**Namespace :-**  System.Data.SqlClient

#### System.Data.SqlClient

This assembly (namespace) of .NET Framework contains all of the classes required to connect to the databases, read/write data to the databases.

Errors, Success reports are generated by it. SqlError is generated for errors, and the success codes are executed and shown on the screen.

We will be using this namespace directly, and the classes contained by it in our code, to connect to the database. To execute the commands to read the data from it, or to update the records or to add new ones as whole.

**Connecting to a database**

Connection to a database requires a connection string. This string has the information about the server you're going to connect, the database you will require and the credentials that you can use to connect. Each database has its own properties, its own server, name and type of login information, using which you can connect to the database to read/write the data from it.

**Quote:**

In computing, a connection string is a string that specifies information about a data source and the means of connecting to it. It is passed in code to an underlying driver or provider in order to initiate the connection. Whilst commonly used for a database connection, the data source could also be a spreadsheet or text file.

From this above statement one can easily conclude that the connection string is a basic information about the data source, data, and the method to connect to it.

<http://www.connectionstrings.com/> is the website where you can easily find the connection string for your database. They provide the strings, for almost all of the database services and their types. Do try them! In the code I am providing, the string was caught from their website for testing my own database too. :-)

In the namespace I have talked about, the SqlConnection class does the job for us. We can use the following code, to connect to the SQL Database,

using(SqlConnection conn = new SqlConnection())

{

conn.ConnectionString = "Server=[server\_name];Database=[database\_name];Trusted\_Connection=true";

*// using the code here...*

}

This would create a new connection to the SQL Server database that will be connected using the ConnectionString provided. You will not require to have the SQL Server installed on your system. Since, we've already said the connectionString is just a string to tell the underlying code about the location and the schema of the data software provider. So we can even have the SQL Server connection established that is on a seperate developmen environment. The above connection string has

1. Server  
   This part of the string tells the underlying code, the name of the server to connect to. Your server's name would differ in this manner.
2. Database  
   This is the name of the database you're going to connect to.

In all of the databases, there are two types of login methods. Windows authentication and Database authentication. In Windows authentication, the database is authenticated using the user's credentials from the windows (OS), and in the database authentication you pass the username and password, in order to connect to the database.

In my case, the authenticaion was Windows, so I had to write the Trusted\_Connection part inside the string. If you're using the database authentication you will provide the username and password fields in the string.

#### Executing the Commands

Once connected to the database, you can execute the set of commands that you're having and which would execute on the server (or the data provider) to execute the function you're trying to do, such as query for data, inserting the data, updating records and so on and so forth.

SQL has the basic syntax for the commands, and in my opinion the simples syntax of commands and near to the human-understandable commands in the programming world. In the namespace the class, SqlCommand does this job for us. An example of a command would be like,

Hide   Copy Code

SqlCommand command = new SqlCommand("SELECT \* FROM TableName", conn);

..each and every command on the SQL Server would be executed like this, first parameter is the command, and the second one is the connection on which the command would execute. You can pass any command into it, and the underlying code would convert it back to the command which would execute on the server where the data is present and then will return the result to you, whether an error or a success report.

Sometimes you might want to use the command of the INSERT INTO clause, to work that out, you will need to add parameters to the command, so that your database is safe from [SQL Injections](https://en.wikipedia.org/wiki/SQL_injection). Using parameters, would reduce the chances of your database being attacked, by throwing an error if the user tries to add some commands through the form into the database server.

#### Parameterizing the data

Parameterzing the query is done by using the SqlParameter passed into the command. For example, you might want to search for the records where a criteria matches. You can denote that criteria, by passing the variable name into the query and then adding the value to it using the SqlParameter object. For instance, the following is your SqlCommand to be passed on to the server,

Hide   Copy Code

*// Create the command*

SqlCommand command = new SqlCommand("SELECT \* FROM TableName WHERE FirstColumn = @0", conn);

*// Add the parameters.*

command.Parameters.Add(new SqlParameter("0", 1));

In the above code, the variable added is 0, and the value to it is passed. You can use any variable but

it **must** start with a **@** sign. Once that has been done, you can then add the parameters to that name. In this case, value 1 has been hardcoded you can add variable value here too.

Remember, the connection must be opened in order to run this code, you can use conn.Open() to open the connection if asked.

As explained in the code, I have used the parameter as a number (0) which can also be a name. For example, you can also write the code as

Hide   Copy Code

*// Create the command*

SqlCommand command = new SqlCommand("SELECT \* FROM TableName WHERE FirstColumn = @firstColumnValue", conn);

*// Add the parameters.*

command.Parameters.Add(new SqlParameter("firstColumnValue", 1));

This way, it will be easier for you to keep them in mind. I am better in working with numbers and

indexes like in array so I used 0, you can use name, combination of alphanumeric characters and so on. Just pass the name in the SqlParameter object and you'll be good to go!

#### Reading the data returned

In SQL you mostly use the SELECT statement to get the data from the database to show, CodeProject would do so to show the recent articles from their database, Google would do so to index the results etc. But how to show those data results in the application using C#? That is the question here. Well, in the namespace we're talking, there is class SqlDataReader present for the SqlCommand which returns the Reader object for the data. You can use this to read through the data and for each of the column provide the results on the screen.

The following code would get the results from the command once executed,

Hide   Copy Code

*// Create new SqlDataReader object and read data from the command.*

using (SqlDataReader reader = command.ExecuteReader())

{

*// while there is another record present*

while (reader.Read())

{

*// write the data on to the screen*

Console.WriteLine("{0} \t | {1} \t | {2} \t | {3}",

*// call the objects from their index*

reader[0], reader[1], reader[2], reader[3]);

}

}

This is the same code, it will execute and once done executing it will let framework handle the job and close the resources as per best method.

#### Adding data to the SQL Server

Similar method is implemented for adding the data to the database. Only the command would change, and we know in database, INSERT INTO clause is used to add the data. So, the command would become,

Hide   Copy Code

SqlCommand insertCommand = new SqlCommand("INSERT INTO TableName (FirstColumn, SecondColumn, ThirdColumn, ForthColumn) VALUES (@0, @1, @2, @3)", conn);

You can then use the SqlParameter objects to add the values to the parameters. This way, when the command would execute the data would be added to the table that you've specified.

#### Catching the errors from SQL Server

SQL Server generates the errors for you to catch and work on them. In the namespace we're working there are two classes that work with the errors and exceptions thrown by SQL Server,

1. SqlError
2. SqlException

These are used to get the error details or to catch the exceptions in the code and print the data results respectively. If you're going to use a try catch block you're more likely to use the SqlException thing in your code.

For this to work, we are going to pass a command that we know is going to throw an error.

Hide   Copy Code

SqlCommand errorCommand = new SqlCommand("SELECT \* FROM someErrorColumn", conn);

Now we know that this is faulty, but this won't generate any error, untill we execute it. To do so, we will try to execute it like this,

Hide   Copy Code

errorCommand.ExecuteNonQuery();

Once this is executed, SQL Server would complain saying there is no such table present. To catch it

you can simply use the try catch block with the SqlException in the catch block to be caught. For a working code, you can see the following code block in the live example of my article. That explains the usage of the try catch block with the SqlException here.

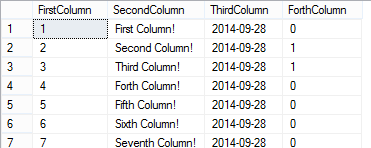
## Working example

In the article there is an example assosiated for you to download if you require to work it out.

You will be required to use a SQL Server, Database and the tables to make sure the program works. If the server name, database name or the tables don't match. The program won't run. There was no way for me to attach a database in the example. Since the databases require SQL Server database that will be always available using a connection. I won't use this database again, so I have not provided the database connection string.

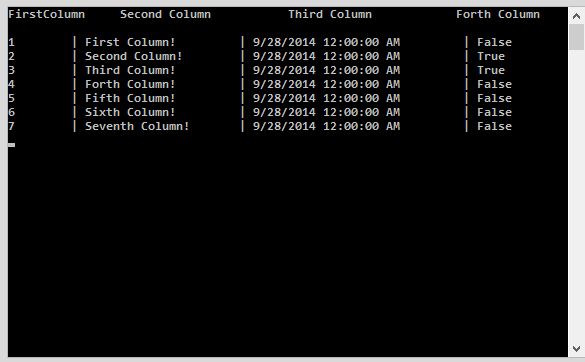
#### Database table

The database table in my system was like the following



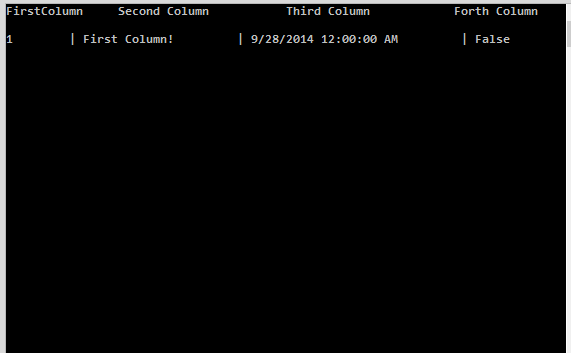
#### SELECT without WHERE

You can run the console and you'll see the result on your screen. This is the code where the SELECT query ran and the out put of the columns returned was printed,



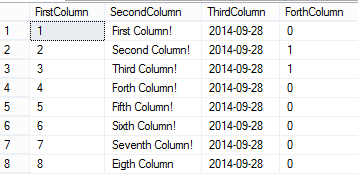
#### SELECT with a WHERE

We can at the same time add a few more parameters to the SELECT query, so that only the data we want would be extracted from the database. For example, if we add a WHERE clause to the SELECT query, the following result would generate.



#### Inserting the data

Once  done, you can move on to the next stage. This part, is related to the second command, where we add the records to the table. This statement once executed would add the data to the table, the table now is like this



This is the example, of what happens when the above code would be used in our application. I am also going to write down the code used in this console application. Comments have been added with each block to make you understand the working of the code.

#### Source code

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

using System;

using System.Data;

using System.Data.SqlClient;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SqlTest\_CSharp

{

class Program

{

static void Main(string[] args)

{

*// Create the connection to the resource!*

*// This is the connection, that is established and*

*// will be available throughout this block.*

using (SqlConnection conn = new SqlConnection())

{

*// Create the connectionString*

*// Trusted\_Connection is used to denote the connection uses Windows Authentication*

conn.ConnectionString = "Server=[server\_name];Database=[database\_name];Trusted\_Connection=true";

conn.Open();

*// Create the command*

SqlCommand command = new SqlCommand("SELECT \* FROM TableName WHERE FirstColumn = @0", conn);

*// Add the parameters.*

command.Parameters.Add(new SqlParameter("0", 1));

*/\* Get the rows and display on the screen!*

*\* This section of the code has the basic code*

*\* that will display the content from the Database Table*

*\* on the screen using an SqlDataReader. \*/*

using (SqlDataReader reader = command.ExecuteReader())

{

Console.WriteLine("FirstColumn\tSecond Column\t\tThird Column\t\tForth Column\t");

while (reader.Read())

{

Console.WriteLine(String.Format("{0} \t | {1} \t | {2} \t | {3}",

reader[0], reader[1], reader[2], reader[3]));

}

}

Console.WriteLine("Data displayed! Now press enter to move to the next section!");

Console.ReadLine();

Console.Clear();

*/\* Above code was used to display the data from the Database table!*

*\* This following section explains the key features to use*

*\* to add the data to the table. This is an example of another*

*\* SQL Command (INSERT INTO), this will teach the usage of parameters and connection.\*/*

Console.WriteLine("INSERT INTO command");

*// Create the command, to insert the data into the Table!*

*// this is a simple INSERT INTO command!*

SqlCommand insertCommand = new SqlCommand("INSERT INTO TableName (FirstColumn, SecondColumn, ThirdColumn, ForthColumn) VALUES (@0, @1, @2, @3)", conn);

*// In the command, there are some parameters denoted by @, you can*

*// change their value on a condition, in my code they're hardcoded.*

insertCommand.Parameters.Add(new SqlParameter("0", 10));

insertCommand.Parameters.Add(new SqlParameter("1", "Test Column"));

insertCommand.Parameters.Add(new SqlParameter("2", DateTime.Now));

insertCommand.Parameters.Add(new SqlParameter("3", false));

*// Execute the command, and print the values of the columns affected through*

*// the command executed.*

Console.WriteLine("Commands executed! Total rows affected are " + insertCommand.ExecuteNonQuery());

Console.WriteLine("Done! Press enter to move to the next step");

Console.ReadLine();

Console.Clear();

*/\* In this section there is an example of the Exception case*

*\* Thrown by the SQL Server, that is provided by SqlException*

*\* Using that class object, we can get the error thrown by SQL Server.*

*\* In my code, I am simply displaying the error! \*/*

Console.WriteLine("Now the error trial!");

*// try block*

try

{

*// Create the command to execute! With the wrong name of the table (Depends on your Database tables)*

SqlCommand errorCommand = new SqlCommand("SELECT \* FROM someErrorColumn", conn);

*// Execute the command, here the error will pop up!*

*// But since we're catching the code block's errors, it will be displayed inside the console.*

errorCommand.ExecuteNonQuery();

}

*// catch block*

catch (SqlException er)

{

*// Since there is no such column as someErrorColumn (Depends on your Database tables)*

*// SQL Server will throw an error.*

Console.WriteLine("There was an error reported by SQL Server, " + er.Message);

}

}

*// Final step, close the resources flush dispose them. ReadLine to prevent the console from closing.*

Console.ReadLine();

}

}

}

The above code is the source code used in this application to work.

Points of Interest

SQL Server does not require that you install the SQL Server on your machine. You can connect to an instance of SQL Server which is present on the seperate environment, but you need to make sure the connection has been established and that you can connect to the server which would provide the data. Databases can be present on different locations only thing that would connect them would be the correct connection string.

Connection string must be accurate so that the server can provide the exact data.

**Important point**: The tables and the database schema provided here is associated with database and tables I had. The results on your screen might (surely) differ. Results, errors, thrown here might not be the same as on your system. This is just example! And depends on the database tables and their data and properties.