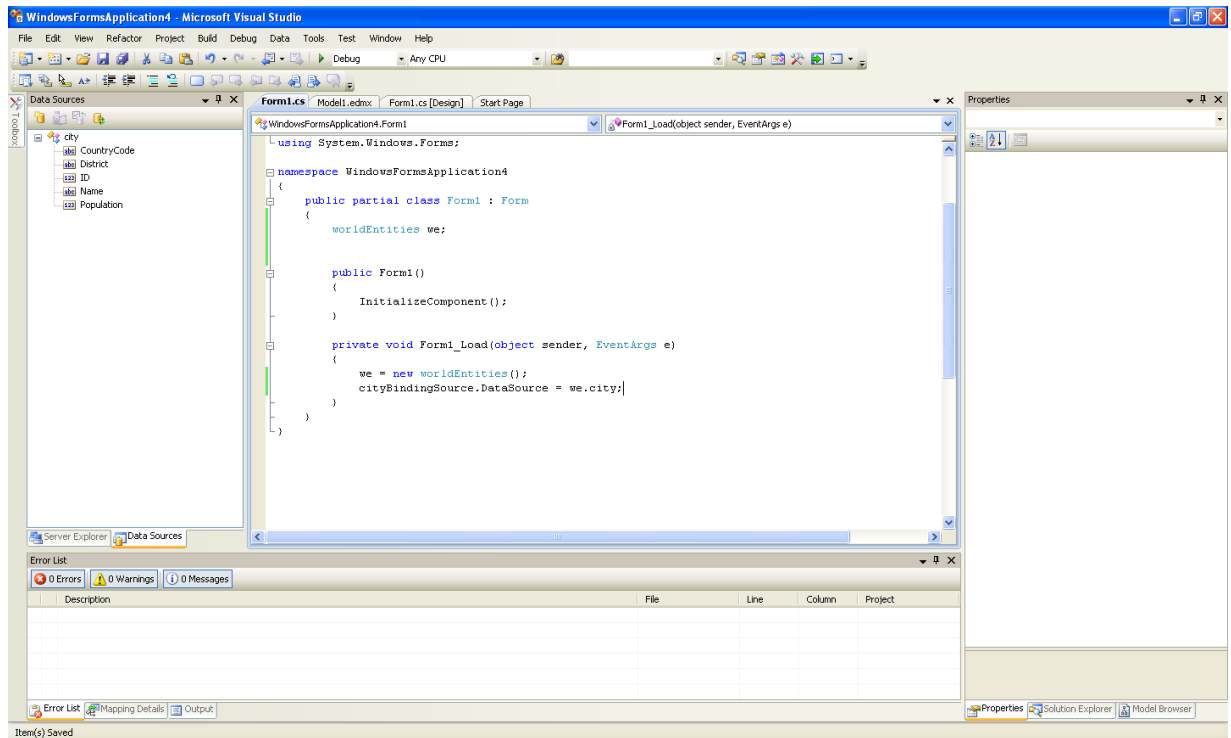
2. Save and rebuild the solution before continuing.

Adding Code to Populate the Data Grid View

You are now ready to add code to ensure that the Data Grid View control will be populated with data from the City database table.

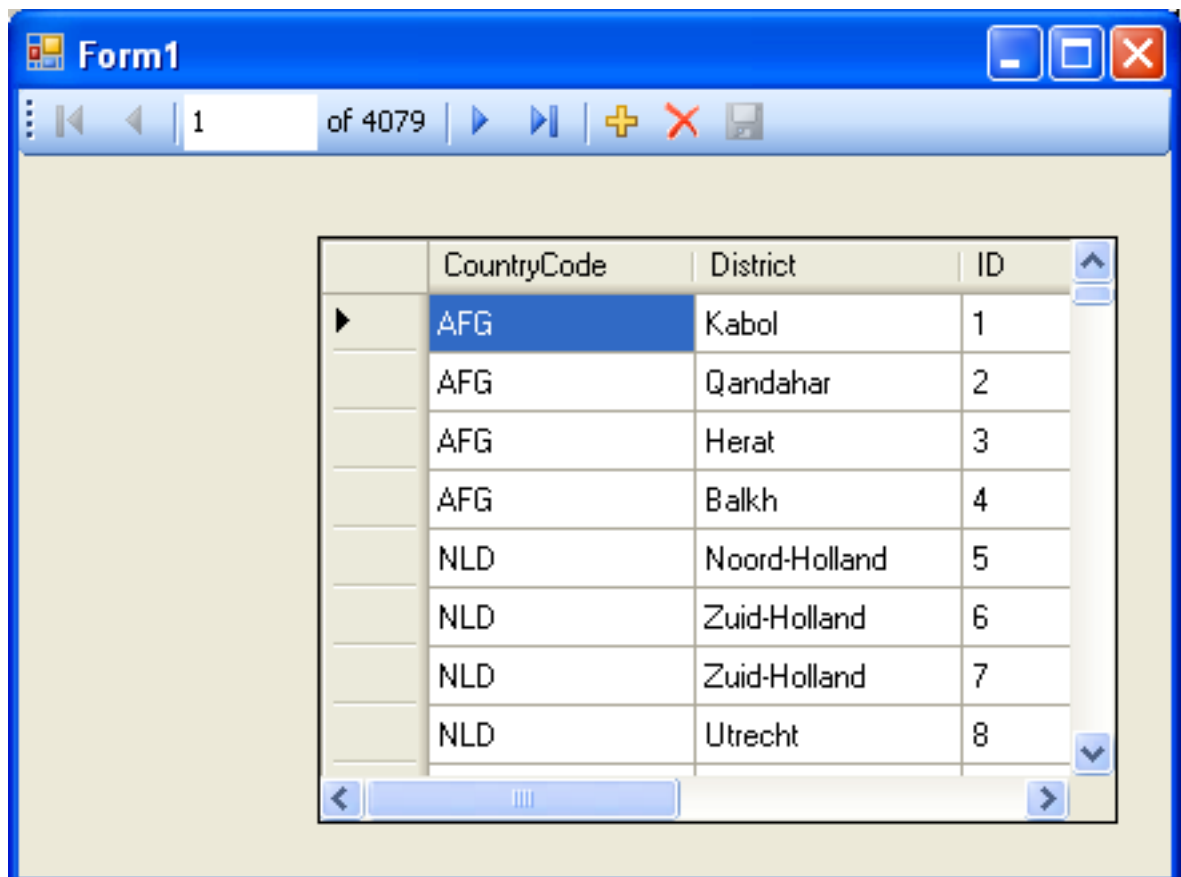
1. Double click the form to access its code.
2. Add code to instantiate the Entity Data Model's EntityContainer object and retrieve data from the database to populate the control.

Figure 3.28. Adding Code to the Form



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

Figure 3.29. The Populated Grid Control



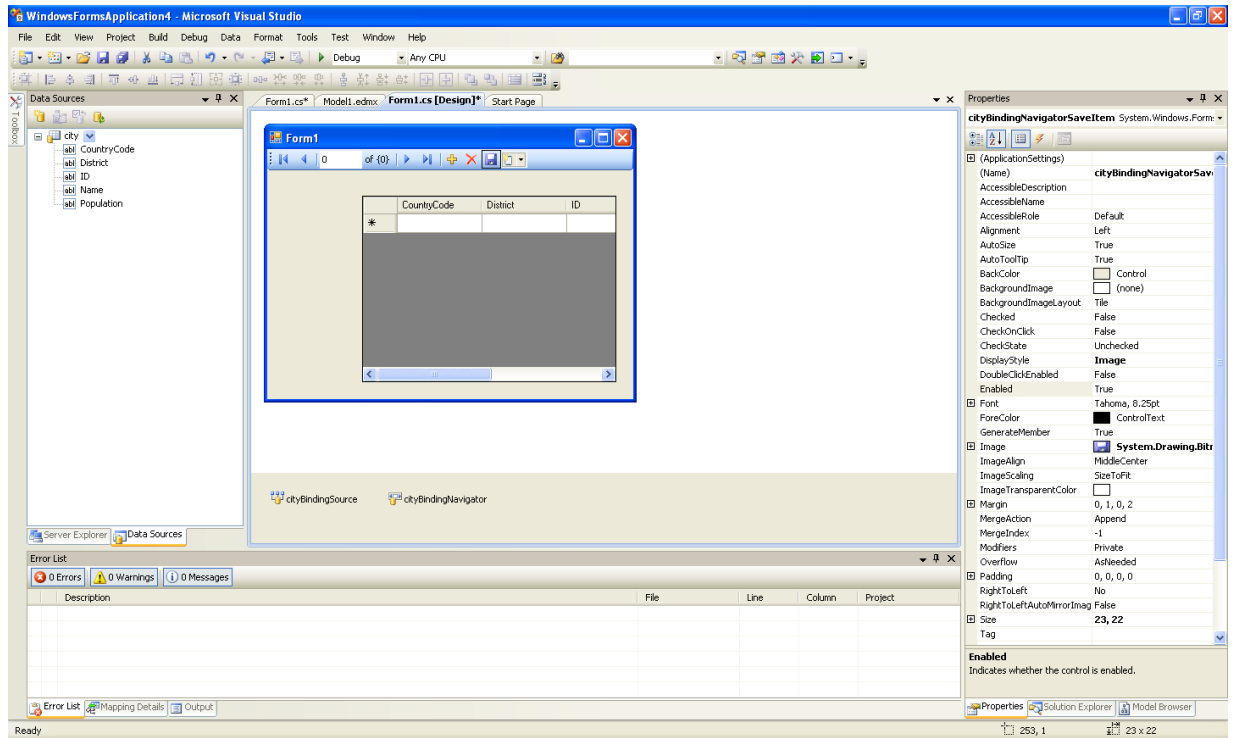
Adding Code to Save Changes to the Database

You will now add code to enable you to save changes to the database.

The Binding source component ensures that changes made in the Data Grid View control are also made to the Entity classes bound to it. However, that data needs to be saved back from the entities to the database itself. This can be achieved by the enabling of the Save button in the Navigator control, and the addition of some code.

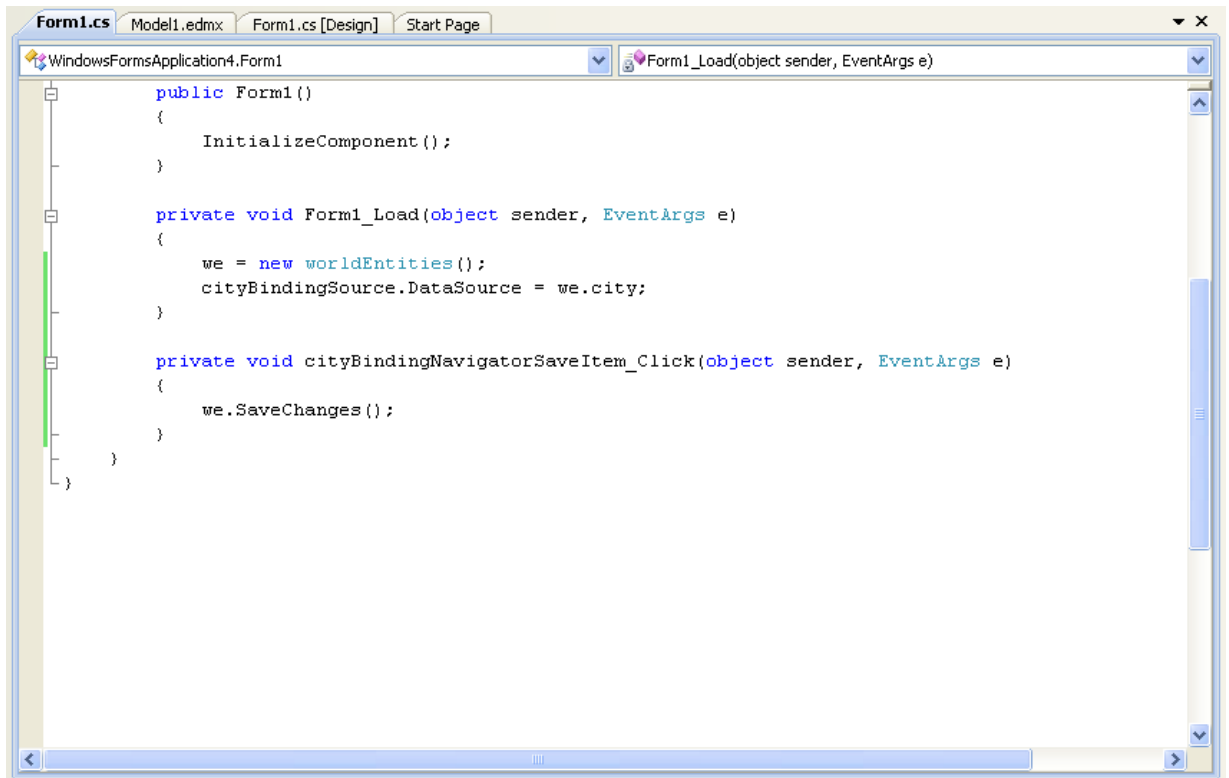
1. In the Form Designer click on the Save icon in the Form toolbar and ensure that its Enabled property is set to True.

Figure 3.30. Save Button Enabled



2. Double click the Save icon in the Form toolbar to display its code.
3. You now need to add code to ensure that data is saved to the database when the save button is click in the application.

Figure 3.31. Adding Save Code to the Form



4. Once the code has been added, save the solution and rebuild it. Run the application and verify that changes made in the grid are saved.

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.

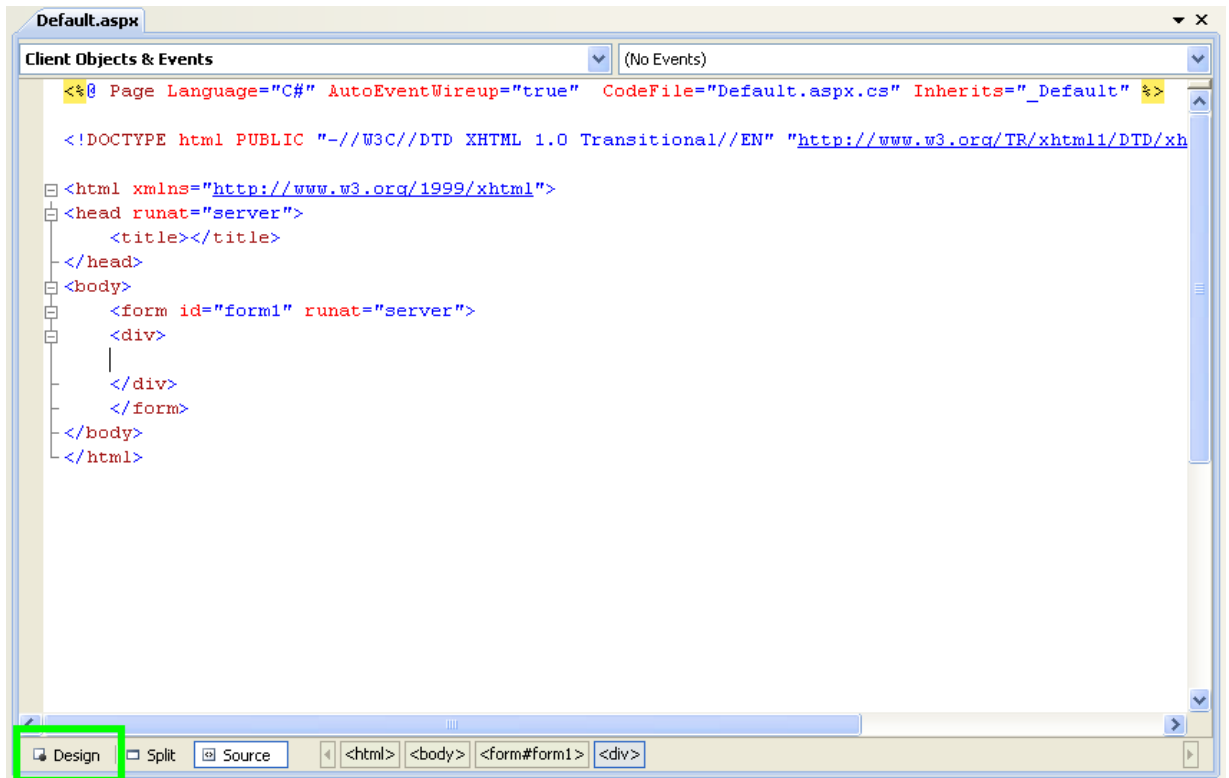
If you have not already done so, you should install the World example database prior to attempting this tutorial. Instructions on where to obtain the database and instructions on how to install it where given in the tutorial [Section 3.9.1, "Tutorial: Using an Entity Framework Entity as a Windows Forms 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.

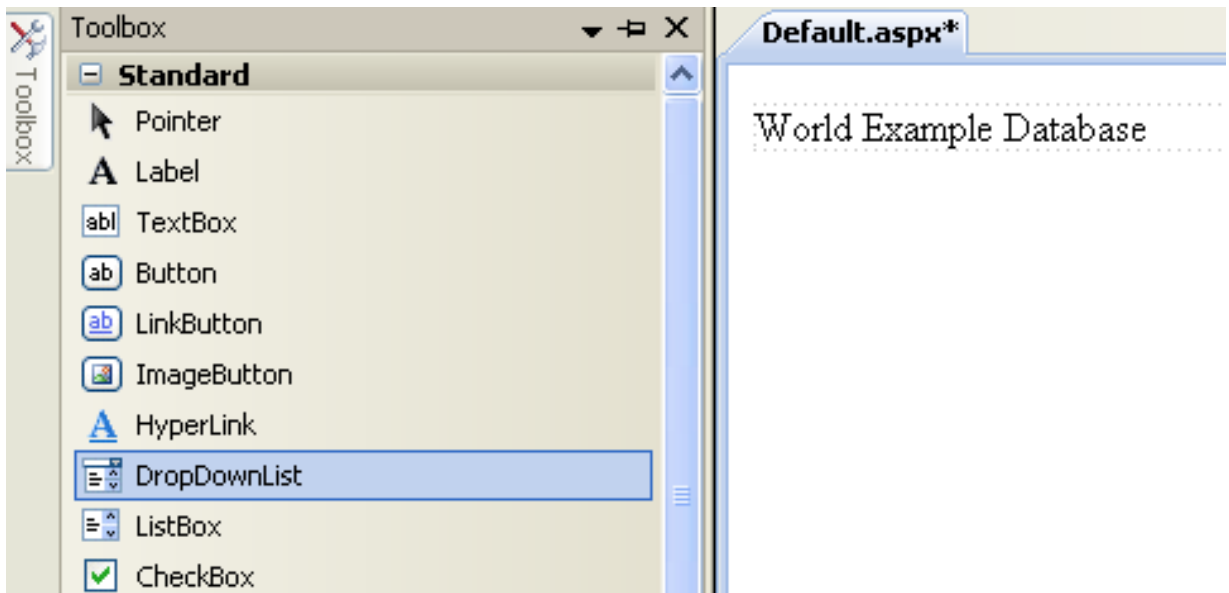
1. From the Visual Studio main menu select **FILE, NEW, WEB SITE...**
2. From the Visual Studio installed templates select **ASP.NET WEB SITE**. Click OK. You will be presented with the Source view of your web page by default.
3. Click the Design view tab situated underneath the Source view panel.

Figure 3.32. The Design Tab



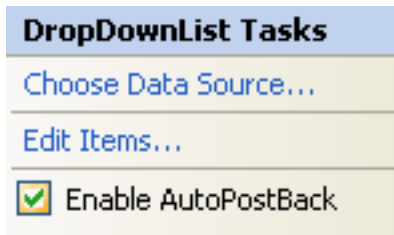
4. In the Design view panel, enter some text to decorate the blank web page.
5. Click on Toolbox. From the list of controls select **DROPPDOWNLIST**. Drag and drop the control to a location beneath the text on your web page.

Figure 3.33. Drop Down List



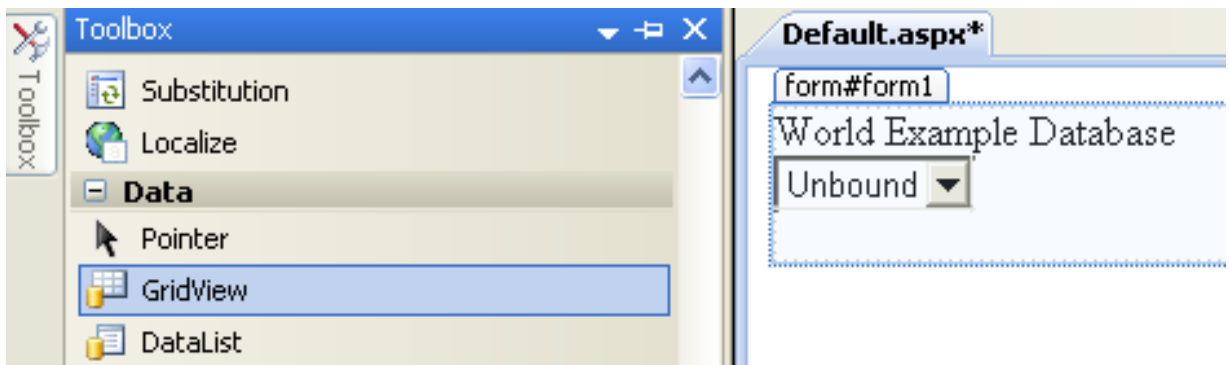
6. From the **DROPPDOWNLIST** control's context menu, ensure that the **ENABLE AUTOPOSTBACK** check box is enabled. This will ensure the control's event handler is called when an item is selected. The user's choice will in turn be used to populate the **GRIDVIEW** control.

Figure 3.34. Enable AutoPostBack



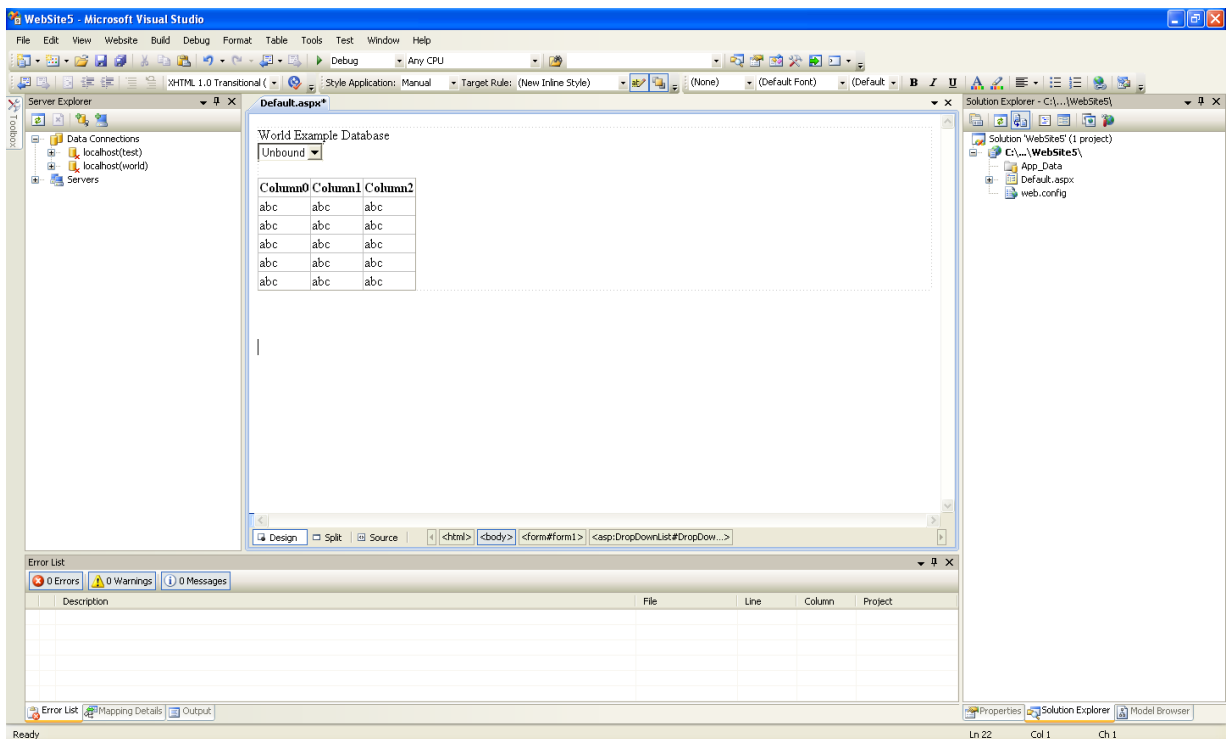
- From the Toolbox select the **GRIDVIEW** control.

Figure 3.35. Grid View Control



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

Figure 3.36. Placed Grid View Control



- At this point it is recommended that you save your solution, and build the solution to ensure that there are no errors.
- 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 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 appear.

At this point you have a working Drop Down List control, populated by a LINQ query on your entity data model.

Populating a Grid View control using an entity LINQ query

In the last part of this tutorial you will populate the Grid View Control using a LINQ query on your entity data model.

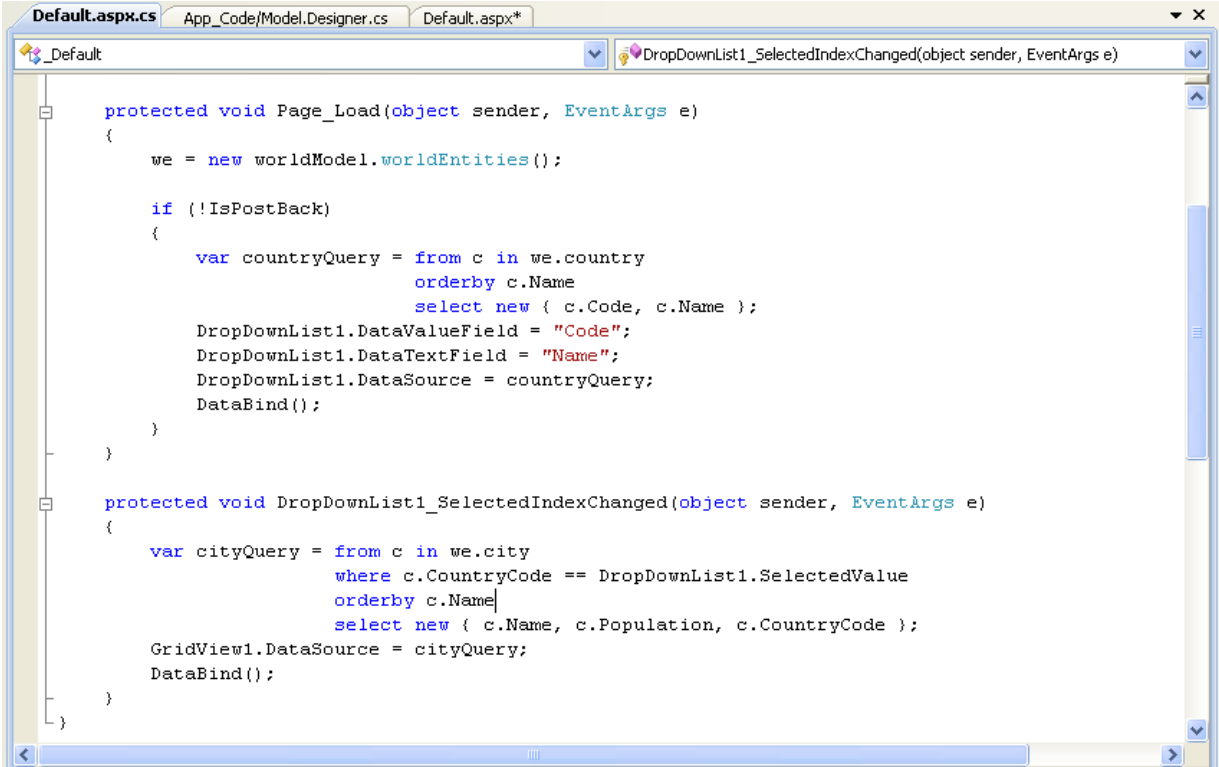
1. In the Design view double click on the **DROPDOWNLIST** control. This causes its [SelectedIndexChanged](#) code to be displayed. This method is called when a user selects an item in the list control and thus fires an `AutoPostBack` event.
2. Modify the relevant section of code accordingly to the following 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 3.37. Source Code



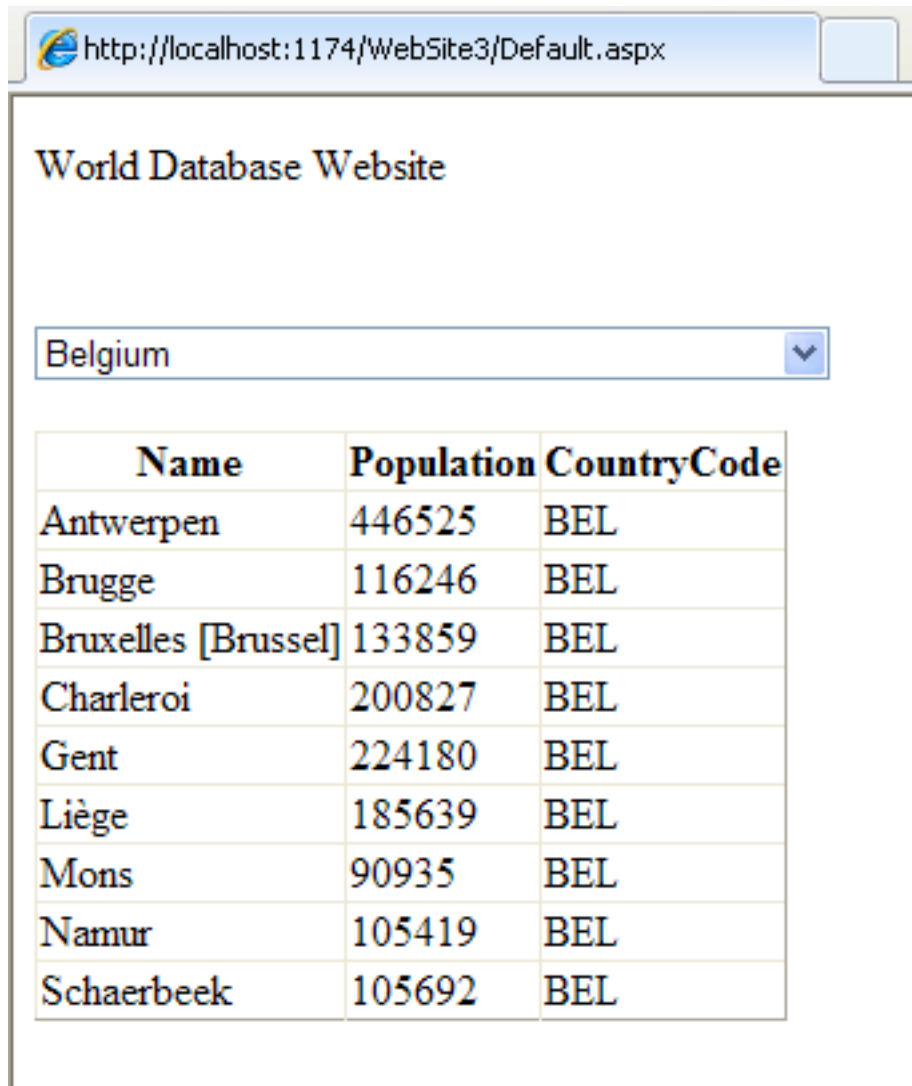
```
protected void Page_Load(object sender, EventArgs e)
{
    we = new worldModel.worldEntities();

    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 3.38. The Working Web Site



The screenshot shows a web browser window with the address bar containing `http://localhost:1174/WebSite3/Default.aspx`. The page title is "World Database Website". Below the title is a dropdown menu with "Belgium" selected. Underneath the dropdown is a table with three columns: "Name", "Population", and "CountryCode". The table lists ten cities in Belgium with their respective populations and country codes.

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.

Chapter 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.

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](#).

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 4.2](#), “Connecting to MySQL Using Connector/NET”. A reference containing a list of supported connection string options can also be found in [Section 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 database 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.

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 `MySqlDataReader` object. The `MySqlDataReader` object contains the results generated by the SQL executed on the command object. Once the results have been obtained in a `MySqlDataReader` object, the results can be processed. In this case the information is simply printed out as part of a `while` loop. Finally, the `MySqlDataReader` object is disposed of by running its `Close` method on it.

In the next example you will see how to use the `ExecuteNonQuery` method.

The procedure for performing an `ExecuteNonQuery` method call is simpler, as there is no need to create an object to store results. This is because `ExecuteNonQuery` is only used for inserting, updating and deleting data. The following example illustrates a simple update to the `Country` table:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial3
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "INSERT INTO Country (Name, HeadOfState, Continent) VALUES ('Disneyland', 'Mickey 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 `ExecuteScalar` on

the command object.

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 illustrates 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 database, you need to load it into a Data Set. This is one of the jobs of the `MySqlDataAdapter` object, and is carried out with its `Fill` method. The following example code illustrates this:

```
DataSet dsCountry;
...
dsCountry = new DataSet();
...
daCountry.Fill(dsCountry, "Country");
```

Note the `Fill` method is a `MySqlDataAdapter` method, the Data Adapter knows how to establish a connection 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 described above is shown below, and is portable.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using MySql.Data;
using MySql.Data.MySqlClient;

namespace WindowsFormsApplication5
{
    public partial class Form1 : Form
    {
        MySqlDataAdapter daCountry;
        DataSet dsCountry;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            string connStr = "server=localhost;user=root;database=world;port=3306;password=*****";
            MySqlConnection conn = new MySqlConnection(connStr);
            try
            {
                label2.Text = "Connecting to MySQL...";

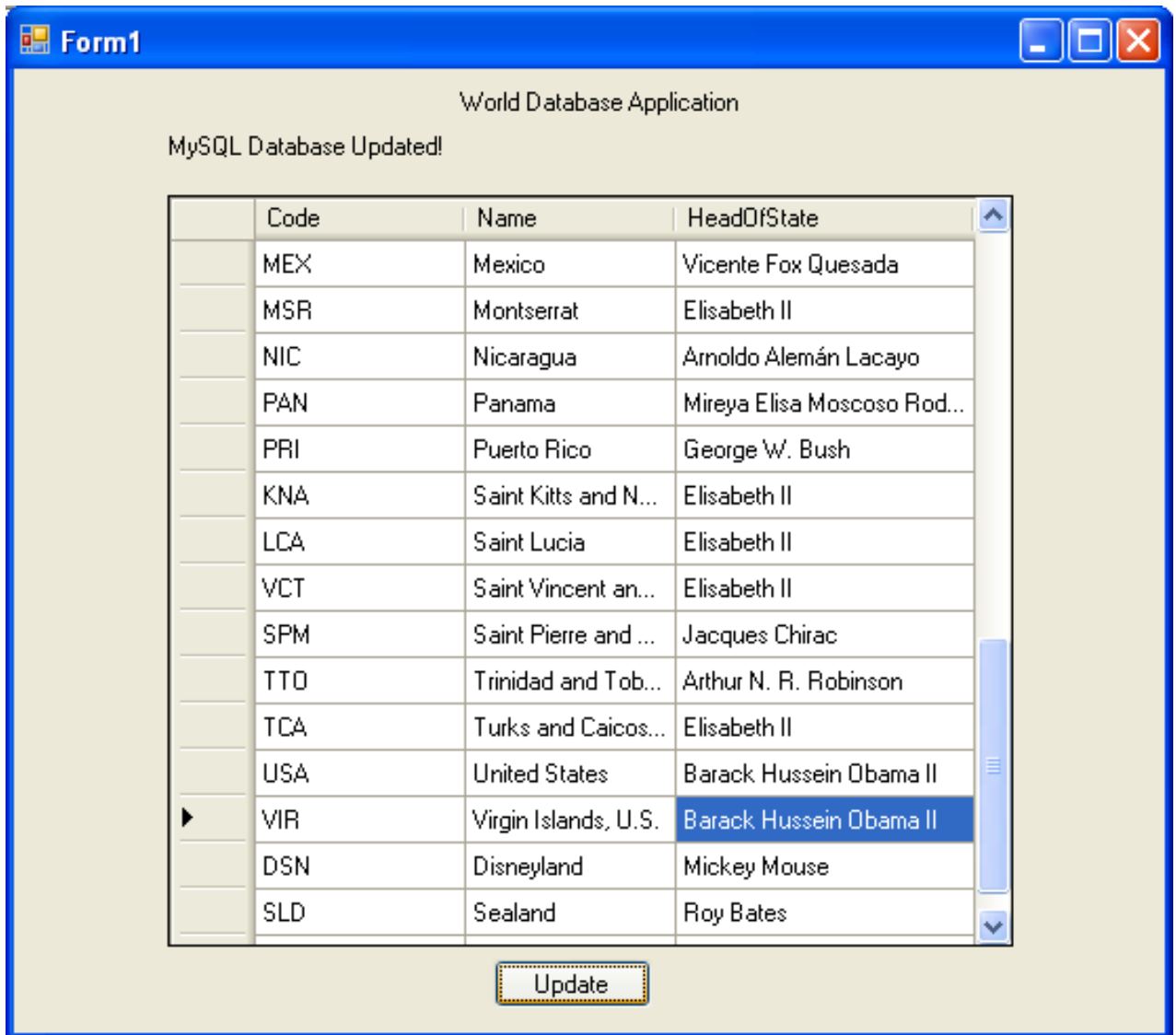
                string sql = "SELECT Code, Name, HeadOfState FROM Country WHERE Continent='North America'";
                daCountry = new MySqlDataAdapter (sql, conn);
                MySqlCommandBuilder cb = new MySqlCommandBuilder(daCountry);

                dsCountry = new DataSet();
                daCountry.Fill(dsCountry, "Country");
                dataGridView1.DataSource = dsCountry;
                dataGridView1.DataMember = "Country";
            }
            catch (Exception ex)
            {
                label2.Text = ex.ToString();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            daCountry.Update(dsCountry, "Country");
            label2.Text = "MySQL Database Updated!";
        }
    }
}
```

The application running is shown below:

Figure 4.1. World Database Application



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 constructed from user input:

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

If the string `user_continent` came from a Text Box control, there would potentially be no control over the string entered by the user. The user could enter a string that generates a 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 using 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.

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 works as expected 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 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.

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 instantiated, 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.

4.3. Creating a Connection String

The `MySqlConnection` object is configured using a connection string. A connection string contains several 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 4.3, "Creating a Connection String"](#). The `Old Syntax` connection string option has now been deprecated.

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.

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;");
```

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 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.

4.5. Connector/NET Connection String Options Reference

Name	Default	Description
<code>Allow Batch</code>	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 ';'. ;
<code>Allow User Variables</code>	false	Setting this to <code>true</code> indicates that the provider expects user variables in the SQL. This option was added in Connector/NET version 5.2.2.
<code>Allow Zero Datetime</code>	false	True to have <code>MySqlDataReader.GetValue()</code> return a <code>MySqlDateTime</code> for date or datetime columns that have illegal values. False will cause a <code>System.DateTime</code> object to be returned for legal values and an exception will be thrown for illegal values.
<code>AutoEnlist</code>	true	
<code>BlobAsUTF8ExcludePattern</code>	null	
<code>BlobAsUTF8IncludePattern</code>	null	
<code>CharSet, Character Set</code>		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.
<code>Connect Timeout, Connection Timeout</code>	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
<code>Connection Reset</code>	false	
<code>Convert Zero Datetime</code>	false	True to have <code>MySqlDataReader.GetValue()</code> and <code>MySqlDataReader.GetDateTime()</code> return <code>DateTime.MinValue</code> for date or datetime columns that have illegal values.
<code>Default Command Timeout</code>		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
<code>Encrypt, UseSSL</code>	false	For Connector/NET 5.0.3 and later, when <code>true</code> , SSL encryption is

		used for all data sent between the client and server if the server has a certificate installed. Recognized values are <code>true</code> , <code>false</code> , <code>yes</code> , and <code>no</code> . In versions before 5.0.3, this option had no effect.
<code>FunctionsReturnString</code>	<code>false</code>	
<code>Host, Server, Data Source, DataSource, Address, Addr, Network Address</code>	<code>localhost</code>	The name or network address of the instance of MySQL to which to connect. Multiple hosts can be specified separated by <code>&</code> . 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.
<code>Ignore Prepare</code>	<code>true</code>	When true, instructs the provider to ignore any calls to <code>MySqlCommand.Prepare()</code> . 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.
<code>Initial Catalog, Database</code>	<code>mysql</code>	The name of the database to use intially
<code>InteractiveSession</code>	<code>false</code>	
<code>Logging</code>	<code>false</code>	When true, various pieces of information is output to any configured TraceListeners.
<code>Old Syntax, OldSyntax</code>	<code>false</code>	Allows use of '@' symbol as a parameter marker. See <code>MySqlCommand</code> for more info. This option was deprecated in Connector/NET 5.2.2. All future code should be written using the '@' symbol.
<code>Password, pwd</code>		The password for the MySQL account being used.
<code>Persist Security Info</code>	<code>false</code>	When set to <code>false</code> or <code>no</code> (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. Re-setting the connection string resets all connection string values including the password. Recognized values are <code>true</code> , <code>false</code> , <code>yes</code> , and <code>no</code> .
<code>Pipe Name, Pipe</code>	<code>mysql</code>	When set to the name of a named pipe, the <code>MySQLConnection</code> will attempt to connect to MySQL on that named pipe. This settings only applies to the Windows platform.
<code>Port</code>	<code>3306</code>	The port MySQL is using to listen for connections. This value is ignored if Unix socket is used.
<code>Procedure Cache Size</code>	<code>25</code>	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.
<code>Protocol</code>	<code>socket</code>	Specifies the type of connection to make to the server. Values can be: <code>socket</code> or <code>tcp</code> for a socket connection, <code>pipe</code> for a named pipe connection, <code>unix</code> for a Unix socket connection, <code>memory</code> to use MySQL shared memory.
<code>Respect Binary Flags</code>	<code>true</code>	Setting this option to <code>false</code> 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.
<code>Shared Memory Name</code>	<code>MYSQL</code>	The name of the shared memory object to use for communication if the connection protocol is set to memory.
<code>TreatBlobsAsUTF8</code>	<code>false</code>	
<code>Treat Tiny As Boolean</code>	<code>true</code>	Setting this value to <code>false</code> indicates that <code>TINYINT(1)</code> will be treated as an <code>INT</code> . See also Overview of Numeric Types for a further explanation of the <code>TINYINT</code> and <code>BOOL</code> data types.
<code>Use Affected Rows</code>	<code>false</code>	When <code>true</code> the connection will report changed rows instead of found rows. This option was added in Connector/NET version 5.2.6.
<code>Use Procedure Bodies</code>	<code>true</code>	Setting this option to <code>false</code> indicates that the user connecting to the database does not have the <code>SELECT</code> privileges for the <code>mysql.proc</code>

		(stored procedures) table. When to set to <code>false</code> , 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.
<code>User Id, Username, Uid, User name</code>		The MySQL login account being used.
<code>Use Compression</code>	false	Setting this option to <code>true</code> enables compression of packets exchanged between the client and the server. This exchange is defined by the MySQL client-server protocol. Compression is used if both client and server support ZLIB compression, and the client has requested compression using this option. A compressed packet header is: packet length (3 bytes), packet number (1 byte), and Uncompressed Packet Length (3 bytes). The Uncompressed Packet Length is the number of bytes in the original, uncompressed packet. If this is zero 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.
<code>Use Usage Advisor</code>	false	
<code>Use Performance Monitor</code>	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
<code>Cache Server Configuration, CacheServerConfiguration, CacheServerConfig</code>	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.
<code>Connection Lifetime</code>	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 <code>Connection Lifetime</code> . 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.
<code>Max Pool Size</code>	100	The maximum number of connections allowed in the pool.
<code>Min Pool Size</code>	0	The minimum number of connections allowed in the pool.
<code>Pooling</code>	true	When <code>true</code> , the <code>MySqlConnection</code> object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are <code>true</code> , <code>false</code> , <code>yes</code> , and <code>no</code> .
<code>Reset Pooled Connections, ResetConnections, ResetPooledConnections</code>	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.

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.

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, MessageBoxIcon.Error)
End Try
```

C# Example

```
MySQL.Data.MySqlClient.MySqlConnection conn;
MySQL.Data.MySqlClient.MySqlCommand cmd;
conn = new MySQL.Data.MySqlClient.MySqlConnection();
cmd = new MySQL.Data.MySqlClient.MySqlCommand();
conn.ConnectionString = strConnection;
try
{
    conn.Open();
    cmd.Connection = conn;
    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, @number, @text)";
    cmd.Prepare();
    cmd.Parameters.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);
}
```

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 out of a MySQL stored procedure through use of the `MySqlCommand.Parameters` collection.

Note

When you call a stored procedure, the command object makes an additional `SELECT` call to determine the parameters of the stored procedure. You must ensure that the user calling the procedure has the `SELECT` privilege on the `mysql.proc` table to enable them to verify the parameters. Failure to do this will result in an error when calling the procedure.

This section will not provide in-depth information on creating Stored Procedures. For such information, please refer to <http://dev.mysql.com/doc/mysql/en/stored-routines.html>.

A sample application demonstrating how to use stored procedures with Connector/NET can be found in the `Samples` directory of your Connector/NET installation.

4.7.1. 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, MessageBoxIcon.Error)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
conn.ConnectionString = "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.

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, MessageBoxIcon.Error)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
conn.ConnectionString = "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 `MySqlConnection.Parameters` collection.

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.

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.

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.

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.

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.

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.

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.

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:

- CrystalDecisions.CrystalReports.Engine
- CrystalDecisions.ReportSource
- CrystalDecisions.Shared
- CrystalDecisions.Windows.Forms

The following code assumes that you created your report using a data set saved using the code shown in [Section 4.9.1, "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.

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.

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.

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](#).

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.

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.

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 their 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 machine's .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 tool-

bar 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.

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 metadata.

Some examples of situations that may return incorrect metadata are:

- Execution of `SHOW PROCESSLIST`. Some of the columns will be returned as binary even though they only hold string data.
- When a 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`.

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 exclude 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.

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.

Chapter 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](#).

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/>.

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 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 `admindemo`. 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/>.

5.3. Connector/NET Change History

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

Appendix A. MySQL Connector/NET Change History

A.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 behavior)
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](#))

A.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](#))

A.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](#))

A.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](#))

A.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](#))

A.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](#))

A.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](#))

A.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 `requiresQuestionAndAnswer="false"`, a `SqlNullValueException` was generated when using `MembershipUser.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)

A.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, PublicKeyToken=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/resetskipkgs' to re-enable package loading."
```

(Bug#40726)

A.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 `autogenerates-chema="true"`. (Bug#39072)
- Connector/NET called hashed password methods not supported in Mono 2.0 Preview 2. (Bug#38895)

A.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)

A.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.QuoteIdentifier` 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 retrieving 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 procedure, 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.

A.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 of issues were identified in the case, connection and schema 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 membership 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 in 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.

A.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 led 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 defined 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 handler 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)

A.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)

A.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 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)

A.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 `enablePasswordRetrieval = 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.QuoteIdentifier` 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)

A.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)

A.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 caused by some old legacy code preventing their use. (Bug#32093)
- A date string could be returned incorrectly by `MySqlDateTime.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 led 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 defined 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 handler 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)

A.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)

A.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)

A.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 before 4.1 would be incorrectly parsed, resulting in an exception. (Bug#23342)
- Fixed password property on `MySQLConnectionStringBuilder` to use `PasswordPropertyText` attribute. This causes dots to show instead of actual password text.

A.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!).

A.24. Changes in MySQL Connector/NET 5.0.10 (Not yet released)

Bugs fixed:

- If, when using the `MySqlConnection` 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 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](#))

A.25. Changes in MySQL Connector/NET 5.0.9 (Not yet released)

Bugs fixed:

- The `DbCommandBuilder.QuoteIdentifier` 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 caused by some old legacy code preventing their use. ([Bug#32093](#))
- A date string could be returned incorrectly by `MySqlDateTime.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 led 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 defined 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 handler 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](#))

A.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.

A.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 `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 `charset` property would not be identified during a connection (also affected Visual Studio 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 before 4.1 would be incorrectly parsed, resulting in an exception. (Bug#23342)

A.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)

A.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 correct 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`.

A.30. Changes in MySQL Connector/NET 5.0.4 (Not released)

This is a new Beta development release, fixing recently discovered bugs.

A.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 using the `BeginExecuteNonQuery` and `EndExecuteNonQuery` methods.
- Metadata from stored 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](#))

A.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 number of procedures that are cached 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](#))

A.33. Changes in MySQL Connector/NET 5.0.1 (01 October 2006)

Bugs fixed:

- Connector/NET on a Turkish 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)

A.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 implementation 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](#))

A.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 caused 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 handler 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](#))

A.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 `MySqlConnection.GetConnection`. ([Bug#24373](#))

A.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 number of procedures that are cached 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)

A.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 Turkish 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)
- Calling `MySqlCommandBuilder.DeriveParameters` for a stored procedure that has no parameters 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)

A.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)

A.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)

A.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)

A.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.

A.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 Databases 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 `IndexOutOfRangeException` 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.

A.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.

A.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

A.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.

A.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.
- Improved 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).

A.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 infinitely (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).

A.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).

A.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 `charset` 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 Perken-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 Perken-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!).

A.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).

A.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 Perens-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 `CoonnectionPooling` 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. PostgreSQL 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.

A.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).

A.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.

A.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.

A.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.