# Code Repository Exercises

## ADO.NET

**Purpose:** The purpose of this exercise is to familiarize you with ADO.NET, the foundational data-access component of the .NET framework. You will see how to write code that creates a connection to a database, runs a query, and returns the result to a page using ADO.NET classes. You will also learn the difference between connected (provider) and disconnected data objects and see how both are used together to provide data access.

**Directory Name:** ADO

**Instructions:**

1. Download the starting files for this exercise. Place the files in the following locations:
    a. *Pubs.mdf* (SQL Server Database file): Place in the App_Data folder of your repository application. If this folder does not yet exist, add it by right clicking the application's root directory in the server explorer window, then choose 'Add ASP.NET Folder' and select 'App_Data'.
    b. *Sprocs.sql*: This file contains SQL code for creating five stored procedures in the Pubs database. These stored procedures select, update, insert, and delete author information, respectively. Add these stored procedures to your database by doing the following:
        i. Open the 'sprocs.sql' file in Notepad or Visual Studio
        ii. Open the Pubs database in the Server Explorer window of Visual Studio by double-clicking on the *Pubs.mdf* file in your App_Data folder.
        iii. Right-click on the 'Stored Procedures' directory and select 'Add New Stored Procedure'
        iv. Delete the boilerplate code from the new stored procedure window and paste the code from the sprocs.sql file into this window
        v. Select all the code, right click, and select 'Run Selection'
        vi. This should create all of the stored procedures in the database. You can verify that they are there by right clicking on the 'Stored Procedures' directory in the Server Explorer window and clicking 'Refresh'
    c. All other files: Place in the *ADO* directory you created for this exercise
2. Add the following connection string to your Web.Config file inside the <connectionStrings> element (you can leave any existing connection strings in place):

```
<add name="Pubs" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|pubs.mdf;Integrated
Security=True;User Instance=True" providerName="System.Data.SqlClient"/>
```

3. Run the *ConnectionTester.aspx* page that you downloaded in the starting files to test the connection to your database. If successful, the page will report the server version and show that a connection was opened and then closed. Study the content of *ConnectionTester.aspx* and *ConnectionTester.aspx.cs* to understand how this page works.
4. Complete the *AuthorManager.aspx* page by writing the required code for each method in the code-behind file (the method declarations are already in place with comments indicating the textbook pages where the code can be found). Be sure you take time to understand what each line of code is doing as you go.
5. Copy the *AuthorManager.aspx* page and paste it in again as a new page. Re-name this page to *AuthorManager2.aspx*. In this page, change the *FillAuthorList()* method to use a disconnected DataSet object as shown on pages 465-466.
6. The DataSet is a heavyweight object that can be used to represent an entire database. In most instances, the DataSet is overkill. An alternative approach to that used in step 5 is to create a DataTable object directly, and then use the Load() method of this object to load the contents of a DataReader into the DataTable. This way, you can have the advantages disconnected object (the DataTable) without the unnecessary overhead of a full DataSet. Copy the *AuthorManager2.aspx* file you created in step 5 and paste it in again as a new page, *AuthorManager3.aspx*. Modify the *FillAuthorList()* method so that it uses a DataReader object like the original *AuthorManager.aspx*, but instead of using the Reader.Read() method, you will instead create a DataTable object and use its Load() method to load the contents of the reader into the DataTable. You can then loop through each of the rows of the DataTable to populate the DropDownList as you did in *AuthorManager2.aspx*.
7. Rather than writing your SQL query in the application code, you can write the query in a stored procedure and then execute this stored procedure using the SqlCommand object. The stored procedures you created in step 1 perform each of the select, insert, update, and delete queries needed by the AuthorManager page. Copy your *AuthorManager3.aspx* file paste it as another file named *AuthorManagerSproc.aspx*. Modify the code in this page to utilize the stored procedures you created. (Hint: you will need to change the CommandType property of each of the SqlCommand objects, and set the CommandText to the name of the stored procedure).

**Header on Default.aspx:** ADO.NET

**Pages linked from Default.aspx:** *AuthorManager.aspx, AuthorManager2.aspx, AuthorManager3.aspx, AuthorManagerSproc.aspx*