

2PGDCA4 (B)-Programming With ASP. Net

UNIT-IV

1. Data Base Programming for Websites

Accessing Data from a database is an important aspect of any programming language. It is necessary for any programming language to have the ability to work with databases. ASP.Net has the ability to work with different types of databases. It can work with the most common databases such as Oracle and Microsoft SQL Server. It also has the ability to work with new forms of databases such as MongoDB and MySQL.

1.1 Fundamentals of Database connectivity

ASP.Net has the ability to work with a majority of databases. The most common being Oracle and Microsoft SQL Server. But with every database, the logic behind working with all of them is mostly the same.

While working with databases, the following concepts which are common across all databases.

1. **Connection** – To work with the data in a database, the first obvious step is the connection. The connection to a database normally consists of the below-mentioned parameters.
 1. **Database name or Data Source** – The first important parameter is the database name. Each connection can only work with one database at a time.
 2. **Credentials** – The next important aspect is the 'username' and 'password'. This is used to establish a connection to the database.
 3. **Optional parameters** - You can specify optional parameters on how .net should handle the connection to the database. For example, one can specify a parameter for how long the connection should stay active.
2. **Selecting data from the database** – Once the connection is established, data is fetched from the database. ASP.Net has the ability to execute 'sql' select command against the database. The 'sql' statement can be used to fetch data from a specific table in the database.
3. **Inserting data into the database** – ASP.Net is used to insert records into the database. Values for each row that needs to be inserted in the database are specified in ASP.Net.
4. **Updating data into the database** – ASP.Net can also be used to update existing records into the database. New values can be specified in ASP.Net for each row that needs to be updated into the database.
5. **Deleting data from a database** – ASP.Net can also be used to delete records from the database. The code is written to delete a particular row from the database.

2. Creating Connection

Let's now look at the code, which needs to be kept in place to create a connection to a database. In our example, we will connect to a database which has the name of Demodb.

The credentials used to connect to the database are given below

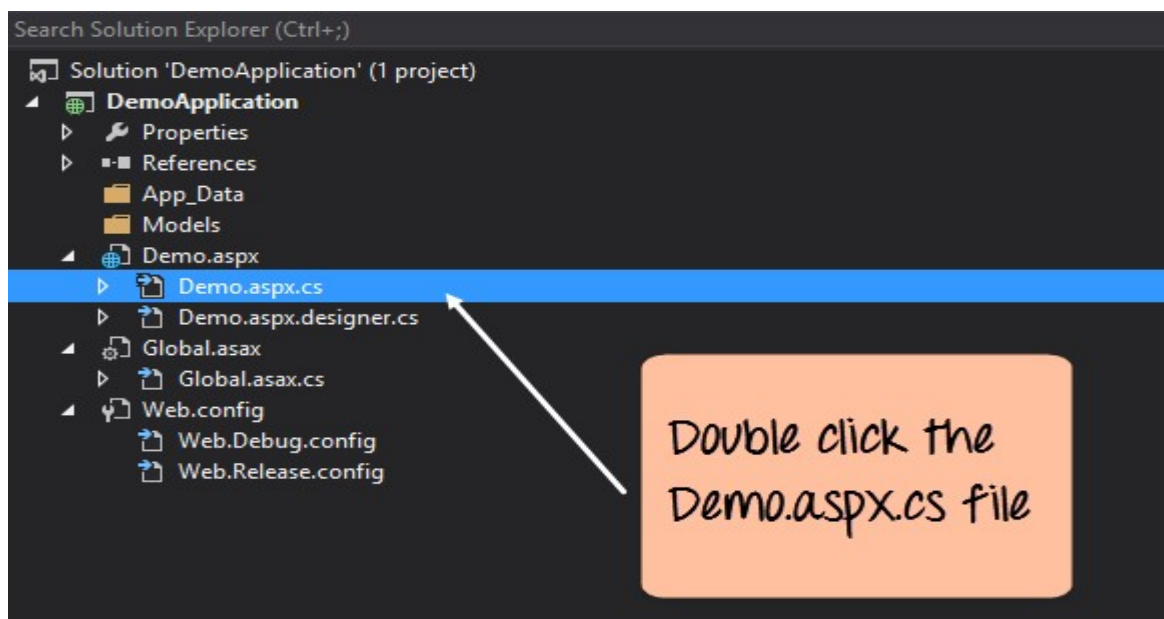
- **Username – sa**
- **Password – demo123**

Let's work with our current web application created in the earlier sections.

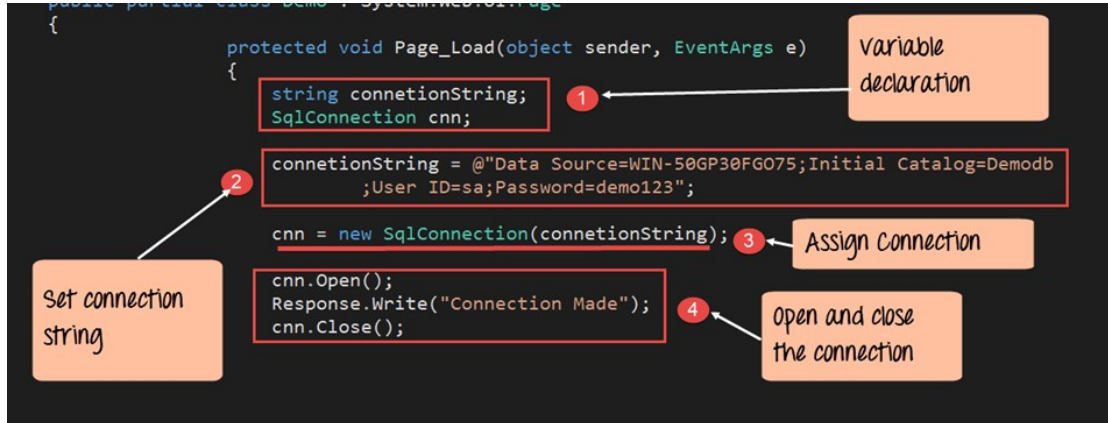
- Start adding database operations to it.
- Our example look's at establishing a simple connection. This connection is made to the Demodb database. This is done when the page is first launched.
- When the connection is established, a message will be sent to the user. The message will indicate that the connection has been established.

Let's follow the below-mentioned steps to achieve this.

Step 1) Let's first ensure that you have your web application (DemoApplication) opened in Visual Studio. Double click the 'demo.aspx.cs' file to enter the code for the database connection.



Step 2) Add the below code which will be used to establish a connection to the database.



```
namespace DemoApplication
{
    public partial class Demo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            string connetionString;
            SqlConnection cnn;

            connetionString = @"Data Source=WIN-50GP30FG075;Initial
Catalog=Demodb ;User ID=sa;Password=demo123";

            cnn = new SqlConnection(connetionString);

            cnn.Open();

            Response.Write("Connection MAd");
            conn.Close();
        }
    }
}
```

Code Explanation:-

1. The first step is to create variables. It will be used to create the connection string and the connection to the SQL Server database.
2. The next step is to actually create the connection string. The connection string consists of the following parts

- Data Source – This is the name of the server on which the database resides. In our case, it resides on a machine called WIN- 50GP30FGO75.
 - The Initial Catalog is used to specify the name of the database
 - The UserID and Password are the credentials required to connect to the database.
3. Next, we assign the connecting string to the variable 'cnn'.
- The variable cnn is of type SqlConnection. This is used to establish a connection to the database.
 - SqlConnection is a class in ASP.Net, which is used to create a connection to a database.
 - To use this class, you have to first create an object of this class. Hence, here we create a variable called 'cnn' which is of the type SqlConnection.
4. Next, we use the open method of the cnn variable to open a connection to the database. We display a message to the user that the connection is established. This is done via the 'response.write' method. We then close the connection to the database.

When the above code is set, and the project is executed using Visual Studio. You will get the below output. Once the form is displayed, click the Connect button.

Output:-



The output message displayed in the browser will show that the connection to the database is made.

3. ADO.NET Data Set

It is a collection of data tables that contain the data. It is used to fetch data without interacting with a Data Source that's why, it also known as **disconnected** data access method. It is an in-memory data store that can hold more than one table at the same time. We can use DataRelation object to relate these tables. The Data Set can also be used to read and write data as XML document.

ADO.NET provides a Data Set class that can be used to create Data Set object. It contains constructors and methods to perform data related operations.

3.1 DataSet Class Signature

```
public class DataSet : System.ComponentModel.MarshalByValueComponent, System.ComponentModel.IListSource, System.ComponentModel.ISupportInitializeNotification, System.Runtime.Serialization.ISerializable, System.Xml.Serialization.IXmlSerializable
```

3.2 DataSet Constructors

Constructor	Description
DataSet()	It is used to initialize a new instance of the DataSet class.
DataSet(String)	It is used to initialize a new instance of a DataSet class with the given name.
DataSet(SerializationInfo, StreamingContext)	It is used to initialize a new instance of a DataSet class that has the given serialization information and context.
DataSet(SerializationInfo, StreamingContext, Boolean)	It is used to initialize a new instance of the DataSet class.

3.3 Data Set Properties

Properties	Description
CaseSensitive	It is used to check whether DataTable objects are case-sensitive or not.
DataSetName	It is used to get or set name of the current DataSet.
DefaultViewManager	It is used to get a custom view of the data contained in the DataSet to allow filtering and searching.
HasErrors	It is used to check whether there are errors in any of the DataTable objects within this DataSet.
IsInitialized	It is used to check whether the DataSet is initialized or not.
Locale	It is used to get or set the locale information used to compare strings within the table.
Namespace	It is used to get or set the namespace of the DataSet.
Site	It is used to get or set an ISite for the DataSet.
Tables	It is used to get the collection of tables contained in the DataSet.

3.4 Data Set Methods

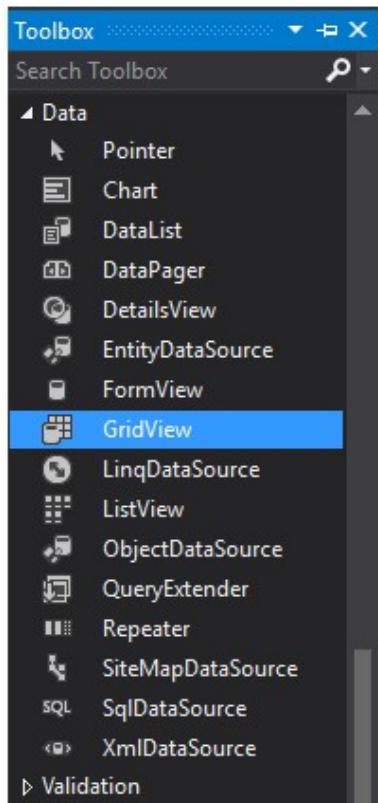
The following table contains some commonly used methods of DataSet.

Method	Description
BeginInit()	It is used to begin the initialization of a DataSet that is used on a form.
Clear()	It is used to clear the DataSet of any data by removing all rows in all tables.
Clone()	It is used to copy the structure of the DataSet.
Copy()	It is used to copy both the structure and data for this

	DataSet.
CreateDataReader(DataTable[])	It returns a DataTableReader with one result set per DataTable.
CreateDataReader()	It returns a DataTableReader with one result set per DataTable.
EndInit()	It ends the initialization of a DataSet that is used on a form.
GetXml()	It returns the XML representation of the data stored in the DataSet.
GetXmlSchema()	It returns the XML Schema for the XML representation of the data stored in the DataSet.
Load(IDataReader, LoadOption, DataTable[])	It is used to fill a DataSet with values from a data source using the supplied IDataReader.
Merge(DataSet)	It is used to merge a specified DataSet and its schema into the current DataSet.
Merge(DataTable)	It is used to merge a specified DataTable and its schema into the current DataSet.
ReadXml(XmlReader, XmlReadMode)	It is used to read XML schema and data into the DataSet using the specified XmlReader and XmlReadMode.
Reset()	It is used to clear all tables and removes all relations, foreign constraints, and tables from the DataSet.
WriteXml(XmlWriter, XmlWriteMode)	It is used to write the current data and optionally the schema for the DataSet using the specified XmlWriter and XmlWriteMode.

Example:

Here, in this example, we are implementing **DataSet** and displaying data into a gridview. Create a web form and drag a gridview from the **toolbox** to the form. We can find it inside the data category.



// DataSetDemo.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DataSetDemo.aspx.
cs"
Inherits="DataSetExample.DataSetDemo" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
</div>
<asp:GridView ID="GridView1" runat="server" CellPadding="4" ForeColor="#33
3333" GridLines="None">
<AlternatingRowStyle BackColor="White" />
<EditRowStyle BackColor="#2461BF" />
<FooterStyle BackColor="#507CD1" Font-
Bold="True" ForeColor="White" />
```



```

        <HeaderStyle BackColor="#507CD1" Font-
Bold="True" ForeColor="White" />
        <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Cente
r" />
        <RowStyle BackColor="#EFF3FB" />
        <SelectedRowStyle BackColor="#D1DDF1" Font-
Bold="True" ForeColor="#333333" />
        <SortedAscendingCellStyle BackColor="#F5F7FB" />
        <SortedAscendingHeaderStyle BackColor="#6D95E1" />
        <SortedDescendingCellStyle BackColor="#E9EBEF" />
        <SortedDescendingHeaderStyle BackColor="#4870BE" />
    </asp:GridView>
</form>
</body>
</html>

```

CodeBehind

```

// DataSetDemo.aspx.cs
using System;
using System.Data.SqlClient;
using System.Data;
namespace DataSetExample
{
    public partial class DataSetDemo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            using (SqlConnection con = new SqlConnection("data source=.; database=stu
dent; integrated security=SSPI"))
            {
                SqlDataAdapter sde = new SqlDataAdapter("Select * from student", con);
                DataSet ds = new DataSet();
                sde.Fill(ds);
                GridView1.DataSource = ds;
                GridView1.DataBind();
            }
        }
    }
}

```

4. ADO.NET Data Adapter

The Data Adapter works as a bridge between a Data Set and a data source to retrieve data. Data Adapter is a class that represents a set of SQL commands and a database connection. It can be used to fill the Data Set and update the data source.

4.1 Data Adapter Class Signature

```
public class DataAdapter : System.ComponentModel.Component, System.Data.IDataAdapter
```

4.2 Data Adapter Constructors

Constructors	Description
Data Adapter()	It is used to initialize a new instance of a Data Adapter class.
Data Adapter(Data Adapter)	It is used to initialize a new instance of a Data Adapter class from an existing object of the same type.

4.3 Methods

Method	Description
CloneInternals()	It is used to create a copy of this instance of Data Adapter.
Dispose(Boolean)	It is used to release the unmanaged resources used by the Data Adapter.
Fill(DataSet)	It is used to add rows in the Data Set to match those in the data source.
FillSchema(DataSet, SchemaType, String, IDataReader)	It is used to add a Data Table to the specified Data Set.
GetFillParameters()	It is used to get the parameters set by the user when executing an SQL SELECT statement.
ResetFillLoadOption()	It is used to reset FillLoadOption to its

	default state.
ShouldSerializeAcceptChangesDuringFill()	It determines whether the AcceptChangesDuringFill property should be persisted or not.
ShouldSerializeFillLoadOption()	It determines whether the FillLoadOption property should be persisted or not.
ShouldSerializeTableMappings()	It determines whether one or more DataTableMapping objects exist or not.
Update(DataSet)	It is used to call the respective INSERT, UPDATE, or DELETE statements.

Example

// DataSetDemo.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DataSetDemo.aspx.cs"
```

```
Inherits="DataSetExample.DataSetDemo" %>
```

```
<!DOCTYPE html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
```

```
<title></title>
```

```
</head>
```

```
<body>
```

```
<form id="form1" runat="server">
```

```
<div>
```

```
</div>
```

```
<asp:GridView ID="GridView1" runat="server" CellPadding="3" BackColor="#DEBA84"
```

```
BorderColor="#DEBA84" BorderStyle="None" BorderWidth="1px" CellSpacing="2"
```

```
>
```

```
<FooterStyle BackColor="#F7DFB5" ForeColor="#8C4510" />
```

```
<HeaderStyle BackColor="#A55129" Font-Bold="True" ForeColor="White" />
```

```
<PagerStyle ForeColor="#8C4510" HorizontalAlign="Center" />
```

```
<RowStyle BackColor="#FFF7E7" ForeColor="#8C4510" />
```

```

        <SelectedRowStyle BackColor="#738A9C" Font-
Bold="True" ForeColor="White" />
        <SortedAscendingCellStyle BackColor="#FFF1D4" />
        <SortedAscendingHeaderStyle BackColor="#B95C30" />
        <SortedDescendingCellStyle BackColor="#F1E5CE" />
        <SortedDescendingHeaderStyle BackColor="#93451F" />
    </asp:GridView>
</form>
</body> </html>

```

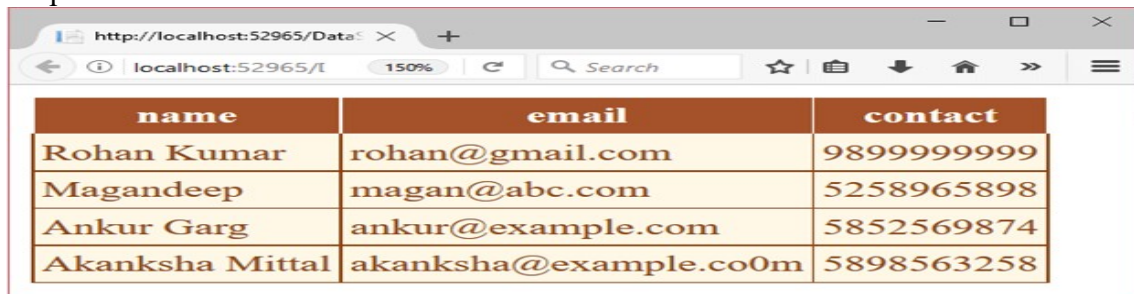
Code Behind

```

using System;
using System.Data.SqlClient;
using System.Data;
namespace DataSetExample
{
    public partial class DataSetDemo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            using (SqlConnection con = new SqlConnection("data source=.; database=stud
ent; integrated security=SSPI"))
            {
                SqlDataAdapter sde = new SqlDataAdapter("Select * from student", con);
                DataSet ds = new DataSet();
                sde.Fill(ds);
                GridView1.DataSource = ds;
                GridView1.DataBind();
            }
        }
    }
}

```

Output:



The screenshot shows a web browser window with the address bar displaying 'http://localhost:52965/DataSetDemo.aspx'. The browser's address bar also shows 'localhost:52965/'. The page content displays a table with three columns: 'name', 'email', and 'contact'. The table contains four rows of data:

name	email	contact
Rohan Kumar	rohan@gmail.com	98999999999
Magandeep	magan@abc.com	5258965898
Ankur Garg	ankur@example.com	5852569874
Akanksha Mittal	akanksha@example.co0m	5898563258

5. ASP.NET Read Database using SqlDataReader

To show data accessed using Asp.Net, let us assume the following artifacts in our database.

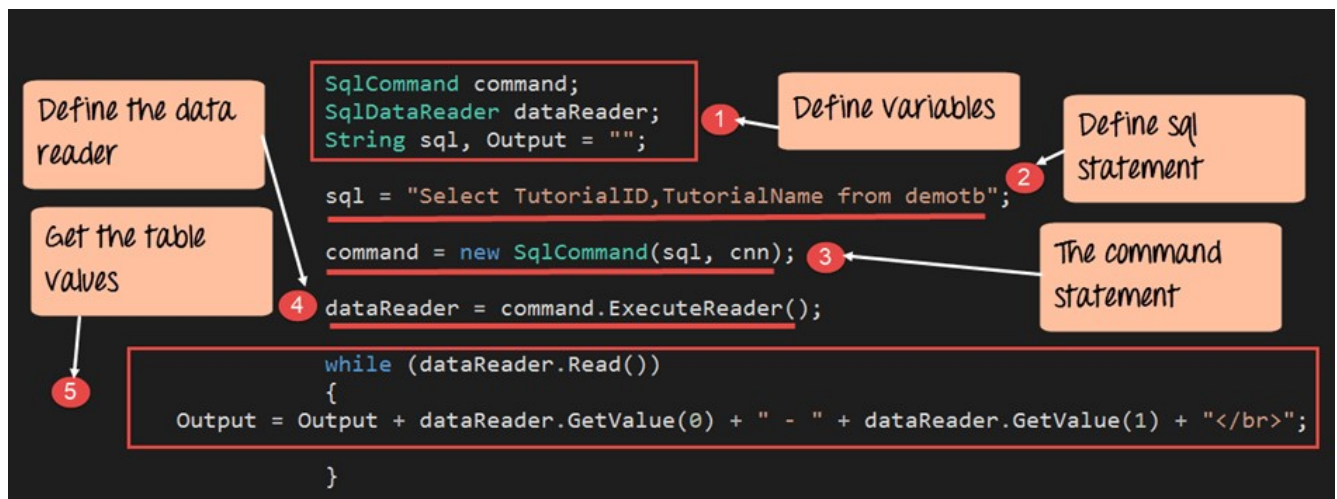
1. A table called demotb. This table will be used to store the ID and names of various Tutorials.
2. The table will have two columns, one called "TutorialID" and the other called "TutorialName."
3. For the moment, the table will have two rows as shown below.

TutorialID	TutorialName
1	C#
2	ASP.Net

Let's change the code so that we can query for this data and display the information on the web page itself. Note that the code entered is in continuation to that written for the data connection module.

Step 1) Let's split the code into two parts,

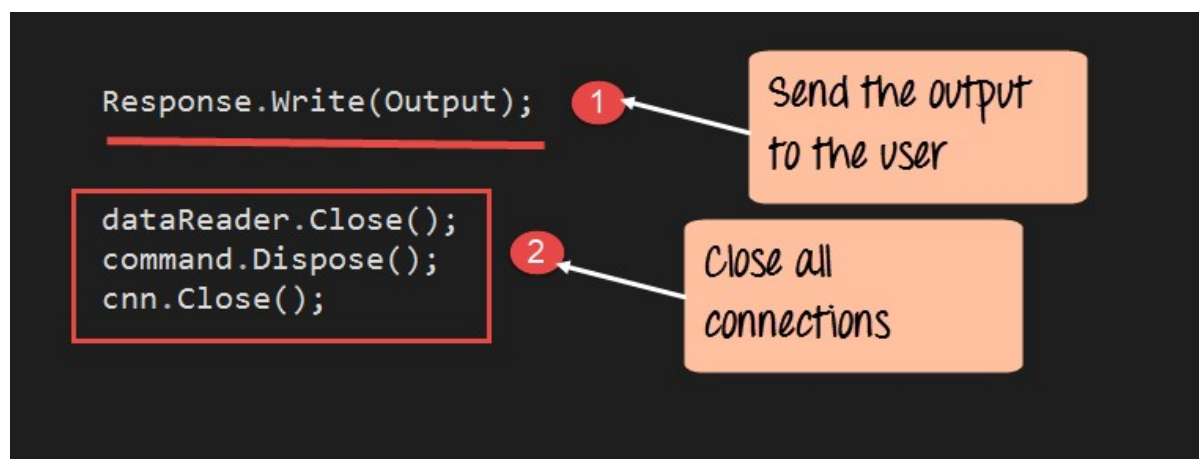
- The first part will be to construct our "select" statement. It will be used to read the data from the database.
- We will then execute the "select" statement against the database. This will fetch all the table rows accordingly.



Code Explanation:-

1. The first step is to create the following variables
 - **SqlCommand** – The 'SqlCommand' is a class defined within C#. This class is used to perform operations of reading and writing into the database. Hence, the first step is to make sure that we create a variable type of this class. This variable will then be used in subsequent steps of reading data from our database.
 - The **DataReader** object is used to get all the data specified by the SQL query. We can then read all the table rows one by one using the data reader.
 - We then define two string variables. One is "SQL" to hold our SQL command string. The next is the "Output" which will contain all the table values.
2. The next step is to actually define the SQL statement. This will be used against our database. In our case, it is "Select TutorialID, TutorialName from demotb". This will fetch all the rows from the table demotb.
3. Next, we create the command object which is used to execute the SQL statement against the database. In the SQL command, you have to pass the connection object and the SQL string.
4. Next, we will execute the data reader command, which will fetch all the rows from the demotb table.
5. Now that we have all the rows of the table with us, we need a mechanism to access the row one by one.
 - For this, we will use the 'while' statement.
 - The 'while' statement will be used to access the rows from the data reader one at a time.
 - We then use the 'GetValue' method to get the value of TutorialID and TutorialName.

Step 2) In the final step, we will just display the output to the user. Then we will close all the objects related to the database operation.



```

namespace DemoApplication
{
    public partial class Demo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            SqlCommand command;
            SqlDataReader dataReader;
            String sql, Output = " ";
            sql = "Select TutorialID,TutorialName from demotb";

            command = new SqlCommand(sql, cnn);

            dataReader = sqlquery.ExecuteReader();
            while (dataReader.Read())
            {
                Output = Output + dataReader.GetValue(0) + "-" +
dataReader.GetValue(1) + "<br>";
            }

            Response.Write(Output);
            dataReader.Close();
            command.dispose();
            conn.Close();

        }
    }
}

```

Code Explanation:-

1. We will continue our code by displaying the value of the Output variable. This is done using the Response.Write method.
2. We finally close all the objects related to our database operation. Remember this is always a good practice.

When the above code is set, and the project is run using Visual Studio, you will get the below output.

Output:-



From the output, you can clearly see that the program was able to get the values from the database. The data is then displayed in the browser to the user.

6. Database Operations

6.1 Insert into Database

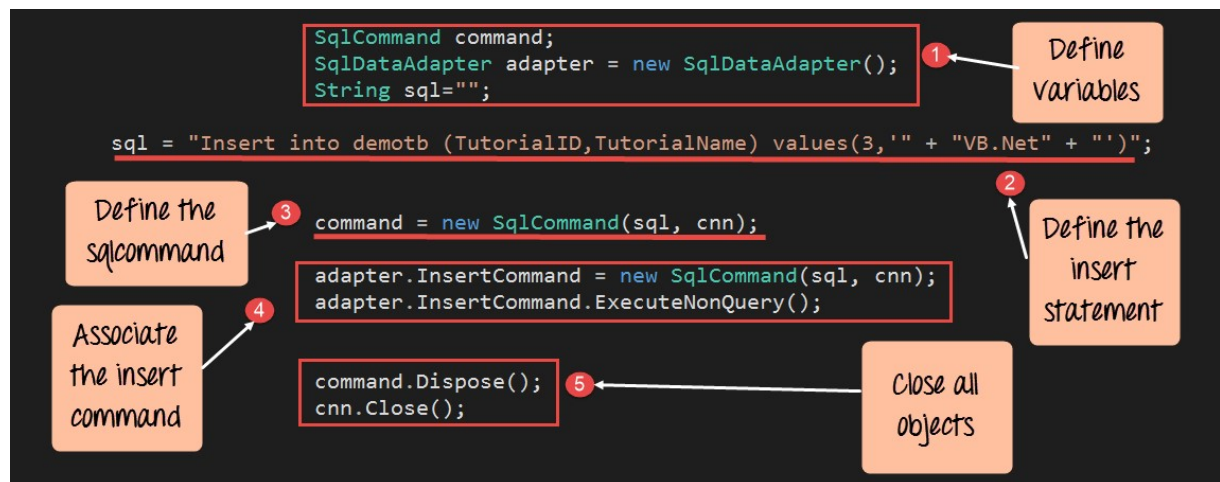
Just like Accessing data, C# has the ability to insert records into the database as well. To showcase how to insert records into our database, let's take the same table structure which was used above.

TutorialID	TutorialName
1	C#
2	ASP.Net

Let's change the code in our form, so that we can insert the following row into the table

TutorialID	TutorialName
3	VB.Net

So let's add the following code to our program. The below code snippet will be used to insert an existing record in our database.

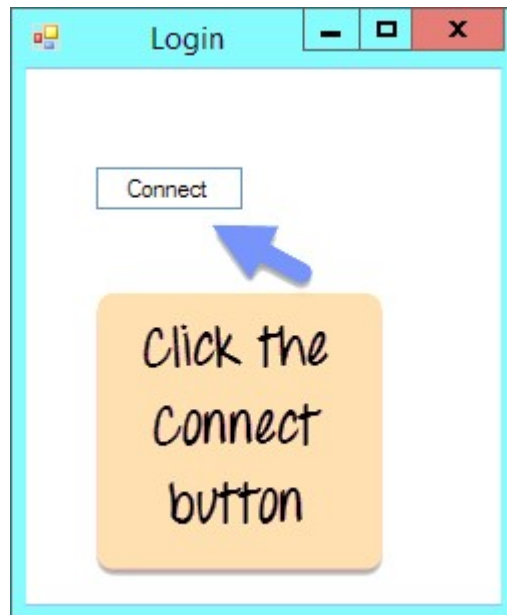


Code Explanation:-

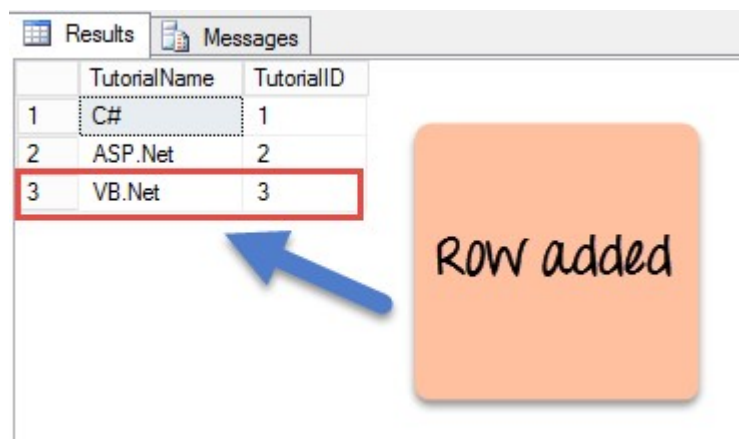
1. The first step is to create the following variables
 1. **SqlCommand** – This data type is used to define objects which are used to perform SQL operations against a database. This object will hold the SQL command which will run against our SQL Server database.
 2. The **DataAdapter** object is used to perform specific SQL operations such as insert, delete and update commands.
 3. We then define a string variable, which is "SQL" to hold our SQL command string.
2. The next step is to actually define the SQL statement which will be used against our database. In our case, we are issuing an insert statement, which will insert the record of TutorialID=1 and TutorialName=VB.Net
3. Next, we create the command object which is used to execute the SQL statement against the database. In the SQL command, you have to pass the connection object and the SQL string
4. In our data adapter command, we now associate the insert SQL command to our adapter. We also then issue the `ExecuteNonQuery` method which is used to execute the Insert statement against our database. The 'ExecuteNonQuery' method is used in C# to issue any DML statements against the database. By DML statements, we mean the insert, delete, and update operation. In C# , if you want to issue any of these statements against a table, you need to use the `ExecuteNonQuery` method.
5. We finally close all the objects related to our database operation. Remember this is always a good practice.

When the above code is set, and the project is executed using Visual Studio, you will get the below output. Once the form is displayed, click the Connect button.

Output:-



If you go to SQL Server Express and see the rows in the demotb table, you will see the row inserted as shown below



A screenshot of the SQL Server Express Results window. The window has two tabs: "Results" and "Messages". The "Results" tab is active, showing a table with two columns: "TutorialName" and "TutorialID". The table contains three rows of data. The third row, which contains "VB.Net" and "3", is highlighted with a red border. A blue arrow points from the highlighted row to an orange box on the right that contains the text "Row added".

	TutorialName	TutorialID
1	C#	1
2	ASP.Net	2
3	VB.Net	3

6.2 Update Database

Just like Accessing data, C# has the ability to update existing records from the database as well. To showcase how to update records into our database, let's take the same table structure which was used above.

TutorialID	TutorialName
1	C#
2	ASP.Net
3	VB.Net

Let's change the code in our form, so that we can update the following row. The old row value is TutorialID as "3" and Tutorial Name as "VB.Net". Which we will update it to "VB.Net complete" while the row value for Tutorial ID will remain same.

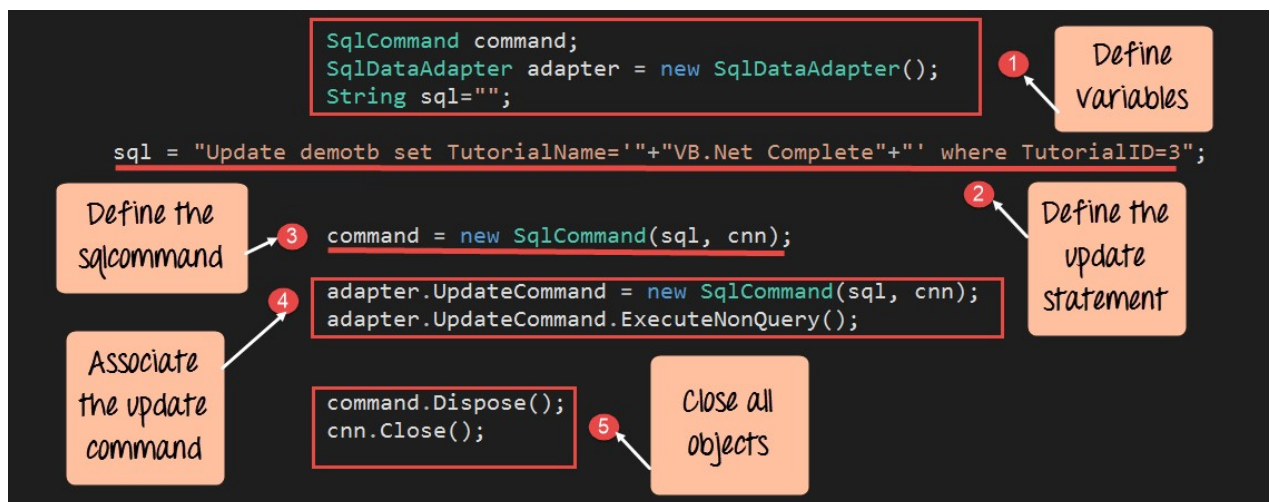
Old row

TutorialID	TutorialName
3	VB.Net

New row

TutorialID	TutorialName
3	VB.Net complete

So let's add the following code to our program. The below code snippet will be used to update an existing record in our database.

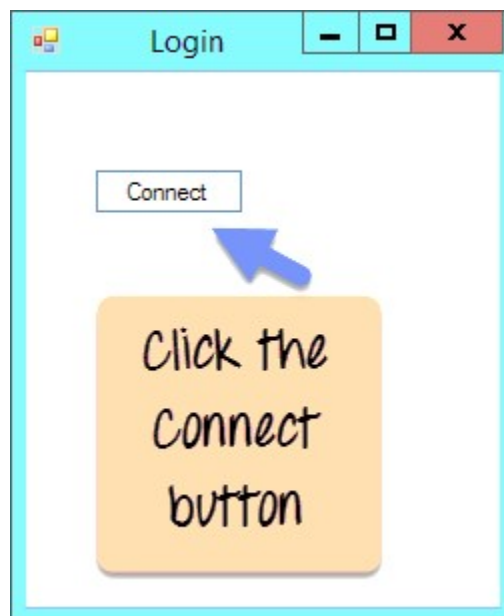


Code Explanation:-

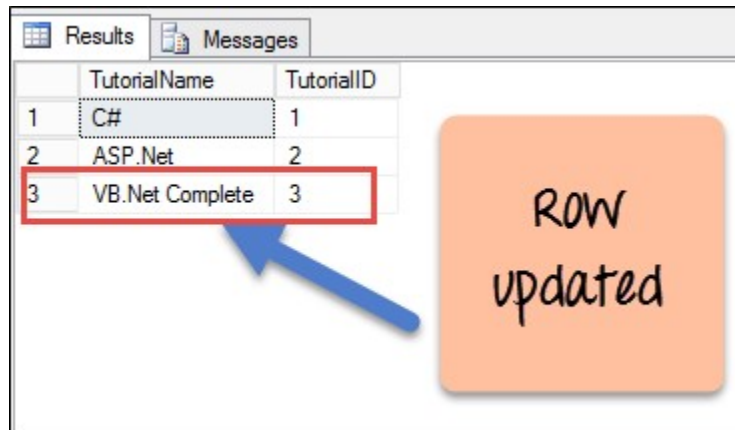
1. The first step is to create the following variables
 1. SqlCommand – This data type is used to define objects which are used to perform SQL operations against a database. This object will hold the SQL command which will run against our SQL Server database.
 2. The dataadapter object is used to perform specific SQL operations such as insert, delete and update commands.
 3. We then define a string variable, which is SQL to hold our SQL command string.
2. The next step is to define the SQL statement which will be used against our database. In our case we are issuing an update statement, this will update the Tutorial name to "VB.Net Complete" while the TutorialID is unchanged and kept as 3.
3. Next, we will create the command object, which is used to execute the SQL statement against the database. In the SQL command, you have passed the connection object and the SQL string.
4. In our data adapter command, we now associate the insert SQL command to our adapter. We also then issue the ExecuteNonQuery method which is used to execute the Update statement against our database.
5. We finally close all the objects related to our database operation. Remember this is always a good practice.

When the above code is set, and the project is executed using Visual Studio, you will get the below output. Once the form is displayed, click the Connect button.

Output:-



If you actually go to SQL Server Express and see the rows in the demotb table, you will see the row was successfully updated as shown below.



	TutorialName	TutorialID
1	C#	1
2	ASP.Net	2
3	VB.Net Complete	3

6.3 Deleting Records

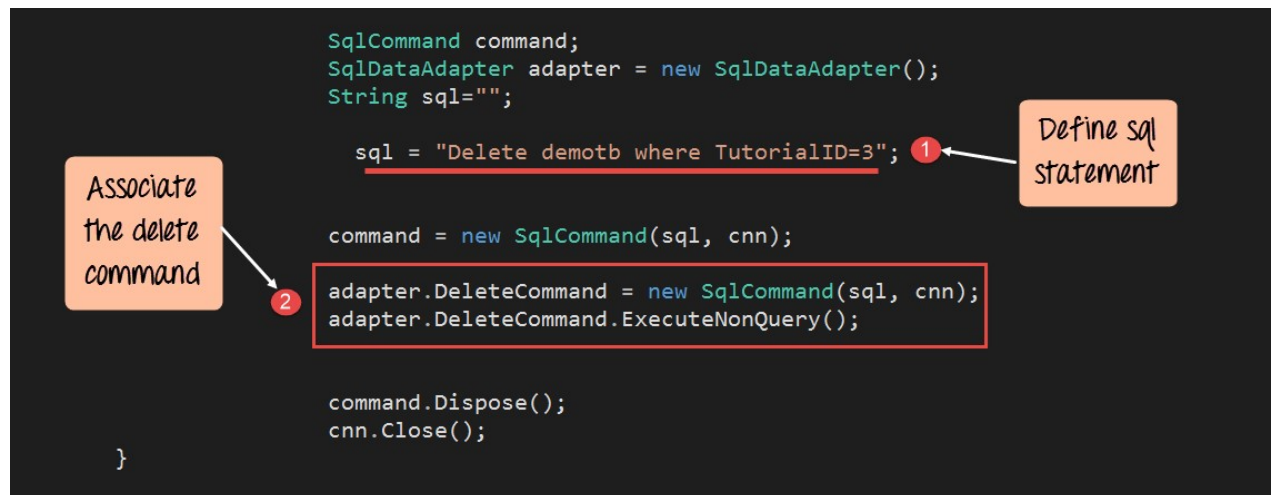
Just like Accessing data, C# has the ability to delete existing records from the database as well. To showcase how to delete records into our database, let's take the same table structure which was used above.

TutorialID	TutorialName
1	C#
2	ASP.Net
3	VB.Net complete

Let's change the code in our form, so that we can delete the following row

TutorialID	TutorialName
3	VB.Net complete

So let's add the following code to our program. The below code snippet will be used to delete an existing record in our database.

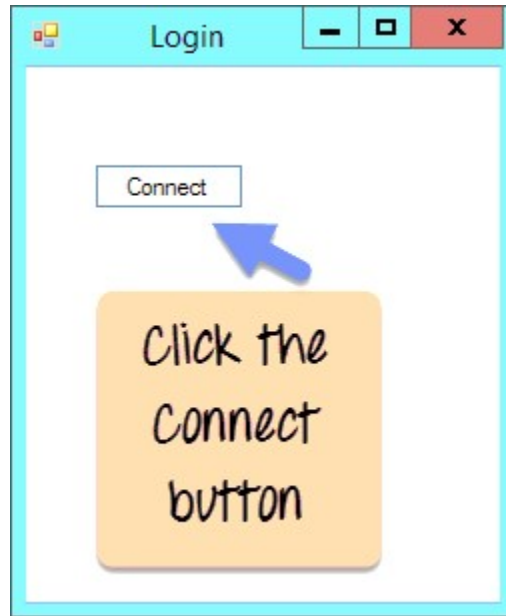


Code Explanation:-

1. The Key difference in this code is that we are now issuing the delete SQL statement. The delete statement is used to delete the row in the demotb table in which the TutorialID has a value of 3.
2. In our data adapter command, we now associate the insert SQL command to our adapter. We also then issue the `ExecuteNonQuery` method which is used to execute the Delete statement against our database.

When the above code is set, and the project is executed using Visual Studio, you will get the below output. Once the form is displayed, click the Connect button.

Output:-



If you actually go to SQL Server Express and see the rows in the demotb table, you will see the row was successfully deleted as shown below.

A screenshot of the SQL Server Enterprise Manager interface. The 'Results' tab is active, showing a table with two columns: 'TutorialName' and 'TutorialID'. The table contains two rows: one with 'C#' and '1', and another with 'ASP.Net' and '2'. A blue arrow points from an orange callout box labeled 'Row deleted' to the second row of the table.

	TutorialName	TutorialID
1	C#	1
2	ASP.Net	2

7. Data Bind Controls

ASP.NET allows powerful feature of data binding, you can bind any server control to simple properties, collections, expressions and/or methods. When you use data binding, you have more flexibility when you use data from a database or other means.

Data binding is binding controls to data from databases. With data binding we can bind a control to a particular column in a table from the database or we can bind the whole table to the data grid. Data binding provides simple, convenient, and powerful way to create a

read/write link between the controls on a form and the data in their application.

Data binding allows you to take the results of properties, collection, method calls, and database queries and integrate them with your ASP.NET code. You can combine data binding with Web control rendering to relieve much of the programming burden surrounding Web control creation. You can also use data binding with ADO.NET and Web controls to populate control contents from SQL select statements or stored procedures.

Data binding uses a special syntax: `<%# %>`

The `<%#`, which instructs ASP.NET to evaluate the expression. The difference between a data binding tags and a regular code insertion tags `<%` and `%>` becomes apparent when the expression is evaluated. Expressions within the data binding tags are evaluated only when the `DataBind` method in the Page objects or Web control is called. Data Bind Control can display data in connected and disconnected model.

Following are data bind controls in ASP.NET:

- Repeater Control
- DataGrid Control
- DataList Control
- GridView Control
- DetailsView
- FormView
- DropDownList
- ListBox
- RadioButtonList
- CheckBoxList
- BulletList
- etc.

To display backend result set (tuple collection) Repeater Control, DataGrid Control, DataList Control, GridView Control are used.

Repeater Control, DataList Control and FormView Control are unformatted controls.

DetailsView and ForView controls display single tuple result at a time.

DropDownList, ListBox, RadioButtonList, CheckBoxList and BulletList controls displays single column values.

7.1 Types of Data Binding

User can bind the data with the controls of the forms. This process is known as data binding. There are two types of data binding in ASP.NET known as simple data binding and declarative data binding.

Simple data binding

In simple data binding, the control is bounded to a data set. The properties of the control are used for binding with the value. Depending on the control to be bounded, the binding's property is set.

Declarative data binding

The process of binding a component like listbox, DataGrid, record list with the dataset is known as declarative binding. When there is more than one element in the database, the declarative binding is used.

Some of the controls used for the declarative data binding are listed below.

1. DataGrid: The data from multiple records is displayed using the DataGrid view. The DataSource property of the control is used for binding the specific element data.
2. ListBox: The data for a column from different dataset is displayed. The DataSource property is used for binding the control. The control binds to the specific element using the DisplayMember property.
3. ComboBox: The DisplayMember property is used for binding the control to the specific data element. The DataSource property is used for binding the control to the data source.

The following objects are needed for data binding in ASP.NET.

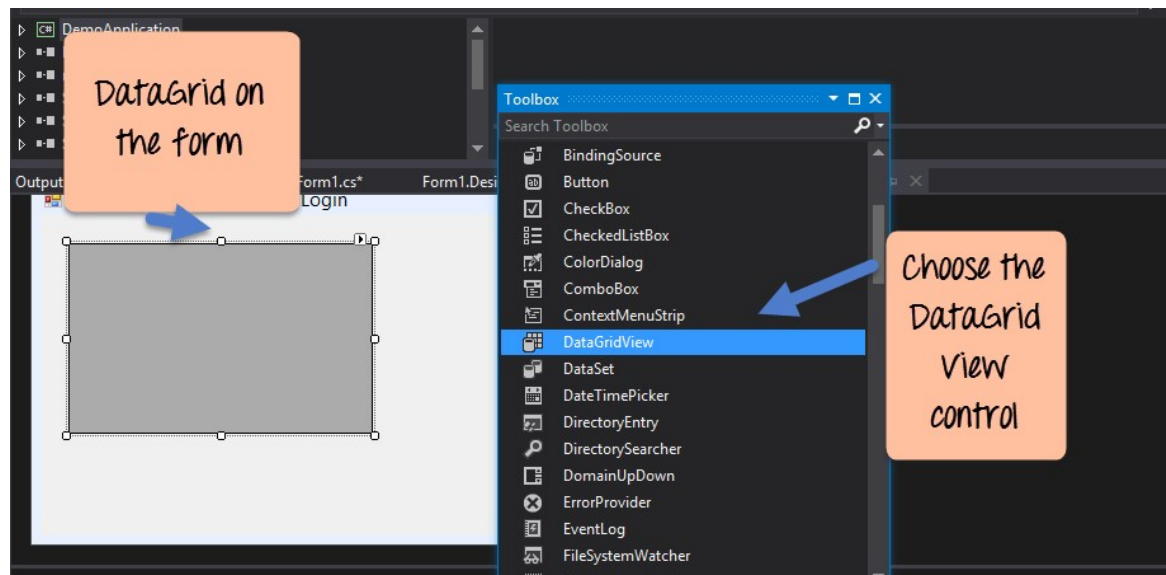
1. The data accessed from the database is stored in the dataset.
2. The data provider is used for accessing data through the command object
3. The data adapter is used for selecting, updating, inserting, deleting the data using commands.

7.2 Data Grid View

Data Grids are used to display data from a table in a grid-like format. When a user sees's table data, they normally prefer seeing all the table rows in one shot. This can be achieved if we can display the data in a grid on the form.

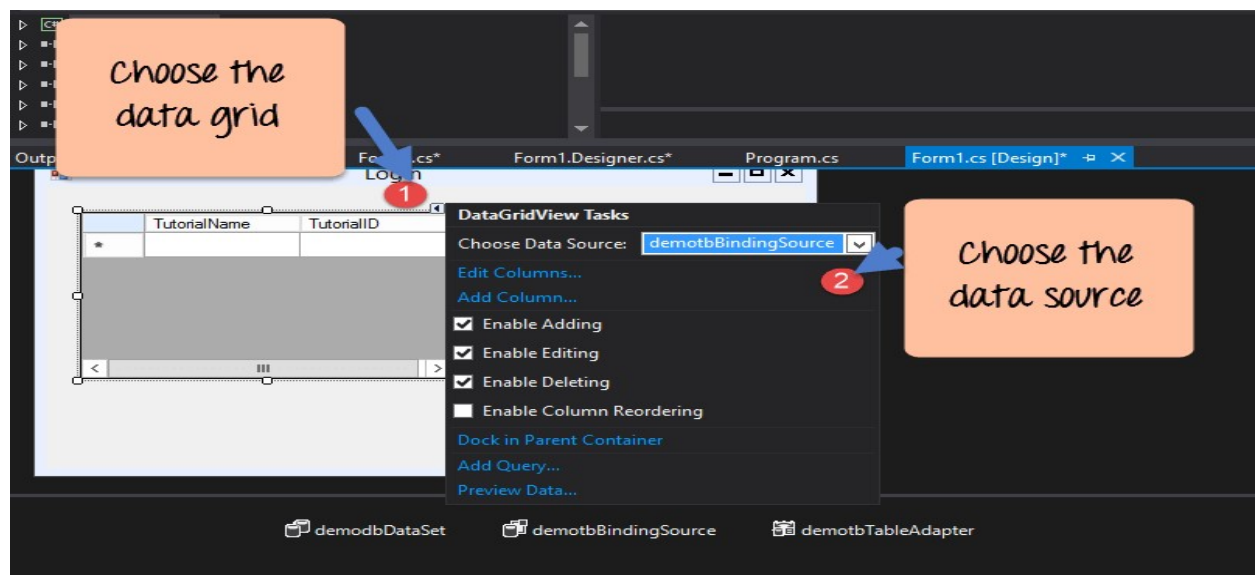
C# and Visual Studio have inbuilt data grids, this can be used to display data. Let's take a look at an example of this. In our example, we will have a data grid, which will be used to display the Tutorial ID and Tutorial Name values from the demotb table.

Step 1) Drag the DataGridView control from the toolbox to the Form in Visual Studio. The DataGridView control is used in Visual Studio to display the rows of a table in a grid-like format.



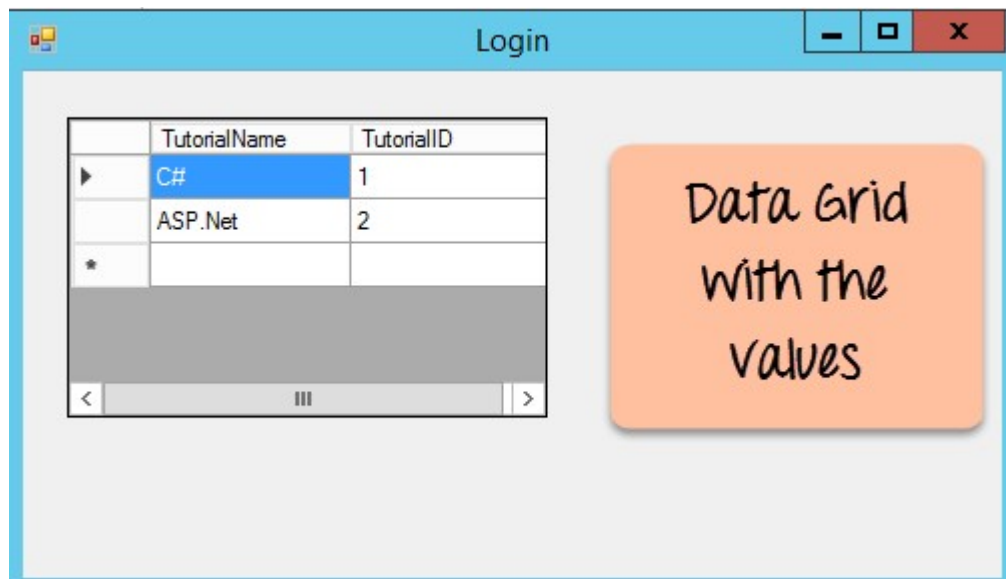
Step 2) In the next step, we need to connect our data grid to the database. In the last section, we had created a project data source. Let's use the same data source in our example.

1. First, you need to choose the grid and click on the arrow in the grid. This will bring up the grid configuration options.
2. In the configuration options, just choose the data source as demotbBindingSource which was the data source created in the earlier section.



If all the above steps are executed as shown, you will get the below-mentioned output.

Output:-



From the output, you can see that the grid was populated by the values from the database.

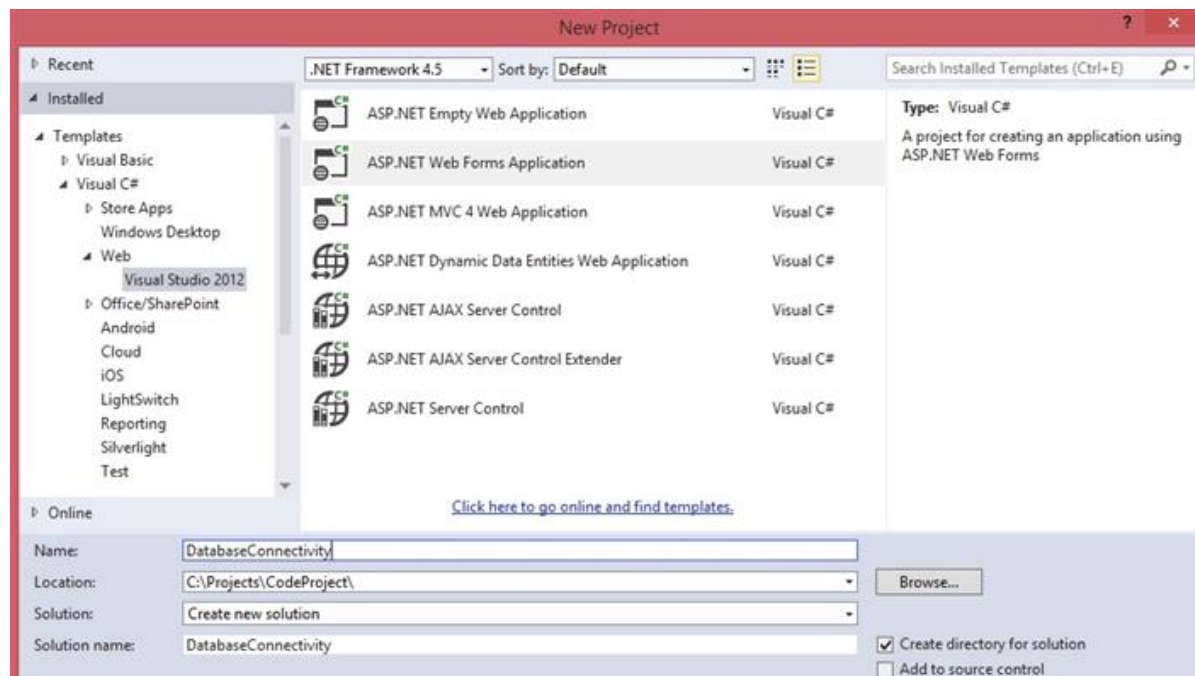
8. Project in ASP. Net using Database

In every language, the most important part is database connectivity. I have covered all the aspects of database connectivity in the form of Create Operation. It is useful for beginners and will help them understand the different steps in database connectivity. It will take 5 easy steps for connectivity.

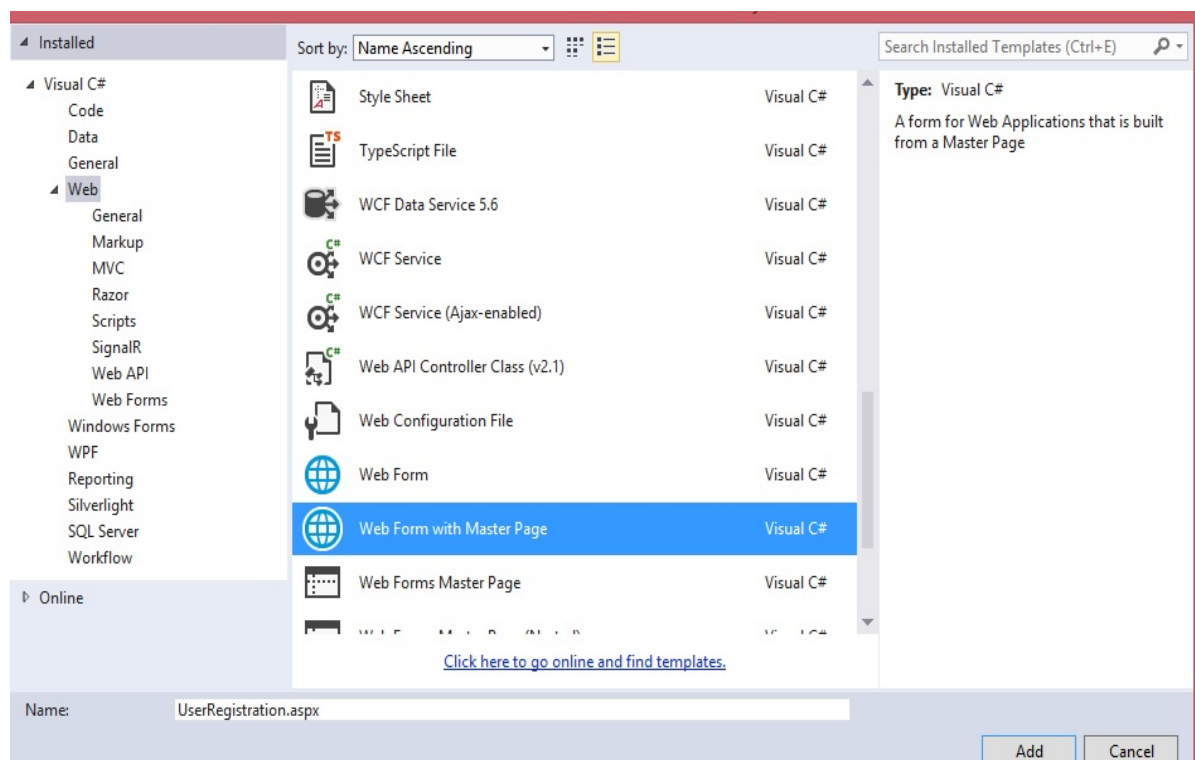
There can be several ways to connect to a database through C#. Here, I have covered a simple ADO.NET approach and I have done this through Stored Procedure and Grid View. So it will be good to have a little knowledge of SQL server and ASP.NET prior to going for this. I have used Visual Studio 2012 and SQL Server 2012, but the code will be the same for all the Visual Studio and SQL Server versions.

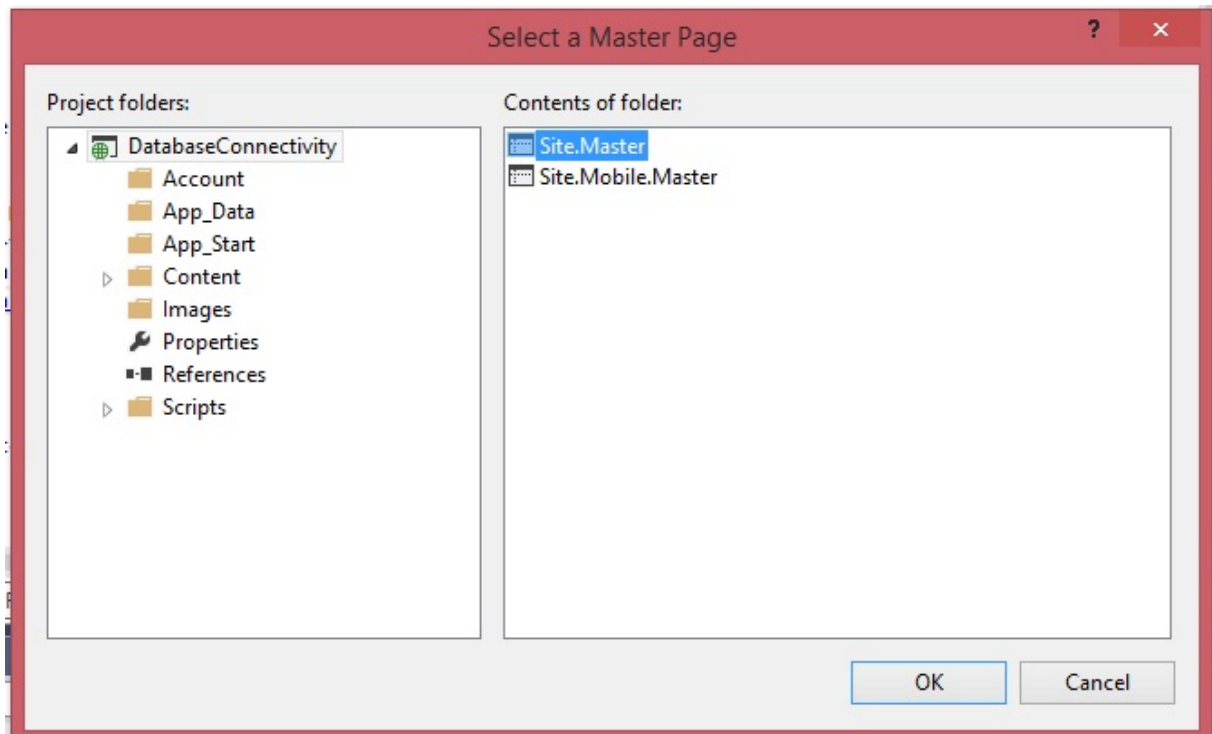
8.1 Using the Code

Add a new project and give a name to it (In my case, the name is **DatabaseConnectivity**) as shown below:



Add a new Web Form with Master Page and give a name (In my case, the name is *UserRegistration.aspx*) and select Master Page as below:





The next step is to create a Registration Form as below and add the below code inside Content Place holder whose **Id** is **Content3** as below:

```
<asp:Content ID="Content3"
ContentPlaceHolderID="MainContent" runat="server">
    <table>
        <tr>
            <td>
                Name
            </td>
            <td>
                <asp:TextBox ID="txtName"
                    runat="server"
                    required="true"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>
                Email
            </td>
            <td>
                <asp:TextBox ID="txtEmail"
                    runat="server" required="true"
                    type="Email"></asp:TextBox>
            </td>
        </tr>
    </table>
```

```

</tr>
<tr>
    <td>
        Password
    </td>
    <td>
        <asp:TextBox ID="txtPassword"

        runat="server" required="true"

        type="Password"></asp:TextBox>
    </td>
</tr> <tr>
    <td>
        Confirm Password
    </td>
    <td>
        <asp:TextBox ID="txtConfirmPassword"

        runat="server" required="true"

        type="Password"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>
        Address
    </td>
    <td>
        <asp:TextBox ID="txtAddress"

        runat="server" required="true"

        TextMode="MultiLine"></asp:TextBox>
    </td>
</tr>
<tr>
    <td></td>
    <td>
        <asp:Button ID="btnSubmit"

        runat="server" Text="Submit" />
    </td>
</tr>
</table>
</asp:Content>

```

Run the application after setting this page as a Start Page (Right click on *UserRegistration.aspx* and click on Set as Start page). Once run, we will get a page as below.

your logo here

Register

Log in

Home

About

Contact

Name

Email

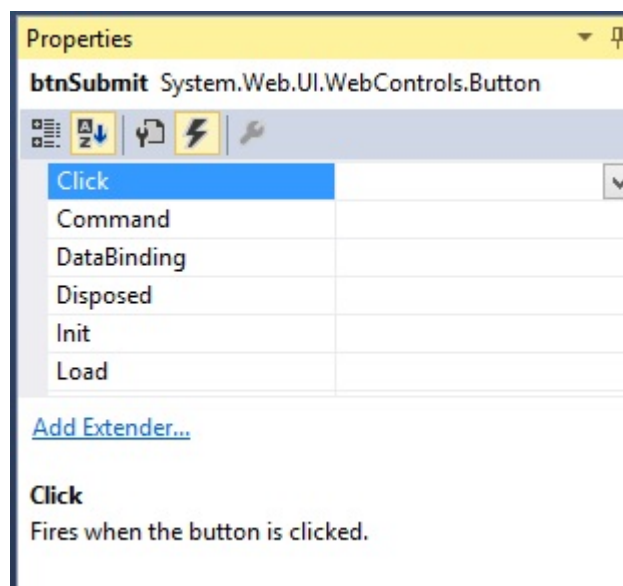
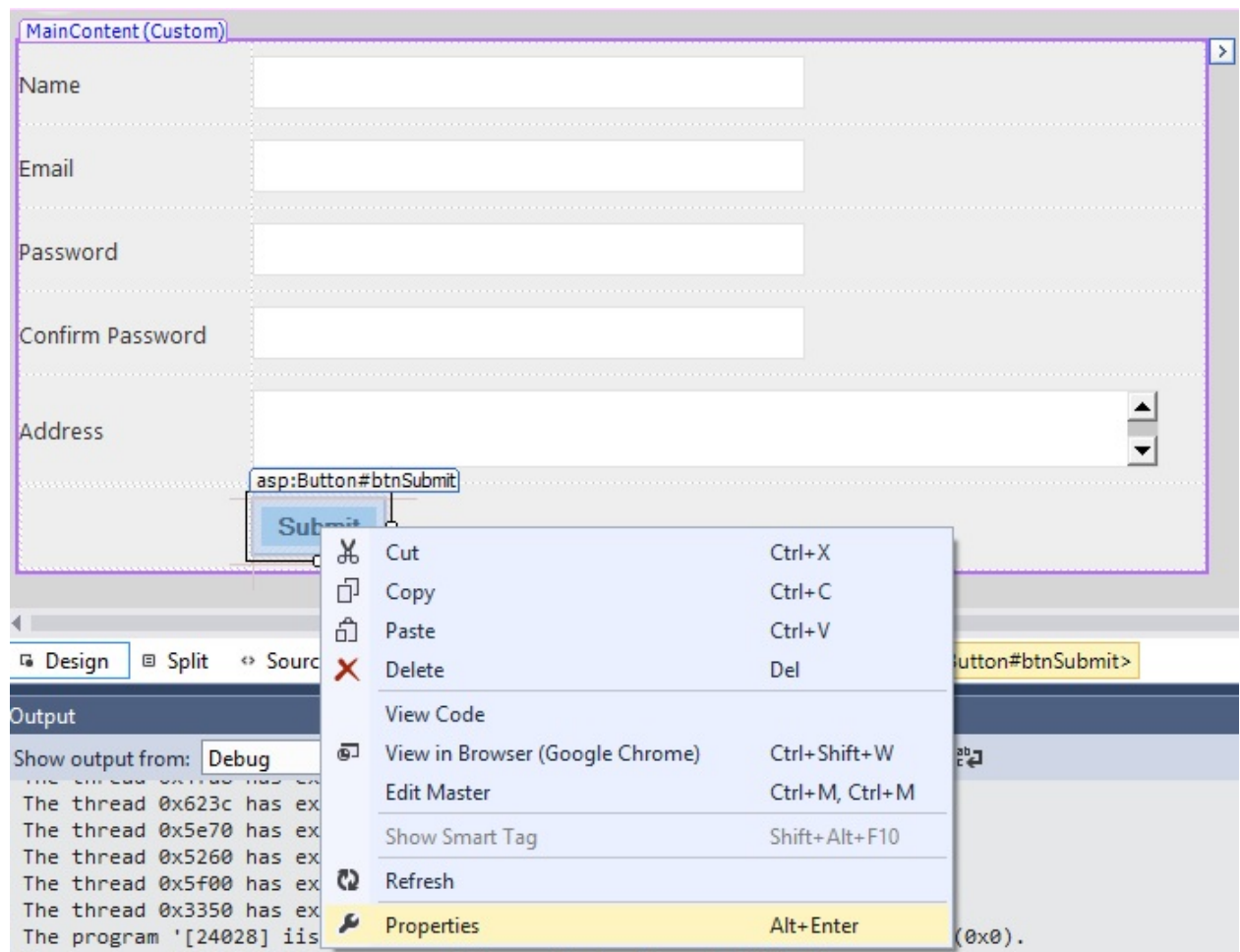
Password

Confirm Password

Address

Submit

The next step is to generate a Button Click Event. For this, just right click on the Button and go to Properties and double click on click event (or directly double click on the button).



Once you will double click on the button, an event will generated on the code behind page as below:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
}
}
```

Whatever you want to perform when you will click on Button, you will write inside this click event. In our case, we will do database connectivity here and will save all the values in database. For this, I created a database named "Database Connectivity" and a table inside it named "**tblUser**" with the below script:

```
CREATE TABLE [dbo].[tblUser](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) NULL,
    [Email] [nvarchar](50) NULL,
    [Password] [nvarchar](50) NULL,
    [Address] [nvarchar](50) NULL,
    CONSTRAINT [PK_tblUser] PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, _
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON
[PRIMARY]
) ON [PRIMARY]
```

This script will create a table in database having identity column in ID with Primary key. I will use Stored Procedure for this, so create a stored procedure as below:

```
create proc spInsertUser
@Name Nvarchar(50),
@email Nvarchar(50),
@Password Nvarchar(50),
@Address Nvarchar(50)
as
Insert into tblUser(Name,Email>Password,Address)
values(@Name,@Email,@Password,@Address)
```

In Code behind, first retrieve all the text box values:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    string name = txtName.Text;
    string email = txtEmail.Text;
    string password = txtPassword.Text;
    string address = txtAddress.Text;
}
```

Now, we will connect to database and will save these values in **tblUser** table. For this, first add two below namespaces on page:

```
using System.Data;
using System.Data.SqlClient;
```

Now, we will save these values in database in 5 steps:

Step 1: Make a Connection

To make a connection with database, ADO.NET provides a class named **SqlConnection**. So, we will create an object of this class and will pass the connection string:

```
SqlConnection con = new SqlConnection
("Data Source=.;Initial Catalog =
DatabaseConnectivity;Trusted_Connection=true;");
```

con is the object of SQL Connection Class. In Connection String, the meanings of different attributes are:

- **Data Source:** In Data Source, we will provide the Machine Name where we create the database. (.) Means Database is in your local Machine.
- **Initial Catalog:** Initial Catalog is the database Name (In my case, it is **DatabaseConnectivity**).
- **Trusted_Connection:** **Trusted_Connection** should be **true** if you are using window authentication while connecting to database. If you are using SQL authentication, you will have to pass userid and password.

Step 2: Open Connection

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

Step 3: Prepare Command

To prepare a command, ADO.NET gives us a class named **SqlCommand** which we will use as below:

```
SqlCommand com = new SqlCommand(); // Create a object of SqlCommand class
com.Connection = con; //Pass the connection object to Command
com.CommandType = CommandType.StoredProcedure; // We will use stored
procedure.
com.CommandText = "spInsertUser"; //Stored Procedure Name
```

Step 4: Add Parameters If Any

Add Parameters if you have any to your command object as below:

```
com.Parameters.Add("@Name", SqlDbType.NVarChar).Value = name;
com.Parameters.Add("@Email", SqlDbType.NVarChar).Value = email;
com.Parameters.Add("@Password", SqlDbType.NVarChar).Value = password;
com.Parameters.Add("@Address", SqlDbType.NVarChar).Value = address;
```

Step 5: Execute Your Command

Execute your command as below:

```
com.ExecuteNonQuery();
```

The complete code for this is as follows:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    string name = txtName.Text;
    string email = txtEmail.Text;
    string password = txtPassword.Text;
    string address = txtAddress.Text;
    SqlConnection con = new SqlConnection
        ("Data Source=.;Initial Catalog =
DatabaseConnectivity;Trusted_Connection=true;");
    SqlCommand com = new SqlCommand();

    try
    {
        con.Open();
        // Create a object of SqlCommand class
        com.Connection = con; //Pass the connection object to Command
        com.CommandType = CommandType.StoredProcedure; // We will use
        stored procedure.
        com.CommandText = "spInsertUser"; //Stored Procedure Name

        com.Parameters.Add("@Name", SqlDbType.NVarChar).Value = name;
        com.Parameters.Add("@Email", SqlDbType.NVarChar).Value = email;
        com.Parameters.Add("@Password", SqlDbType.NVarChar).Value =
password;
        com.Parameters.Add("@Address", SqlDbType.NVarChar).Value = address;

        com.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
    }
}
```

```

finally
{
    con.close();

}
}

```

Run the application and fill the form and click on submit:

your logo here

Name	<input type="text" value="Vijay"/>
Email	<input type="text" value="vijayrana1091@gmail.com"/>
Password	<input type="password" value="..."/>
Confirm Password	<input type="password" value="..."/>
Address	<input type="text" value="Delhi"/>
<input type="button" value="Submit"/>	

Results		Messages			
	ID	Name	Email	Password	Address
1	1	Vijay	vijayrana1091@gmail.com	123	Delhi

Note: I have created a project with insert operation in a database if you want to perform more operations on database then follow the database operations topic from the same unit.

References:

- <https://www.guru99.com/insert-update-delete-asp-net.html>
- <https://www.c-sharpcorner.com/uploadfile/puranindia/data-bind-controls-in-Asp-Net/>
- <https://www.go4expert.com/articles/data-binding-aspnet-t34155/>
- <https://www.codeproject.com/Tips/1038536/Day-Database-Connectivity-ASP-NET-Csharp-with-SQL>