C# MySQL tutorial

last modified September 25, 2021

C# MySQL tutorial shows how to program MySQL in C#. It covers the basics of MySQL programming with C#. C# tutorial is a comprehensive tutorial on C# language.

MySQL

MySQL is a leading open source database management system. It is a multi user, multithreaded database management system. MySQL is especially popular on the web. MySQL database is available on most important OS platforms. It runs on BSD Unix, Linux, Windows or Mac OS. MySQL comes in two versions: MySQL server system and MySQL embedded system.

ADO.NET

ADO.NET is an important part of the .NET framework. It is a specification that unifies access to relational databases, XML files and other application data. *MySql.Data* is an implementation of the ADO.NET specification for the MySQL database. It is a driver written in C# language and is available for all .NET languages.

```
$ dotnet add package MySql.Data
```

We include the package to our .NET Core project.

The `MySqlConnection`, `MySqlCommand`, `MySqlDataReader`, `DataSet`, and `MySqlDataProvider` are the core elements of the .NET data provider model. The `MySqlConnection` creates a connection to a specific data source. The `MySqlCommand` object executes an SQL statement against a data source. The `MySqlDataReader` reads streams of data from a data source.

The `DataSet` object is used for offline work with a mass of data. It is a disconnected data representation that can hold data from a variety of different sources. Both `MySqlDataReader` and `DataSet` are used to work with data; they are used under different circumstances. If we only need to read the results of a query, the `MySqlDataReader` is the better choice. If we need more extensive processing of data, or we want to bind a Winforms control to a database table, the `DataSet` is preferred.

C# MySQL version

If the following program we check the version of the MySQL server.

Program.cs

```
using System;
using MySql.Data.MySqlClient;

string cs = @"server=localhost;userid=dbuser;password=s$cret;database=testdb";

using var con = new MySqlConnection(cs);
con.Open();

Console.WriteLine($"MySQL version : {con.ServerVersion}");
```

We connect to the database and get some info about the MySQL server.

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

We import the elements of the MySQL data provider.

```
string cs = @"server=localhost;userid=dbuser;password=s$cret;database=testdb";
```

This is the connection string. It is used by the data provider to establish a connection to the database. We specify the host name, user name, password and a database name.

```
using var con = new MySqlConnection(cs);
```

A `MySQLConnection` object is created. This object is used to open a connection to a database. The `using` statement releases the database connection resource when the variable goes out of scope.

```
con.Open();
```

This line opens the database connection.

```
Console.WriteLine($"MySQL version : {con.ServerVersion}");
```

Here we print the version of MySQL using the `ServerVersion` property of the connection object.

```
$ dotnet run
MySQL version : 5.5.5-10.1.36-MariaDB
```

C# MySQL SELECT statement

The following example determines the version of MySQL with a SELECT statement.

Program.cs

```
using System;
using MySql.Data.MySqlClient;

string cs = @"server=localhost;userid=dbuser;password=s$cret;database=mydb";

using var con = new MySqlConnection(cs);
con.Open();

var stm = "SELECT VERSION()";
var cmd = new MySqlCommand(stm, con);

var version = cmd.ExecuteScalar().ToString();
Console.WriteLine($"MySQL version: {version}");
```

We check for the version of the MySQL database. This time using an SQL query.

```
var stm = "SELECT VERSION()";
```

This is the SQL SELECT statement. It returns the version of the database. The `VERSION()` is a built-in MySQL function.

```
var cmd = new MySqlCommand(stm, con);
```

The `MySqlCommand` is an object which is used to execute a query on the database. The parameters are the SQL statement and the connection object.

```
var version = cmd.ExecuteScalar().ToString();
```

There are queries which return only a scalar value. In our case, we want a simple string specifying the version of the database. The `ExecuteScalar` is used in such situations.

C# MySQL create table

In the following example, we create a database table and fill it with data.

Program.cs

```
using System;
using MySql.Data.MySqlClient;

string cs = @"server=localhost;userid=dbuser;password=s$cret;database=testdb";

using var con = new MySqlConnection(cs);
con.Open();
```

```
using var cmd = new MySqlCommand();
cmd.Connection = con;

cmd.CommandText = "DROP TABLE IF EXISTS cars";
cmd.ExecuteNonQuery();

cmd.CommandText = @"CREATE TABLE cars(id INTEGER PRIMARY KEY AUTO_INCREMENT,
        name TEXT, price INT)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Audi',52642)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Mercedes',57127)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Skoda',9000)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Volvo',29000)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Bentley',350000)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Citroen',21000)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Hummer',41400)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Volkswagen',21600)";
cmd.ExecuteNonQuery();

Console.WriteLine("Table cars created");
```

In the example, we create a `cars` table with eight rows.

```
cmd.CommandText = "DROP TABLE IF EXISTS cars";
cmd.ExecuteNonQuery();
```

First we drop the table if it already exists. We use the `ExecuteNonQuery()` method if we do not want a result set, for example for `DROP`, `INSERT`, or `DELETE` statements.

```
cmd.CommandText = @"CREATE TABLE cars(id INTEGER PRIMARY KEY AUTO_INCREMENT,
        name TEXT, price INT)";
cmd.ExecuteNonQuery();;
```

The `cars` table is created. The `AUTO_INCREMENT` keyword makes the column auto-incremented in MySQL.

```
cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Audi',52642)";
cmd.ExecuteNonQuery();

cmd.CommandText = "INSERT INTO cars(name, price) VALUES('Mercedes',57127)";
cmd.ExecuteNonQuery();
...
```

Here we insert two rows into the table.

```
$ dotnet run
Table cars created
```

We run the program.

```
$ mysql -u dbuser -p
```

We connect to the MySQL server with the `mysql` tool.

```
MariaDB [(none)]> use testdb;
Database changed
MariaDB [testdb]> SELECT * FROM cars;
+----+------------+--------+
| id | name       | price  |
+----+------------+--------+
|  1 | Audi       |  52642 |
|  2 | Mercedes   |  57127 |
|  3 | Skoda      |   9000 |
|  4 | Volvo      |  29000 |
|  5 | Bentley    | 350000 |
|  6 | Citroen    |  21000 |
|  7 | Hummer     |  41400 |
|  8 | Volkswagen |  21600 |
+----+------------+--------+
8 rows in set (0.00 sec)
```

We verify the data. The `cars` table was successfully created.

C# MySQL prepared statements

Prepared statements increase security and performance. When we write prepared statements, we use placeholders instead of directly writing the values into the statements.

Program.cs

```
using System;
using MySql.Data.MySqlClient;

string cs = @"server=localhost;userid=dbuser;password=s$cret;database=testdb";

using var con = new MySqlConnection(cs);
```

```
con.Open();

var sql = "INSERT INTO cars(name, price) VALUES(@name, @price)";
using var cmd = new MySqlCommand(sql, con);

cmd.Parameters.AddWithValue("@name", "BMW");
cmd.Parameters.AddWithValue("@price", 36600);
cmd.Prepare();

cmd.ExecuteNonQuery();

Console.WriteLine("row inserted");
```

We add a new car to the `cars` table. We use a parameterized command.

```
var sql = "INSERT INTO cars(name, price) VALUES(@name, @price)";
using var cmd = new MySqlCommand(sql, con);
```

When we write prepared statements, we use placeholders instead of directly writing the values into the statements. Prepared statements are faster and guard against SQL injection attacks. The `@name` and `@price` are placeholders, which are going to be filled later.

```
cmd.Parameters.AddWithValue("@name", "BMW");
cmd.Parameters.AddWithValue("@price", 36600);
cmd.Prepare();
```

Values are bound to the placeholders with the `AddWithValue()` method.

```
cmd.ExecuteNonQuery();
```

The prepared statement is executed. We use the `ExecuteNonQuery()` method of the `MySQLCommand` object when we don't expect any data to be returned.

C# MySqlDataReader

The `MySqlDataReader` is an object used to retrieve data from the database. It provides fast, forward-only, read-only access to query results. It is the most efficient way to retrieve data from tables.

Program.cs

```
using System;
using MySql.Data.MySqlClient;

string cs = @"server=localhost;userid=dbuser;password=s$cret;database=testdb";

using var con = new MySqlConnection(cs);
con.Open();
```

```
string sql = "SELECT * FROM cars";
using var cmd = new MySqlCommand(sql, con);

using MySqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())
{
    Console.WriteLine("{0} {1} {2}", rdr.GetInt32(0), rdr.GetString(1),
            rdr.GetInt32(2));
}
```

We get all rows from the `cars` table and print them to the console.

```
using MySqlDataReader rdr = cmd.ExecuteReader();
```

To create a `MySQLDataReader`, we call the `ExecuteReader` method of the `MySqlCommand` object.

```
while (rdr.Read())
{
    Console.WriteLine("{0} {1} {2}", rdr.GetInt32(0), rdr.GetString(1),
            rdr.GetInt32(2));
}
```

The `Read` method advances the data reader to the next record. It returns `true` if there are more rows; otherwise `false`. We can retrieve the value using the array index notation, or use a specific method to access column values in their native data types. The latter is more efficient.

```
$ dotnet run
1 Audi 52642
2 Mercedes 57127
3 Skoda 9000
4 Volvo 29000
5 Bentley 350000
6 Citroen 21000
7 Hummer 41400
8 Volkswagen 21600
9 BMW 36600
```

C# MySQL column headers

In the following example we print column headers with the data from a database table.

Program.cs

```
using System;
using MySql.Data.MySqlClient;

string cs = @"server=localhost;userid=dbuser;password=s$cret;database=testdb";
```

```
using var con = new MySqlConnection(cs);
con.Open();

var sql = "SELECT * FROM cars";

using var cmd = new MySqlCommand(sql, con);

using MySqlDataReader rdr = cmd.ExecuteReader();
Console.WriteLine($"{rdr.GetName(0),-4} {rdr.GetName(1),-10} {rdr.GetName(2),10}");

while (rdr.Read())
{
    Console.WriteLine($"{rdr.GetInt32(0),-4} {rdr.GetString(1),-10}
{rdr.GetInt32(2),10}");
}
```

In the example, we select all rows from the `cars` table with their column names.

```
Console.WriteLine($"{rdr.GetName(0),-4} {rdr.GetName(1),-10} {rdr.GetName(2),10}");
```

We get the names of the columns with the `GetName()` method of the reader.

```
while (rdr.Read())
{
    Console.WriteLine($"{rdr.GetInt32(0),-4} {rdr.GetString(1),-10}
{rdr.GetInt32(2),10}");
}
```

We print the data that was returned by the SQL statement to the terminal.

```
$ dotnet run
id   name            price
1    Audi            52642
2    Mercedes        57127
3    Skoda            9000
4    Volvo           29000
5    Bentley        350000
6    Citroen         21000
7    Hummer          41400
8    Volkswagen      21600
9    BMW             36600
```

In this tutorial, we have shown how to program MySQL databases in C#.