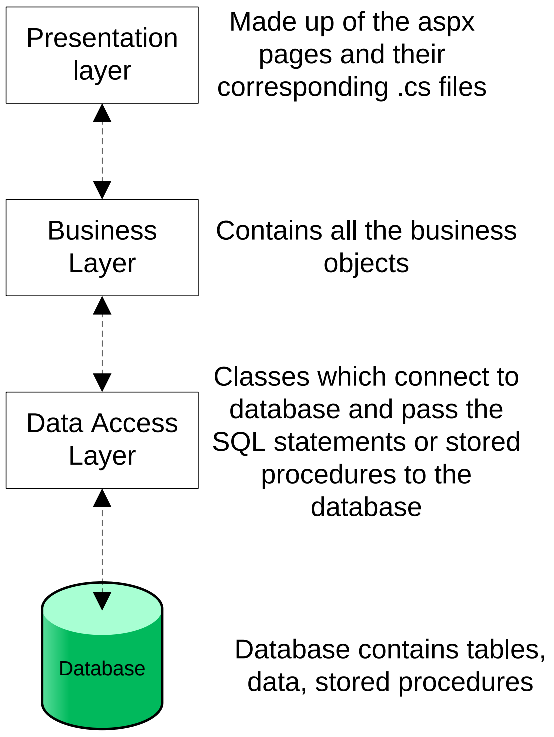
Creating a data access layer – Part 1

Contents

Creating a data access layer is not as intuitive as creating the user interface. You need to think about what your application needs to do with the database. First things that come to mind are:

- Select data
- Insert data
- Update data
- Delete data

These are typically what you need to be able to do to your database.



Presentation layer
Made up of the aspx pages and their corresponding .cs files
Business Layer
Contains all the business objects
Data Access Layer
Classes which connect to database and pass the SQL statements or stored procedures to the database
Database
Database contains tables, data, stored procedures

In the above diagram the part we are focussing on is the Data Access Layer which sits just above the database.

Review of ADO.NET

To understand what is required of the data access layer we need to have an understanding of ADO.NET. ADO.NET is a Data access technology built into the .NET framework. ADO.NET provides a wide range of classes to work with databases. The System.Data.SqlClient namespace contains the classes for connecting to Microsoft SQL server.

Data base connections

To be able to execute SQL statements you need to first open a connection to the database. This is achieved by the sqlConnection object.

⇘ Example

```
SqlConnection objConnection;
objConnection = new SqlConnection(connectionString);
```

Where connectionString is the connection string of your database.

Connection strings

Contain the details needed to connect to database

- Name of database server
- Name of database
- Security access
- Connection timeout

It can (and should) be setup in the web.config file. As most of your application will use the same connection.

⇘ Example:

```
strConnection = “server=shep\navy3757; database=northwind; integrated security=SSPI;
connect timeout=30 ”
```

or

```
strconnection = "Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\NORTHWND.MDF;Integrated
Security=True;User Instance=True"
```

depending on if it is a SQLServer or SQLExpress connection.

The connectionString should actually be stored in the web.config file and read from that file:

Web.config file:

```
<connectionStrings>
      <add name="ConnectionString" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\NORTHWND.MDF;Integrated
Security=True;User Instance=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Code behind:

```
string strConnection;

strConnection =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
```

TIP: The connection string can be generated for you automatically by using the SQLDataSource wizard.

Opening a connection

The connection string is just a string it just holds the value of the database connection but it does not actually connect to the database. To connect to the database you need to use the connection object.

To open a connection you need to:

1. Set up connection string
2. Create connection object
3. Open connection using open() method
4. Every opened connection must eventually be closed by using the close() method

⇘ Example

```
//set up connection string
string strConnection;

strConnection =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;

//set up connection object
SqlConnection objConnection;
objConnection = new SqlConnection(strConnection);
```

```
//open connection
objConnection.Open();

//close connection
objConnection.Close();
```

Executing an SQL statement

The command object is used to execute SQL statements on the database. It need to be configured so that it knows 2 things:

- The SQL statement (or stored procedure name) it needs to execute
- The database it needs to execute it at. The database is already supplied in the connection object.

Example:

```
//set up connection string
string strConnection;

strConnection =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;

//set up connection object
SqlConnection objConnection;
objConnection = new SqlConnection(strConnection);

//open connection
objConnection.Open();

//set up SQL to be executed
string strSQL;
strSQL = "select categoryID, categoryName from categories";

//set up command object
SqlCommand objCommand = new SqlCommand(strSQL, objConnection);

//close connection
objConnection.Close();
```

The above code sets up the command object so it is ready to excute but at this stage it has not yet executed.

The dataReader object

The dataReader object can be used to store data retrieved from database. It contains a Read() method used to open the dataReader for reading. The data reader is used for read only forward only data access.

Example:

```
//set up connection string
string strConnection;

strConnection =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;

//set up connection object
SqlConnection objConnection;
objConnection = new SqlConnection(strConnection);

//open connection
objConnection.Open();

//set up SQL to be executed
string strSQL;
strSQL = "select categoryID, categoryName from categories";

//set up command object
SqlCommand objCommand = new SqlCommand(strSQL, objConnection);
```

```
//set up date reader to hold data
SqlDataReader objDataReader;

//execute SQL and store in data reader
objDataReader = objCommand.ExecuteReader();

//set datasource for grid view to
//data reader just populated
gvCategories.DataSource = objDataReader;
gvCategories.DataBind();

//close connection
objConnection.Close();
```

Activity 1. Populate a gridview from codebehind

default.aspx

In this exercise we will use sqlClient classes to connect to the northwind database and select the category id and name from the category table then bind the data to the gridview.

Add a gridview control to the aspx page and name it gvCategories. Add the following code to the codebehind:

At the top of the file we need to include the following name space:

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

In the page_load sub add the following code:

```
//set up connection string
        string strConnection;

        strConnection =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;

        //set up connection object
        SqlConnection objConnection;
        objConnection = new SqlConnection(strConnection);

        //open connection
        objConnection.Open();

        //set up SQL to be executed
        string strSQL;
        strSQL = "select categoryID, categoryName from categories";

        //set up command object
        SqlCommand objCommand = new SqlCommand(strSQL, objConnection);

        //set up date reader to hold data
        SqlDataReader objDataReader;

        //execute SQL and store in data reader
        objDataReader = objCommand.ExecuteReader();

        //set datasource for grid view to
        //data reader just populated
        gvCategories.DataSource = objDataReader;
        gvCategories.DataBind();

        //gvCategories.DataSource = objCommand.ExecuteReader();
        ////bind datato gridview
        //gvCategories.DataBind();

        //close connection
        objConnection.Close();
```

Note instead of the lines:

```
//set up date reader to hold data
```

```
SqlDataReader objDataReader;

//execute SQL and store in data reader
objDataReader = objCommand.ExecuteReader();

//set datasource for grid view to
//data reader just populated
gvCategories.DataSource = objDataReader;
gvCategories.DataBind();
```

You could have written:

```
 gvCategories.DataSource = objCommand.ExecuteReader();
//bind data to gridview
 gvCategories.DataBind();
```

Activity 2. Display product names and price

products.aspx

Using the same method as above this time display the product name and unit price.

What changed between activity 1 and activity 2?

Most likely the name of the gridview to something like gvProducts

The SQL statement

The rest stayed the same.

Data Access layer

What we have just gone through can be achieved with no code at all. The aim of the previous exercise was to introduce ADO.NET. The code we wrote is the code which is actually executed for us when we use the SQLDataSource control to obtain a connection and connect to the database. The code we created is still 2 tiers we are accessing the database from the code behind of the aspx page. The code behind should only deal with presentation logic. Now if we had many pages in our application all needing to connect to the same database and execute different SQL statements there is a lot of code that would be replicated in each page. You would still need to set up your connection string:

```
string strConnection;

strConnection =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
```

and the connection object:

```
SqlConnection objConnection;
objConnection = new SqlConnection(strConnection);

//open connection
objConnection.Open();
```

The SQL statement would be different depending on what you want brought back:

```
//set up SQL to be executed
string strSQL;
strSQL = "select categoryID, categoryName from categories";
```

The command object still needs to be set up:

```
//set up command object
SqlCommand objCommand = new SqlCommand(strSQL, objConnection);
```

And the SQL executed:

```
//set up date reader to hold data
SqlDataReader objDataReader;

//execute SQL and store in data reader
objDataReader = objCommand.ExecuteReader();
```

The databinding has to happen in the presentation layer because that is the only layer that deals with the user interface.

```
gvCategories.DataSource = objDataReader;
gvCategories.DataBind();
```

And the connection needs to be closed each time:

```
//close connection
 objConnection.Close();
```

If we take the replicated code and place it inside a class we don't need to copy it each time we need it rather we need to just call the code.

Creating the sqlHelper class

We will create a class named sqlHelper which will handle all the database functionality. The code should be placed in the App_code folder.

Activity 3. Create SQLHelper

sqlHelper.cs1

Create the app_code folder.

Create a new class in the app_code folder and name it sqlHelper

Add the following line at the top of the file:

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

The above line includes the sqlClient and the data namespaces to the file so that the classes in the sqlClient and data namespaces are available from within this file.

Each time we want to connect to the database (i.e. make use of the sqlHelper class) we need to get the connection string from the web.config file. So it is a good idea to store the connectionstring in a variable inside the class. Since the connection string is not needed outside of this class we can set it up as a private variable:

```
private string _Conn;
```

Whenever we need to use the sqlHelper class it needs to read the connection string from the web.config file so the reading of this file should be done when the sqlHelper class is used.

The constructor of a class is executed whenever an instance of the class is created. So it makes sense to place the code in the constructor.

```
public sqlHelper()
{
        //get connection string from web.config file
        _Conn =
ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;
}
```

So now we have the connection string read each time the class is instantiated and stored in a private variable called _Conn. This private variable will be available to the class sqlHelper but not outside of the class.

Next we need to create a method that will create a connection and command object and execute the SQL. This method will return a dataReader. This method needs to be accessible from outside of the class so it needs to be defined as a public method.

```
public SqlDataReader executeSQL(string strSQL)
{
        //create connection object get connection string from private
        //variable _Conn
        SqlConnection objConnection = new SqlConnection(_Conn);

        //create command object
        //get SQL to execute from the strSql parameter
        //passed as input to this function
        SqlCommand objCommand = new SqlCommand(strSQL, objConnection);

        //open database connection
        objConnection.Open();

        //execute SQL and return dataReader
        return objCommand.ExecuteReader(CommandBehavior.CloseConnection);
}
```

Note the line

```
    return objCommand.ExecuteReader(CommandBehavior.CloseConnection);
```

This line does a lot it executes the SQL statement using the method executeReader from the command object. The parameter passed in the brackets CommandBehavior.CloseConnection specifies that the connection to the database should be closed as soon as the data has been read. This means we do not need to include a objConnection.close() statement. The commandBehavior belongs to the data namespace which is why we needed to include it at the top of the file. The return key word returns the dataReader to the calling code.

Activity 4. Use the sqlHelper class

categoriesWithClass.aspx

Now that we have a class containing all the code we need to be able to execute some SQL we can call the executeReader method from anywhere within the web application. The following code reproduces the same output as the first exercise we did but this time using the newly created class.

Create a web page and place a gridview on it. Name it gvCategories from the code behind add the following code to the page_load sub:

```
protected void Page_Load(object sender, EventArgs e)
{
        //create sqlHelper object
        sqlHelper objSqlHelper = new sqlHelper();

        //create SQL statement that needs to be passed
        string strSQL;
        strSQL = "Select categoryID, categoryName from categories";

        //populate gridview
        gvCategories.DataSource = objSqlHelper.executeSQL(strSQL);
        gvCategories.DataBind();
}
```

You can format the gridview the same way as you normally would when we were setting up the SQL code in a sqlDataSource control.

Executing a query that returns a single value

In some cases your SQL statement will only return back a single value (a scalar). There is a method of the command object which executes an SQL statement and returns back a single value. The method is executeScalar to cater for the need to execute SQL statements that only return back a single value we can add a new method to the sqlHelper class.

Activity 5. executeScalar

sqlHelper.cs2

Add the following code to the sqlHelper class

```
public object scalarSQL(string strSQL)
    {
        //create connection object get connection string from private
        //variable _Conn
        SqlConnection objConnection = new SqlConnection(_Conn);

        //create command object
        //get SQL to execute from the strSql parameter
        //passed as input to this function
        SqlCommand objCommand = new SqlCommand(strSQL, objConnection);

        //create a variable that will hold the return value
        object objRetValue;

        //open database connection
        objConnection.Open();

        //execute SQL
        objRetValue = objCommand.ExecuteScalar();

        //close connection
        objConnection.Close();

        //execute SQL and return value
        return objRetValue;
    }
```

Activity 6. Using the executeScalar

usingExecuteScalar.aspx

To try out the newly created method create a web page with following on it:

```
<div>
At the moment there are
<asp:Label ID="lblCount" runat="server"></asp:Label>
products in the database
</div>
```

Add the following code to the code behind page:

```
protected void Page_Load(object sender, EventArgs e)
{
        //create sqlHelper object
        sqlHelper objSqlHelper = new sqlHelper();

        //create SQL statement that needs to be passed
        string strSQL;
        strSQL = "Select count(*) from products";

        int intProductCount;

        //execute SQL and store result in an integer
        intProductCount = (int)objSqlHelper.scalarSQL(strSQL);

        //display the result;
        lblCount.Text = intProductCount.ToString();
}
```

Executing an insert, update or delete

Executing an insert update or delete statement does not actually bring data back. For this we need to use the executeNonQuery method of the command object. The executeNonQuery method of the command object does actually return a value. For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command. For CREATE TABLE and DROP TABLE statements, the return value is 0. For all other types of statements, the return value is -1.

Activity 7. executeNonQuery

sqlHelper.cs3

Add the following code to the sqlHelper class:

```
public int NonQuerySQL(string strSQL)
    {
        //create connection object get connection string from private
        //variable _Conn
        SqlConnection objConnection = new SqlConnection(_Conn);

        //create command object
        //get SQL to execute from the strSql parameter
        //passed as input to this function
        SqlCommand objCommand = new SqlCommand(strSQL, objConnection);

        //create a variable that will hold the return value
        int intRetValue;

        //open database connection
        objConnection.Open();

        //execute SQL
        intRetValue = objCommand.ExecuteNonQuery();

        //close connection
        objConnection.Close();

        //execute SQL and return value
        return intRetValue;
    }
```

Activity 8. Executing an insert statement

insertCategory.aspx

In this exercise we will insert a new category. Create a new web page and add the following code:

```
<div>
Insert a category<br />
<asp:Label ID="lblResult" runat="server"></asp:Label>
many rows were inserted
</div>
```

Now add the following code to the codebehind:

```
protected void Page_Load(object sender, EventArgs e)
{
        //create SQL helper object
        sqlHelper objSqlHelper = new sqlHelper();

        //create SQL statement that needs to be passed
        string strSQL;
        strSQL = "insert into categories(categoryName, description)
values('testCategory', 'testDescription')";

        int intResult;

        //execute SQL and store result in an integer
        intResult = (int)objSqlHelper.NonQuerySQL(strSQL);

        //display the result;
        lblResult.Text = intResult.ToString();
}
```

So what is wrong with this ??? We are hard coding the values the values should be coming from the user.

Passing parameters

For each of the above 3 methods we should be able to pass parameters in for example we may want to return a list of products that belong to a category. We would need to pass the category id to the executeReader method. Furthermore we can't predict how many parameters will be passed what if we then want to display all products that are part of a particular category and over a certain price. We would need to pass 2 parameters in this case the category ID and the price. Since we can't predict how many parameters need to be passed we can pass a collection or an array of parameters.

Activity 9.   Creating an executeReader that takes in parameters

sqlHelper.cs4

Add the following code to the sqlHelper class:

```
public SqlDataReader executeSQL(string strSQL, SqlParameter[] parameters)
    {
        //create connection object get connection string from private
        //variable _Conn
        SqlConnection objConnection = new SqlConnection(_Conn);

        //create command object
        //get SQL to execute from the strSql parameter
        //passed as input to this function
        SqlCommand objCommand = new SqlCommand(strSQL, objConnection);

        //fill parameters
        fillParameters(objCommand, parameters);

        //open database connection
        objConnection.Open();

        //execute SQL and return dataReader
        return objCommand.ExecuteReader(CommandBehavior.CloseConnection);
    }

    private void fillParameters(SqlCommand objCommand, SqlParameter[] parameters)
    {
        int i;
        //for each parameter passed in add it to the commad object
        //parameters collection
        for (i = 0; i < parameters.Length; i++)
        {
            objCommand.Parameters.Add(parameters[i]);
        }
    }
```

Activity 10. Passing parameters

productsByCategory.aspx

Create the following web page:

```
<div>
Select the Category:<br />
<asp:DropDownList ID="ddlCategory" runat="server">
</asp:DropDownList><br />
<br />
<asp:Button ID="btnView" runat="server" Text="View Products" /><br />
<br />
<asp:GridView ID="gvProducts" runat="server">
</asp:GridView>
</div>
```

Add the following code to the code behind file:

Add the following at the top of the file:

```
using System.Data.sqlClient
```

This allows for the use of the sqlParameter class

```
protected void Page_Load(object sender, EventArgs e)
{
   //populate drop down list only the first time the page is loaded
   if (Page.IsPostBack == false)
   {
      //create SQL object
      sqlHelper objSqlHelper = new sqlHelper();

      //create SQL statement that needs to be passed
      string strSQL;
      strSQL = "select categoryID, categoryName from categories";

      //populate drop down list
      ddlCategory.DataSource = objSqlHelper.executeSQL(strSQL);
      ddlCategory.DataTextField = "categoryName";
      ddlCategory.DataValueField = "categoryID";
      ddlCategory.DataBind();
   }

}
protected void btnView_Click(object sender, EventArgs e)
{
   //create SQL object
   sqlHelper objSqlHelper = new sqlHelper();

   //create SQL statement that needs to be passed
   string strSQL;
   strSQL = "select productName, unitPrice from products where categoryID =
@categoryID";

   //set up parameters
   SqlParameter[] objParams;
   objParams = new SqlParameter[1];
   objParams[0] = new SqlParameter("@categoryID", DbType.Int16);
   objParams[0].Value = int.Parse(ddlCategory.SelectedValue);

   //populate gridview
   gvProducts.DataSource = objSqlHelper.executeSQL(strSQL, objParams);
   gvProducts.DataBind();
}
```

Now we need to do the same thing for the executeScalar and executeNonQuery methods. Note that they will both use the fillParameters method.

Activity 11.	passing parameters to executeScalar and executeNonQuery

Add the following code to your sqlHelper class

```
public int NonQuerySQL(string strSQL, SqlParameter[] parameters)
    {
        //create connection object get connection string from private
        //variable _Conn
        SqlConnection objConnection = new SqlConnection(_Conn);

        //create command object
        //get SQL to execute from the strSql parameter
        //passed as input to this function
        SqlCommand objCommand = new SqlCommand(strSQL, objConnection);

        //fill parameters
        fillParameters(objCommand, parameters);

        //create a variable that will hold the return value
        int intRetValue;

        //open database connection
        objConnection.Open();

        //execute SQL
        intRetValue = objCommand.ExecuteNonQuery();

        //close connection
        objConnection.Close();

        //execute SQL and return dataReader
        return intRetValue;
    }
public object scalarSQL(string strSQL, SqlParameter[] parameters)
    {
        //create connection object get connection string from private
        //variable _Conn
        SqlConnection objConnection = new SqlConnection(_Conn);

        //create command object
        //get SQL to execute from the strSql parameter
        //passed as input to this function
        SqlCommand objCommand = new SqlCommand(strSQL, objConnection);

        //fill parameters
        fillParameters(objCommand, parameters);

        //create a variable that will hold the return value
        object objRetValue;

        //open database connection
        objConnection.Open();

        //execute SQL
        objRetValue = objCommand.ExecuteScalar();

        //close connection
        objConnection.Close();

        //execute SQL and return dataReader
        return objRetValue;
    }
```

Activity 12. Using these methods

Using the executeScalar with parameters

Aspx page:

```
Select the Category:<br />
<asp:DropDownList ID="ddlCategory" runat="server">
</asp:DropDownList><br />
<br />
<asp:Button ID="btnView" runat="server" Text="View Products" /><br />
<br />
<asp:Label ID="lblCount" runat="server"></asp:Label>
</div>
```

Code behind

```
protected void Page_Load(object sender, EventArgs e)
{
   //populate drop down list only the first time the page is loaded
   if (Page.IsPostBack == false)
   {
      //create SQL object
      sqlHelper objSqlHelper = new sqlHelper();

      //create SQL statement that needs to be passed
      string strSQL;
      strSQL = "select categoryID, categoryName from categories";

      //populate drop down list
      ddlCategory.DataSource = objSqlHelper.executeSQL(strSQL);
      ddlCategory.DataTextField = "categoryName";
      ddlCategory.DataValueField = "categoryID";
      ddlCategory.DataBind();
   }
}
protected void btnView_Click(object sender, EventArgs e)
{
   //create SQL object
   sqlHelper objSqlHelper = new sqlHelper();

   //create SQL statement that needs to be passed
   string strSQL;
   strSQL = "select count(*) from products where categoryID = @categoryID";

   //set up parameters
   SqlParameter[] objParams;
   objParams = new SqlParameter[1];
   objParams[0] = new SqlParameter("@categoryID", DbType.Int16);
   objParams[0].Value = int.Parse(ddlCategory.SelectedValue);

   int intCount;
   intCount = (int)objSqlHelper.scalarSQL(strSQL, objParams);

   lblCount.Text = intCount.ToString();
}
```

Part 2

Aspx file

```
<div>
Category Name:<br />
<asp:TextBox ID="txtCategoryName" runat="server"></asp:TextBox><br />
Category Description:<br />
<asp:TextBox ID="txtDescription" runat="server"></asp:TextBox><br />
<br />
<asp:Button ID="btnSend" runat="server" Text="Insert" /><br />
<br />
<asp:Label ID="lblResult" runat="server"></asp:Label>
</div>
```

Code behind

```
protected void btnSend_Click(object sender, EventArgs e)
{
   //create SQL object
   sqlHelper objSqlHelper = new sqlHelper();

   //create SQL statement that needs to be passed
   string strSQL;
   strSQL = "insert into categories(categoryName, description) values(@categoryName,
@categoryDescription)";

   //set up parameters
   SqlParameter[] objParams;
   objParams = new SqlParameter[2];
   objParams[0] = new SqlParameter("@categoryName", DbType.String);
   objParams[0].Value = txtCategoryName.Text;
   objParams[1] = new SqlParameter("@categoryDescription", DbType.String);
   objParams[1].Value = txtDescription.Text;

   int intResult = objSqlHelper.NonQuerySQL(strSQL, objParams);

   //display result
   lblResult.Text = intResult.ToString();
}
```

Is that it?

No! what we have created is a simple data access layer without error checking. We have also bypassed the business layer. In our exercises we have gone from the presentation layer (our aspx files) straight to the data access layer (sqlHelper).