Certificate in C# Advanced Programming

Creating Web Applications in C#

By Randal Root

# Module 07: Data-Driven Applications

*“At the beginning of this book, you learned that ASP.NET is just one component in Microsoft’s ambitious .NET platform. As you know, .NET also includes modern languages and a toolkit of classes that allows you to do everything from handling errors to analyzing XML documents. In this chapter, you’ll explore another one of the many features in the .NET Framework: the ADO.NET data access model.” (Beginning ASP.NET 4.5 in C#, Chapter 14, MacDonald)*

The ADO.NET programming model consists of **five core objects:**

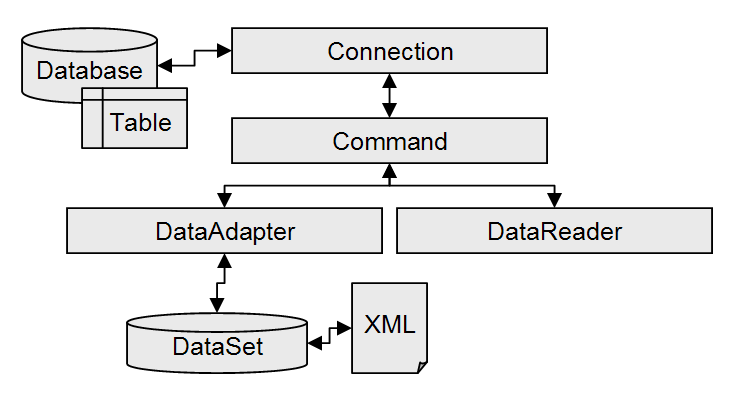
\* **Connection** object

\* **Command** object

\* **DataReader** object

\* **DataAdapter** object

\* **DataSet** object



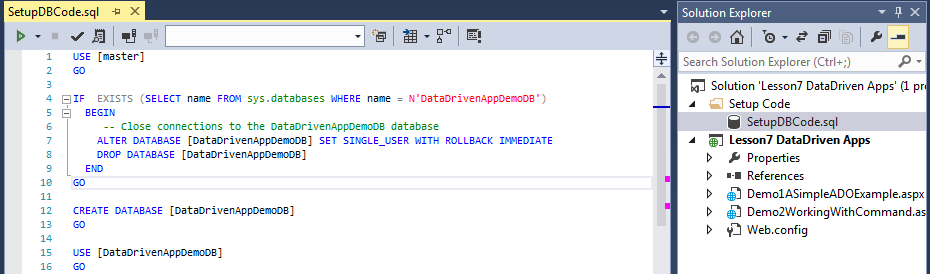
The figure above shows an overview of **how the different components work together**.

* The **Connection** object provides the **connection to the database**.
* The **Command** object **executes a command** against a data source. It can execute non-query commands, such as INSERT, UPDATE, or DELETE, or return a DataReader with the results of a SELECT command.
* The **DataReader** object provides a **forward-only, read-only, connected recordset**.
* The **DataAdapter** object populates a **disconnected DataSet or DataTable** with data and performs updates.
* The **DataSet** class, while not truly part of ADO.NET, it is most often used with the DataAdapter to **hold an updateable copy of the results from a query**. In fact, the DataAdapter is really not much use without it. Updates to the results in a DataSet can be sent back to the original data source or saved to a local XML file.

Here is a simple example of how these objects are used…

**NOTE**: You need to run the SetupDBCode.sql SQL file before you can test the Module 7 demos.





Demo1ASimpleADOExample.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

//1. Make a Connection

System.Data.OleDb.**OleDbConnection** objOleCon;

objOleCon = new System.Data.OleDb.OleDbConnection();

string strOledbConnection = @"Provider=SQLOLEDB;Data Source=(local);Integrated Security=SSPI;Initial Catalog=DataDrivenAppDemoDB";

objOleCon.ConnectionString = strOledbConnection;

objOleCon.Open();

//2. Issue a Command

System.Data.OleDb.**OleDbCommand** objCmd;

objCmd = new System.Data.OleDb.OleDbCommand("Select Count(\*) From Products", objOleCon);

int intProductCount = (int)objCmd.ExecuteScalar();

//3. Process the Results (if any)

Label1.Text = "<p>Number of items: " + intProductCount.ToString() + "</p>";

//4. Free up resources

objOleCon.Close();

}

</script>

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

<head runat="server">

<title></title>

</head>

<body>

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

<div>

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

</div>

</form>

</body>

</html>

## The Connection Class

The Connection class sets up the information necessary to attach to a database. It has **several properties** available to specify this information, such as username and password, as well as, of course, the location of the database.

***Important****: The properties change depending on which Namespace/Classes you use!*

**C#, SQL Server Connection**

using System.Data.SQLClient;

…

**Sql**Connection nwindConn =

new SqlConnection("Data Source=localhost; Integrated Security=SSPI; Initial Catalog=northwind");

nwindConn.Open();

**C#, ODBC Connection**

using System.Data.Odbc

…

**Odbc**Connection nwdCn =

new OdbcConnection("Driver={SQL Server};Server=localhost; Trusted\_Connection=yes;Database=northwind");

nwdCn.Open();

After creating the Connection object and setting its properties correctly, you’ll call the **Open() method** of the Connection object, which **creates the connection** to the database and opens it.

Both **Trusted\_Connection=yes** or **Integrated Security=SSPI** options log onto the server **using Windows Authentication**, which means your Windows logon rather than using a username/password combination. Windows Authentication is **considered more secure** since you don’t have to embed password information into your code **but** **requires a database administrator to create a database logon that maps to your Windows account**.

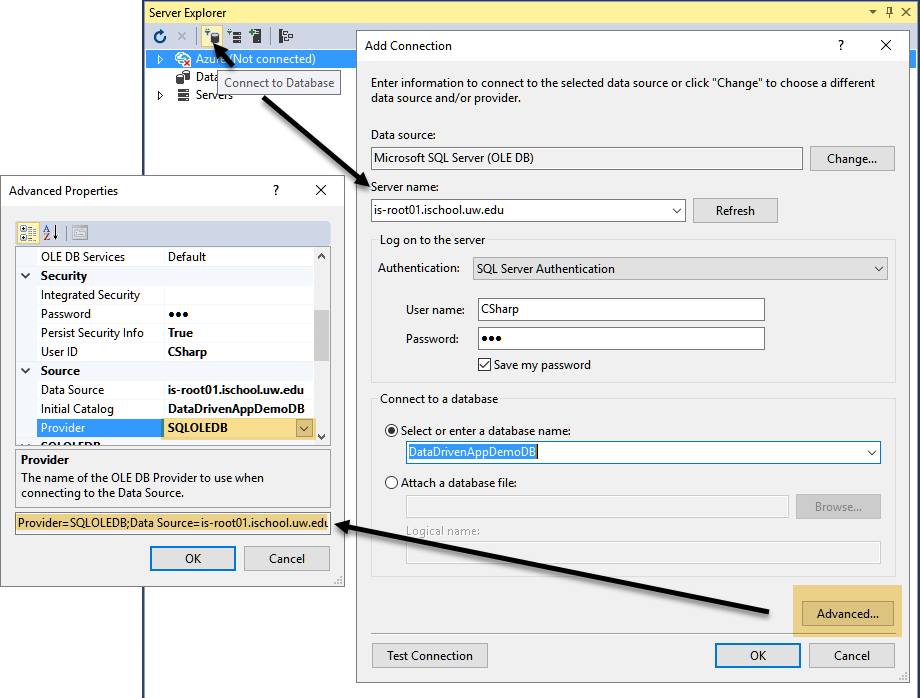
Windows accounts work **well for Internal sites**, where users are logged into the same domain, but not for External sites on the Internet. For those types of connections, you use to use a SQL Login.

@"Provider=SQLOLEDB; Data Source=is-root01.ischool.uw.edu; User ID=CSharp; Password=sql; Initial Catalog=DataDrivenAppDemoDB";

***NOTE:*** *The newer version of Windows really wants you to join Microsoft's external domain via a Hotmail, Live, or Office account. This way you can log in once and connect to multiple services on that domain. This would include web services that accept these domain accounts!*

## Making a Connection Visually

If you are using the full version of Visual Studio and not just the Express version, you can use Server Explorer to create a connection. (Not SQL Server Explorer) After you have selected the Data Source, you can configure both simple and advanced options. Once you create a connection in Server Explorer you use it much like you would the SQL Management Studio, including creating tables, views, and stored procedures.



***Important****: You can copy the Connection string using the Advanced button on the Add Connection dialog window.*

## LAB 1: A Simple ADO.NET Page

In this lab, you will:

1) Create a web page that indicates how many databases are on a given SQL Server.

**Tip:** Use the table/view called SysDatabases in the Master database.

**This lab should take about 5 to 10 minutes**

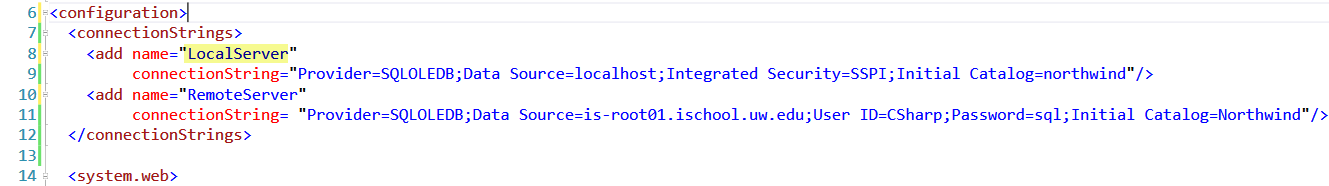
# Connections Strings

If you are going to use the **connection in many web pages**, you should **move your connection string to the** **web.config file** and reference it from your web pages using the following code in the web.config file:

connectionString="Provider=SQLOLEDB;Data Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind"/>

connectionString= "Provider=SQLOLEDB;Data Source=is-root01.ischool.uw.edu;User ID=CSharp;Password=sql;Initial Catalog=Northwind"/>

Goes into...



Followed by this code in your ASP.NET pages:

//1. Make a Connection

System.Data.OleDb.OleDbConnection objOleCon;

objOleCon = new System.Data.OleDb.OleDbConnection();

string strOledbConnection = ConfigurationManager.ConnectionStrings["LocalServer"].ConnectionString;

objOleCon.ConnectionString = strOledbConnection;

objOleCon.Open();

## LAB 2: Connection Strings

In this lab, you will:

1) Make a copy of your Lab1 web page, then modify the new web page so that it uses a connection string from the Web Config file.

**This lab should take about 5 minutes**

## Working with Commands

The ADO Command object let you send code to the database engine for processing. It also **identifies which type of result you expect to get back** (**nothing**, a **single value**, or a **table of values**).

### Demo2WorkingWithCommand.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">

protected void Button1\_Click(object sender, EventArgs e)

{

//1. Make a Connection

System.Data.OleDb.OleDbConnection objOleCon;

objOleCon = new System.Data.OleDb.OleDbConnection();

objOleCon.**ConnectionString** = ConfigurationManager.ConnectionStrings["LocalServer"].ConnectionString;

objOleCon.**Open**();

//2. Issue a Command

System.Data.OleDb.OleDbCommand objCmd;

objCmd = new System.Data.OleDb.OleDbCommand();

objCmd.**Connection** = objOleCon;

objCmd.**CommandType** = System.Data.**CommandType.Text**;

//Not normally typed out since Text is the default anyway

//3. Process the Results

//3a. Commands without Results

//When you don't need results back, like these two examples, use ExecuteNonQuery()

objCmd.**CommandText** = @"Create Table Demo1 (ID int Primary Key, Name nVarchar(50))";

Label1.Text = "<p>No Results</p>";

try

{

int RowsAffected = objCmd.**ExecuteNonQuery**();

// Ex. Inserts, Updates, Deletes SQL commands

Label1.Text += "<p>Table Created</p>";

}

catch (Exception ex)

{

Label1.Text += ex.Message.ToString() + "<hr/>";

}

objCmd.**CommandText** = @"Insert Into Demo1 (ID, Name) Values(1, 'Test Data')";

try

{

int RowsAffected = objCmd.**ExecuteNonQuery**();

// Ex. Inserts, Updates, Deletes SQL commands

Label1.Text += "<p>Row Added</p>";

}

catch (Exception ex)

{

Label1.Text += ex.Message.ToString() + "<hr/>";

}

//3b. Commands with one Result

objCmd.**CommandText** = "Select Count(\*) From Demo1";

try

{

int intResult = (int)objCmd.**ExecuteScalar**();

Label2.Text = "<p>Scalar Result</p>";

Label2.Text += "<p>Number of Row in Table: " + intResult.ToString() + "</p>";

}

catch (Exception ex)

{

Label2.Text += ex.ToString();

}

//3c. Commands multiple Results

objCmd.**CommandText** = "Select ID, Name From Demo1";

System.Data.OleDb.OleDbDataReader objDR;

try

{

objDR = objCmd.**ExecuteReader**();

Label3.Text = "<p>Multiple Results</p>";

while (objDR.Read() == true)

{

Label3.Text += "<p>"

+ objDR["ID"].ToString() + ", " + objDR["Name"].ToString()

+ @"</p>";

}

objDR.**Close**();

}

catch (Exception ex)

{

Label3.Text = ex.ToString();

}

//4. Free up Resources

objOleCon.**Close**();

}

</script>

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

<head runat="server">

<title></title>

</head>

<body>

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

<div>

<asp:Label ID="Label1" runat="server" Text="Label1"></asp:Label>

<br />

<asp:Label ID="Label2" runat="server" Text="Label2"></asp:Label>

<br />

<asp:Label ID="Label3" runat="server" Text="Label3"></asp:Label>

<br />

</div>

<asp:Button ID="Button1" runat="server" OnClick="Button1\_Click" Text="Button" />

</form>

</body>

</html>

## LAB 3: Displaying a list of Products in an ADO.NET Page

In this lab, you will:

1) Create a web page that displays a list of product and category names in an HTML table.

**Tip:** Use the view called vProductsByCategories in the DataDrivenAppDemoDB database.

SELECT CategoryName, ProductName FROM vProductsByCategories

**This lab should take about 10 to 15 minutes**

# Using SQL Abstraction Layers

In the last lab, we used a SQL view to "project" data from two tables. **Querying views, instead of tables,** is the "**best practice**." When working with **transactional data processing**, using SQL **stored procedures** is the "**best practice**."

A very simple insert stored procedure might look like this one:

Create Procedure pInsCategories

(@CategoryName [nvarchar](50), **@NewCategoryID** int Output)

As

Begin

Declare @StatusCode int = 0;

Begin Try

Insert Into [Categories]([CategoryName]) Values(@CategoryName);

Set **@NewCategoryID** = @@IDENTITY;

Set @StatusCode = +100;

End Try

Begin Catch

Set @StatusCode = -100;

End Catch

**Return @StatusCode;**

End

Go

You would execute this "Sproc" using the following code:

Declare @RC int, @NewID int;

Exec @RC = pInsCategories @CategoryName = 'Cat1', @NewCategoryID = @NewID Output;

Select @RC, @NewID;

The code in your web page now changes to call the stored procedure and pass in the parameter values.

* Configuring Parameters and Parameter Data Types
  + <https://msdn.microsoft.com/en-us/library/yy6y35y8(v=vs.110).aspx>

### Demo3WorkingWithSprocs.aspx

<%@ Page Language="C#" %>

<script runat="server">

protected void Page\_Load(object sender, EventArgs e)

{

//1. Make a Connection

System.Data.OleDb.OleDbConnection objOleCon;

objOleCon = new System.Data.OleDb.OleDbConnection();

objOleCon.ConnectionString = ConfigurationManager.ConnectionStrings["LocalServer"].ConnectionString;

objOleCon.Open();

//2. Issue a Command

System.Data.OleDb.OleDbCommand objCmd;

objCmd = new System.Data.OleDb.OleDbCommand("pInsCategories", objOleCon);

objCmd.CommandType = System.Data.CommandType.StoredProcedure;

//2.1 Add Parameters

// Declare @RC int, @NewID int;

// Exec @RC = pInsCategories @CategoryName = 'Cat1', @NewCategoryID = @NewID Output;

System.Data.OleDb.OleDbParameter objRC;

objRC = new System.Data.OleDb.OleDbParameter("@RC", System.Data.OleDb.OleDbType.Integer);

objRC.Direction = System.Data.ParameterDirection.ReturnValue;

objCmd.Parameters.Add(objRC);

System.Data.OleDb.OleDbParameter objCatName;

objCatName = new System.Data.OleDb.OleDbParameter("@CategoryName", System.Data.OleDb.OleDbType.VarWChar, 50);

objCatName.Direction = System.Data.ParameterDirection.Input;

objCatName.Value = "Cat from Web";

objCmd.Parameters.Add(objCatName);

System.Data.OleDb.OleDbParameter objNewID;

objNewID = new System.Data.OleDb.OleDbParameter("@NewCategoryID", System.Data.OleDb.OleDbType.Integer);

objNewID.Direction = System.Data.ParameterDirection.Output;

objNewID.DbType = System.Data.DbType.Int32;

objCmd.Parameters.Add(objNewID);

//2.2 Execute the code

objCmd.ExecuteNonQuery();

//3. Process the Results (if any)

// Select @RC, @NewID;

Label1.Text += "<p>Return Code: " + objCmd.Parameters["@RC"].Value.ToString() + "</p>";

Label1.Text += "<p>New Category ID: " + objCmd.Parameters["@NewCategoryID"].Value.ToString() + "</p>";

//4. Free up resources

objOleCon.Close();

}

</script>

<!DOCTYPE html>

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

<head runat="server">

<title></title>

</head>

<body>

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

<div>

<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

</div>

</form>

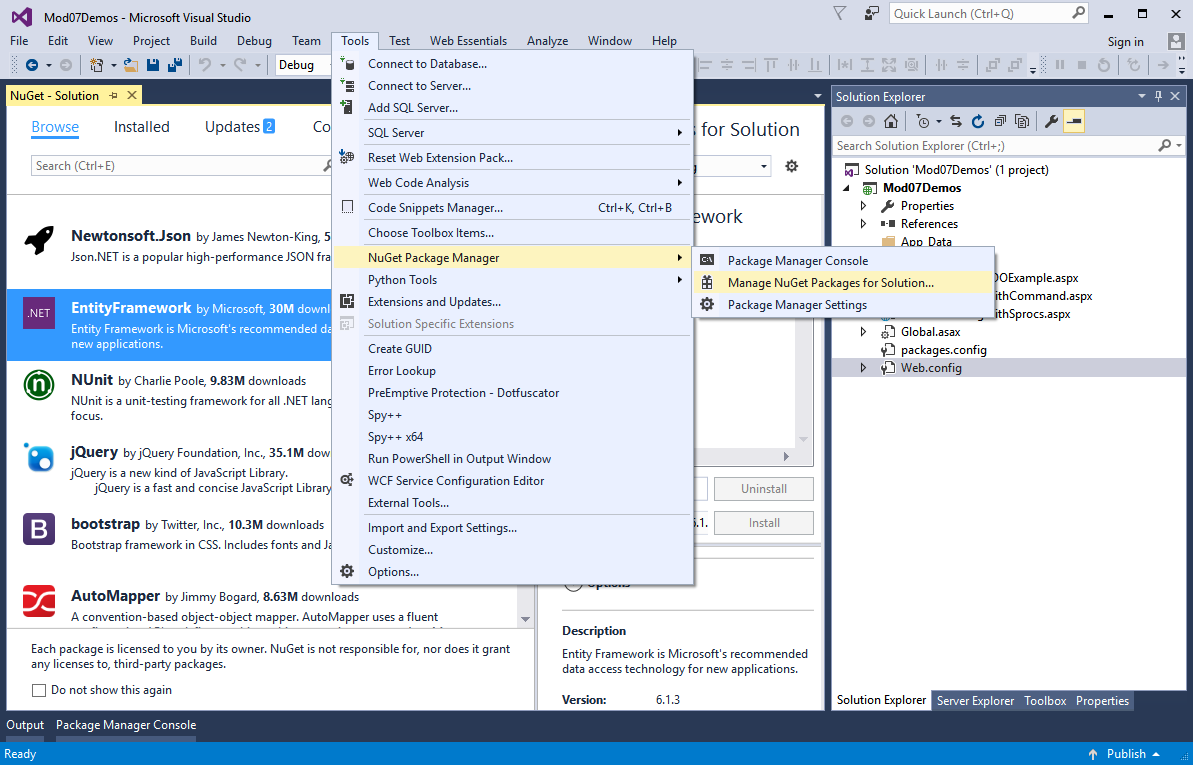
</body>

</html>

# Creating a Data-Driven Web Site using the Entity Framework

The web applications may also consist of an Entity Framework Data Model over ADO.NET connecting to a SQL Server Database. The Entity Framework still **uses the core ADO.NET objects but provides an abstraction layer** over the top of it.

Depending on the version of Visual Studio, the Entity Framework (EF) may already be installed, but if not you can install the Entity Framework using the NuGet Console. You can find more information about how this works on the Nuget website here: <https://www.nuget.org/packages/EntityFramework/5.0.0>



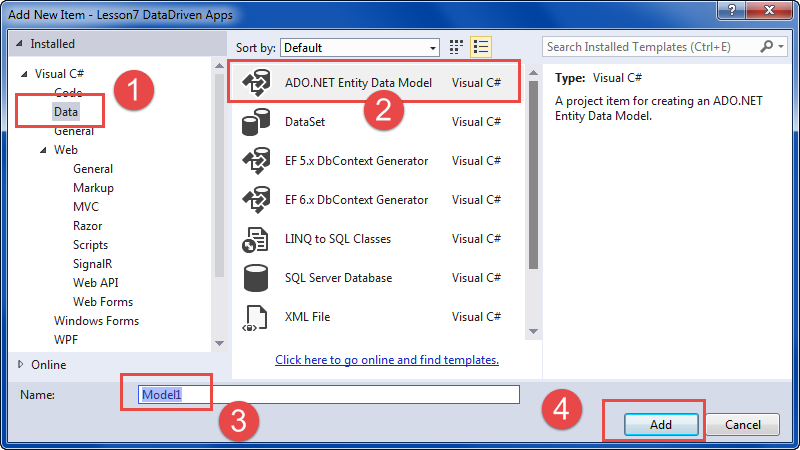
You can also add and execute the following command to add the EF to your project.

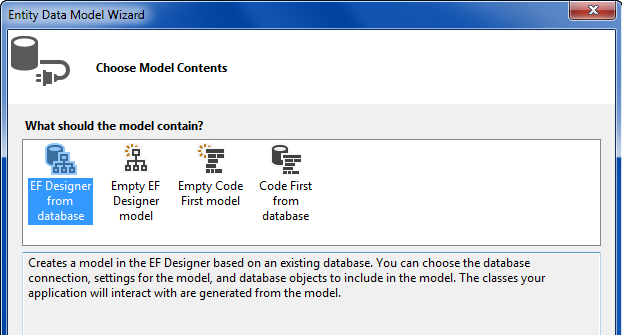
PM> Install-Package EntityFramework -Version 5.0.0

**NOTE**: Review the script file the installation make before you move on.

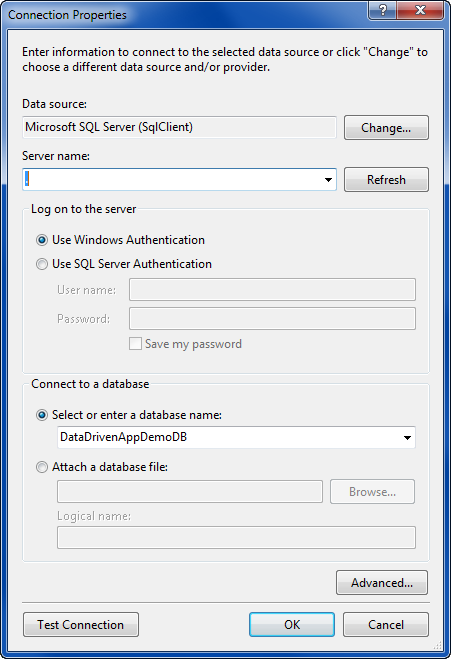
Follow these instructions to create a new EF data model:

1) Add a new Data Model to the Models folder of the project called Model1. **Right-Click** your **Project** icon in Solution Explorer, then **select Add->New** Item from the context menu. **Select the ADO .NET Entity Data Model** option.



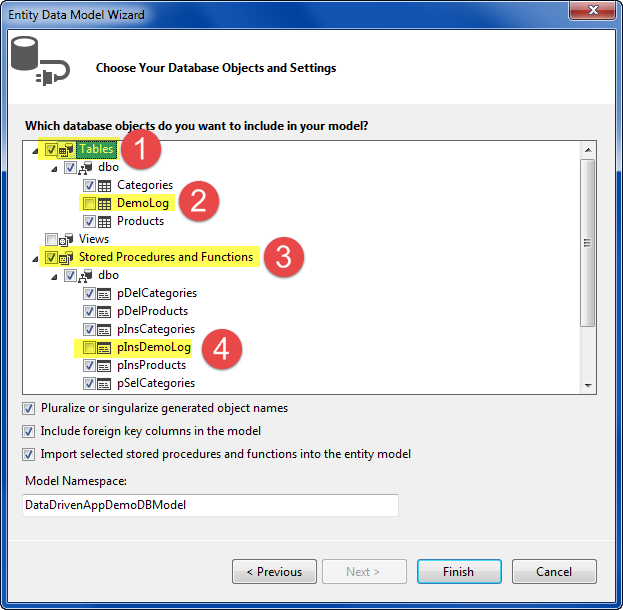
2) When the Entity Data Model Wizard appears, **select the “EF Designer from database”** option.

3) **Connect to your server** and the new DataDrivenAppDemoDB.



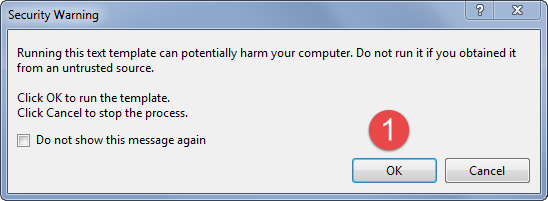
4) IF it asks you, **choose to use EF 6** for this demo.

5) Choose the Categories and Products tables and their associated stored procedures. Do not include the DemoLog table and its pInsDemoLog stored procedure. **Leave the Views unchecked** for now as well.



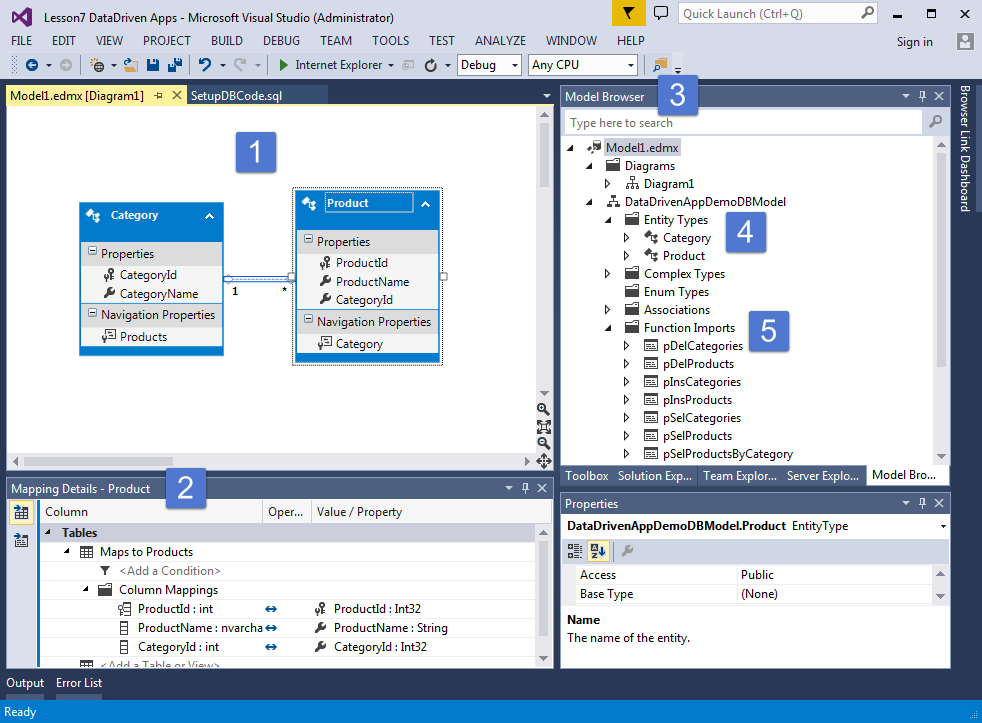
**Tip**: Note the Model namespace for later.

6) Respond to the Security warning by clicking the OK button.



7) Review the new EF Diagram.

1. The Model diagram
2. Mapping Details
3. The Model Browser
4. The Entity Types
5. The imported Stored Procedures and Functions



## Creating web pages with the Entity Framework

With an EF model created and configured, we let it handle the tasks of connecting, issuing commands, and returning results.

#### Demo4WorkingWithEF.aspx

<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">

Mod07Demos.DataDrivenAppDemoDBEntities **objEF**;

protected void Page\_Load(object sender, EventArgs e)

{

objEF = new Mod07Demos.DataDrivenAppDemoDBEntities();

}

protected void Button1\_Click(object sender, EventArgs e)

{ //Select From a Table

Label1.Text = "";

foreach (Mod07Demos.Product row in objEF.Products)

{

Label1.Text += "<p>" + row.ProductId.ToString()

+ ", " + row.ProductName.ToString()

+ ", " + row.CategoryId.ToString()

+ "</p>";

}

}

protected void Button2\_Click(object sender, EventArgs e)

{

//Insert using a stored procedure

Label2.Text = "";

try

{

objEF.pInsProducts(TextBoxProductName.Text, int.Parse(TextBoxCategoryId.Text));

Label2.Text = "Inserted";

}

catch (Exception ex)

{ Label2.Text = ex.Message.ToString(); }

}

</script>

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

<head runat="server">

<title>Demo4</title>

</head>

<body>

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

<div>

<asp:Button ID="Button1" runat="server" OnClick="Button1\_Click" Text="Select Button" />

<br />

<asp:Label ID="Label1" runat="server" Text="Label1"></asp:Label>

<br />

<asp:Button ID="Button2" runat="server" OnClick="Button2\_Click" Text="Insert Button" />

<br />

<asp:Label ID="LabelProductName" runat="server" Text="ProductName: "></asp:Label>

<asp:TextBox ID="TextBoxProductName" runat="server"></asp:TextBox>

<br />

<asp:Label ID="LabelCategoryId" runat="server" Text="CategoryId: "></asp:Label>

<asp:TextBox ID="TextBoxCategoryId" runat="server"></asp:TextBox>

<br />

<asp:Label ID="Label2" runat="server" Text="Label2"></asp:Label>

<br />

</div>

</form>

</body>

</html>

# EF vs ADO.NET

Remember that the **EF is an abstraction layer over the top of ADO.NET** and using it comes with a cost. On a Windows form application, you have one user for the application and the performance is not much of an issue, but with a Web Site it can be. External, or any site with a lot of users, need to be programmed to have the best performance possible. ADO.NET will still provide you with that at the cost of having to do a lot more manually coding. Take a look at the following Microsoft web page on this topic: <https://msdn.microsoft.com/en-us/data/hh949853>

## LAB 4: Displaying a list of Products in an ADO.NET Page

In this lab, you will:

1) Create a web page that displays a list of product and category names in an HTML table, this time using the Entity Framework.

**Tip:** Use the view called vProductsByCategories in the DataDrivenAppDemoDB database, but remember, you have to add it to the EF model first.

SELECT CategoryName, ProductName FROM vProductsByCategories

**This lab should take about 10 to 15 minutes**

# Creating a Middle-Tier

**Consider moving your data access layer into a class library** if you want to reuse it in other applications or if you feel that changing the UI layer of your current application my happen in the future. For example, imaging that you have a WebForms application that you want to convert to an MVC application, in this scenario, having the data access layer is separate from the WebForms pages **make the upgrade easier**.

