Chapter 8

Accessing Data in ASP.NET

We have provided a very detailed discussion on database programming with Visual C#.NET using the Windows-based applications in the previous chapters. Starting with this chapter, we will concentrate on database programming with Visual C#.NET using Web-based applications. To develop the Web-based application and allow users to access the database through the Internet, you need to understand an important component: Active Server Page.NET or ASP.NET.

Essentially, ASP.NET allows users to write software to access databases through a Web browser rather than a separate program installed on their computers. With the help of ASP.NET, the users can easily create and develop an ASP.NET Web application and run it on the server as a server-side project. The user then can send requests to the server to download any Web page and to access the database to retrieve, display, and manipulate data via the Web browser. The actual language used in the communications between the client and the server is Hypertext Markup Language (HTML).

When finished this chapter, you will:

- Understand the structure and components of ASP.NET Web applications.
- Understand the structure and components of .NET Framework.
- Select data from the database and display data in a Web page.
- Understand the Application state structure and implement it to store global variables.
- Understand the AutoPostBack property and implement it to communicate with the server effectively.
- Insert, update, and delete data from the database through a Web page.
- Use the stored procedure to perform the data actions against the database via a Web application.
- Use LINQ to SQL query to perform the data actions against the database via a Web application.
- Perform client-side data validation in Web pages.

In order to help readers to successfully complete this chapter, first we need to provide a detailed discussion about the ASP.NET. But the prerequisite to understanding the ASP.NET is the .NET Framework since the ASP.NET is a part of .NET Framework, or in other words, the .NET Framework is the foundation of the ASP.NET. So we need first to give a detailed discussion about the .NET Framework.

Practical Database Programming With Visual C#.NET, by Ying Bai

8.1 WHAT IS .NET FRAMEWORK?

The .NET Framework is a model that provides a foundation to develop and execute different applications at an integrated environment such as Visual Studio.NET. In other words, the .NET Framework can be considered as a system to integrate and develop multiple applications such as Windows applications, Web applications, or XML Web Services by using a common set of tools and codes such as Visual C#.NET or Visual Basic.NET.

The current version of the .NET Framework is 3.5. Basically, the .NET Framework consists of the following components:

- The Common Language Runtime—CLR (called runtime). The runtime handles runtime services such as language integration, security, and memory management. During the development stage, the runtime provides features that are needed to simplify the development.
- Class Libraries. Class libraries provide reusable codes for most common tasks such as data access, XML Web service development, and Web and Windows forms.

The main goal to develop the .NET Framework is to overcome several limitations on Web applications since different clients may provide different client browsers. To solve these limitations, .NET Framework provides a common language called Microsoft Intermediate Language (MSIL) that is language independent and platform independent, and allows all programs developed in any .NET-based language to be converted into this MSIL. The MSIL can be recognized by the Common Language Runtime (CLR), and the CLR can compile and execute the MSIL codes by using the Just-In-Time compiler located at the local machines or clients.

You access the .NET Framework by using the class libraries provided by the .NET Framework, and you implement the .NET Framework by using the tools such as Visual Studio.NET provided by the .NET Framework, too. All class libraries provided by the .NET Framework are located at the different namespaces. All .NET-based languages access the same libraries. A typical .NET Framework model is shown in Figure 8.1.

The .NET Framework supports three types of user interfaces:

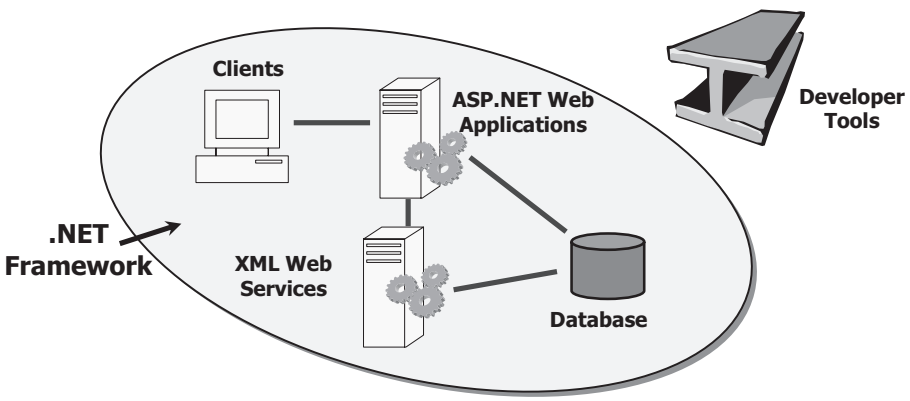

Figure 8.1 A .NET Framework model.

- Windows Forms that run on Windows 32 client computers. All projects we developed in the previous chapters used this kind of user interface.
- Web Forms that run on Server computers through ASP.NET and the Hypertext Transfer Protocol (HTTP).
- The Command Console.

The advantages of using the .NET Framework to develop Windows-based and Web-based applications include but are no limited to:

- The .NET Framework is based on Web standards and practices, and it fully supports Internet technologies, including HTML, HTTP, XML, Simple Object Access Protocol (SOAP), XML Path Language (XPath), and other Web standards.
- The .NET Framework is designed using unified application models, so the functional of any class provided by the .NET Framework is available to any .NET-compatible language or programming model. The same piece of code can be implemented in Windows applications, Web applications, and XML Web Services.
- The .NET Framework is easy for developers to use since the code in the .NET Framework is organized into hierarchical namespaces and classes. The .NET Framework provides a common type system, which is called the unified type system, and it can be used by any .NET-compatible language. In the unified type system, all language elements are objects that can be used by any .NET application written in any .NET-based language.

Now let's take a closer look at the ASP.NET.

8.2 WHAT IS ASP.NET AND ASP.NET 3.5?

ASP.NET is a programming framework built on the .NET Framework, and it is used to build Web applications. Developing ASP.NET Web applications in the .NET Framework is very similar to developing Windows-based applications. An ASP.NET Web application is composed of many different parts and components, but the fundamental component of ASP.NET is the Web Form. A Web Form is the Web page that users view in a browser, and an ASP.NET Web application can contain one or more Web Forms. A Web Form is a dynamic page that can access server resources.

The current version of the ASP.NET is 3.5, which is combined with .NET Framework 3.5 to provide a professional and convenient way to help users build and develop a variety of data-driven applications in .NET programming languages. Compared with the progression from ASP.NET 2.0, the features in ASP.NET 3.5 are additive, which means that the core assemblies installed from the .NET Framework 2.0 are still used by the 3.5 version. In short, ASP.NET 3.5 does not change, take away, or break any function, concepts, or code present in 2.0, but it simply adds new types and features and capabilities to the framework. Therefore, ASP.NET 3.5 is a rather minor upgrade from ASP.NET 2.0; that is, there are not many new ASP.NET-specific features in the .NET Framework 3.5.

There are three new features worth noting in ASP.NET 3.5:

- Integrated ASP.NET AJAX support
- The ListView control
- The DataPager control

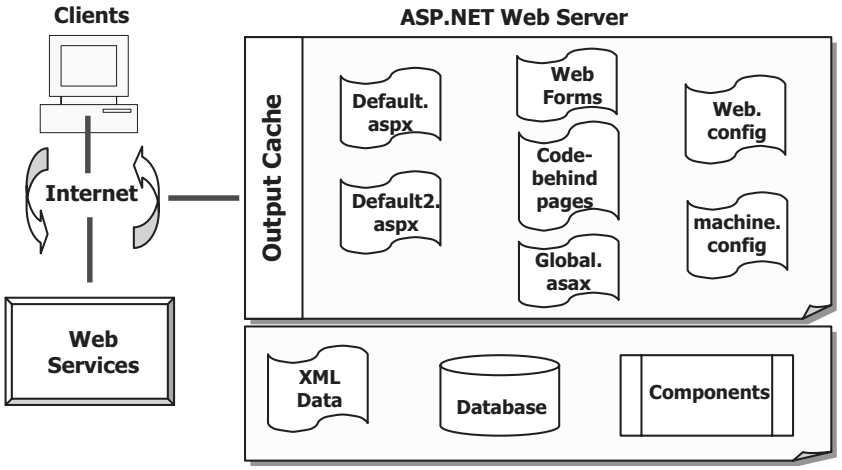

Figure 8.2 Structure of an ASP.NET Web application.

Besides these new features, one of the most significant differences between ASP.NET 3.5 and ASP.NET 2.0 is that the LINQ support is added to ASP.NET 3.5. LINQ provides a revolutionary solution between the different query syntaxes used in the different databases and bridges the gap between the world of objects and the world of data.

A completed structure of an ASP.NET Web application is shown in Figure 8.2.

Unlike a traditional Web page that can run scripts on the client, an ASP.NET Web Form can also run server-side codes to access databases, to create additional Web Forms, or to take advantage of built-in security of the server. In addition, since an ASP.NET Web Form does not rely on client-side scripts, it is independent on the client's browser type or operating system. This independence allows users to develop a single Web Form that can be viewed on any device that has Internet access and a Web browser.

Because ASP.NET is part of the .NET Framework, the ASP.NET Web application can be developed in any .NET-based language.

The ASP.NET technology also supports XML Web Services. XML Web Services are distributed applications that use XML for transferring information between clients, applications, and other XML Web Services.

The main parts of an ASP.NET Web application include:

- Web Forms or Default.aspx pages. The Web Forms or Deafult.aspx pages provide the user interface for the Web application, and they are very similar to the Windows Forms in the Windows-based application. The Web Forms files are indicated with an extension of **.aspx**.
- Code-behind pages. The so-called code-behind pages are related to the Web Forms and contain the server-side codes for the Web Form. This code-behind page is very similar to the code window for the Windows Forms in a Windows-based application we discussed in the previous chapters. Most event methods or handlers associated with controls on the Web Forms are located in this code-behind page. The code-behind pages are indicated with an extension of **.aspx.cs**.
- Web Services or **.asmx** pages. Web services are used when you create dynamic sites that will be accessed by other programs or computers. ASP.NET Web Services may be supported by a code-behind page designed by the extension of **.asmx.cs**.

- Configuration files. The Configuration files are XML files that define the default settings for the Web application and the Web server. Each Web application has one **Web.config** configuration file, and each Web server has one **machine.config** file.
- **Global .asax** file. The Global.asax file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events that are raised by ASP.NET or by HttpModules. At runtime, Global.asax is parsed and compiled into a dynamically generated .NET Framework class that is derived from the HttpApplication base class. This dynamic class is very similar to the Application class or main thread in Visual C++, and this class can be accessed by any other objects in the Web application.
- XML Web service links. These links are used to allow the Web application to send and receive data from an XML Web service.
- Database connectivity. The Database connectivity allows the Web application to transfer data to and from database sources. Generally, it is not recommended to allow users to access the database from the server directly because of security issues. Instead, in most industrial and commercial applications, the database can be accessed through the application layer to strengthen the security of the databases.
- Caching. Caching allows the Web application to return Web Forms and data more quickly after the first request.

8.2.1 ASP.NET Web Application File Structure

When you create an ASP.NET Web application, Visual Studio.NET creates two folders to hold the files that relate to the application. When the project is compiled, a third folder is created to store the terminal dll file. In other words, the final or terminal file of an ASP.NET Web application is a dynamic linked library file (.dll). Figure 8.3 shows a typical file structure of an ASP.NET Web application.

The folders on the left side in Figure 8.3 are very familiar to us since they are created by the Windows-based applications. But the folders on the right side are new to us, and the functions of those folders are:

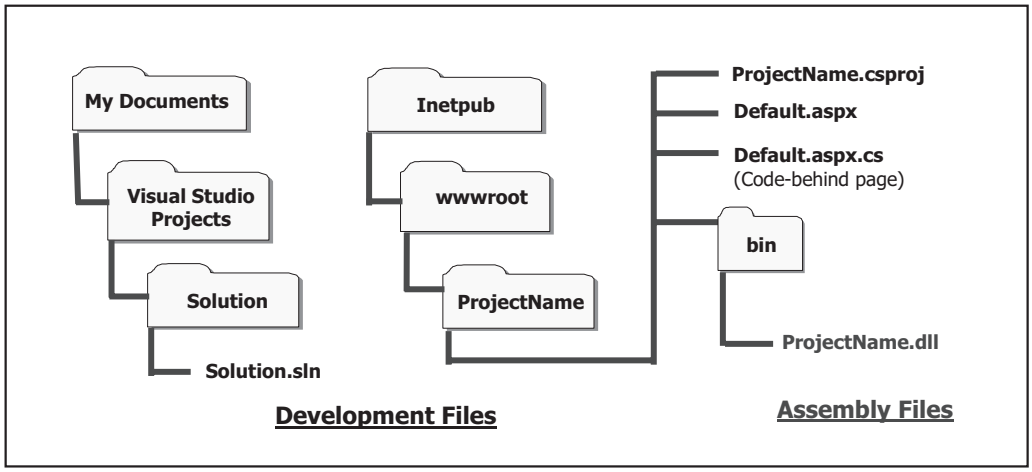

Figure 8.3 ASP.NET Web application file structure.

- The Inetpub folder contains another folder named wwwroot, and it is used to hold the root address of the Web project whose name is defined as ProjectName. The project file ProjectName.csproj is an XML file that contains references to all project items, such as forms and classes.
- The bin folder contains the assembly file or the terminal file of the project with the name of ProjectName.dll. All ASP.NET Web applications will be finally converted to a dll file and stored in the server's memory.

8.2.2 ASP.NET Execution Model

When you finished an ASP.NET Web application, the Web project is compiled and two terminal files are created:

1. Project Assembly files (.dll). All code-behind pages (.aspx.cs) in the project are compiled into a single assembly file that is stored as ProjectName.dll. This project assembly file is placed in the **bin** directory of the Web site and will be executed by the Web server as a request is received from the client at running time.
2. AssemblyInfo.cs file. This file is used to write the general information, specially assembly version and assembly attributes, about the assembly.

As a Web project runs and the client requests a Web page for the first time, the following events occur:

1. The client browser issues a GET HTTP request to the server.
2. The ASP.NET parser interprets the course code.
3. Based on the interpreting result, ASP.NET will direct the request to the associated assembly file (.dll) if the code has been compiled into the dll files. Otherwise, the ASP.NET invokes the compiler to convert the code into the dll format.
4. Runtime loads and executes the Microsoft Intermediate Language (MSIL) codes and sends back the required Web page to the client in the HTML file format.

For the second time when the user requests the same Web page, no compiling process is needed, and the ASP.NET can directly call the dll file and execute the MSIL code to speed up this request.

From this execution sequence, it looks like the execution or running of a Web application is easy and straightforward. However, in practice, a lot of data round trips occurred between the client and the server. To make it clear, let's continue the discussion and analysis of this issue and see what really happens between the client and the server as a Web application is executed.

8.2.3 What Really Happens When a Web Application Is Executed?

The key point is that a Web Form is built and run on the Web server. When the user sends a request from the user's client browser to request that Web page, the server needs to build that form and send it back to the user's browser in the HTML format. Once the

Web page is received by the client's browser, the connection between the client and the server is terminated. If the user wants to request any other page or information from the server, additional requests must be submitted.

To make this issue more clear, we can use our LogIn Form as an example. The first time the user sends a request to the server to ask to start a logon process, the server builds the LogIn Form and sends it back to the client in the HTML format. After that, the connection between the client and the server is gone. After the user receives the LogIn Web page and enters the necessary logon information such as the username and password to the LogIn Form, the user needs to send another request to the server to ask the server to process those pieces of logon information. After the server receives and processes the logon information, if the server finds that the logon information is invalid, the server needs to rebuild the LogIn Form and resend it to the client with some warning message. Therefore, you can see how many round trips occurred between the client and the server as a Web application is executed.

A good solution to try to reduce these round trips is to make sure that all the information entered from the client side is as correct as possible. In other words, try to make as much validation as possible on the client side to reduce the burden of the server.

Now we have finished the discussion about the .NET Framework and ASP.NET as well as the ASP.NET Web applications. Next we will create and develop some actual Web projects using the ASP.NET Web Forms to illustrate how to access the database through the Web browser to select, display, and manipulate data on Web pages.

8.2.4 Requirements to Test and Run a Web Project

Before we can start to create our real Web project using the ASP.NET, we need the following requirements to test and run our Web project:

1. Web server: To test and run our Web project, you need a Web server either on your local computer or on your network. By default, if you installed the Internet Information Services (IIS) on your local computer before the .NET Framework is installed on your computer, the FrontPage Server Extension 2000 should have been installed on your local computer. This software allows your Web development tools such as Visual Studio.NET to connect to the server to upload or download pages from the server.
2. In this chapter, in order to make our Web project simple and easy, we always use our local computer as a pseudoserver. In other words, we always use the localhost, which is the IP name of our local computer, as our Web server to communicate with our browser to perform the data accessing and manipulating.

If you have not installed the IIS on your computer, follow the steps below to install this component on your computer:

- Click on Start, then click on Control Panel, and click on Add or Remove Programs.
- Click on Add/Remove Windows Components. The Windows Components Wizard appears, which is shown in Figure 8.4.
- Check the checkbox for the Internet Information Services (IIS) from the list to add the IIS to your computer. To confirm that this installation contains the installation of the FrontPage 2000 Server Extensions, click on the Details button to open the IIS dialog box.

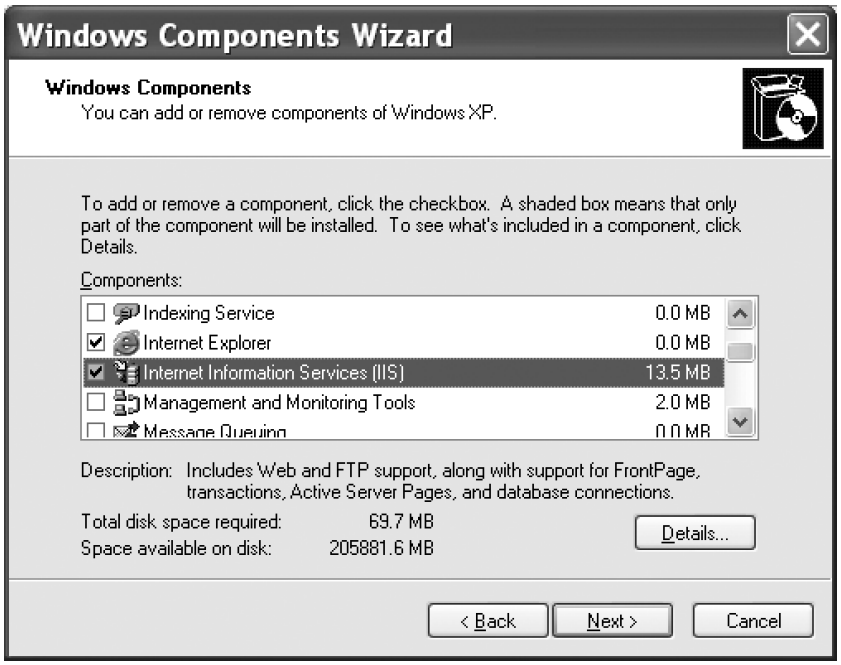

Figure 8.4 Opened Windows Components dialog box.

Check on the checkbox for the FrontPage 2000 Server Extensions to select it if it is not checked. Although Microsoft has stopped supporting this version of the server and the current version is FrontPage 2002 Server Extensions, you can still use it without any problem.

- Click on the OK button to close the IIS dialog box.
- Click on the Next button to begin to install the IIS and the FrontPage 2000 Server Extensions to your computer.

You may be asked to insert the Windows XP SP2 Operating System CD into your CD drive since this installation needs some files in that system disk. Just follow the instructions to do that to complete this installation. Click on the Finish button to close the Windows Components Wizard when this installation is finished. You may need to contact the administrator at your college to get this system CD if you do not have one. You must reboot your computer to make this installation complete.

As you know, the .NET Framework includes two Data Providers for accessing enterprise databases: the .NET Framework Data Provider for OLE DB and the .NET Framework Data Provider for SQL Server. Because there is no significant difference between the Microsoft Access database and the SQL Server database, in this chapter we only use the SQL Server database and the Oracle database as our target databases to illustrate how to select, display, and manipulate data against our sample database through the Web pages.

This chapter is organized as follows:

1. Develop ASP.NET Web application to select and display data from the Microsoft SQL Server database.

2. Develop ASP.NET Web application to insert data into the Microsoft SQL Server database.
3. Develop ASP.NET Web application to update and delete data against the Microsoft SQL Server database.
4. Develop ASP.NET Web application to select and manipulate data against the Microsoft SQL Server database using LINQ to SQL query.
5. Develop ASP.NET Web application to select and display data from the Oracle database.
6. Develop ASP.NET Web application to insert data into the Oracle database.
7. Develop ASP.NET Web application to update and delete data against the Oracle database.

Let's start with the first one in this list to create and build our ASP.NET Web application.

8.3 DEVELOP ASP.NET WEB APPLICATION TO SELECT DATA FROM SQL SERVER DATABASES

Let's start a new ASP.NET Web application project SQLWebSelect to illustrate how to access and select data from the database via the Internet. Open the Visual Studio.NET and click on the File|New Web Site to create a new ASP.NET Web application project. On the opened New Web Site dialog box, which is shown in Figure 8.5, keep the default template ASP.NET Web Site selected and the default content of Location box unchanged. Select Visual C# from the Language box and then enter the project name SQLWebSelect into the box that is next to the Browse button, as shown in Figure 8.5.

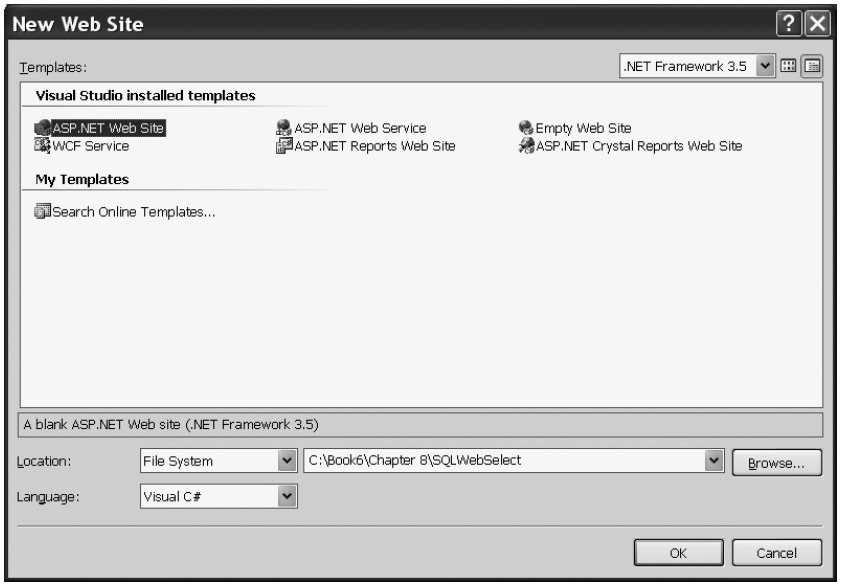

Figure 8.5 Opened Template dialog box.

You can place your new project in any folder you like on your computer. In our case, we place it in the folder C:\Book 6\Chapter 8. Click on the OK button to create this new Web application project.

On the opened new project, the default Web form is named Default.aspx, and it is located at the Solution Explorer window. This is the Web form that works as a user interface on the server side. Now let's perform some modifications to this form to make it our LogIn form page.

8.3.1 Create the User Interface—LogIn Form

Right-click on this Default.aspx item and select the Rename item from the pop-up menu and change the name of this Web Form to LogIn.aspx since we want to use this default page as our LogIn page.

Three buttons are located at the bottom of this window: Design, Split, and Source. The Design button is used to open the Web form page to allow users to insert any control onto that page. The Source button is used to open the Source file that basically is an HTML file that contains the related codes for all the controls you added into this Web Form in the HTML format. The Split button is used to divide the full window into two parts: the Design view and Source view. Compared with the codes in the code-behind page, the difference between them is that the Source file is used to describe all controls you added into the Web Form in HTML format, but the code-behind page is used to describe all controls you added into the Web form in Visual C#.NET code format.

Note that the code line is inside the code body: <form id="form1" runat="server">. This coding line indicates that this Web form will be run at the server side as the project runs.

Now let's click on the View Designer button from the Solution Explorer window to open the Design view to design our Web form window.

Unlike the Windows-based application, by default the user interface in the Web-based application has no background color. You can modify the Web form by adding another Style Sheet and format the form as you like. Also if you want to change some styles such as the header and footer of the form applied to all of your pages, you can add a Master Page to do that. But in this project we prefer to use the default window as our user interface and each page in our project has a different style.

We need to add the controls into our LogIn user interface or Web page as shown in Table 8.1. Note that there is no Name property available for any control in the Web form object, instead the property ID is used to replace the Name property and it works as a unique identifier for each control you added into the Web form.

Another difference with the Windows-based form is that when you add these controls into our Web form, first you must locate a position for the control to be added using the Space key and the Enter key on your keyboard in the Web form, and then pick up a control from the Toolbox window and drag it to that location. You cannot pick and drag a control to a random location in this Web form, and this is a significant difference between the Windows-based form and the Web-based form windows. Your finished user interface should match the one shown in Figure 8.6.

Table 8.1 Controls for the LogIn Form

Type	ID	Text	TabIndex	BackColor	TextMode	Font
Label	Label1	Welcome to CSE DEPT	0	#E0E0E0		Bold/Large
Label	Label2	User Name	1			Bold/Small
Textbox	txtUserName		2			
Label	Label3	Pass Word	3			Bold/Small
Textbox	txtPassWord		4		Password	
Button	cmdLogIn	LogIn	5			Bold/Medium
Button	cmdCancel	Cancel	6			Bold/Medium

Figure 8.6 Finished LogIn Web form.

Before we can add the codes into the code-behind page in response to the controls to perform the logon process, first we must run the project to allow the web.config file to recognize those controls added into the Web form. Click on the Start Debugging button on the toolbar to run our project. Click on OK to a prompted window to add a Web.config file with debugging enabled. Your running Web page should match the one shown in Figure 8.6. Click on the Close button located at the upper-right corner of the form to close this page.

Now let's develop the codes to access the database to perform the logon process.

8.3.2 Develop Codes to Access and Select Data from Database

Open the code-behind page by clicking on the View Code button from the Solution Explorer window. First, we need to add an SQL Server data provider-related namespace as we did for those projects in the previous chapters. Add the following namespace to the top of this code window to involve the namespace of the SQL Server Data Provider:

_Default | Page_Load()

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

   public partial class _Default : System.Web.UI.Page
   {
B     public SqlConnection sqlConnection;
      protected void Page_Load(object sender, EventArgs e)
      {
C        string sqlString = "Server=localhost;Data Source=.\\SQLEXPRESS;" +
                            "Database=C:\\database\\SQLServer\\CSE_DEPT.mdf;Integrated Security=SSPI";
D        sqlConnection = new SqlConnection(sqlString);
E        Application["sqlConnection"] = sqlConnection;          //define a global connection object
F        if (sqlConnection.State == ConnectionState.Open)
            sqlConnection.Close();
G        sqlConnection.Open();
H        if (sqlConnection.State != ConnectionState.Open)
            Response.Write("<script>alert('Database connection is Failed')</script>");
      }
   }
```

Figure 8.7 Coding for the Page_Load method.

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

Next we need to create a class or field-level variable, sqlConnection, for our connection object. Enter the following code under the class header:

```
public SqlConnection sqlConnection;
```

This connection object will be used by all Web forms in this project later.

Now we need to perform the coding for the Page_Load() method, which is similar to the Form_Load() method in the Windows-based application. Open this event method and enter the codes shown in Figure 8.7 into this method.

Let's take a closer look at this piece of code to see how it works.

A. An SQL Server data provider related namespace is added into this project since we need to use those data components to perform data actions against our sample SQL Server database later.

B. A class-level Connection object is declared first, and this object will be used by all Web forms in this project later to connect to our sample database.

C. As we did for the Form_Load() method in the Windows-based applications, we need to perform the database connection job in this Page_Load() method. A connection string is created with the database server name, database name, and security mode.

D. A new database Connection object is created with the connection string as the argument.

E. The Connection object sqlConnection is added into the Application state function, and this object can be used by any pages in this application by accessing this Application state function later. Unlike global variables in the Windows-based applications, one cannot access a class variable by prefixing the form's name before the class variable declared in that form from other pages. In the Web-based application, the Application state function is a good place to store any global variable. In ASP.NET Web application, the Application state is stored in an instance of the HttpApplicationState class, which can be accessed

through the Application property of the HttpContext class in the server side and is faster than storing and retrieving information in a database.

F. First, we need to check whether this database connection has been done. If it has, we need first to disconnect this connection by using the Close() method.

G. Then we can call the Open() method to set up the database connection.

H. By checking the database connection state property, we can confirm the connection we did. If the connection state is not equal to Open, which means that the database connection has failed, a warning message is displayed and the procedure is exited.

One significant difference in using the Message box to display some debugging information in the Web form is that you cannot use a Message box as you did in the Windows-based applications. In the Web form development, no Message box is available, and you can only use the Javascript alert() method to display a Message box in ASP.NET. Two popular objects are widely utilized in the ASP.NET Web applications: The Request and the Response objects. The ASP Request object is used to get information from the user, and the ASP Response object is used to send output to the user from the server. The Write() method of the Response object is used to display the message sent by the server. You must add the script tag <script>……</script> to indicate that the content is written in Javescript language.

Now let's perform the coding for the LogIn button's Click method. The function of this piece of coding is to access the LogIn table located in our sample SQL Server database based on the username and password entered by the user to try to find the matched logon information. Currently, since we have not created our next page—Selection page—we just display a Message box to confirm the success of the logon process if it is. Click on the View Design button from the Solution Explorer window and then double-click on the LogIn button to open its Click method. Enter the codes shown in Figure 8.8 into this method.

Let's take a closer look at this piece of code to see how it works.

A. An SQL query statement is declared first since we need to use this query statement to retrieve the matched username and password from the LogIn table. Because this query statement is relatively long, we split it into two substrings.

B. All data objects related to the SQL Server Data Provider are created here, such as the Command object and DataReader object.

C. The Command object is initialized and built by assigning it with the Connection object, commandType, and Parameters collection properties of the Command class. The Add() method is utilized to add two actual dynamic parameters to the Parameters collection of the Command class.

D. The ExecuteReader() method of the Command class is executed to access the database, retrieve the matched username and password, and return them to the DataReader object.

E. If the HasRows property of the DataReader is true, at least one matched username and password has been found and retrieved from the database. A successful message is created and sent back from the server to the client to display it in the client browser.

F. Otherwise, no matched username or password has been found from the database, and a warning message is created and sent back to the client and displayed in the client browser.

G. The used objects such as the Command and the DataReader are released.

_Default | cmdLogIn_Click()

```
protected void cmdLogIn_Click(object sender, EventArgs e)
{
A       string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
        cmdString += "WHERE (user_name=@name) AND (pass_word=@word)";
B       SqlCommand sqlCommand = new SqlCommand();
        SqlDataReader sqlReader;
C       sqlCommand.Connection = sqlConnection;
        sqlCommand.CommandType = CommandType.Text;
        sqlCommand.CommandText = cmdString;
        sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = txtUserName.Text;
        sqlCommand.Parameters.Add("@word", SqlDbType.Char, 8).Value = txtPassWord.Text;
D       sqlReader = sqlCommand.ExecuteReader();
E       if (sqlReader.HasRows == true)
        {
            Response.Write("<script>alert('LogIn is successful!')</script>");
        }
F       else
            Response.Write("<script>alert('No matched username/password found!')</script>");
G       sqlCommand.Dispose();
        sqlReader.Close();
}
```

Figure 8.8 Coding for the LogIn button's Click method.

_Default | cmdCancel_Click()

```
protected void cmdCancel_Click(object sender, EventArgs e)
{
A       if (sqlConnection.State == ConnectionState.Open)
            sqlConnection.Close();
B       Response.Write("<script>window.close()</script>");
}
```

Figure 8.9 Coding for the Cancel button's Click method.

Next let's make the coding for the Cancel button's Click method. The function of this method is to close the current Web page if one clicks on this Cancel button, which means that the user wants to terminate the ASP.NET Web application. Double-click on the Cancel button from the Design View of the LogIn Form to open this method and enter the codes shown in Figure 8.9 into this method.

The function of this piece of code is:

A. First, we need to check whether the database is still connected to our Web form. If it is, we need to close this connection before we can terminate our Web application.

B. The server sends back a command with the Response object's method Write() to issue a Javascript statement window.close() to close the Web application.

At this point, we have finished developing the codes for the LogIn Web form. Before we can run the project to test our Web page, we need to add some data validation functions in the client side to reduce the burden of the server.

Table 8.2 Validation Controls

Validation Control	Functionality
RequiredFieldValidator	Validate whether the required field has valid data (not blank).
RangeValidator	Validate whether a value is within a given numeric range. The range is defined by the MaximumValue and MinimumValue properties provided by users.
CompareValidator	Validate whether a value fits a given expression by using the different Operator property such as 'equal', 'greater than', 'less than' and the type of the value, which is setting by the Type property.
CustomValidator	Validate a given expression using a script function. This method provides the maximum flexibility in data validation but one needs to add a function to the Web page and sends it to the server to get the feedback from it.
RegularExpressionValidator	Validate whether a value fits a given regular expression by using the ValidationExpression property, which should be provided by the user.

8.3.3 Validate Data on Client Side

As we mentioned in Section 8.2.3, in order to reduce the burden of the server, we should make every effort to perform the data validation on the client side. In other words, before we can send requests to the server, we need to make sure that the information to be sent to the server should be as correct as possible. ASP.NET provides some tools to help us to complete this data validation. These tools include five validation controls that are shown in Table 8.2. All of these five controls are located at the Validation tab in the Toolbox window in Visual Studio.NET environment.

Here we want to use the first control, RequiredFieldValidator, to validate our two textboxes, txtUserName and txtPassWord, in the LogIn page to make sure that both are not empty when the user clicks on the LogIn button as the project runs.

Open the Design View of the LogIn Web form, go to the Toolbox window, and click on the Validation tab to expand it. Drag the RequiredFieldValidator control from the Toolbox window and place it next to the username textbox. Set the following properties to this control in the property window:

- ErrorMessage UserName is required
- ControlToValidate txtUserName

Perform the similar dragging and placing operations to place the second RequiredFieldValidator just next to the password textbox. Set the following properties for this control in the property window:

- ErrorMessage PassWord is required
- ControlToValidate txtPassWord

Your finished LogIn Web form should match the one shown in Figure 8.10.

Now run our project to test this data validation by clicking on the Start Debugging button, without entering any data into two textboxes, and then click on the LogIn button. Immediately two error messages, which are created by the RequiredFieldValidators, are displayed to ask users to enter these two pieces of information. After entering the username and password, click on the LogIn button again; a successful login message is displayed. So you can see how the RequiredFieldValidator works to reduce the processing load for the server.

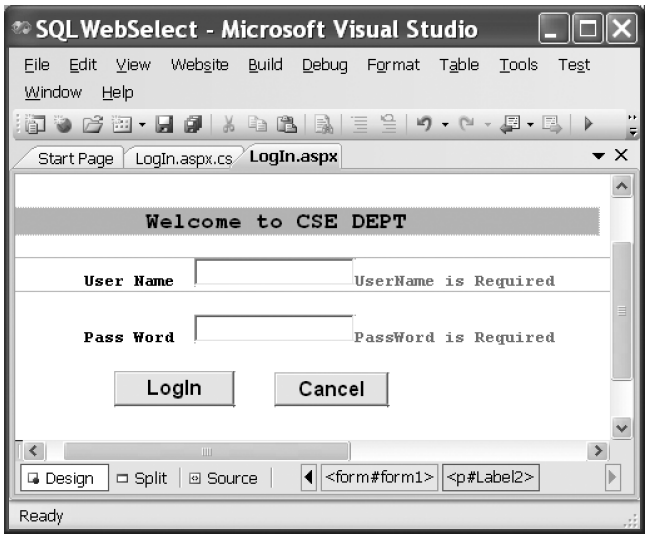

Figure 8.10 Adding the data validation—RequiredFieldValidator.

One good thing always brings some bad thing, which is true to our project, too. After the RequiredFieldValidator is added into our Web page, the user cannot close the page by clicking on the Cancel button if both username and password textboxes are empty. This is because the RequiredFieldValidator is performing the validation checking and no further action can be taken by the Web page until both textboxes are filled with some valid information. Therefore, if you want to close the Web page now, you have to enter a valid username and password, and then you can close the page by clicking on the Cancel button.

8.3.4 Create Second User Interface—Selection Page

Now let's continue to develop our Web application by adding another Web page, the Selection page. As we did in previous chapters, after the logon process is finisfhed, the next step is to allow users to select different functions from the Selection form to perform the associated database actions.

The function of this Selection page is to allow users to visit different pages to perform the different database actions such as selecting, inserting, updating, or deleting data in the the database via the different tables by selecting the different items. Therefore, this Selection page needs to perform the following operations:

1. Provide and display all available selections to allow users to select them.
2. Open the associated page based on the users' selection.

Now let's build this page. To do that, we need to add a new Web page. Right-click on the project icon from the Solution Explorer window and select the Add New Item from the pop-up menu. On the opened window, keep the default Template Web Form selected, and enter Selection.aspx into the Name box as the name for this new page,

Table 8.3 Controls for the Selection Form

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	Make Your Selection:	0	#E0E0E0	Bold/Large
DropDownList	ComboSelection		1		
Button	cmdSelect	Select	2		Bold/Medium
Button	cmdExit	Exit	3		Bold/Medium

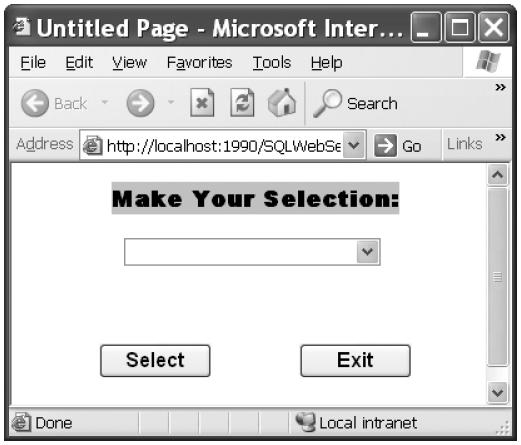

Figure 8.11 Finished Selection page.

and then click on the Add button to add this page into our project. On the opened Web form, add the controls listed in Table 8.3 into this page.

As we mentioned in the last section, before you pick up those controls from the Toolbox window and drag them into the page, you must first use the Space or the Enter keys on the keyboard to locate the positions on the page for those controls. Your finished Selection page should match the one shown in Figure 8.11.

Next let's create the codes for this Selection page to allow users to select the different page to perform the associated data actions.

8.3.5 Develop Codes to Open Other Page

First, let's run the Selection page to build the Web configuration file. Click on the Start Debugging button to run this page, and then click on the Close button located at the upper-right corner of the page to close it.

Click on the View Code button from the Solution Explorer window to open the code page for the Selection Web form. First, let's add an SQL Data Provider–related namespace to the top of this page to provide a reference to all data components of the SQL Data Provider:

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

Then enter the codes shown in Figure 8.12 into the Page_Load() method to add all selection items into the combobox control ComboSelection to allow users to make their selection as the project runs.

Selection | Page_Load()

```
protected void Page_Load(object sender, EventArgs e)
{
      ComboSelection.Items.Add("Faculty Information");
      ComboSelection.Items.Add("Course Information");
      ComboSelection.Items.Add("Student Information");
}
```

Figure 8.12 Coding for the Page_Load method of the Selection page.

Selection | cmdSelect_Click()

```
protected void cmdSelect_Click(object sender, EventArgs e)
{
      if (ComboSelection.Text == "Faculty Information")
         Response.Redirect("Faculty.aspx");
      else if (ComboSelection.Text == "Student Information")
         Response.Redirect("Student.aspx");
      else if (ComboSelection.Text == "Course Information")
         Response.Redirect("Course.aspx");
}
```

Figure 8.13 Coding for the Select button's Click method.

The function of this piece of code is straightforward. Three pieces of CSE Dept–related data are added into the combobox ComboSelection by using the Add() method, and these pieces of data will be selected by the user as the project runs.

Next we need to create the codes for two buttons' Click methods. First, let's do the coding for the Select button. Click on the View Designer button from the Solution Explorer window to open the Selection Web form, and then double-click on the Select button to open its method. Enter the codes shown in Figure 8.13 into this method.

The function of this piece of code is easy. Based on the information selected by the user, the related Web page is opened by using the server's Response object, that is, by using the Redirect() method of the server's Response object. These three pages will be created and discussed in the following sections.

Finally let's take care of the coding for the Exit button's Click method. The function of this piece of code is to close the database connection and close the Web application. Double-click on the Exit button from the Design View of the Selection page to open this method. Enter the codes shown in Figure 8.14 into this method.

First, we need to check if the database is still connected to our application. If it is, the global connection object stored in the Application state is activated with the Close() method to close the database connection. Then the Write() method of the server Response object is called to close the Web application. A key point is that the Application state function stores an object, the Connection object in this case. In order to access and use that Connection object stored in the Application global function, a casting (SqlConnection) must be clearly prefixed before that object; therefore, a two-layer parenthesis is used to complete this casting. Otherwise a compiling error will be encountered since the compiler cannot recognize and convert the general object stored in the Application state function to a specific Connection object.

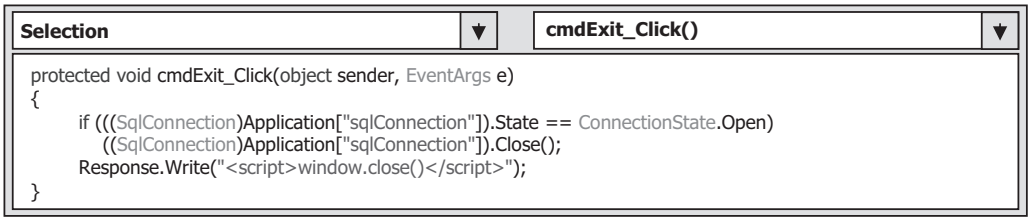

Figure 8.14 Coding for the Exit button's Click method.

Now we have finished the coding for the Selection page. Before we can run the project to test this page, we need to do some modifications to the coding in the LogIn button's Click method in the LogIn page to allow the application to switch from the LogIn page to the Selection page as the LogIn process is successful.

Open the LogIn page and the LogIn button's Click method, and replace the code body located inside the `if` block:

```
Response.Write("<script>alert('LogIn is successful!')</script>");
```

with the following code:

```
Response.Redirect("Selection.aspx");
```

In this way, as long as the logon process is successful, the next page, the Selection page, will be opened by executing the Redirect() method of the server Response object. The argument of this method is the URL address of the Selection page. Since the Selection page is located at the same application as the LogIn page, a direct page name is used.

Now let's run the application to test two pages. Make sure that the LogIn page is the starting page for our application. To do that, right-click on the LogIn.aspx from the Solution Explorer window and select the item Set As Start Page from the pop-up menu. Click the Start Debugging button to run our project. Enter the suitable username and password such as ybai and reback to the username and password boxes; then click on the LogIn button. The Selection page is displayed if the logon process is successful, as shown in Figure 8.15. Click on the Exit button to close our application. Now let's begin to develop our next page, the Faculty page.

8.3.6 Create Third User Interface—Faculty Page

Right-click on our project folder from the Solution Explorer window and select the Add New Item from the pop-up menu. On the opened dialog box, keep the default Template Web Form selected, and then enter Faculty.aspx into the Name box as the name for this new page, and click on the Add button to add this new page into our project. On the opened Web form, add the controls shown in Table 8.4 into this page.

As we mentioned in the last section, before you can drag those controls from the Toolbox window and place them into the page, you must first use the Space or the Enter keys on the keyboard to locate the positions on the page for those controls. You cannot just place a control in a random position on the form as you did in the Windows-based applications since the Web-based applications have special layout requirements.

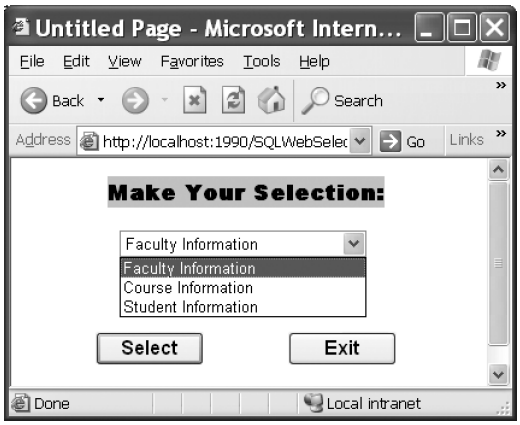

Figure 8.15 Running status of the second page—Selection page.

Table 8.4 Controls for the Faculty Form

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	CSE DEPT Faculty Page	0	#E0E0E0	Bold/Large
Image	PhotoBox		22		
Label	Label2	Faculty Name	1		Bold/ Small
DropDownList	ComboName		2		
Label	Label3	Faculty ID	3		Bold/ Small
TextBox	txtID		4		
Label	Label4	Name	5		Bold/ Small
TextBox	txtName		6		
Label	Label5	Title	7		Bold/ Small
TextBox	txtTitle		8		
Label	Label6	Office	9		Bold/ Small
TextBox	txtOffice		10		
Label	Label7	Phone	11		Bold/ Small
TextBox	txtPhone		12		
Label	Label8	College	13		Bold/ Small
TextBox	txtCollege		14		
Label	Label9	Email	15		Bold/ Small
TextBox	txtEmail		16		
Button	cmdSelect	Select	17		Bold/ Small
Button	cmdInsert	Insert	18		Bold/ Small
Button	cmdUpdate	Update	19		Bold/ Small
Button	cmdDelete	Delete	20		Bold/ Small
Button	cmdBack	Back	21		Bold/ Small

One important point to note is the position of the Image control on the page form. After you drag an Image control from the Toolbox window and place it into the page window, you should set the ImageAlign property to Left. In this way, we can continue to add all other controls such as labels and textboxes to this Web page without any overlapping problem.

Now you can enlarge this Image and place it on the left side of this Web page. Your finished Faculty page should match the one shown in Figure 8.16.

Although we have added five buttons into this Faculty page, in this section we only take care of the Select and the Back buttons, that is, two buttons' Click methods, since

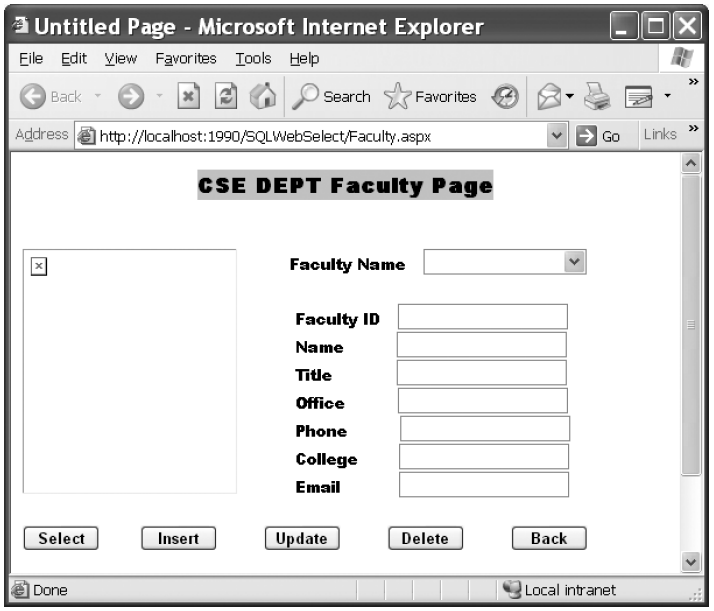

Figure 8.16 Finished Faculty page.

we want to discuss how to retrieve the queried data from our database and display them in this Faculty page. Other buttons will be used in the following sections later. Now let's begin to develop the codes for the Faculty page.

8.3.7 Develop Codes to Select Desired Faculty Information

First, let's run the project to build the configuration file web.config to configure all controls we just added into the Faculty page. Click on the Start Debugging button to run the project, and enter the suitable username and password to open the Selection page. Select the Faculty Information item from this page to open the Faculty page. Click on the Close button located at the upper-right corner of this page to close the project.

Open the code page of the Faculty form, and, as we did before, let's first add an SQL Data Provider–related namespace to the top of this page to provide a reference to all data components of the SQL Data Provider:

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

The coding for this page can be divided into three parts: Coding for the Page_Load() method, coding for the Select button's Click method, and coding for other methods. First, let's take care of the coding for the Page_Load() method.

8.3.7.1 Develop Codes for Page_Load Method

In the opened code page, open the Page_Load() method and enter the codes shown in Figure 8.17 into this method. Let's that a closer look at this piece of code to see how it works.

Faculty | Page_Load()

```
.....
A  using System.Data.SqlClient;
   public partial class Faculty : System.Web.UI.Page
   {
B     private TextBox[] FacultyTextBox = new TextBox[7];
      protected void Page_Load(object sender, EventArgs e)
      {
C        if (((SqlConnection)Application["sqlConnection"]).State != ConnectionState.Open)
            ((SqlConnection)Application["sqlConnection"]).Open();
D        if (!IsPostBack)
         {
            ComboName.Items.Add("Ying Bai");
            ComboName.Items.Add("Satish Bhalla");
            ComboName.Items.Add("Black Anderson");
            ComboName.Items.Add("Steve Johnson");
            ComboName.Items.Add("Jenney King");
            ComboName.Items.Add("Alice Brown");
            ComboName.Items.Add("Debby Angles");
            ComboName.Items.Add("Jeff Henry");
         }
      }
   }
```

Figure 8.17 Coding for the Page_Load method.

A. An SQL Server Data Provider–related namespace is added into this namespace area since we need to use some SQL Server data components located in that namespace.

B. A field-level textbox array is created first since we need this array to hold six pieces of faculty information and display them in these six textboxes later.

C. Before we can perform the data actions against the database, we need to make sure that a valid database connection is set to allow us to transfer data between our project and the database. An Application state, which is used to hold our global connection object variable, is utilized to perform this checking and connecting to our database if it has not been connected.

D. As the project runs, each time as the user clicks on the Select button to perform a data query, a request is sent to the database server and the Web server (it can be the same server as the database server). Then the Web server will post back a refreshed Faculty page to the client when it received this request (IsPostBack = true). Each time this happens, the Page_Load() method will be activated again, and the duplicated eight faculty members are attached to the end of the combobox control ComboName again. To avoid this duplication, we need to check the IsPostBack property of the page and add eight faculty members into the combobox control only one time when the project starts (IsPostBack = false). Refer to Section 8.3.8.1 for more detailed discussion about the AutoPostBack property.

Next we need to develop the coding for the Select button's Click method to perform the data query actions against the database.

8.3.7.2 Develop Codes for Select Button Method

The function of this coding is to make queries to the database to retrieve the faculty information based on the selected faculty member by the user from the combobox control

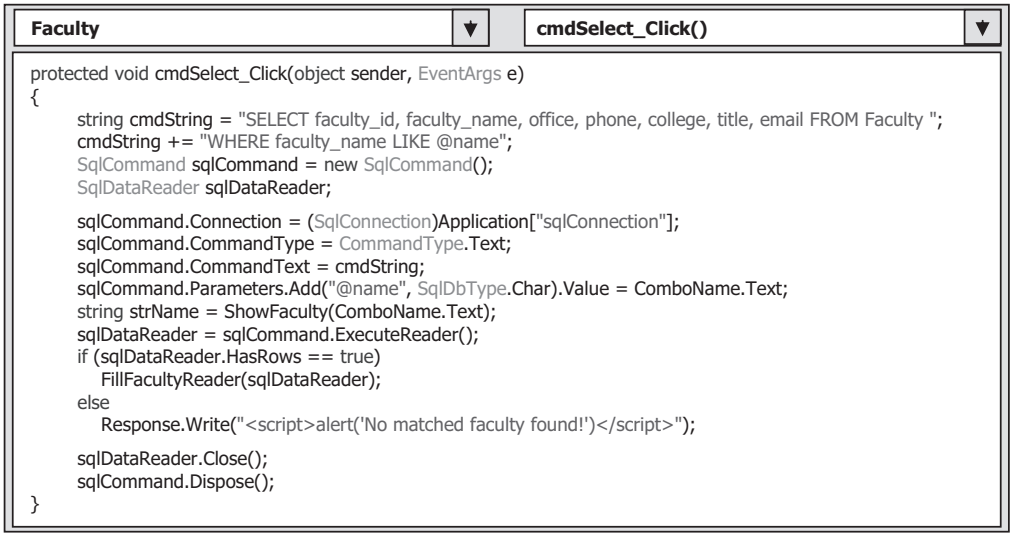

Figure 8.18 Coding for the Select button's Click method.

ComboName, and then display that retrieved information in the six textbox controls on the Faculty page.

Open this Select button's Click method by double-clicking on this button from the Design View of the Faculty Form, and enter the codes shown in Figure 8.18 into this method.

Let's take a look at this piece of code to see how it works.

A. The query string that contains a SELECT statement is declared here since we need to use this string as our command text. The dynamic parameter of this query is the @name, which is the selected faculty name by the user as the project runs.

B. All data components such as the Command and DataReader objects are declared here since we need to use them to perform the data query later.

C. The Command object is initialized by assigning the associated components to it. These components include the global Connection object that is stored in the Application state function, the CommandType, and the CommandText properties.

D. The Parameter object is initialized by assigning the dynamic parameter's name and value to it.

E. The user-defined ShowFaculty() method is called to display the selected faculty photo in the Image control on the Faculty page. We will develop this method in the next section.

F. The ExecuteReader() method of the Command object is called to execute the query command to retrieve the selected faculty information, and assign it to the DataReader object.

G. By checking the HasRows property of the DataReader, we can determine whether this query is successful or not. If this property is greater than zero, which means that at least one row is retrieved from the Faculty table in the database and therefore the query is successful, a user-defined method FillFacultyReader(), which we will develop in the next part, is called to fill the six textboxes on the Faculty page with the retrieved faculty information.

H. Otherwise if the HasRows property is equal to zero, which means that no row has been retrieved from the database, the query has failed. A warning message is displayed to the client by calling the Write() method of the server Response object.

I. All data components used for this data query are released after this query.

At this point we have finished the coding for the Select button's Click method.

8.3.7.3 Develop Codes for Other Methods

Next let's take care of the coding for other methods in this Faculty page, which includes the coding for the following methods:

1. User-defined ShowFaculty() method
2. User-defined FillFacultyReader() method
3. User-defined MapFacultyTable() method
4. Back button's Click method

The third method, MapFacultyTable(), is used and called by the second user-defined FillFacultyReader() method in our project. Now let's discuss the coding for these methods one by one.

First, let's see the coding for the ShowFaculty() method. The function of this method is to get the matched faculty photo from the default location based on the input faculty name and display it in the Image control on the Faculty page. The so-called default location for the photo file is exactly the current ASP.NET Web application folder. In our case, it is C:\Book6\Chapter 8\SQLWebSelect. You must place all faculty photo files in this location before you can run this project to pick up the desired faculty information from the database and display it in the Faculty page. Of course, you can place your faculty photo files in any folder in your computer. In that case, you must provide the full name for the faculty photo, including the drive, path, and the name of the photo file. In this project, to make it simple, we used the default folder to store our faculty photo files.

Open the code page of the Faculty page, and type and create this method, which is shown in Figure 8.19. Let's take a look at the coding for this method to see how it works.

A. A local string variable FacultyImage is created, and it is used to hold the name of the matched faculty photo file.

B. A switch…case structure is utilized to find the matched faculty photo file based on the input faculty name. The name of the matched faculty photo file is assigned to the local string variable FacultyImage if it is found.

C. The name of the matched faculty photo file is assigned to the ImageUrl property of the Image control to display that photo if the FacultyImage is not equal to "No Match".

D. Otherwise, a warning message is displayed in the client using the Write() method of the server Response object.

E. The FacultyImage is returned to the calling method.

One significant difference in displaying an image between the Windows-based and the Web-based application is that the Image.Url property, which belongs to the control System.Web.UI.WebControls.Image, is utilized to access the matched faculty photo file and only the name of an image file is needed to display the associated image in the

Faculty | ShowFaculty()

```
private string ShowFaculty(string fName)
{
A       string FacultyImage;
B       switch (fName)
        {
           case "Black Anderson":
              FacultyImage = "Anderson.jpg";
              break;
           case "Ying Bai":
              FacultyImage = "Bai.jpg";
              break;
           case "Satish Bhalla":
              FacultyImage = "Satish.jpg";
              break;
           case "Steve Johnson":
              FacultyImage = "Johnson.jpg";
              break;
           case "Jenney King":
              FacultyImage = "King.jpg";
              break;
           case "Alice Brown":
              FacultyImage = "Brown.jpg";
              break;
           case "Debby Angles":
              FacultyImage = "Angles.jpg";
              break;
           case "Jeff Henry":
              FacultyImage = "Henry.jpg";
              break;
           default:
              FacultyImage = "No Match";
              break;
        }
C       if (FacultyImage != "No Match")
           PhotoBox.ImageUrl = FacultyImage;
D       else
           Response.Write("<script>alert('No matched faculty image found!')</script>");
E       return FacultyImage;
}
```

Figure 8.19 Coding for the ShowFaculty method.

Web-based application. In the Windows-based application, a System.Drawing() method must be used to display an image based on the image file's name.

The next method is the FillFacultyReader(). Open the code page of the Faculty Web form, type the codes shown in Figure 8.20 to create this user-defined method inside the Faculty class.

The function of this method is to pick up each data column from the retrieved data that is stored in the DataReader and assign it to the associated textbox on the Faculty page to display it.

Let's see how this piece of code works.

A. A loop counter intIndex is declared.

B. Seven instances of the object array, textbox array, are created and initialized. These seven objects are mapped to seven columns in the Faculty table in our sample database.

Faculty | FillFacultyReader()

```
private void FillFacultyReader(SqlDataReader FacultyReader)
{
A       int intIndex = 0;
B       for (intIndex = 0; intIndex <= 6; intIndex++)       //Initialize the object array
           FacultyTextBox[intIndex] = new TextBox();
C       MapFacultyTable(FacultyTextBox);
D       while (FacultyReader.Read())
        {
E          for (intIndex = 0; intIndex <= FacultyReader.FieldCount - 1; intIndex++)
              FacultyTextBox[intIndex].Text = FacultyReader.GetString(intIndex);
        }
}
```

Figure 8.20 Coding for the FillFacultyReader method.

C. Another user-defined method MapFacultyTable() is called to set up the correct mapping between the seven textbox controls on the Faculty page window and seven columns in the query string cmdString.

D. A `while` loop is executed as long as the loop condition Read() method is `true`, which means that a valid data is read out from the DataReader. This method will return a `false` if no valid data can be read out from the DataReader, which means that all data has been read out. In this application, in fact, this `while` loop is only executed once since we have only one row (one record) read out from the DataReader.

E. A `for` loop is utilized to pickup each data read out from the DataReader object, and assign each of them to the associated textbox control on the Faculty page window. The intIndex is used here to identify each item from the DataReader.

The detailed codes for the user-defined MapFacultyTable() method are shown in Figure 8.21. The function of this method, as we mentioned, is to set up a correct mapping relationship between seven textboxes in the textbox array on the Faculty page and the seven data columns in the query string since the order in which the textboxes are displayed in the Faculty page may not be identical with the order of the data columns in the query string cmdString we created at the beginning of the Select button's Click method.

Open the code page of the Faculty Web form, type the codes shown in Figure 8.21 to create this method inside the Faculty class.

The order of the seven textboxes on the right-hand side of the equals operator should be equal to the order of the queried columns in the query string—cmdString. By performing this assignment, the seven textbox controls on the Faculty page window have a correct one-to-one relation with the queried columns in the query string cmdString.

Finally, let's take care of the coding for the Back button's click method. The function of this piece of coding is to return to the Selection page as this button is clicked. Double-click on the Back button from the Faculty page window to open this method, and enter the coding line into this method, which is shown in Figure 8.22.

This coding is straightforward and easy to understand. The Redirect() method of the server Response object is executed to direct the client from the current Faculty page back to the Selection page when this button is clicked on by the user; that is, the server resends the Selection page to the client when this button is clicked on and a request is sent to the server.

Faculty | MapFacultyTable()

```
private void MapFacultyTable(Object[] fText)
{
      fText[0] = txtID;
      fText[1] = txtName;
      fText[2] = txtOffice;          //The order must be identical
      fText[3] = txtPhone;           //with the real order in the query string
      fText[4] = txtCollege;
      fText[5] = txtTitle;
      fText[6] = txtEmail;
}
```

Figure 8.21 Coding for the MapFacultyTable method.

Faculty | cmdBack_Click()

```
protected void cmdBack_Click(object sender, EventArgs e)
{
      Response.Redirect("Selection.aspx");
}
```

Figure 8.22 Coding for the Back button's method.

Now we have finished all coding development for the Faculty page. It is time for us to run the project to test our pages. However, before you can run the project, make sure that you have stored all faculty photo files in the default location. Now click on the Start Debugging button to run our project. Enter the suitable username and password such as jhenry and test to the LogIn page, and select the Faculty Information from the Selection page to open the Faculty page. Select one faculty member from the combobox control such as Ying Bai, and then click on the Select button to retrieve the selected faculty information from the database. All information related to that selected faculty is retrieved and displayed on this Faculty page, as shown in Figure 8.23.

Click on the Back button to return to the Selection page, and then click on the Exit button to terminate our project. So far our Web application is successful.

Next we need to create our last Web page, Course page, and add it into our project to select and display all courses taught by the selected faculty member.

8.3.8 Create Fourth User Interface—Course Page

To create a new Web page and add it to our project, go to the Solution Explorer window and right-click on our project folder, select Add New Item from the pop-up menu to open the Add New Item dialog box. On the opened dialog box, keep the default template Web Form selected, enter `Course.aspx` into the Name box as the name for our new page, and click on the Add button to add it to our project. On the opened Web form, add the controls shown in Table 8.5 into this page.

As we mentioned before, you cannot place a control in any position on the form as you like. Now we introduce a technology to place a control at your desired position on the page window just as you did for your Windows-based application. The key element

Figure 8.23 Running status of the Faculty page.

Table 8.5 Controls on the Course Form

Type	ID	Text	TabIndex	BackColor	Font	AutoPostBack
Panel	Panel1		16	#C0C0FF		
Label	Label1	Faculty Name	0		Bold/Small	
DropDownList	ComboName		1			
ListBox	CourseList		17		Bold/Small	True
Panel	Panel2		18	#C0C0FF		
Label	Label2	Course Title	2		Bold/Small	
TextBox	txtCourse		3			
Label	Label3	Schedule	4		Bold/Small	
TextBox	txtSchedule		5			
Label	Label4	Classroom	6		Bold/Small	
TextBox	txtClassRoom		7			
Label	Label5	Credit	8		Bold/Small	
TextBox	txtCredit		9			
Label	Label6	Enrollment	10		Bold/Small	
TextBox	txtEnrollment		11			
Button	cmdSelect	Select	12		Bold/ Small	
Button	cmdInsert	Insert	13		Bold/ Small	
Button	cmdUpdate	Update	14		Bold/ Small	
Button	cmdDelete	Delete	15		Bold/ Small	
Button	cmdBack	Back	16		Bold/ Small	

is the Position property for each control you added into the page window. For example, in this Course page, we added two panel controls, one listbox control, five textbox controls, and five button controls. In order to locate those controls at the desired position on the page, you must set up the Format|Position property to Absolute for all controls added to this page except the label controls.

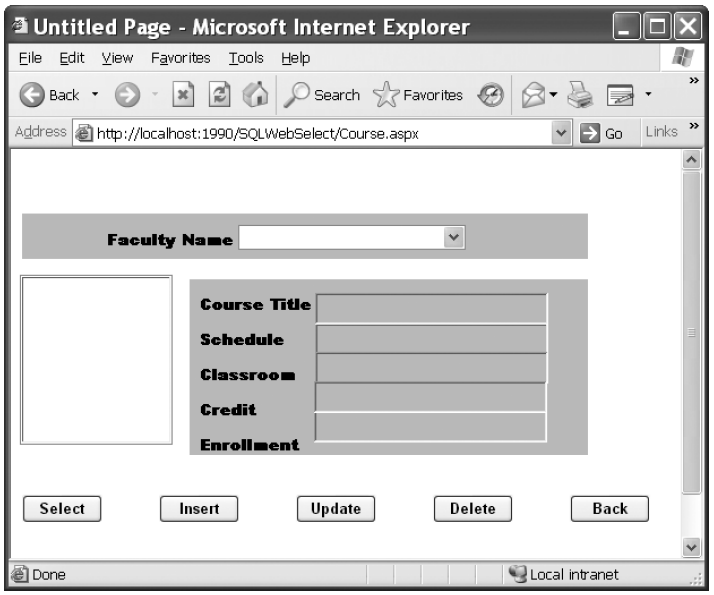

Figure 8.24 Finished Course Web page.

Another issue is the panel size and background color. As you add a new panel into the page, the background color of that new panel is transparent, which means that you cannot see the new added panel. To see that panel, you need to define the size of the panel by performing the following steps:

1. Click on the new added panel to select it and go to the Property window.
2. Set the Height and the Width properties to the desired dimensions in pixels (px).

In this application, set the Height and Width properties for panel1 and panel2 to:

- Panel1: Height 40 px Width 500 px
- Panel2: Height 148 px Width 352 px

Your finished Course page should match the one shown in Figure 8.24. Before we can continue to develop the codes for this page, we must emphasize one key point for the listbox control used in the Web-based applications. There is a significant difference between the Windows-based and Web-based applications for the listbox control.

8.3.8.1 AutoPostBack Property of Listbox Control

One important property is the AutoPostBack property for the listbox control CourseList. Unlike the listbox control used in the Windows-based application, a SelectedIndexChanged event will not be created on the server side if the user clicked on and selected an item from the listbox. The reason for that is because the default value for the AutoPostBack property of a listbox control is set to `false` when you add a new listbox to your Web form. This means that even if the user clicked on and changed the item from the listbox, a SelectedIndexChanged event can only be created on the client side, and it cannot be

sent to the server. As you know, the listbox is running on the server side when your project runs. Therefore, no matter how many times you clicked on and changed the items in the listbox at the client side, no event can be created on the server side. Therefore, the project will not respond to your clicking on the listbox control as the project runs.

However, in this project we need to use this SelectedIndexChanged event to trigger its event method to perform the course information query. To solve this problem, the AutoPostBack property should be set to `true`. In this way, each time you click on an item to select it from the listbox, this AutoPostBack property will be set to post back to the server to indicate that the user has interacted with that control.

In this section, we only discuss the coding for the Select and the Back buttons' click methods to perform the course data query. The operations for the other buttons such as Insert, Update, and Delete will be discussed in the following sections when we perform the data inserting or updating in the database using the Web pages.

Now let's develop the codes for the Select and the Back buttons' click methods to pick up the course data from the database using the Course Web page.

8.3.9 Develop Codes to Select Desired Course Information

The functions of the Course page are:

1. When the user selects the desired faculty member from the Faculty Name combobox control and clicks on the Select button, the IDs of all courses taught by the selected faculty should be retrieved from the database and displayed in the listbox control CourseList on the Course page.
2. When the user clicks any `course_id` from the listbox control CourseList, the detailed course information related to the selected `course_id` in the listbox will be retrieved from the database and displayed in five textboxes on the Course page form.

Based on the function analysis above, we need to concentrate on the coding for two methods: one is the Select button's click method and the second is the SelectedIndexChanged event method of the listbox control CourseList. The first coding is used to retrieve and display all `course_id` related to courses taught by the selected faculty in the listbox control CourseList, and the second coding is to retrieve and display the detailed course information such as course title, schedule, classroom, credit, and enrollment related to the selected `course_id` from the CourseList.

The above coding operations can be divided into four parts:

1. Coding for the Course page loading and ending event methods. These methods include the Page_Load() and the Back button's click method.
2. Coding for the Select button's click method.
3. Coding for the SelectedIndexChanged event method of the listbox control CourseList.
4. Coding for other user-defined methods.

Before we can start the first coding operation, we need to add an SQL Server Data Provider related namespace to the top of the Course page. Open the code window of the Course page and enter the following namespace to the top of that page:

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

Course | Page_Load()

```
.....
A  using System.Data.SqlClient;
   public partial class Course : System.Web.UI.Page
   {
B     private TextBox[] CourseTextBox = new TextBox[6];
      protected void Page_Load(object sender, EventArgs e)
      {
C        if (((SqlConnection)Application["sqlConnection"]).State != ConnectionState.Open)
            ((SqlConnection)Application["sqlConnection"]).Open();

D        if (!IsPostBack)                    //these items can only be added into the combo box in one time
         {
              ComboName.Items.Add("Ying Bai");
              ComboName.Items.Add("Satish Bhalla");
              ComboName.Items.Add("Black Anderson");
              ComboName.Items.Add("Steve Johnson");
              ComboName.Items.Add("Jenney King");
              ComboName.Items.Add("Alice Brown");
              ComboName.Items.Add("Debby Angles");
              ComboName.Items.Add("Jeff Henry");
         }
      }
   }
```

Figure 8.25 Coding for the Page_Load method.

8.3.9.1 Coding for Course Page Loading and Ending Methods

Open the Page_Load() method and enter the codes shown in Figure 8.25 into this method.

Let's take a closer look at this piece of code to see how it works.

A. The SQL Server Data Provider–related namespace is added into this page since we need to use some data components stored in that namespace to perform the data actions against the Course table in our sample database.

B. This coding fragment is similar to one we did for the Faculty Form. Five textbox controls in the Course Form page are used to display the detailed course information that is related to the selected faculty from the Faculty Name combobox. The Course table has seven columns, although we only need five of them, we still set the size of this TextBox array to 6. Each element or each TextBox control in this array is indexed from 0 to 5.

C. The function of this code segment is: Before we can perform any data query, we need to check whether a valid database connection is available. Since we created a class-level connection instance in the LogIn page and stored it in the Application state, now we need to check this connection object and reconnect it to the database if our application has not been connected to the database.

D. This piece of codes is used to initialize the Faculty Name combobox control, and the Add() method is utilized to add all faculty members into this combobox control to allow users to select one to get the course information as the project runs. Here a potential bug exists for this piece of code. As we mentioned in Section 8.3.8.1, an AutoPostBack property will be set to `true` whenever the user clicked on and selected an item from the listbox control, and this property will be sent back to the server to indicate that an action has been taken by the user to this listbox. After the server received this property, it will resend a refreshed

Course page to the client. Therefore, the Page_Load() method of the Course page will be triggered and run again as a refreshed Course page is sent back. The result of execution of this Page_Load() method is to attach another copy of all faculty members to the end of those faculty members that have been already added into the combobox control ComboName when the Course page is displayed in the first time. As the number of the item you clicked on from the CourseList box increases, the number of copies of all faculty members will also increase and be displayed the ComboName box. To avoid this duplication, we only need to add all faculty members the first time as the Course page is displayed, but do nothing if any another AutoPostBack property occurs.

The coding for the Back button's click method is similar to that for the Back button in the Faculty page. When this button is clicked on by the user, the Course page should be switched back to the Selection page. The Redirect() method of the server Response object is used to fulfill this switching back function. Double-click on the Back button from the Course page window and enter the following codes into this method:

```
Response.Redirect("Selection.aspx");
```

Let's continue with the coding for the Select button's click method.

8.3.9.2 Coding for Select Button Click Method

As we mentioned at the beginning of this section, the function of this method is: When the user selects the desired faculty member from the Faculty Name combobox control and clicks on the Select button, all `course_id` related to courses taught by the selected faculty should be retrieved from the database and displayed in the listbox control CourseList on the Course page.

Double-click on the Select button from the Course page window to open its Click method, and enter the codes shown in Figure 8.26 into this method.

Let's take a look at this piece of code to see how it works.

Course | cmdSelect_Click()

```
protected void cmdSelect_Click(object sender, EventArgs e)
{
A       string strCourse = "SELECT Course.course_id, Course.course FROM Course JOIN Faculty ";
        strCourse += "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.faculty_name LIKE @name)";
B       SqlCommand sqlCommand = new SqlCommand();
        SqlDataReader sqlDataReader;

C       sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
        sqlCommand.CommandType = CommandType.Text;
        sqlCommand.CommandText = strCourse;
        sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = ComboName.Text;
D       sqlDataReader = sqlCommand.ExecuteReader();
E       if (sqlDataReader.HasRows == true)
            FillCourseReader(sqlDataReader);
F       else
            Response.Write("<script>alert('No matched course found!')</script>");
G       sqlDataReader.Close();
        sqlCommand.Dispose();
}
```

Figure 8.26 Coding for the Select button's Click method.

A. The joined table query string is declared at the beginning of this method. Here two columns are queried. The first one is the `course_id` and the second is the course name. The reason for this is that we need to use the `course_id`, not course name, as the identifier to pick up each course' detailed information from the Course table when the user clicks on and selects the `course_id` from the CourseList box. We use the `course_id` with the course name together in this joined table query, and we will use that `course_id` later. The comparator LIKE is used to replace the original equals symbol for the criteria in the ON clause in the definition of the query string, and this is required by SQL Server database operation. For a more detailed description about the joined table query, refer to Section 5.19.2.5 in Chapter 5.

B. Some SQL data objects such as the Command and DataReader are created here. All of these objects should be prefixed by the keyword sql to indicate that all those components are related to the SQL Server database.

C. The sqlCommand object is initialized with the connection string, command type, command text, and command parameter. The parameter's name must be identical with the dynamic nominal name @name, which is defined in the query string and is located after the LIKE comparator in the ON clause. The parameter's value is the content of the Faculty Name combobox, which should be entered by the user as the project runs later.

D. The ExecuteReader() method of the Command class is executed to read back all courses (`course_id`) taught by the selected faculty and assign them to the DataReader object.

E. If the HasRows property of the DataReader is `true`, which means that at least one row data has been retrieved from the database, the FillCourseReader() method is called to fill the `course_id` into the CourseList box.

F. Otherwise, this joined query has failed and a warning message is displayed.

G. Finally some cleaning is preformed to release objects used for this query.

Now let's take care of the coding for the user-defined FillCourseReader() method, which is shown in Figure 8.27. Open the code page of the Course Web form and enter the codes shown in Figure 8.27 to create this method inside the Course class.

Let's see how this piece of code works.

A. A local string variable strCourse is created, and this variable can be considered an intermediate variable used to temporarily hold the queried data from the Course table.

B. The CourseList box is cleaned up before it can be filled by the `course_id`.

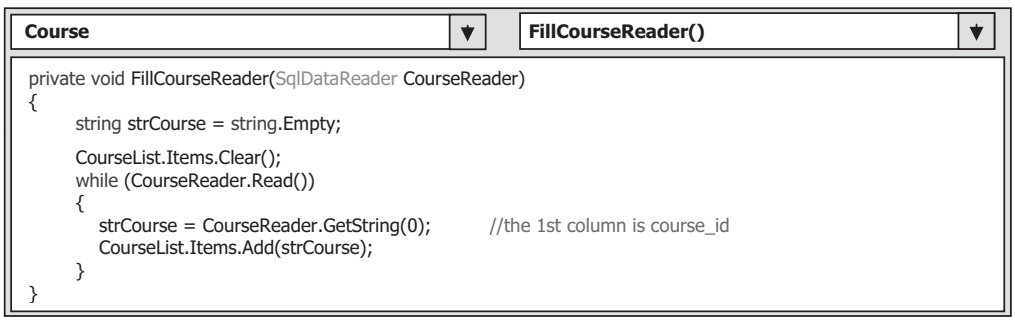

Figure 8.27 Coding for the FillCourseReader method.

C. A `while` loop is utilized to retrieve the first column for each row (GetString(0)), whose column index is 0 and the data value is the `course_id`. The queried data is first assigned to the intermediate variable strCourse, and then it is added into the CourseList box by using the Add() method.

Now let's start to develop the codes for the SelectedIndexChanged event method of the listbox control CourseList. The function of this method is: When the user clicks any `course_id` from the listbox control CourseList, the detailed course information related to the selected `course_id` in the listbox such as course title, schedule, credit, classroom, and enrollment will be retrieved from the database and displayed in five textboxes on the Course page form.

8.3.9.3 Coding for SelectedIndexChanged Method of Listbox Control

Double-click on the listbox control CourseList from the Course Web form to open this event method, and then enter the codes shown in Figure 8.28 into this method.

Let's take a closer look at this piece of code to see how it works.

A. The query string is created with six queried columns such as course, credit, classroom, schedule, enrollment, and course_id. The query criterion is course_id, which was selected by the user by clicking on a valid `course_id` from the CourseList box control as the project runs.

B. Two SQL data objects are created, and these objects are used to perform the data operations between the database and our project. All of these objects should be prefixed by the keyword sql since in this project we used an SQL Server data provider.

C. The sqlCommand object is initialized with the connection object, command type, command text, and command parameter. The parameter's name must be identical with the dynamic nominal name @courseid, which is defined in the query string, exactly after the LIKE comparator in the WHERE clause. The parameter's value is the `course_id` selected by the user from the CourseList box control.

Course | CourseList_SelectedIndexChanged()

```
protected void CourseList_SelectedIndexChanged(object sender, EventArgs e)
{
A       string cmdString = "SELECT course, credit, classroom, schedule, enrollment, course_id FROM Course ";
        cmdString += "WHERE course_id LIKE @courseid";
B       SqlCommand sqlCommand = new SqlCommand();
        SqlDataReader sqlDataReader;

C       sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
        sqlCommand.CommandType = CommandType.Text;
        sqlCommand.CommandText = cmdString;
        sqlCommand.Parameters.Add("@courseid", SqlDbType.Char).Value = CourseList.SelectedItem;
D       sqlDataReader = sqlCommand.ExecuteReader();
E       if (sqlDataReader.HasRows == true)
            FillCourseReaderTextBox(sqlDataReader);
F       else
            Response.Write("<script>alert('No matched course information found!')</script>");
G       sqlDataReader.Close();
        sqlCommand.Dispose();
}
```

Figure 8.28 Coding for the SelectedIndexChanged event method.

D. The ExecuteReader() method is executed to read back the detailed information for the selected course and assign it to the DataReader object.

E. If the HasRows property of the DataReader is `true`, which means that at least one row data has been retrieved from the database, the user-defined method FillCourseReaderTextBox() is called to enter those pieces of data into five textboxes.

F. Otherwise, this query has failed and a warning message is displayed.

G. Finally some cleaning jobs are preformed to release objects used for this query.

The coding for other user-defined methods includes the coding for the user-defined FillCourseReaderTextBox() and MapCourseTable() methods. The function of the user-defined MapCourseTable() method is to set up a one-to-one relationship between each textbox in the Course Form page and each queried column from our database since the order of the textbox in the Course Form page may not be identical to the order of the data column queried from the database based on the query string.

8.3.9.4 Coding for Other User-Defined Methods

First, let's develop the coding for the FillCourseReaderTextBox() method. On the opened code page of the Course Web form, enter the codes shown in Figure 8.29 into the Course class to create this user-defined method.

Let's take a look at this piece of code to see how it works.

A. A loop counter intIndex is first created, and it is used for the loop of creation of the textbox object array and the loop of retrieving data from the DataReader later.

B. The first `for` loop is used to create the textbox object array and perform the initialization for those objects.

C. Another user-defined method MapCourseTable() is executed to set up a one-to-one relationship between each textbox control on the Course page and each queried column in the query string. This step is necessary since the distribution order of five textbox controls on the Course page may differ from the column order in the query string.

D. A `while` and the second `for` loop are used to pick up all five pieces of course-related information from the DataReader one by one. The Read() method is used as the `while` loop condition. A returned `true` means that a valid data is read out from the DataReader,

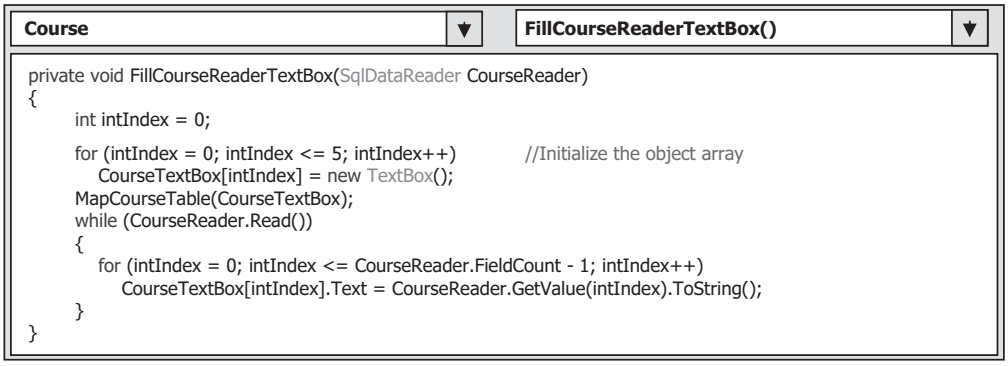

Figure 8.29 Coding for the FillCourseReaderTextBox method.

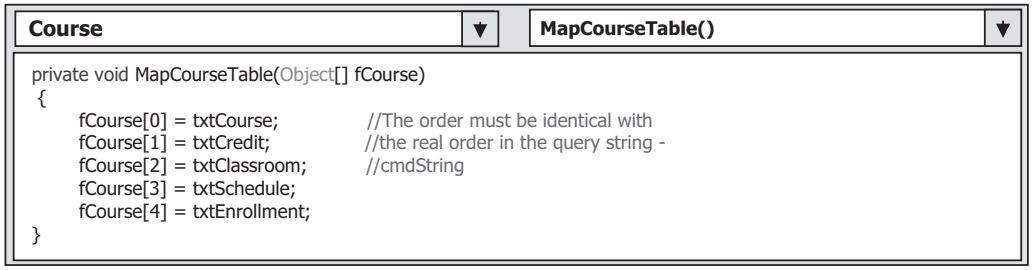

Figure 8.30 Coding for the subroutine MapCourseTable.

and a returned `false` means that no valid data has been read out from the DataReader. In other words, no more data is available and all data has been read out. The second `for` loop uses the FieldCount—1 as the termination condition since the index of the first data column is 0, not 1, in the DataReader object. Each read-out data is converted to a string and assigned to the associated textbox control in the textbox object array.

The coding for the user-defined MapCourseTable() method is shown in Figure 8.30.

The function of this piece of coding is straightforward with no tricks. The order of the textboxes on the right-hand side of the equals operator is aligned with the column order of the query string, cmdString, by assigning each column of queried data to each of its partner, the textbox in the textbox object array in this order. A one-to-one relationship between each column of queried data and the associated textbox control on the Course page is built, and the data retrieved from the DataReader can be mapped exactly to the associated textbox control and displayed there.

At this point, we have finished all coding developments for the Course Web form. Now let's run the project to test the functionality of this form. Click on the Start Debugging button to run the project. Enter the suitable username and password such as jhenry and test to the LogIn page, and select the Course Information item from the Selection page to open the Course page. On the opened page, select a faculty member from the combobox control and then click on the Select button to retrieve all courses taught by that faculty and display them in the CourseList box. Your running result should match the one shown in Figure 8.31.

Click on any `course_id` from the CourseList box to select it. Immediately the detailed information related to that selected `course_id` is displayed in the five textboxes, which is shown in Figure 8.32.

Click on the Back button to return to the Selection page, and you can click on any other item from the Selection page to perform the associated information query or you can click on the Exit button to terminate the application. Our Web application is successful.

A complete Web application project SQLWebSelect, which is used for data query from the SQL Server database, can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

When running the project, a possible bug is the font size of those five textbox controls in this Course Form page. Sometimes it looks like it does not work to set the Font size property for these textboxes in the Properties window. Instead, you have to do that by going to the Format|Font menu item for each textbox.

In the next section, we will discuss how to insert a new record into the SQL Server database via Web page applications.

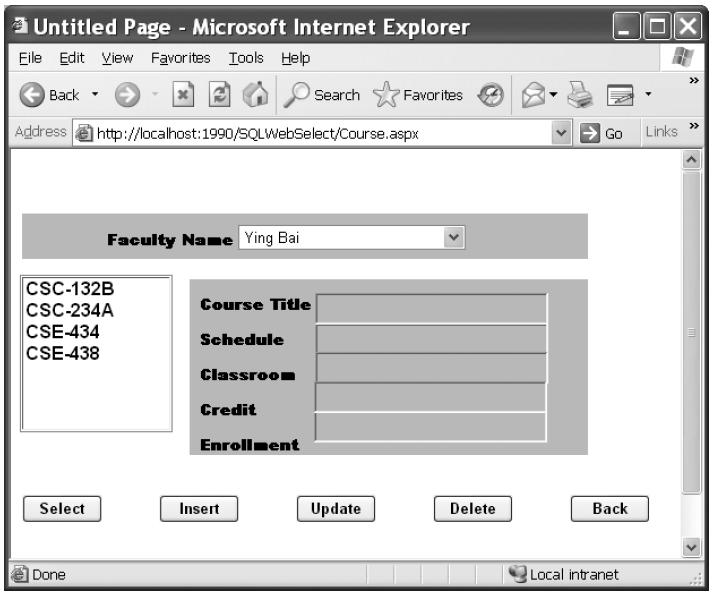

Figure 8.31 Running status of the Course page.

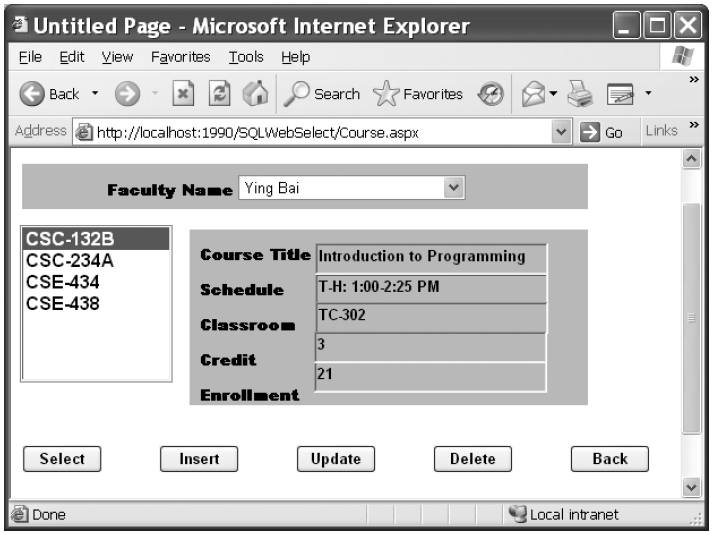

Figure 8.32 Detailed course information.

8.4 DEVELOP ASP.NET WEB APPLICATION TO INSERT DATA INTO SQL SERVER DATABASES

In this section we discuss how to insert a new faculty record into the SQL Server database from the Web page. To do that, we need to create a new Web page called Insert page and add it into our new project. To save time and space, we can modify an existing project,

Table 8.6 Controls for the Insert Form

Type	ID	Text	TabIndex	BackColor	Font
Panel	Panel1		19	#C0C0FF	
Label	Label1	Faculty Photo	0		x-small
TextBox	txtPhoto		1		
Panel	Panel2		20	#C0C0FF	
Label	Label2	Faculty ID	2		x-small
TextBox	txtID		3		
Panel	Panel3		21	#C0C0FF	
Image	PhotoBox		4		
Label	Label3	Name	5		Small
TextBox	txtName		6		
Label	Label4	Title	7		Small
TextBox	txtTitle		8		
Label	Label5	Office	9		Small
TextBox	txtOffice		10		
Label	Label6	Phone	11		Small
TextBox	txtPhone		12		
Label	Label7	College	13		Small
TextBox	txtCollege		14		
Label	Label8	Email	15		Small
TextBox	txtEmail		16		
Button	cmdInsert	Insert	17		Bold/Small
Button	cmdBack	Back	18		Bold/Small

SQLWebSelect developed in the previous section, and make it our new Web application project named SQLWebInsert.

Open Windows Explorer and create a new folder Chapter 8 if you have not done that. Then copy the project SQLWebSelect from the folder DBProjects\Chapter8 located at the accompanying ftp site (see Chapter 1) and paste it to our new folder C:\Chapter 8. Rename this project to SQLWebInsert.

As we mentioned, to add a new record into the database, we need a user interface to do that job. So we need to create and add a new Web page Insert.aspx into our new project to perform the inserting data function.

8.4.1 Create New Web Page Insert.aspx

Right-click on our new project icon from the Solution Explorer window and select Add New Item from the pop-up menu to open the Add New Item dialog box. Keep the default template Web Form selected and then enter `Insert.aspx` into the Name box as the name for this new page. Click on the Add button to add this new page into our new project. Add the controls shown in Table 8.6 into this Web page.

The key points to add these controls to the page are: For three panels you need to set the Height and Width properties to the following dimensions:

- Panel1: Height 36 px Width 230 px
- Panel2: Height 36 px Width 230 px
- Panel3: Height 200 px Width 474 px

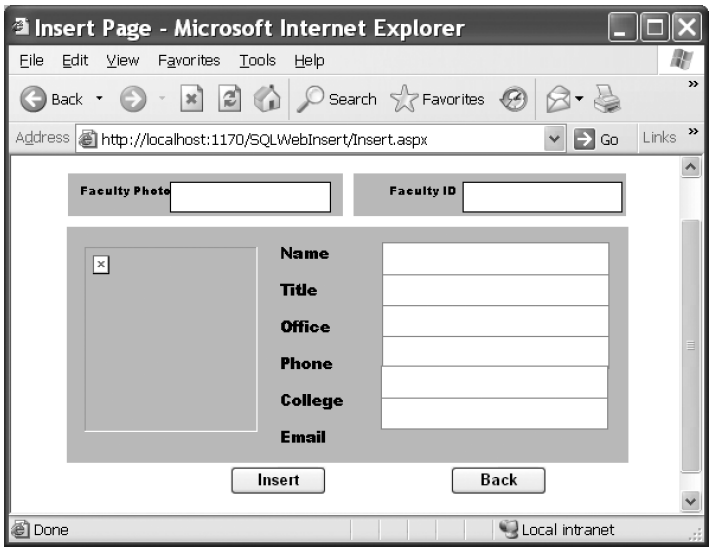

Figure 8.33 Insert Web page.

Also set the Format|Position properties for all Panel, Image, Label, and TextBox controls to Absolute position. You can move, copy, and paste those Label and TextBox controls one by one on the page form to save time. Your finished Insert page should match the one shown in Figure 8.33.

8.4.2 Develop Codes to Perform Data Insertion Function

The function of this Insert page are:

1. While the project is running, you need to open the Insert page by clicking on the Insert button from the Faculty page.
2. To insert a new faculty record into the database, you need to enter seven pieces of new information into seven textboxes on the insert page. The information includes the faculty_id, faculty name, title, office, phone, college, and email.
3. The Faculty Photo textbox is optional, which means that you can either enter a new faculty photo name with this new record or leave it blank. If you leave it blank, a default photo will be displayed when this new record is validated later.
4. After all information has been entered into all textboxes, you can click on the Insert button to insert this new record into the Faculty table in our sample database via the Web page.
5. The Back button is used to return to the Faculty page to perform the data validation to confirm this data insertion.

Now let's start creating the codes for this Insert page. The coding can be divided into three parts: the coding for the Page_Load() and Back button's click method, the coding for the Insert button's click method, and the coding for other user-defined methods.

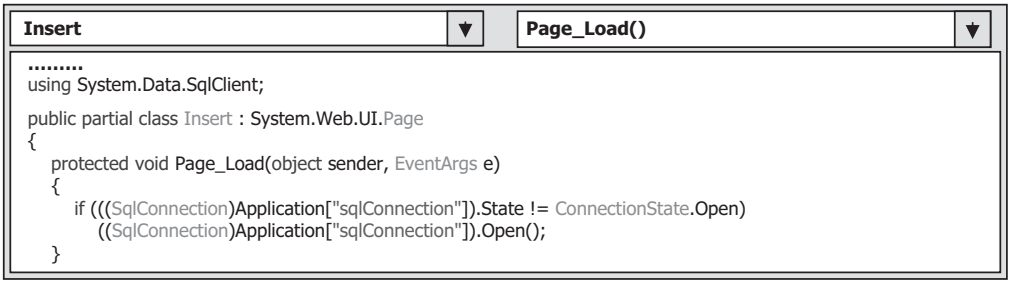

Figure 8.34 Coding for the Page_Load method.

8.4.3 Develop Codes for Page_Load and Back Button Click Methods

Open the code page window of the Insert Web form, and enter the following SQL Server Data Provider–related namespace at the top of this page:

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

Open the Page_Load() method and enter the codes shown in Figure 8.34 into this method.

The function of this piece of coding is:

A. The SQL Server Data Provider–related namespace is added into the namespace area in this page since we need to use some data components stored in that namespace to perform the data insertion.

B. The global connection object sqlConnection stored in the Application state will be checked to make sure that a valid database connection exists before this data insertion. Redo this connection if no valid connection existed.

The coding for the Back button's Click method is simple. As this button is clicked, the current page will be returned to the Faculty page to perform the data validation. Open this method by double-clicking on the Back button from the Insert Web form, and enter the following code into this method:

```
Response.Redirect("Faculty.aspx");
```

The Redirect() method of the server's Response object is used to redirect the current page to the Faculty page.

8.4.4 Develop Codes for Insert Button Click Method

Open the Insert button's Click method by double-clicking on the Insert button from the Insert Web form, and enter the codes shown in Figure 8.35 into this method.

Let's take a closer look at this piece of code to see how it works.

A. The Insert query string is declared first, and it contains seven pieces of information related to seven columns in the Faculty table in the database.

Insert | cmdInsert_Click()

```
protected void cmdInsert_Click(object sender, EventArgs e)
{
A       string cmdString = "INSERT INTO Faculty (faculty_id, faculty_name, office, phone, college, title, email) " +
                           "VALUES (@faculty_id,@faculty_name,@office,@phone,@college,@title,@email)";
B       SqlCommand sqlCommand = new SqlCommand();
        string FacultyImage;
        int intInsert = 0;
C       FacultyImage = txtPhoto.Text;                          //reserve the new inserted faculty photo location
        if (FacultyImage == string.Empty)
           FacultyImage = "Default.jpg";
D       Application["FacultyImage"] = FacultyImage;            //reserve faculty image for validation
        PhotoBox.ImageUrl = FacultyImage;
E       Application["FacultyName"] = txtName.Text;             //reserve faculty name for validation
F       sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
        sqlCommand.CommandType = CommandType.Text;
        sqlCommand.CommandText = cmdString;
G       InsertParameters(sqlCommand);
H       intInsert = sqlCommand.ExecuteNonQuery();
I       sqlCommand.Dispose();
J       if (intInsert == 0)
           Response.Write("<script>alert('The data insertion is failed')</script>");
K       cmdInsert.Enabled = false;                             //disable the Insert button
}
```

Figure 8.35 Coding for the Insert button's method.

B. The data components and local variables used in the method are declared here. The FacultyImage string variable is used to hold the faculty photo name, and the local integer variable intInsert is used to hold the returned data from the execution of the data insertion command.

C. The faculty photo name is reserved to the FacultyImage string variable. If this variable contains an empty string, which means that the user did not enter any photo name to the txtPhoto box, a default faculty image will be assigned and displayed as the project runs. Otherwise, the selected faculty photo will be displayed based on the photo name entered by the user.

D. The name of the faculty photo file, which is entered by the user, is reserved to the FacultyImage string variable. Also this faculty photo file is stored into the Application state as a global variable since we need to use it later as we perform data validation in the Faculty page. The reason we use the Application state to store this global variable is that each time when the client sends a request to the server, the server will post back a refreshed page to the client after the server received that request. The values of all global variables created in the client side will be lost after this refreshed page is sent back. Therefore, in order to keep those values, we must use this Application state to reserve them.

E. The faculty name entered by the user is reserved to the Application state as a global variable since we need to use it later as we perform data validation in the Faculty page. The reason we use the Application state to store this global variable is same as that we discussed in step D for the FacultyImage.

F. The Command object is initialized by assigning it with connection object stored in the Application state, the command type, and the command text objects, respectively.

G. The user-defined subroutine InsertParameters() is executed to assign all seven input parameters to the Parameters collection of the command object.

Insert | InsertParameters()

```
private void InsertParameters(SqlCommand cmd)
{
      cmd.Parameters.Add("@faculty_id", SqlDbType.Char).Value = txtID.Text;
      cmd.Parameters.Add("@faculty_name", SqlDbType.Char).Value = txtName.Text;
      cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text;
      cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text;
      cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text;
      cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text;
      cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text;
}
```

Figure 8.36 Coding for the subroutine InsertParameters.

H. The ExecuteNonQuery() method of the command object is called to run the insert query to perform this data insertion.

I. A cleaning job is performed to release all objects used in the method.

J. The ExecuteNonQuery() method will return a data value to indicate whether this data insertion is successful or not. The value of this returned data equals to the number of rows that have been successfully inserted into the Faculty table in the database. If a zero is returned, which means that no row has been inserted into the database, a warning message is displayed to indicate this situation. Otherwise, the data insertion is successful.

K. The Insert button is disabled after the current record is inserted into the database. This is to avoid the multiple insertions of the same record into the database. The Insert button will be enabled again when the content of the Faculty ID textbox is changed, which means that a new different faculty record will be inserted.

The detailed coding for the user-defined InsertParameters() method is shown in Figure 8.36.

This coding is straightforward and easy to understand. Each piece of new faculty information is assigned to the associated input parameter by using the Add() method of the Parameters collection of the Command object.

8.4.5 Develop Codes for Other Methods

The coding for other methods includes the coding for the Insert button's Click method in the Faculty page. The function of this piece of code is to direct the project from the Faculty page to the Insert page as the user clicks on this Insert button from the Faculty page.

Open the Insert button's Click method in the Faculty page by double-clicking on the Insert button from the Faculty Web form window, and enter the following code into this method to direct the project to the Insert page:

```
Response.Redirect("Insert.aspx");
```

Now we can run the project to test the data insertion function via the website. However, before we can start the project, make sure that a default faculty photo file named Default.jpg and a faculty photo file named Mhamed.jpg have been stored in the default folder in which our project is located since we need to use those photo files to

test our project. Also make sure that the start page is LogIn page by setting it up as the start page using the Start Options menu item.

Click on the Start Debugging button to run the project. Enter the suitable username and password such as `jhenry` and `test` to the LogIn page, and select the Faculty Information item from the Selection page to open the Faculty page. Click on the Insert button to open the Insert page and enter the following data as the information for inserting a new faculty member:

- Mhamed.jpg — Faculty Photo textbox
- M56789 — Faculty ID textbox
- Ali Mhamed — Faculty Name textbox
- Professor — Title textbox
- MTC-353 — Office textbox
- 750-378-3355 — Phone textbox
- University of Main — College textbox
- amhamed@college.edu — Email textbox

Your finished new faculty information page should match the one shown in Figure 8.37.

Click on the Insert button to insert this new record into the database. The Insert button is immediately disabled and the associated faculty image is displayed in the PhotoBox, which is shown in Figure 8.37.

Click on the Back button to return to the Faculty page, and next we need to perform data validation to confirm our data insertion is successful.

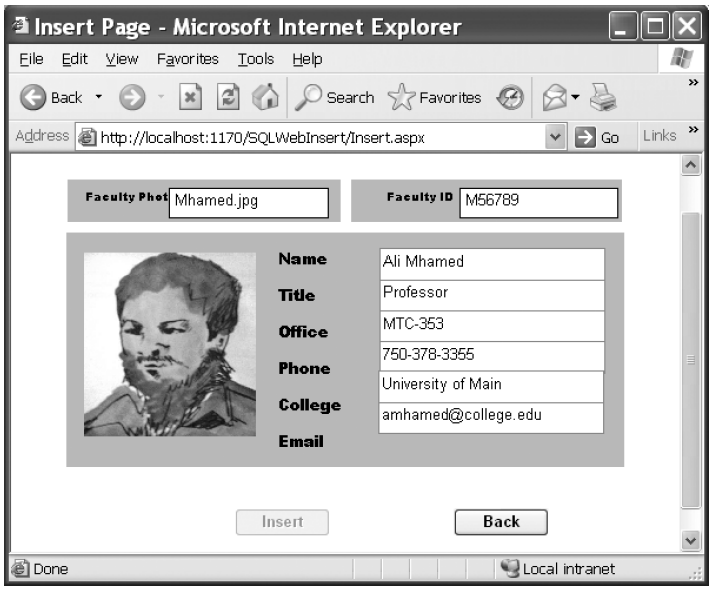

Figure 8.37 Running status of the Insert page.

8.4.6 Validate Data Insertion

To validate the new inserted data from the Faculty page, we need to do some modifications to the coding in the Faculty page and add some codes to this page to allow us to retrieve the new inserted record from the database and display it on this page. The following methods need to be modified:

1. Page_Load() method
2. ShowFaculty() method

Let's first take care of the first method, Page_Load() method. After a new faculty record is inserted into the database from the Insert page and returned to the Faculty page, the new inserted faculty name should be added into the combobox control ComboName to allow the user to select it to retrieve the new inserted information for the selected faculty. To do this, we need to add the codes shown in Figure 8.38 into the Page_Load() method of the Faculty page.

The codes we developed in the previous section have been highlighted with shading. The function of this piece of new added code is: Each time, when the server posts a refreshed Faculty page to the client, we need to inspect whether a new faculty record has been inserted into the database by checking the global variable FacultyName, which is stored in the Application state. If this global variable is empty, which means that no data insertion occurred, we need to do nothing. However, if this variable contains a valid faculty name, which means that a data insertion has occurred, we need to add this new inserted faculty name into the combobox control ComboName to allow users to select this new faculty from the control to perform the associated data actions against the database. After this new faculty name is added into the combobox control, we need to reset this global variable to avoid the multiple additions of the same faculty name into the ComboName control.

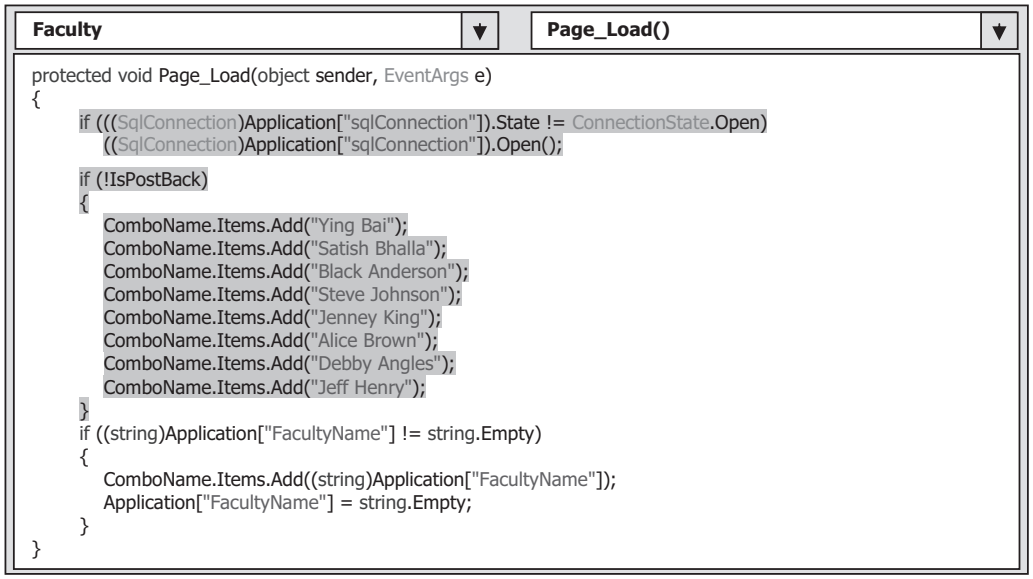

Figure 8.38 Modified coding for the Page_Load method.

During the data insertion process, the user may want to insert a faculty photo by entering the name of the faculty photo file into the Faculty Photo textbox. However, another possibility is that the user may not want to insert any faculty photo with that data insertion. In that case, the content of the Faculty Photo textbox should be empty. Recall that when we developed the coding for the Insert button's Click method in the Insert page (refer to Section 8.4.4), we used a global variable FacultyImage that is stored in the Application state to store the name of the new inserted faculty photo file. Now when we validate that data insertion, we need to confirm whether the user inserted a faculty photo or not by checking that global variable FacultyImage.

Open the user-defined ShowFaculty() method and add the codes shown in Figure 8.39 into this method. The modified codes have been highlighted in bold. The function

Faculty ▼ **ShowFaculty()** ▼

```
private string ShowFaculty(string fName)
{
      string FacultyImage;
      switch (fName)
      {
         case "Black Anderson":
            FacultyImage = "Anderson.jpg";
            break;
         case "Ying Bai":
            FacultyImage = "Bai.jpg";
            break;
         case "Satish Bhalla":
            FacultyImage = "Satish.jpg";
            break;
         case "Steve Johnson":
            FacultyImage = "Johnson.jpg";
            break;
         case "Jenney King":
            FacultyImage = "King.jpg";
            break;
         case "Alice Brown":
            FacultyImage = "Brown.jpg";
            break;
         case "Debby Angles":
            FacultyImage = "Angles.jpg";
            break;
         case "Jeff Henry":
            FacultyImage = "Henry.jpg";
            break;
         default:
            FacultyImage = "No Match";
            break;
      }
      if (FacultyImage != "No Match")
        PhotoBox.ImageUrl = FacultyImage;
      else
        if ((string)Application["FacultyImage"] == string.Empty)
          FacultyImage = "Default.jpg";
        else
          FacultyImage = (string)Application["FacultyImage"];
        PhotoBox.ImageUrl = FacultyImage;
      return FacultyImage;
}
```

Figure 8.39 Modifications to the coding of ShowFaculty method.

of these new added codes is that a default faculty photo file Default.jpg will be assigned to the FacultyImage variable if the global variable FacultyImage is empty, which means that the user does not want to add a new faculty photo with that data insertion and the Faculty Photo textbox in the Insert page is blank. Otherwise the faculty photo file stored in the Application state will be assigned to the FacultyImage variable that will be displayed later in the PhotoBox image control in the Faculty page.

Compared with the original coding we did for this part, it can be found that the warning message box, which will be displayed if no matched faculty image can be found from this ShowFaculty method, has been removed. Right now the default faculty image will be displayed. Either the global variable FacultyImage is empty or no matched faculty image can be found. The reason we made this modification to this part is that we want our project more professional and neat. Of course, if you like to keep that warning message box to indicate the situation in which no matched faculty image can be found, you can add another global variable to monitor whether the Insert page has been executed or not. If the Insert page has not been executed and no matched faculty image can be found, the warning message should be displayed. Otherwise, the default faculty image should be displayed.

Now we have completed all modifications to the coding on our Faculty page, and we can run the project to test our data insertion function via the website. Recall that we have already inserted a new faculty record with the faculty name Ali Mhamed into the Faculty table in the last section. In order to validate this insertion, we need to run the project and insert this new record again. To avoid the duplicated insertion, we need first to open our sample database to delete that new inserted record from the Faculty table. Open the SQL Server 2005|SQL Server Management Studio Express, and then open our sample database `CSE_DEPT.mdf` that is located at the folder C:\database\SQLServer and the Faculty table, and delete that new inserted record.

Click on the Start Debugging button to run our project. Enter the suitable username and password to the LogIn page, and select the Faculty Information item from the Selection page to open the Faculty page. Click on the Insert button to open the Insert page and enter the following data as the information for a new faculty member:

- Mhamed.jpg — Faculty Photo textbox
- M56789 — Faculty ID textbox
- Ali Mhamed — Faculty Name textbox
- Professor — Title textbox
- MTC-353 — Office textbox
- 750-378-3355 — Phone textbox
- University of Main — College textbox
- amhamed@college.edu — Email textbox

Click on the Insert button to insert this new record into the database. Then click on the Back button to return to the Faculty page to perform the data validation.

Go to the ComboName combobox control, and you can find that the new inserted faculty name `Ali Mhamed` is already in there. Click on it to select this faculty and then click on the Select button to retrieve this new inserted record from the database and display it in this page. The inserted record is displayed in this page, which is shown in Figure 8.40.

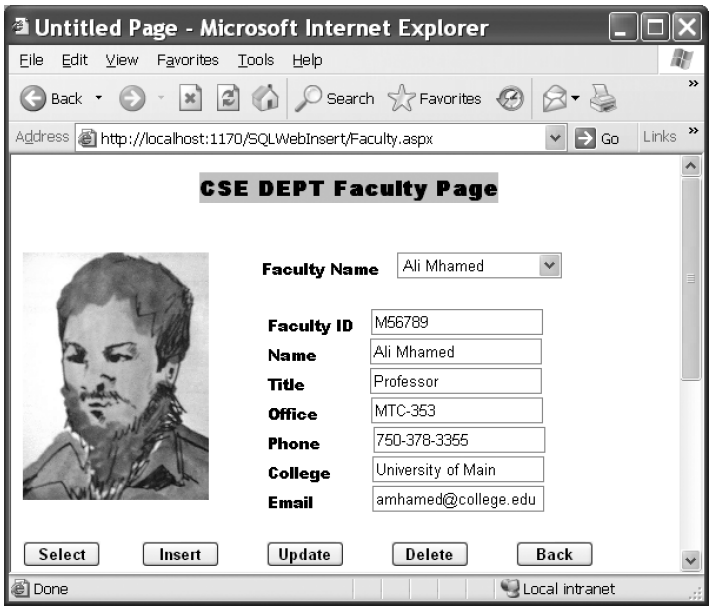

Figure 8.40 Data validation process.

Our data insertion is successful. Click on the Back button and then on the Exit button to close our project. A complete Web application project SQLWebInsert can be found at the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

In the next section, we will discuss how to perform data updating and deleting actions against our SQL Server database via the website.

8.5 DEVELOP WEB APPLICATIONS TO UPDATE AND DELETE DATA IN SQL SERVER DATABASES

To update or delete data against the relational databases is a challenging topic. We have provided a very detailed discussion and analysis of this topic in Section 7.1.1 in Chapter 7. Refer to that section to get more detailed information for these data actions. Here we want to emphasize some important points related to data updating and deleting.

1. When updating or deleting data against related tables in a database, it is important to update or delete data in the proper sequence in order to reduce the chance of violating referential integrity constraints. The order of command execution will also follow the indices of the DataRowCollection in the database. To prevent data integrity errors from being raised, the best practice is to update or delete data against the database in the following sequence:
   - **a.** Child table: delete records.
   - **b.** Parent table: insert, update, and delete records.
   - **c.** Child table: insert and update records.
2. To update existing data in the database, generally it is unnecessary to update the primary key for that record. It is much better to insert a new record with a new primary key into

the database than updating the primary key for an existing record because of the complicated table operations listed above. In practice, it is very rare to update a primary key for an existing record in the database in most real applications. Therefore, in this section, we concentrate our discussion on updating the existing records by modifying all data columns except the primary key column.

3. To delete a record from a relational database, the normal operation sequence listed above must be followed. For example, to delete a record from the Faculty table in our application, one must first delete those records that are related to the data to be deleted in the Faculty table from the child table such as the LogIn and Course tables, and then one can delete the record from the Faculty table. The reason for this deleting sequence is because the `faculty_id` is a foreign key in the LogIn and the Course tables, but it is a primary key in the Faculty table. One must first delete data with the foreign keys and then one can delete the data with the primary key from the database.

Keep these three points in mind. Now let's begin our project. We can modify one of our existing projects, SQLWebInsert, and make it our new project SQLWebUpdateDelete. To do that, open Windows Explorer and create a new folder Chapter 8 if you have not done that. Then copy the project SQLWebInsert from the folder DBProjects\Chapter 8 from the accompanying ftp site (see Chapter 1) and paste it to our new folder C:\Chapter 8. Rename this project SQLWebUpdateDelete.

8.5.1 Application User Interfaces

To update or delete an existing faculty record in our sample database, we don't need any new Web page as our user interface, and we can use the Faculty page as our user interface to perform these data actions. To meet our data actions' requirements, we need to perform some modifications to the Faculty page.

The first modification to the Faculty Web form is to clean up the Faculty ID textbox during the data updating process because we don't want users to modify this piece of information based on our discussion in step 2 in the last section.

8.5.2 Modify Coding for Faculty Page

Besides the coding development for the Update button's Click method we will discuss in the next section, the second modification to this page is to add one more statement into the Select button's Click method, which is shown in step **A** in Figure 8.41.

The new added statement has been highlighted in bold in Figure 8.41. The purpose of this statement is to store the current selected faculty name located at the combobox control ComboName into the Application state function as a global variable. During the data updating process, the faculty name may be updated by the user. If this happens, the updated faculty name stored in the txtName textbox will be added into the combobox control ComboName, and the original faculty name should be removed from that control. In order to remember the original faculty name, we must use this global variable to keep it since this is a Web application, and each time the server posts back a refreshed Faculty page based on the client's request, all contents in all controls on that page will be refreshed and all old data will be lost.

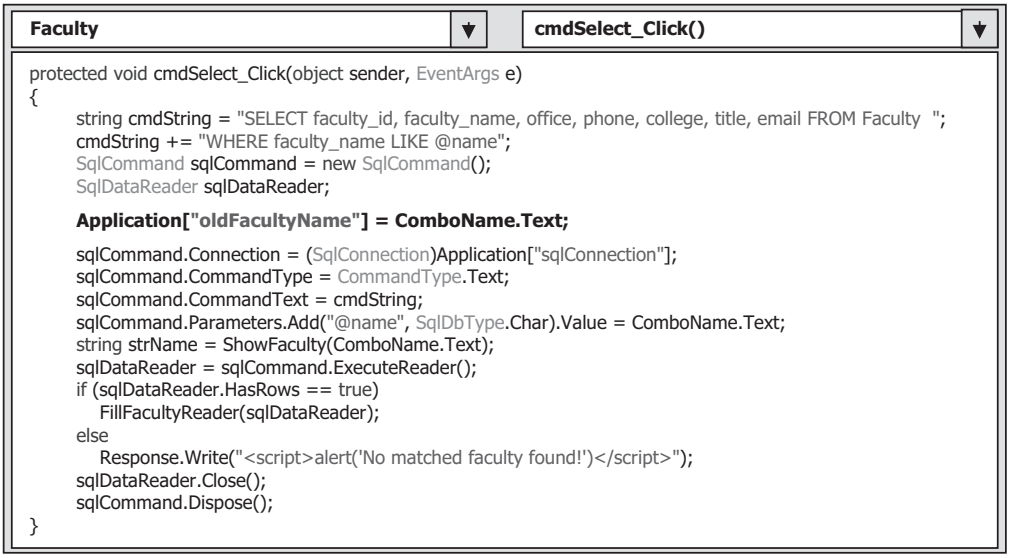

Figure 8.41 Modified Select button's Click method.

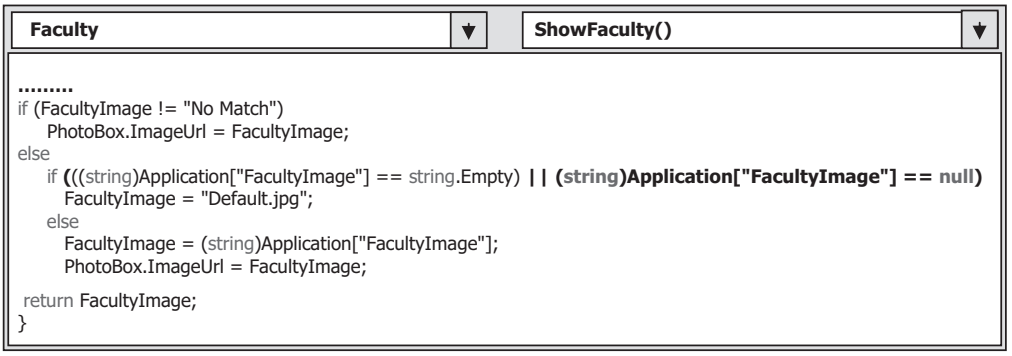

Figure 8.42 Modified codes of the ShowFaculty method.

Another modification to this Faculty page is to add one more statement to the `if` condition in the ShowFaculty() method, which is shown in step **A** in Figure 8.42, to display a default faculty image if no data insertion action is performed. The new added instruction has been highlighted in bold.

The purpose of adding this `or` condition is to display a default image if no data insertion action is performed since this ShowFaculty() method will be used by different data actions, including data selection, insertion, and updating, as the project runs. Without this `or` condition, no faculty image will be displayed if no data insertion occurred, even when data updating is performed with a default faculty image selected by the user. Now let's develop the codes for the Update button's Click method.

Faculty | cmdUpdate_Click()

```
protected void cmdUpdate_Click(object sender, EventArgs e)
{
A       string cmdString = "UPDATE Faculty SET faculty_name = @name, office = @office, phone = @phone,  " +
                           "college = @college, title = @title, email = @email  " +
                           "WHERE (faculty_name LIKE @oldName)";
B       SqlCommand sqlCommand = new SqlCommand();
        int intUpdate = 0;
C       txtID.Text = string.Empty;                                          //clean up the faculty_id textbox
D       if (txtName.Text != (string)Application["oldFacultyName"])          //the faculty name is updated
        {
           ComboName.Items.Add(txtName.Text);
           ComboName.Items.Remove((string)Application["oldFacultyName"]);
        }
E       sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
        sqlCommand.CommandType = CommandType.Text;
        sqlCommand.CommandText = cmdString;
F       UpdateParameters(ref sqlCommand);
G       intUpdate = sqlCommand.ExecuteNonQuery();
H       sqlCommand.Dispose();
I       if (intUpdate == 0)
           Response.Write("<script>alert('The data updating is failed')</script>");
}
```

Figure 8.43 Coding for the Update button's Click method.

8.5.3 Develop Codes for Update Button Click Method

Open this method by double-clicking on the Update button from the Faculty Web form window and enter the codes shown in Figure 8.43 into this method.

Let's take a closer look at this piece of code to see how it works.

A. An updating query string is declared first with the `oldName` as the name of the dynamic parameter. This is because when you want to update the faculty name, the original name stored in the combobox control ComboName becomes the old name, and we need to distinguish this old name from the updated name.

B. The data component, Command object, used in this method is created here. A local integer variable intUpdate is also created, and it is used as a value holder to keep the returned data from executing the ExecutNonQuery() method later.

C. Before we can perform data updating, first we need to clean up the Faculty ID textbox since we don't want to update this piece of information.

D. Now we need to check whether the user wants to update the faculty name or not by comparing the global variable oldFacultyName that is stored in the Application state function during the data selection process in the Select button's click method with the current faculty name stored in the textbox control txtName. If both are different, that means the user has updated the faculty name. In that case, we need to add the updated faculty name into the combobox control ComboName and remove the old faculty name from that control to allow users to select this updated faculty from the combobox list to perform the data actions in the database in the future.

E. The Command object is initialized with the Connection object, Command type and Command text.

Faculty ▼ UpdateParameters() ▼

```
private void UpdateParameters(ref SqlCommand cmd)
{
      cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text;
      cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text;
      cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text;
      cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text;
      cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text;
      cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text;
      cmd.Parameters.Add("@oldName", SqlDbType.Char).Value = ComboName.Text;
}
```

Figure 8.44 Coding for the UpdateParameters method.

F. The user-defined UpdateParameters() method, whose detailed coding is shown in Figure 8.44, is called to assign all input parameters to the command object.

G. The ExecuteNonQuery() method of the Command class is called to execute the data updating operation. This method returns a feedback value to indicate whether this data updating is successful or not, and this returned value is stored into the local integer variable intUpdate.

H. A cleaning job is performed to release all data objects used in this method.

I. The data value returned from calling the ExecuteNonQuery() is exactly equal to the number of rows that have been successfully updated in the database. If this value is zero, which means that no row has been updated and this data updating has failed, a warning message is displayed. Otherwise if this value is nonzero, this data updating is successful.

The detailed coding for the method UpdateParameters() is shown in Figure 8.44.

Seven input parameters are assigned to the Parameters collection property of the command object using the Add() method. One important point for this parameter assignment is the last input parameter or the dynamic parameter oldName. The original or the old faculty name oldFacultyName stored in the Application state function must be used as the value for this parameter. Some readers may disagree with me: The original or the old faculty name is located at the combobox control ComboName, and we can directly get it from that control without using this global variable. Well, this statement is correct for the Windows-based application without any problem. However, for the Web-based application, it is absolutely wrong. Recall that when the users clicked on the Update button to perform a data updating action, this updating request will be sent to the server, and the server will post back a refreshed Faculty page to the client. All old or the original data stored in all textboxes or comboboxes in the previous page will be gone. In other words, the contents stored in all textboxes and comboboxes in this refreshed page are different with the contents stored in the previous pages. A wrong updating may occur if you still use the faculty name stored in the combobox control ComboName in the current or refreshed page.

At this point we have finished all coding jobs for the data updating actions against the SQL Server database in the Faculty page. Before we can run the project to test this data updating function, we must make sure that the starting page is the LogIn page, and a default faculty image file Default.jpg has been stored in our default folder. To check the starting page, right-click on our project icon from the Solution Explorer window, select the Start Options item from the pop-up menu, and then check the Specific page radio button and select the `LogIn.aspx` as the starting page.

Now let's run the project to test the data updating actions. Click on the Start Debugging button to run the project, enter the suitable username and password to the LogIn page, and select the Faculty Information item from the Selection page to open the Faculty page. Select the faculty name `Ying Bai` from the combobox control ComboName and click on the Select button to retrieve the information for this selected faculty from the database and display it on this page.

Now let's test the data updating actions in two steps: First, we update the faculty information without touching the faculty name, and second we update the faculty information with changing the faculty name.

Let's start from the first step now. Enter the following information into the associated textboxes to update this faculty record:

- Professor Title textbox
- MTC-353 Office textbox
- 750-378-3300 Phone textbox

Click on the Update button to perform this data updating. To confirm this data updating, first select another faculty from the combobox control ComboName and click on the Select button to retrieve and display that faculty information. Then select the faculty `Ying Bai` whose information has just been updated from the combobox control and click on the Select button to retrieve and display it. You can see that the selected faculty information has been updated, which is shown in Figure 8.45.

Next let's perform data updating with the second method: Include updating of the faculty name. Still keep the current page unchanged, and then modify the faculty information from the associated textboxes by entering the following data:

Figure 8.45 Data updating process.

- Peter Bai — Faculty Name textbox
- Associate Professor — Title textbox
- MTC-555 — Office textbox
- 750-378-3355 — Phone textbox
- pbai@college.edu — Email textbox

Click on the Update button to update this faculty information. Immediately you will find that the original faculty name `Ying Bai` has disappeared from the combobox control ComboName. To confirm this data updating, in a similar way, let's first select another faculty from the combobox control ComboName and click on the Select button to retrieve and display that faculty information. Then select the faculty `Peter Bai` from the combobox control and click on the Select button to retrieve and display it. You can see that the selected faculty information including the faculty name has been updated, which is shown in Figure 8.46.

One point to note is the faculty photo. When you updated the faculty name, you did not place an updated faculty photo file in our default folder. Therefore, a default faculty photo is displayed for this situation. You can change this situation by placing an updated faculty photo file in our default folder before the project runs if you like the correct faculty photo to be displayed with this data updating. Our data updating action is very successful.

Next let's take care of the data deleting action in the SQL Server database. Similarly to data updating, for data deleting we don't need any new Web page as our user interface, and we can still use the Faculty page to perform data deleting actions.

Figure 8.46 Data updating process—including the faculty name updating.

8.5.4 Develop Codes for Delete Button Click Method

Since deleting a record from a relational database is a complex issue, we divide this discussion into five sections:

1. Relationships between five tables in our sample database
2. Data deleting sequence
3. Using the Cascade deleting option to simplify the data deleting
4. Creating the stored procedure to perform the data deleting
5. Calling the stored procedure to perform data deleting

Let's start with the first section.

8.5.4.1 Relationships Between Five Tables in Our Sample Database

As we discussed at the beginning of this section, to delete a record from a relational database, one must follow the correct sequence. In other words, one must first delete the records related to the record to be deleted in the parent table from the child tables. In our sample database, five tables are related together by using the primary and foreign keys. In order to make these relationships clear, we redraw Figure 2.5 in Chapter 2, which is Figure 8.47 in this section, to illustrate this issue.

If you want to delete a record from the Faculty table, you must first delete the related records from the LogIn, Course, StudentCourse, and Student tables, and then you can delete the desired record from the Faculty table. The reason for that is because the relationships existed between five tables.

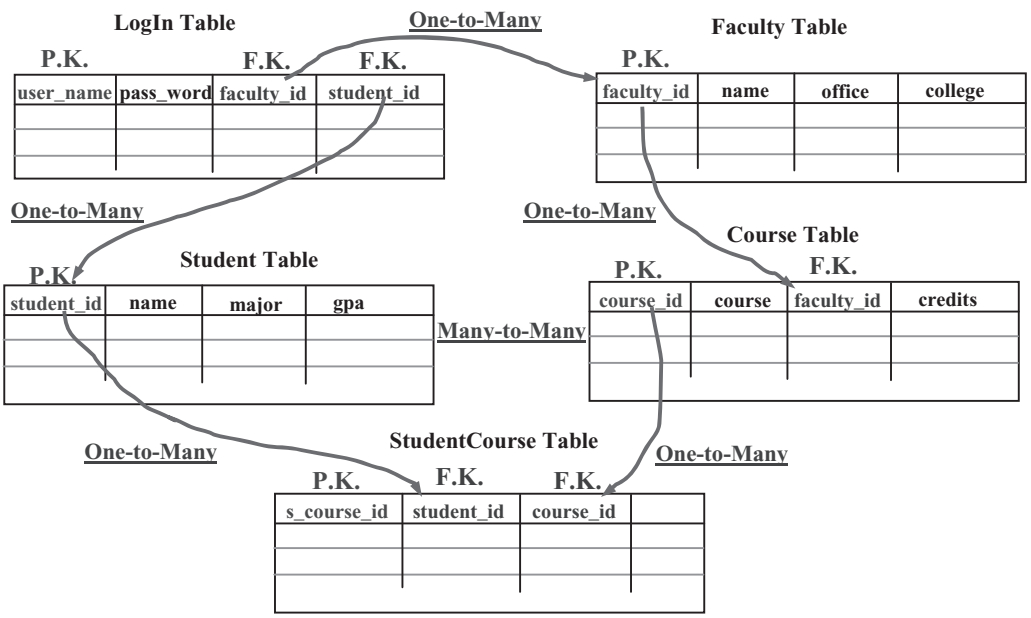

Figure 8.47 Relationships between five tables.

For example, if one wants to delete a faculty record from the Faculty table, one must perform the following deleting operations:

- The `faculty_id` is a primary key in the Faculty table, but it is a foreign key in the LogIn and the Course table. Therefore, the Faculty table is a parent table and the LogIn and the Course are child tables. Before one can delete any record from the Faculty table, one must first delete records that have the `faculty_id` as the foreign key from the child tables. In other words, one must first delete those records that use the `faculty_id` as a foreign key from the LogIn and the Course tables.
- When deleting records that use the `faculty_id` as a foreign key from the Course table, the related `course_id` that is a primary key in the Course table will also be deleted. The Course table right now is a parent table since the `course_id` is a primary key for this table. But as we mentioned, to delete any record from a parent table, one must first delete the related records from the child tables. Now the StudentCourse table is a child table for the Course table; therefore, the records that use the `course_id` as a foreign key in the StudentCourse table should be deleted first.
- After those related records in the child tables are deleted, finally the faculty member can be deleted from the parent table, Faculty table.

8.5.4.2 Data Deleting Order Sequence

Summarily, to delete a record from the Faculty table, one needs to perform the following deleting operations in the **order sequence** shown below:

1. Delete all records that use the `course_id` as the foreign key from the StudentCourse table.
2. Delete all records that use the `faculty_id` as the foreign key from the LogIn table.
3. Delete all records that use the `faculty_id` as the foreign key from the Course table.
4. Delete the desired faculty member from the Faculty table.

You can see how complicated the operations are to delete one record from the relational database from this example.

8.5.4.3 Use Cascade Deleting Option to Simplify Data Deleting

To simplify the data deleting operations, we can use the cascade deleting option provided by the SQL Server 2005 Database Management Studio.

Recall that when we created and built the relationship between our five tables, the following five **relationships** were built between tables:

1. A relationship between the LogIn and the Faculty tables was set up using the **faculty_id** as a foreign key FK_LogIn_Faculty in the LogIn table.
2. A relationship between the LogIn and the Student tables was set up using the **student_id** as a foreign key FK_LogIn_Student in the LogIn table.
3. A relationship between the Course and the Faculty tables was set up using the **faculty_id** as a foreign key FK_Course_Faculty in the Course table.
4. A relationship between the StudentCourse and the Course table was set up using the **course_id** as a foreign key FK_StudentCourse_Course in the StudentCourse table.

5. A relationship between the StudentCourse and the Student table was set up using the **student_id** as a foreign key FK_StudentCourse_Student in the StudentCourse table.

Refer to the data deleting sequence listed in the last section to delete a record from the Faculty table. One needs to perform four deleting operations in that sequence. Compared with those four deleting operations, the first one is the most difficult and the reason for that is: To perform the first data deleting operation, one must first find all course_ids that use the faculty_id as the foreign key from the Course table, and then based on those course_ids, one needs to delete all records that use those course_ids as the foreign keys from the StudentCourse table. For deleting operations in sequences 3 and 4, they are very easy, and each deleting operation only needs one deleting query. The conclusion for this discussion is: How do we find an easy way to complete the deleting operation in sequence 1?

A good solution to this question is to use the Cascade option, as we did in Chapter 2, to perform data deleting and updating in a cascaded mode. This Cascade option allows the SQL Server 2005 database engine to perform that deleting operation in sequence 1 as long as a Cascade option is selected for relationships 4 and 5 listed above.

Now let's use a real example to illustrate how to use this Cascade option to simplify the data deleting operations, especially for the first data deleting in that sequence. Open the SQL Server Management Studio Express by going to Start|All Programs |Microsoft SQL Server 2005|SQL Server Management Studio Express. On the opened Studio Express window, click on the Database and expand our sample database, C:\database\SQLServer\CSE_DEPT.mdf, to display all five tables. Since we are only interested in relationships 4 and 5, expand the dbo.StudentCourse table and expand the Keys folder to display all the keys we set up in Section 2.10.4. Double-click on the key FK_StudentCourse_Course to open it, which is shown in Figure 8.48.

On the opened dialog box, keep our desired foreign key FK_StudentCourse_Course selected from the left pane, and then click on the small plus icon before the item INSERT And UPDATE Specification, and you can find that a **Cascade** mode has been set for both Delete Rule and Update Rule items, which is shown in Figure 8.48.

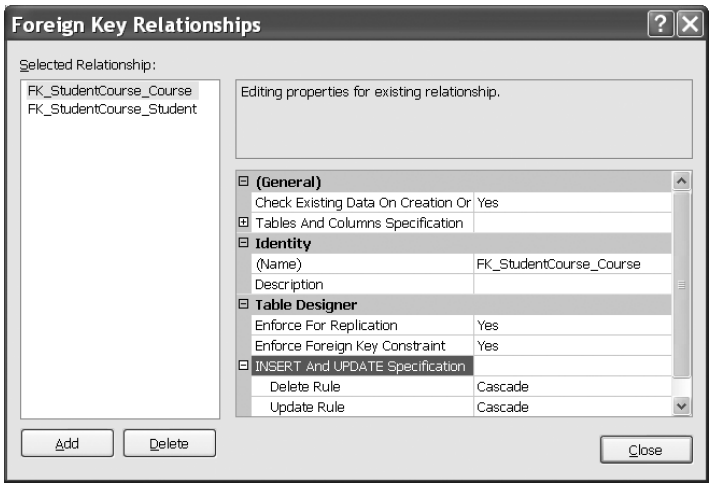

Figure 8.48 Foreign Key Relationship dialog box.

After this Cascade option is set up, each time you want to delete all records that use the `course_id` or the `student_id` as the foreign keys in the StudentCourse table, the SQL Server engine will perform those data deleting operations automatically for you in that cascaded sequence. Therefore, you can see how easy it is to perform data deleting in sequence 1.

Now let's create our codes for the Delete button's Click method to perform this data deleting operation.

8.5.4.4 Develop Codes to Perform Data Deleting

On the opened Visual Studio.NET, go to File|Open Web Site menu item to open our Web application project SQLWebUpdateDelete. Then open the Delete button's Click method from the Faculty Web form window by double-clicking on the Delete button. Enter the codes shown in Figure 8.49 into this method.

Let's take a closer look at this piece of codes to see how it works.

A. The data deleting query string is declared first with the `faculty_name` as the query criterion.

B. The data object and local variable used in this method are declared here. The integer variable intDelete is used to hold the returned value from calling the ExecuteNonQuery() method of the Command class later.

C. The Command object is initialized by using the Connection object, Command Type, Command Text, and Parameters properties.

D. After the Command object is initialized, the ExecuteNonQuery() method of the Command class is called to perform the data deleting action. This method will return a data value, and it is assigned to the local variable intDelete.

E. A cleaning operation is performed to release all objects used in this method.

Faculty | cmdDelete_Click()

```
protected void cmdDelete_Click(object sender, EventArgs e)
{
A    string cmdString = "DELETE FROM Faculty WHERE (faculty_name LIKE @FacultyName)";
B    SqlCommand sqlCommand = new SqlCommand();
     int intDelete = 0;
C    sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
     sqlCommand.CommandType = CommandType.Text;
     sqlCommand.CommandText = cmdString;
     sqlCommand.Parameters.Add("@FacultyName", SqlDbType.Char).Value = ComboName.Text;
D    intDelete = sqlCommand.ExecuteNonQuery();
E    sqlCommand.Dispose();
F    if (intDelete == 0)
        Response.Write("<script>alert('The data deleting is failed')</script>");
G    for (intDelete = 0; intDelete < 7; intDelete++)          // clean up the Faculty textbox array
     {
        FacultyTextBox[intDelete] = new TextBox();
        FacultyTextBox[intDelete].Text = string.Empty;
     }
}
```

Figure 8.49 Coding for the Delete button's Click method.

Table 8.7 Data to Be Added into the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu

Table 8.8 Data to Be Added into the LogIn Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	NULL

Table 8.9 Data to Be Added into the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880

Table 8.10 Data to Be Added into the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1005	J77896	CSC-234A	3	CS/IS
1009	A78835	CSE-434	3	CE
1014	A78835	CSE-438	3	CE
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE

F. The returned value from calling the ExecuteNonQuery() method is exactly equal to the number of rows that have been successfully deleted from our sample database. If this value is zero, which means that no row has been deleted or affected from our database and this data deleting has failed, a warning message is displayed. Otherwise if a nonzero value is returned, at least one row in our database has been deleted, and this data deleting is successful.

G. A cleaning operation is performed to clean up the contents of all textboxes that stored the deleted faculty information.

At this point, we finished all coding operations to delete data in the SQL Server database via Web pages. Before we can run the project to test this deleting function, make sure that the starting page is the LogIn page. After the project runs, enter the suitable username and password to complete the LogIn process. Then open the Faculty page by clicking on the Faculty Information item from the Selection page, keep the default faculty Ying Bai selected from the combobox control, and then click on the Select button to retrieve and display this faculty's information.

Click on the Delete button to delete this faculty record from our database. To confirm this data deleting, keep the deleted faculty member Ying Bai selected in the combobox control ComboName, and click on the Select button to try to retrieve this deleted faculty information. A warning message: "No matched faculty found!" is displayed to indicate that this faculty member has been deleted from our sample database.

Another way to confirm this data deleting is to open our sample database and find that all records related to that deleted faculty, as shown in Tables 8.7 to 8.10, have been deleted from our database. Yes, our data deleting action is successful.

Before we can close the SQL Server Management Studio, it is highly recommended to recover all records that have been deleted from our sample database. To do this recovering operation, you need to take the following actions in the following order:

1. Recover the Faculty table by adding the deleted faculty record into the Faculty table, which is shown in Table 8.7.
2. Recover the LogIn table by adding the deleted login record into the LogIn table, as shown in Table 8.8.
3. Recover the Course table by adding the deleted courses taught by the deleted faculty member into the Course table, which is shown in Table 8.9.
4. Recover the StudentCourse table by adding the deleted courses taken by the associated students into the StudentCourse table, as shown in Table 8.10.

Some readers may have noticed the following interesting point: Although we have developed the codes to clean up all seven textboxes' contents after this deletion action, however, it looks like those pieces of code do not work and the deleted faculty information stored in those seven textboxes are still in there. What is the reason for that? Does our code have something wrong? Try to think about this and find the solution yourself. Yes, the answer is simple. This is a significant difference that exists between the Windows-based and Web-based applications since our project SQLWebUpdateDelete will run at the server side, and each time the server sends back a refreshed Faculty page as an action is performed from the client side. After a deletion action is performed, the server still sends back a refreshed page that contains the original faculty information in seven textboxes to the client.

A complete Web application project SQLWebUpdateDelete can be found at the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

8.6 DEVELOP ASP.NET WEB APPLICATIONS WITH LINQ TO SQL QUERY

In this section, we provide a fundamental end-to-end LINQ to SQL scenario for adding, modifying, and deleting data against our sample database via a Web page. As you know, LINQ to SQL queries can perform not only the data selections, but also the data insertion, updating, and deletion. The standard LINQ to SQL queries include:

- Select
- Insert
- Update
- Delete

To perform any of these operations or queries, we need to use entity classes and DataContext, which we discussed in Section 4.6.1, in Chapter 4 to do LINQ to SQL actions in our sample database. We already created a Console project QueryLINQSQL in that section to illustrate how to use LINQ to SQL to perform data queries, such as data selection, insertion, updating, and deleting, in our sample database CSE_DEPT.mdf. However, in this section, we want to create a Web-based project SQLWebLINQ by

adding a graphic user interface to perform the data selection, and data insertion, and data updating and deleting actions in our sample database CSE_DEPT.mdf using the LINQ to SQL query via Web pages. Now let's perform the following steps to create our new project SQLWebLINQ:

1. Create a new Visual C# Web-based project and name it SQLWebLINQ.
2. Create a new Web form page with (1) five button controls: Select, Insert, Update, Delete, and Exit; (2) eight TextBox controls, (3) one DropDownList control, and (4) one Image Box control.
3. Add a System.Data.Linq reference to this new project by right-clicking on our new project from the Solution Explorer window, selecting the Add Reference item and scroll down the list, and selecting the item System.Data.Linq from the list and clicking on the OK button.
4. Add the following directives at the top of the Faculty page file:
   a. Using System.Data.Linq;
   b. Using System.Data.Linq.Mapping;
5. Follow the steps listed in Section 4.6.1 in Chapter 4 to create entity classes using the Object Relational Designer. The database used in this project is **CSE_DEPT.mdf**, and it is located at the folder C:\database\SQLServer. Open the Server Explorer window and add this database by right-clicking on the Data Connections item and select Add Connection if it has not been added into our project.
6. We need to create five entity classes, and each of them is associated with a data table in our sample database. Drag each table from the Server Explorer window and place it on the Object Relational Designer canvas. The mapping file's name is CSE_DEPT.dbml. Make sure that you enter this name into the Name box in the Object Relational Designer.

Now let's begin the coding process for this project. Since we need to use the Select button's Click method to validate our data insertion and data updating and deleting actions, we need to divide our coding process into the following four parts:

1. Create a new object of the DataContext class and do some initialization coding.
2. Develop the codes for the Select button's Click method to retrieve the selected faculty information using the LINQ to SQL query.
3. Develop the codes for the Insert button's Click method to insert new faculty member using the LINQ to SQL query.
4. Develop the codes for the Update button's Click method to update the selected faculty member using the LINQ to SQL query.
5. Develop the codes for the Delete button's Click method to delete the selected faculty member using the LINQ to SQL query.

Before we can start the coding process, first let's create a new Web form page as our graphic user interface to perform those data actions.

8.6.1 Create New Web Form Page

In the newly created Web site project SQLWebLINQ, click on the View Designer button to open the default Web page, Default.aspx, and add the components listed in Table 8.11 into this Web page. Your finished Web page should match the one shown in Figure 8.50.

Table 8.11 Controls on the Faculty Web Page

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	CSE DEPT Faculty Page	0		Bold/Large
Image	PhotoBox		22		
Label	Label2	Faculty Photo	1		Bold/ Smaller
TextBox	txtPhoto		2		
Label	Label3	Faculty Name	3		Bold/Smaller
DropDownList	ComboName		4		
Label	Label3	Faculty ID	5		Bold/ Smaller
TextBox	txtID		6		
Label	Label4	Name	7		Bold/ Smaller
TextBox	txtName		8		
Label	Label5	Title	9		Bold/ Smaller
TextBox	txtTitle		10		
Label	Label6	Office	11		Bold/ Smaller
TextBox	txtOffice		12		
Label	Label7	Phone	13		Bold/ Smaller
TextBox	txtPhone		14		
Label	Label8	College	15		Bold/ Smaller
TextBox	txtCollege		16		
Label	Label9	Email	17		Bold/ Smaller
TextBox	txtEmail		18		
Button	cmdSelect	Select	19		Bold/ Small
Button	cmdInsert	Insert	20		Bold/ Small
Button	cmdUpdate	Update	21		Bold/ Small
Button	cmdDelete	Delete	22		Bold/ Small
Button	cmdExit	Exit	23		Bold/ Small

A key point in developing this Web page is that you have to set most controls' Position property to Absolute using the Format|Position menu item after you finish dragging each control from the Tool box window and placing it on the Web form page. These controls include all TextBoxes, DropDownList, Image control, and button controls. You also need to set the line-height size in the Style|Block property to 5 px for this Web page to align each label to make them vertically equal.

Another point is that you should not use the Copy|Paste menu items to create those buttons. Instead, you need to create those buttons one by one by dragging each of them from the ToolBox window and placing them on this page. Otherwise, the relationship between each button and its event method may be missed.

Finally, you need to set the Font size of the Text property of all label controls to smaller using the Format|Font menu item. Now let's start our coding process.

8.6.2 Create New Object of DataContext Class

We need to create this new object of the DataContext class since we need to use this object to connect to our sample database to perform data queries. We have connected

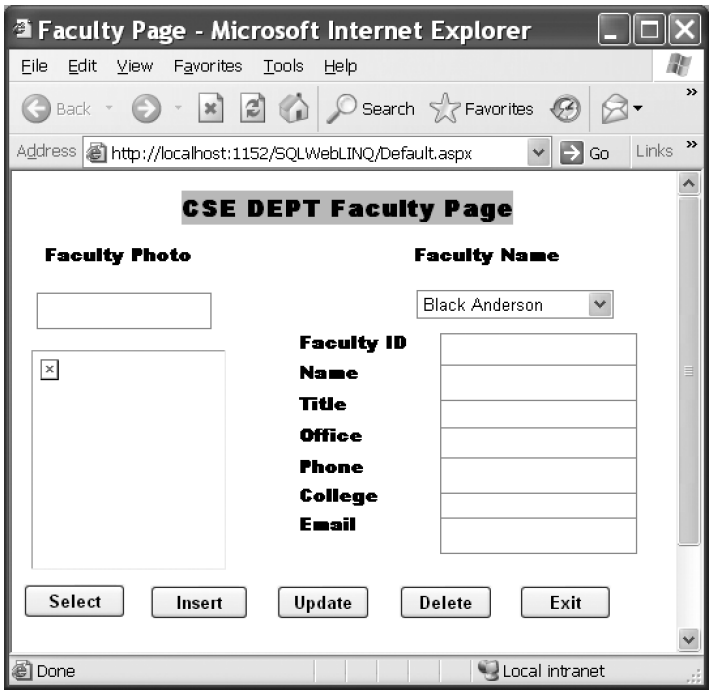

Figure 8.50 Default.aspx Web page.

this DataContext class to our sample database CSE_DEPT.mdf in step 5 in Section 8.6, and the connection string has been added into our web.config file when this connection is done. Therefore we do not need to indicate the special connection string for this object.

Some initialization coding includes retrieving all updated faculty members from the Faculty table in our sample database using the LINQ to SQL query and display them in the ComboName combobox control.

Open the code window and the Page_Load() method of the Faculty Web page, and enter the codes shown in Figure 8.51 into this method.

Let's take a close look at this piece of code to see how it works.

A. A new field-level object of the DataContext class, cse_dept, is created first since we need to use this object to connect our sample database to this Web project to perform the data actions later.

B. A user-defined UpdateFaculty() method is executed to retrieve all updated faculty members from our sample database and display them in the ComboName combobox control to allow users to select a desired faculty later. To avoid multiple displaying of retrieved faculty members, an `if` selection structure is adopted to make sure that we only display those updated faculty members in the combobox control ComboName at the first time as this Web page is loaded, and will not display them each time as the server sends back a refreshed Faculty page to the client when an action is performed in the client.

C. Before we can update the combobox control ComboName by adding the updated faculty members into this control, a cleaning job is performed to avoid the multiple adding and displaying of those faculty members.

Figure 8.51 Initialization codes for the Faculty Web page.

D. The LINQ query is created and initialized with three clauses, from, let, and select. The range variable `fi` is selected from the Faculty entity in our sample database. All current faculty members (`faculty_name`) will be read back using the let clause and assigned to the query variable faculty.

E. The LINQ query is executed to pick up all queried faculty members and add them into the ComboName combobox control in our Faculty Form.

The coding for the Exit button's Click method is easy, just enter the following code line into this method: `Response.Write("<script>window.close()</script>");`. The function of this line is to close this Web project if this Exit button is clicked.

8.6.3 Coding for Data Selection Query

Double-click on the Select button to open its Click method and enter the codes shown in Figure 8.52 into this method. The function of this piece of code is to retrieve all current faculty members from the Faculty table in our sample database and display them in the ComboName combobox control in the Faculty Form window as this Select button is clicked on by the user.

Let's take a close look at this piece of code to see how it works.

A. The user-defined ShowFaculty() method is executed to identify and display a matched faculty image for the selected faculty member. You can copy this piece of code from the Faculty Form class in the previous projects we developed in this chapter.

B. The LINQ query is created and initialized with three clauses: from, where, and select. The range variable `fi` is selected from the Faculty entity in our sample database based on a matched faculty members (`faculty_name`).

_Default | cmdSelect_Click()

```
protected void cmdSelect_Click(object sender, EventArgs e)
{
A    string strName = ShowFaculty(ComboName.Text);
B    var faculty = (from fi in cse_dept.Faculties
                    where fi.faculty_name == ComboName.Text
                    select fi);
C    foreach (var f in faculty)
     {
        txtID.Text = f.faculty_id;
        txtName.Text = f.faculty_name;
        txtTitle.Text = f.title;
        txtOffice.Text = f.office;
        txtPhone.Text = f.phone;
        txtCollege.Text = f.college;
        txtEmail.Text = f.email;
     }
}
```

Figure 8.52 Codes for the Select button Click method.

C. The LINQ query is executed to pick up all columns for the selected faculty member and display them on the associated textbox in our Faculty Form.

It is recommended that you copy the body of the user-defined ShowFaculty() method from any Faculty Form page in the previous project SQLWebUpdateDelete we developed in this chapter, and paste it into this Faculty Form page. Now let's concentrate on the coding for our data insertion actions.

8.6.4 Coding for Data Insertion Query

Double-click on the Insert button from our Faculty Form page to open its Click method, and enter the codes shown in Figure 8.53 into this method.

Let's take a close look at this piece of code to see how it works.

A. A new instance of the Faculty entity class is created since we need to add a new record into the Faculty table in our sample database.

B. Seven pieces of new faculty information stored in seven textbox controls are assigned to the associated columns in the Faculty instance that can be mapped to the Faculty table in our sample database.

C. A system method InsertOnSubmit() is executed to send our new created Faculty instance to our Faculty table via the DataContext class.

D. Another system method SubmitChanges() is executed to perform this data insertion.

E. After a new record has been inserted into our database, we need to update our combobox control ComboName to reflect that insertion. First, we need to clean up all original contents from this control to avoid multiple updating.

F. The user-defined UpdateFaculty() method is called to complete this updating.

G. In case the user wants to insert a new faculty image with that data insertion, the Text property of the textbox control txtPhoto, which stored a valid faculty image file, is assigned to the Application state function that works as a global variable. This global variable will

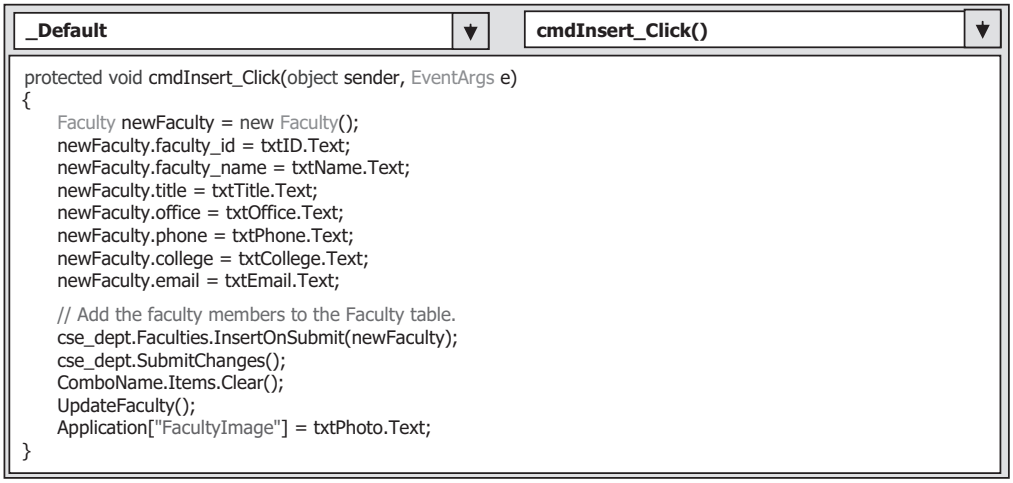

Figure 8.53 Codes for the Insert button Click method.

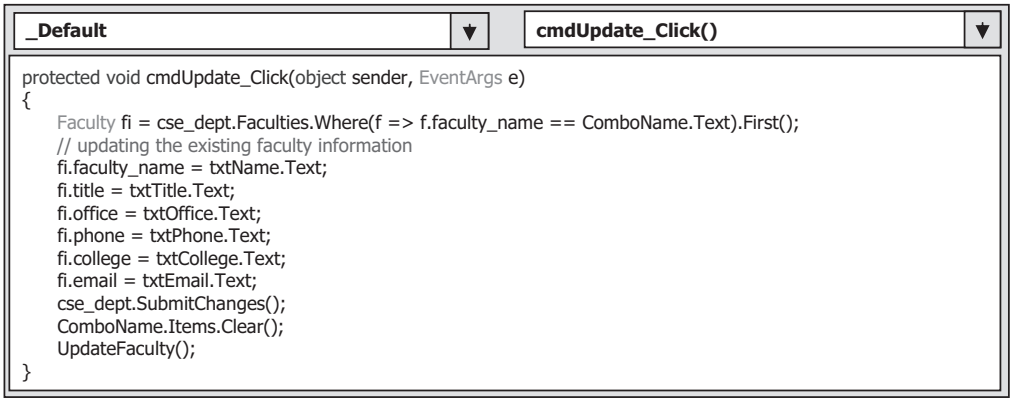

Figure 8.54 Codes for the Update button Click method.

be used later when we perform the confirmation of the data insertion in the Select button's Click method.

Now let's concentrate on the coding for our data updating and deleting actions.

8.6.5 Coding for Data Updating and Deleting Queries

Double-click on the Update button from our Faculty Form page window to open its Click method, and enter the codes shown in Figure 8.54 into this method.

Let's take a close look at this piece of code to see how it works.

A. A selection query is executed using the Standard Query Operator method with the `faculty_name` as the query criterion. The First() method is used to return only the first

_Default | cmdDelete_Click()

```
protected void cmdDelete_Click(object sender, EventArgs e)
{
A   var faculty = (from fi in cse_dept.Faculties
                   where fi.faculty_name == ComboName.Text
                   select fi).Single<Faculty>();

B   cse_dept.Faculties.DeleteOnSubmit(faculty);
C   cse_dept.SubmitChanges();

    // clean up all textboxes
D   txtID.Text = string.Empty;
    txtName.Text = string.Empty;
    txtOffice.Text = string.Empty;
    txtTitle.Text = string.Empty;
    txtPhone.Text = string.Empty;
    txtCollege.Text = string.Empty;
    txtEmail.Text = string.Empty;

E   ComboName.Items.Clear();
F   UpdateFaculty();
}
```

Figure 8.55 Codes for the Delete button Click method.

matched record. It does not matter to our application since we have only one record that is associated with this specified `faculty_name`.

B. All six columns for the selected faculty member are updated by assigning the current value stored in the associated textbox to each column in the Faculty instance in our DataContext class object cse_dept.

C. This data updating can be really performed only after the system method SubmitChanges() is executed.

D. The combobox control ComboName is cleaned up to be ready to be updated.

E. The user-defined UpdateFaculty() method is executed to refresh the updated faculty members stored in that control.

Before we can run our Web project to test these data actions, let's complete the last coding for our data deleting action.

Double-click on the Delete button from our Faculty Form page window to open its Click method, and enter the codes shown in Figure 8.55 into this method.

Let's take a close look at this piece of code to see how it works.

A. A LINQ selection query is first executed to pick up the faculty member to be deleted. This query is initialized with three clauses: from, where, and select. The range variable `fi` is selected from the Faculty, which is exactly an instance of our entity class Faculty, and the `faculty_name` works as the query criterion for this query. All information related to the selected faculty members (`faculty_name`) will be retrieved and stored in the query variable faculty. The Single() means that only a single or the first record is queried.

B. The system method DeleteOnSubmit() is executed to issue a deleting action to the faculty instance, Faculties in our DataContext class object cse_dept.

C. Another system method SubmitChanges() is executed to exactly perform this deleting action against data tables in our sample database. Only after this method is executed is the selected faculty record deleted from our database.

D. All textboxes that stored information related to the deleted faculty are cleaned up by assigning an empty string to each of them.

E. The combobox control ComboName is cleaned up to be ready to be updated.

F. The user-defined UpdateFaculty() method is executed to reflect deleting this faculty record for all faculty members stored in that control.

Now we can build and run our Web project to test the data actions against our sample database. One point we need to note before we can run the project is that we must make sure that all faculty image files should have been stored in the default folder in which our Web project SQLWebLINQ is located. In this application, it should be C:\Chapter 8\SQLWebLINQ.

Unlike other projects we developed in the previous chapters, in which a separate Insert Faculty Form must be used to perform the data insertion action, in this project, we can use the Faculty Form page to perform all data actions, including the data selection, data insertion, and data updating and deleting. As an example, let's run the project to test the data insertion action by inserting a new faculty member with the following information:

• P77777	Faculty ID textbox
• Peter Tom	Faculty Name textbox
• Assistant Professor	Title textbox
• MTC-200	Office textbox
• 750-378-2000	Phone textbox
• University of Miami	College textbox
• ptom@college.edu	Email textbox

Directly enter these new data into each associated textbox after the project runs, and you can select any faculty member from the combobox control ComboName to perform this insertion action. Click on the Insert button when you finish this data entering to all textboxes to perform this data insertion.

To confirm this data action, first select another faculty member from the combobox control ComboName and click on the Select button to retrieve and display that faculty's information. Then select the new inserted faculty Peter Tom, who should already be in the combobox control ComboName, and click on the Select button to try to retrieve that new inserted faculty's information and display it in this form. Your confirmation page should match the one shown in Figure 8.56.

A default faculty image is displayed for this data insertion since we did not include any faculty image file for this insertion. You can test to insert a new faculty with a selected faculty image by entering the name of that faculty image file into the Faculty Photo textbox control txtPhoto located at the upper-left corner of this page if you like.

Note that you had better recover any deleted faculty record if a data deleting action is tested for this project since we want to keep our database neat and complete. Refer to Tables 8.7 to 8.10 in section 8.5.4.4 to recover the deleted records to our sample database.

A complete Web page application project SQLWebLINQ can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1). Next let's take care of the Web applications with the Oracle database.

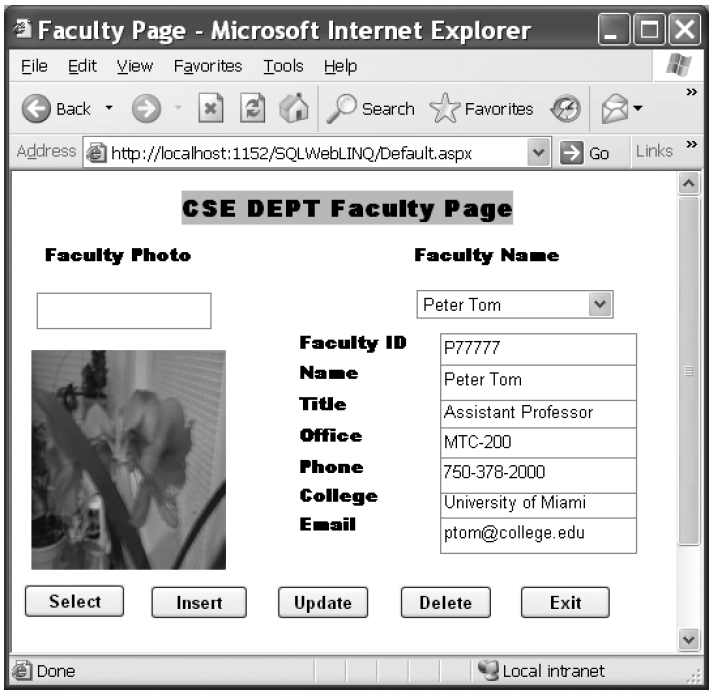

Figure 8.56 Testing status of the data insertion action.

8.7 DEVELOP ASP.NET WEB APPLICATION TO SELECT DATA FROM ORACLE DATABASES

Because of the coding similarity between the SQL Server and Oracle databases, we will emphasize the main differences between the codes in SQL Server and Oracle data actions. Also in order to save time and space, we will modify the existing Web application project SQLWebSelect we developed in the last section to make it our new project OracleWebSelect in this section.

The main coding differences between these two database operations are:

1. Connection string and Connection object in the LogIn page
2. LogIn query string in the LogIn page
3. Query string in the Faculty page
4. Query strings in the Course page, which include the query string in the Select button's Click method and the query string in the SelectedIndexChanged event method of the CourseList box control
5. Namespace and data objects used in the Selection page
6. Prefix for each data object and class used for the Oracle database operations
7. Data type of the passed arguments of methods for Oracle database operations

Now let's begin to modify the project SQLWebSelect based on the seven differences listed above to make it our new project OracleWebSelect. Open Windows Explorer and

create a new folder such as Chapter 8 if you have not created it. Copy the project SQLWebSelect from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1) and paste it in the folder C:\Chapter 8. Rename the project to OracleWebSelect.

Open the Visual Studio.NET, go to the File|Open Web Site menu item, and browse to our folder Chapter 8. Select our new project OracleWebSelect and then click on the Open button to open it.

Let's first modify the codes of the connection string in the LogIn page.

8.7.1 Modify Connection String and Connection Object on LogIn Page

Open the code page of the LogIn Web form by clicking on it from the Solution Explorer window, and then clicking on the View Code button. The first thing we need to do is to add the Oracle data client reference to our project. To do that, right-click on our project icon on the Solution Explorer window and select Add Reference item from the pop-up menu to open the Add Reference dialog box. Browse down on the list until you find the item System.Data.OracleClient. Click on this item to select it, and click on the OK button to add this reference to our project. Now we need to add one more Oracle Data Provider–related namespace to the top of this page to set the namespace that contains the data components for the Oracle Data Provider:

```
using System.Data.OracleClient;
```

Now open the Page_Load() method and perform the following modifications to the codes in this method:

A. Add an Oracle Data Provider–related namespace to this page.

B. Change the prefix for the global connection object from sqlConnection to oraConnection since we need to use the Oracle data components in this section.

C. Change the connection string to contain the User ID and Password related to our sample Oracle database.

D. Create a new instance of Oracle connection class with the Oracle connection string oraString as the argument. Also change the prefix for all Oracle data objects and classes from **Sql** to **Oracle**, and from **sql** to **ora**, respectively.

E. Change the prefix for the global connection object stored in the Application state function from **sql** to **ora**.

F. Change the prefix for all following data components from **sql** to **ora**.

Your finished modifications to the Page_Load() method and the connection string should match the one shown in Figure 8.57. All modified parts have been highlighted in bold.

The next modification is to change the prefix of each Connection object in the Cancel button's Click method from sqlConnection to oraConnection. Your finished modification to this method should match the one shown in Figure 8.58. The modified parts have been highlighted in bold.

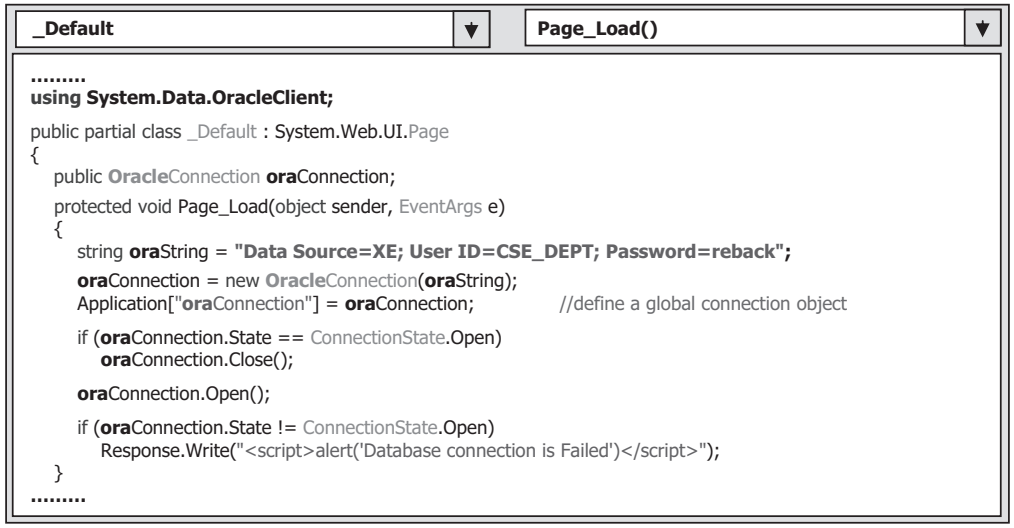

Figure 8.57 Modified connection object and connection string.

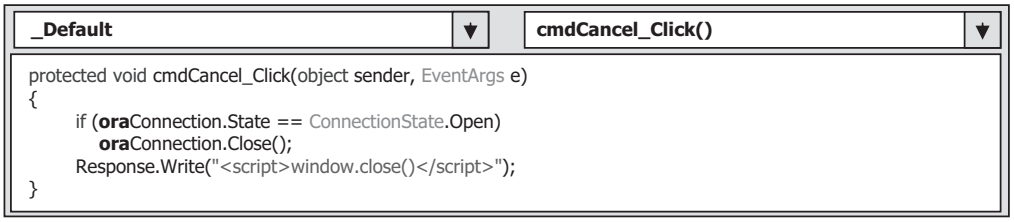

Figure 8.58 Modified connection object in Cancel button method.

8.7.2 Modify Query String in LogIn Page

Now open the LogIn button's Click method and perform the modifications shown in Figure 8.59 to the codes in this method. All modified parts have been highlighted in bold.

Let's take a look at this piece of code to see these modifications.

A. Change the query string from the SQL Server database to the Oracle database. The Oracle database assignment operator =: is used to replace the SQL Server database assignment operator LIKE @.

B. Change the prefix for all data components and classes from sql to ora and from Sql to Oracle, respectively.

C. Change the nominal names for the dynamic parameters from @name to name and from @word to word, respectively. Also change the data type of these two dynamic parameters from SqlDbType to OracleType.

D. Change the prefix for all data components and classes from sql to ora and from Sql to Oracle, respectively.

E. Change the prefix for all data components from sql to ora.

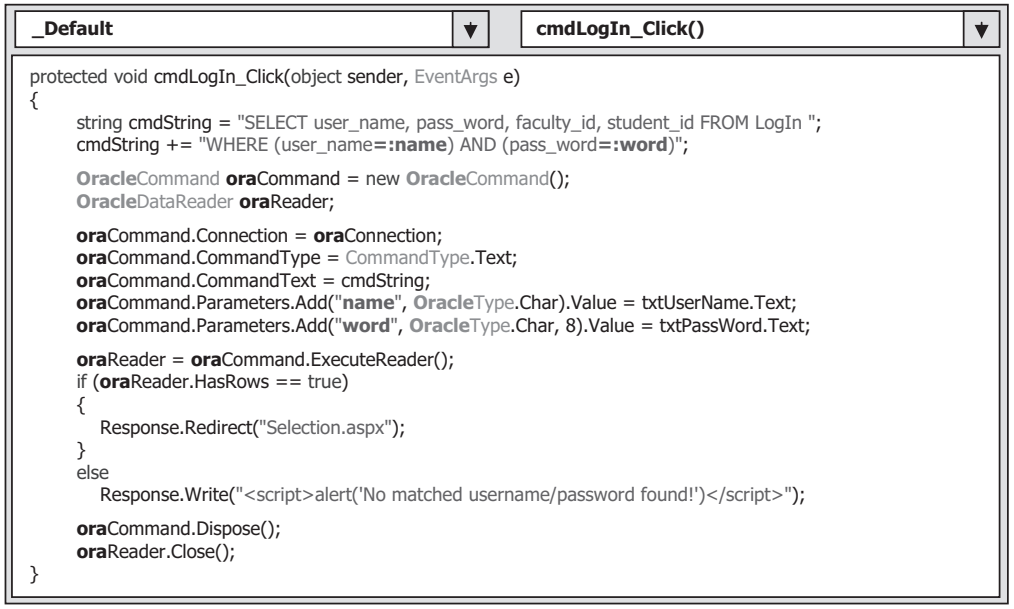

Figure 8.59 Modifications to codes in LogIn button method.

Go to the File|Save All menu item to save these modifications. Now let's continue our modifications to the next page, the Faculty page.

8.7.3 Modify Query String in Faculty Page

The modifications to this page include the following contents:

1. Adding an Oracle Data Provider–related namespace to the top of this page
2. Modifications to the global connection object stored in the Application state function in the Page_Load() method
3. Modifications to the codes in the Select button's Click method
4. Modifications to the data type of the passed argument in the user-defined method FillFacultyReader()

Let's first add an Oracle Data Provider–related namespace to the namespace area located at the top of this page:

```
using System.Data.OracleClient;
```

Open the Page_Load() method and change the connection object stored in the Application state from sqlConnection to oraConnection. Your finished modifications to this method should match the one shown in Figure 8.60. The modified parts have been highlighted in bold.

Now open the Select button's Click method and perform the following modifications:

Faculty | Page_Load()

```
.....
using System.Data.OracleClient;
public partial class Faculty : System.Web.UI.Page
{
   private TextBox[] FacultyTextBox = new TextBox[7];
   protected void Page_Load(object sender, EventArgs e)
   {
      if (((OracleConnection)Application["oraConnection"]).State != ConnectionState.Open)
         ((OracleConnection)Application["oraConnection"]).Open();
      if (!IsPostBack)
      {
         ComboName.Items.Add("Ying Bai");
         ComboName.Items.Add("Satish Bhalla");
         ComboName.Items.Add("Black Anderson");
         ComboName.Items.Add("Steve Johnson");
         ComboName.Items.Add("Jenney King");
         ComboName.Items.Add("Alice Brown");
         ComboName.Items.Add("Debby Angles");
         ComboName.Items.Add("Jeff Henry");
      }
   }
```

Figure 8.60 Modified Page_Load method.

A. Change the query string by replacing the SQL Server database assignment operator LIKE @ with the Oracle database operator =: in the WHERE clause.

B. Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**, respectively.

C. Modify the global Connection object stored in the Application state from the **sqlConnection** to the **oraConnection**.

D. Modify the nominal name of the dynamic parameter `@name` by removing the @ symbol before the parameter `name`.

E. Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**.

Your finished modifications to this method should match the one shown in Figure 8.61. All modified parts have been highlighted in bold.

The modification to the data type of the passed argument in the user-defined method FillFacultyReader() is simple, and just change the data type of that passed argument from the SqlDataReader to the OracleDataReader.

8.7.4 Modify Query Strings in Course Page

The modifications to this page include the following contents:

1. Adding an Oracle Data Provider–related namespace to the top of this page
2. Modifications to the global connection object stored in the Application state in the Page_Load() method
3. Modifications to the codes in the Select button's Click method

Faculty | cmdSelect_Click()

```
protected void cmdSelect_Click(object sender, EventArgs e)
{
      string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty ";
A     cmdString += "WHERE faculty_name =: name";
B     OracleCommand oraCommand = new OracleCommand();
      OracleDataReader oraDataReader;

C     oraCommand.Connection = (OracleConnection)Application["oraConnection"];
      oraCommand.CommandType = CommandType.Text;
      oraCommand.CommandText = cmdString;
D     oraCommand.Parameters.Add("name", OracleType.Char).Value = ComboName.Text;
      string strName = ShowFaculty(ComboName.Text);
E     oraDataReader = oraCommand.ExecuteReader();
      if (oraDataReader.HasRows == true)
         FillFacultyReader(oraDataReader);
      else
         Response.Write("<script>alert('No matched faculty found!')</script>");
      oraDataReader.Close();
      oraCommand.Dispose();
}
```

Figure 8.61 Modifications to the Select button's Click method.

4. Modifications to the codes in the SelectedIndexChanged event method of the listbox control CourseList
5. Modifications to the data type of the passed argument in the user-defined methods FillCourseReader() and FillCourseReaderTextBox()

Let's first add an Oracle Data Provider–related namespace to the namespace area that is located at the top of this page:

```
using System.Data.OracleClient;
```

Open the Page_Load() method and change the Connection object stored in the Application state function from sqlConnection to oraConnection. Your finished modifications to this method should match the one shown in Figure 8.62. The modified parts have been highlighted in bold.

Now open the Select button's Click method and perform the following modifications:

A. The query string applied for the joined table must be modified since the syntax of the query string used for the SQL Server database is ANSI 92 standard, and this standard is up to date. However, this standard cannot be recognized by the Oracle database since the Oracle database still uses an old standard called ANSI 89 standard. In order to match the requirement of the Oracle database, the query string must be modified. Refer to Section 5.19.2.5 in Chapter 5 to get more detailed information about these two standards.

B. Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**, respectively.

C. Modify the global connection object stored in the Application state function from the **sqlConnection** to the **oraConnection**.

D. Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**.

Course | Page_Load()

```
.....
using System.Data.OracleClient;
public partial class Course : System.Web.UI.Page
{
    private TextBox[] CourseTextBox = new TextBox[6];
    protected void Page_Load(object sender, EventArgs e)
    {
        if (((OracleConnection)Application["oraConnection"]).State != ConnectionState.Open)
            ((OracleConnection)Application["oraConnection"]).Open();
        if (!IsPostBack)            //these items can only be added into the combo box in one time
        {
            ComboName.Items.Add("Ying Bai");
            ComboName.Items.Add("Satish Bhalla");
            ComboName.Items.Add("Black Anderson");
            ComboName.Items.Add("Steve Johnson");
            ComboName.Items.Add("Jenney King");
            ComboName.Items.Add("Alice Brown");
            ComboName.Items.Add("Debby Angles");
            ComboName.Items.Add("Jeff Henry");
        }
    }
```

Figure 8.62 Modified Page_Load method in the Course page.

Course | cmdSelect _Click()

```
protected void cmdSelect_Click(object sender, EventArgs e)
{
A       string strCourse = "SELECT Course.course_id, Course.course FROM Course, Faculty ";
        strCourse += "WHERE (Course.faculty_id=Faculty.faculty_id) AND (Faculty.faculty_name=:name)";
B       OracleCommand oraCommand = new OracleCommand();
        OracleDataReader oraDataReader;
C       oraCommand.Connection = (OracleConnection)Application["oraConnection"];
D       oraCommand.CommandType = CommandType.Text;
        oraCommand.CommandText = strCourse;
E       oraCommand.Parameters.Add("name", OracleType.Char).Value = ComboName.Text;
        oraDataReader = oraCommand.ExecuteReader();
        if (oraDataReader.HasRows == true)
            FillCourseReader(oraDataReader);
        else
            Response.Write("<script>alert('No matched course found!')</script>");
        oraDataReader.Close();
        oraCommand.Dispose();
}
```

Figure 8.63 Modified Select button's Click method.

E. Modify the nominal name of the dynamic parameter `@name` by removing the @ symbol before the parameter `name`. Also change the data type for this dynamic parameter from SqlDbType to OracleType.

Your modified Select button's Click method should match one that is shown in Figure 8.63. All modified parts have been highlighted in bold.

Next open the SelectedIndexChanged event method of the listbox control CourseList, and perform the following modifications:

Course | CourseList_SelectedIndexChanged()

```
protected void CourseList_SelectedIndexChanged(object sender, EventArgs e)
{
A       string cmdString = "SELECT course, credit, classroom, schedule, enrollment, course_id FROM Course ";
        cmdString += "WHERE course_id =: courseid";
B       OracleCommand oraCommand = new OracleCommand();
        OracleDataReader oraDataReader;
C       oraCommand.Connection = (OracleConnection)Application["oraConnection"];
        oraCommand.CommandType = CommandType.Text;
        oraCommand.CommandText = cmdString;
D       oraCommand.Parameters.Add("courseid", OracleType.Char).Value = CourseList.SelectedItem.ToString();
E       oraDataReader = oraCommand.ExecuteReader();
        if (oraDataReader.HasRows == true)
            FillCourseReaderTextBox(oraDataReader);
        else
            Response.Write("<script>alert('No matched course information found!')</script>");
        oraDataReader.Close();
        oraCommand.Dispose();
}
```

Figure 8.64 Modified SelectedIndexChanged event method.

A. Modify the assignment operator for the dynamic parameter `courseid` in the query string by replacing the LIKE @ with the Oracle assignment operator =:.

B. Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**, respectively.

C. Modify the global connection object stored in the Application state function from the sqlConnection to the oraConnection.

D. Modify the nominal name of the dynamic parameter `@courseid` by removing the @ symbol before the parameter `courseid`. Also change the data type for this dynamic parameter from SqlDbType to OracleType.

E. Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**.

Your modified SelectedIndexChanged method should match one that is shown in Figure 8.64. All modified parts have been highlighted in bold.

Modifications to the data type of the passed argument in the user-defined methods FillCourseReader() and FillCourseReaderTextBox() are simple, and just change the data type of that passed argument from the SqlDataReader to the OracleDataReader.

The last modification is to add an Oracle Data Provider–related namespace into the Selection page and data objects used in the Selection page. Add the following namespace to the top of this page:

```
using System.Data.OracleClient;
```

Modify the global connection object stored in the Application state function from the sqlConnection to the oraConnection in the Exit button's Click method.

At this point, we have finished all modifications to this new project. Before we can run the project to test the function of our coding, the following two points must be noted:

1. Make sure that all faculty photo files have been stored in our default folder, in which our project file is located.
2. Make sure that the Start page in our Web application is LogIn page.

To confirm the second point, right-click on our project icon from the Solution Explorer window and select the Start Options item from the pop-up menu to open the Property Page. On the opened page window, select the Specific page radio button and click on the ellipsis button that is next to the Specific page box to open the Select Page to Start dialog box. In the opened dialog, click on the `LogIn.aspx` from the list and click on OK to select it as our start page. Finally click on the OK button to the Property Page to finish this setup.

Now you can click on the Start Debugging button to run the project to confirm the functionalities of our coding.

A complete Web application project OracleWebSelect can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

8.8 DEVELOP ASP.NET WEB APPLICATION TO INSERT DATA INTO ORACLE DATABASES

Because of the coding similarity between the SQL Server and the Oracle databases, we only emphasize the important differences in the coding for these two databases. To save time and space, we need to modify an existing project OracleWebSelect by adding some new components and codes to this project.

Unlike other projects we developed in the previous sections, in which a new Insert Faculty Form page needs to be created to allow us to insert a new faculty member into our sample database, in this project, we will use the Faculty Form page as our graphical user interface to perform this data insertion. To do that, we need to perform the following tasks to finish developing this Web application.

First, we need to add the following two controls to the Faculty Form page to complete our graphic user interface design:

1. A Faculty Photo textbox control that allows users to enter a new faculty image file.
2. A label control to indicate the function of the Faculty Photo textbox control.

Next we need to perform some code modifications to the following methods:

1. The Page_Load() method in the Faculty page
2. The ShowFaculty() method in the Faculty page

Finally we need to develop new codes for the following methods:

1. The Insert button's Click method in the Faculty page
2. The user-defined InsertParameters() method
3. The FacultyID TextChanged() method

Now let's begin to perform these tasks to build our new project. First, let's start to modify an existing project OracleWebSelect to make it as our new project OracleWebInsert. Open Windows Explorer and create a new folder such as Chapter 8 if you have not created it. Copy the project OracleWebSelect from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1) and paste it to our folder C:\Chapter 8. Rename the project to OracleWebInsert.

Open the Visual Studio.NET, go to the File|Open Web Site menu item, and browse to our folder Chapter 8. Then select our new project OracleWebInsert and click on the Open button to open it. First, let's complete our graphic user interface design by adding two controls to the Faculty page.

8.8.1 Add Two Controls to Faculty Page

Open the Faculty Form page window and add the two controls shown in Table 8.12 into this Faculty page. Your modified Faculty page should match the one shown in Figure 8.65.

Table 8.12 Added Controls to Faculty Web Page

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	Faculty Photo	0		Bold/Smaller
TextBox	txtPhoto		1		

Next let's perform some code modifications to some methods in the Faculty page.

8.8.2 Modify Codes to Some Methods on Faculty Page

1. First, we need to remove some codes from the Page_Load() method and add some new codes to retrieve the updated faculty members from our sample database to confirm the data insertion action.

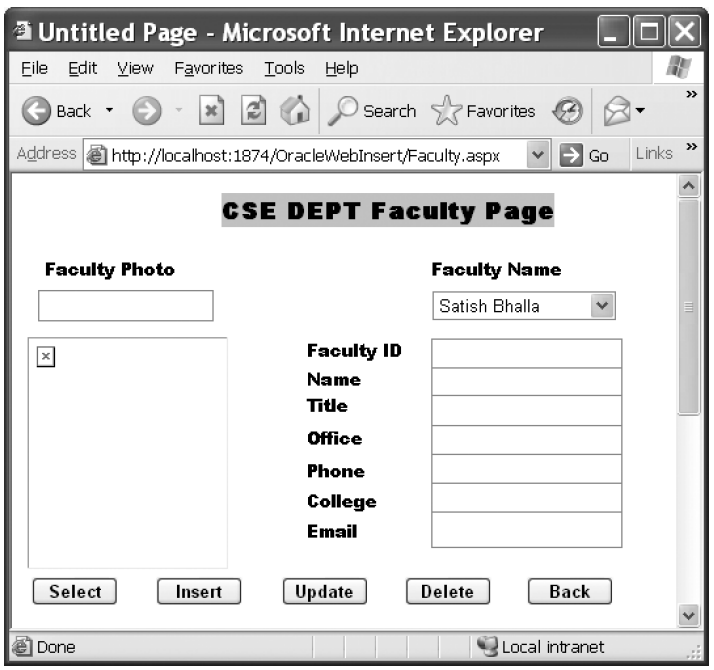

Figure 8.65 Modified Faculty page.

2. Second, we need to modify and add another piece of coding in the ShowFaculty() method to allow the new inserted faculty photo to be displayed as the new inserted data is validated.

Open the Page_Load() method in the Faculty page. Perform the modifications shown in Figure 8.66 to this method. The codes we developed in the previous section have been highlighted with shading.

Let's take a closer look at this piece of modified code to see how it works.

A. Each time the Faculty page starts, we need to update the combobox control ComboName to reflect any new inserted or updated faculty members against our sample database. To do that, a new user-defined UpdateFaculty() method is added into this Page_Load() method to fulfill this task.

B. To perform updating the faculty records in our sample database inside the UpdateFaculty() method, first an instance of the Oracle Command class is created.

C. Then the system ExecuteReader() method is called to read back all updated faculty records from our sample database and assign them to the Oracle Data Reader object.

D. A `while` loop is utilized to retrieve each record and add each into the combobox control ComboName.

E. A cleaning job is performed to release all data objects used in this method.

Now let's take care of the code modifications to the user-defined method ShowFaculty(). The purpose of this modification is to enable users to insert a new faculty image with a new faculty record insertion, and this new inserted faculty image can be displayed as the data insertion validation process to be performed later.

Faculty | **Page_Load()**

```
protected void Page_Load(object sender, EventArgs e)
{
    if (((OracleConnection)Application["oraConnection"]).State != ConnectionState.Open)
        ((OracleConnection)Application["oraConnection"]).Open();

    if (!IsPostBack)
    {
A       UpdateFaculty();
        ComboName.SelectedIndex = 0;
    }
}
public void UpdateFaculty()
{
B   OracleCommand oraCommand = new OracleCommand("SELECT faculty_name FROM Faculty",
                                    (OracleConnection)Application["oraConnection"]);
C   OracleDataReader oraReader = oraCommand.ExecuteReader();
D   while (oraReader.Read())
    {
        ComboName.Items.Add(oraReader[0].ToString());
    }
E   oraReader.Close();
    oraCommand.Dispose();
}
```

Figure 8.66 Modified Page_Load method.

Open the user-defined ShowFaculty() method and make the modifications shown in Figure 8.67 to this method.

Let's take a closer look at these modified codes to see how they work.

A. An `if` structure is used to check whether a valid faculty image has been detected. If it has, the detected faculty image file is assigned to the ImageUrl property of the PhotoBox control to display it.

B. Otherwise, if the global variable FacultyImage stored in the Application state function is an empty string, the user did not want to insert a new faculty image with that data insertion

Faculty ▼ **ShowFaculty()** ▼

```
private string ShowFaculty(string fName)
{
      string FacultyImage;
      switch (fName)
      {
         case "Black Anderson":
            FacultyImage = "Anderson.jpg";
            break;
         case "Ying Bai":
            FacultyImage = "Bai.jpg";
            break;
         case "Satish Bhalla":
            FacultyImage = "Satish.jpg";
            break;
         case "Steve Johnson":
            FacultyImage = "Johnson.jpg";
            break;
         case "Jenney King":
            FacultyImage = "King.jpg";
            break;
         case "Alice Brown":
            FacultyImage = "Brown.jpg";
            break;
         case "Debby Angles":
            FacultyImage = "Angles.jpg";
            break;
         case "Jeff Henry":
            FacultyImage = "Henry.jpg";
            break;
         default:
            FacultyImage = "No Match";
            break;
      }
A     if (FacultyImage != "No Match")
         PhotoBox.ImageUrl = FacultyImage;
      else
B        if (((string)Application["FacultyImage"] == string.Empty) || (string)Application["FacultyImage"] == null)
            FacultyImage = "Default.jpg";
C        else
            FacultyImage = (string)Application["FacultyImage"];
D     PhotoBox.ImageUrl = FacultyImage;
E     return FacultyImage;
}
```

Figure 8.67 Modified user-defined ShowFaculty method.

action. If that global variable is a null object, no data insertion action has been performed. In either case, a default faculty image is displayed.

C. If both above situations are not true, which means that the user did insert a new faculty image with that new data insertion action and the new faculty image file has been stored in the global variable FacultyImage, that global variable is assigned to the local variable FacultyImage, which will be displayed later.

D. The FacultyImage is assigned to the ImageUrl property of the PhotoBox control to display this image.

E. The FacultyImage string is returned to the calling method.

The functionality of these new added codes is that a default faculty photo file Default.jpg will be assigned to the FacultyImage variable if the global variable FacultyImage is empty. Otherwise the faculty photo file stored in the Application state will be assigned to the FacultyImage variable, which will be displayed later in the PhotoBox image control in the Faculty page.

8.8.3 Create Codes to Insert New Faculty on Faculty Page

Now let's handle developing new codes to the data insertion action. This procedure includes adding new codes to the following three methods:

1. The Insert button's Click method

2. The user-defined InsertParameters() method

3. The FacultyID TextChanged() method

Open the Insert button's Click method by double-clicking on the Insert button from the Faculty Form page window and enter the codes shown in Figure 8.68 into this method.

Let's take a closer look at this piece of code to see how it works.

A. An insert query string is declared here with the Oracle database query syntax. One of the most important differences between the SQL Server and Oracle database query syntax is the assignment operator for the dynamic parameter. Instead of using a LIKE @ symbol before the dynamic parameter, a =: operator is used for the Oracle database in the VALUES clause.

B. The Command object and some local variables are created since we need to use them to perform this data insertion action.

C. We need to reserve the faculty image file, that is, the location of this image file, and save it to a local variable since we need it in the data validation process later.

D. If the content of this faculty image file is empty, which means that the user did not want to insert a new faculty image with this data insertion, a default faculty image file is used since we do not want to keep our Faculty Photo box empty during the validation process.

E. This faculty image file is stored into an Application state function as a global variable, and it will be used later in the validation process.

F. The Command object is initialized and built with Connection, CommandType, and CommandText properties.

G. A user-defined InsertParameters() method is executed to fill out all inserted new parameters for the insert query string declared at the beginning of this method.

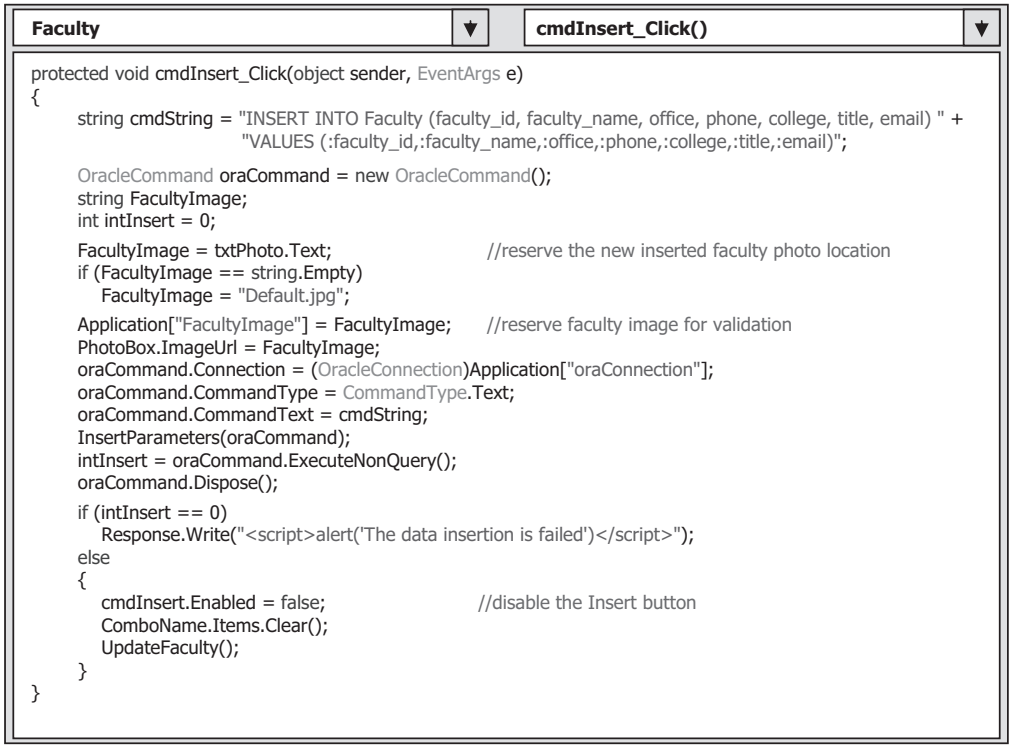

Figure 8.68 Codes for the Insert button's Click method.

H. The system method ExecuteNonQuery() is called to perform this data insertion action against our sample database.

I. A cleaning job is performed to release the object we used in this method.

J. The system ExecuteNonQuery() method will return a data value to indicate whether this data insertion is successful or not. If a zero is returned, which means that no row or zero row has been inserted into our database and this data insertion has failed, in that case, a warning message is displayed.

K. Otherwise, a nonzero value is returned and this means that the data insertion is successful. In order to void multiple duplicated insertions, the Insert button is disabled after this insertion. Also the combobox control ComboName is cleaned up and the user-defined UpdateFaculty() method is executed to refill that control to update the faculty members stored in that combobox.

The detailed codes of the user-defined InsertParameters() method are shown in Figure 8.69.

This piece of code is straightforward and easy to understand. Input parameters or seven pieces of new faculty information are assigned to the associated dynamic parameters in the insert query string with the system Add() method.

Finally let's handle the coding for the FacultyID TextChanged() method. As we know, in order to void multiple duplicated insertions for the same data, the Insert button should be disabled after a new faculty record has been inserted into the database. The

Faculty | InsertParameters()

```
private void InsertParameters(OracleCommand cmd)
{
      cmd.Parameters.Add("faculty_id", OracleType.Char).Value = txtID.Text;
      cmd.Parameters.Add("faculty_name", OracleType.Char).Value = txtName.Text;
      cmd.Parameters.Add("office", OracleType.Char).Value = txtOffice.Text;
      cmd.Parameters.Add("phone", OracleType.Char).Value = txtPhone.Text;
      cmd.Parameters.Add("college", OracleType.Char).Value = txtCollege.Text;
      cmd.Parameters.Add("title", OracleType.Char).Value = txtTitle.Text;
      cmd.Parameters.Add("email", OracleType.Char).Value = txtEmail.Text;
}
```

Figure 8.69 Codes for the user-defined InsertParameters method.

Faculty | txtID_TextChanged()

```
protected void txtID_TextChanged(object sender, EventArgs e)
{
      cmdInsert.Enabled = true;      //enable the Insert button when the faculty_id is changed
}
```

Figure 8.70 Codes for the txtID_TextChanged method.

question is: When should this Insert button be enabled again? The answer is: When another new faculty record is ready to be inserted into our database. How do we know another new faculty record is ready to be inserted? The answer is: When the `faculty_id` stored in the Faculty ID textbox is changed or different with the previous one. A TextChanged event will be created as soon as the content of the Faculty ID textbox is changed, which means that a new faculty record is ready to be inserted to our database, and the associated TextChanged method will be triggered. Therefore, we need to enable this Insert button as long as this situation happened.

Double-click on the Faculty ID textbox to open this method and enter the code shown in Figure 8.70 into this method.

At this point we have finished all modifications to our new project. Before we can run the project to test the data insertion function, make sure that the following three jobs have been done:

1. Make sure that all faculty image files and a default faculty image file **Default.jpg** has been saved to our default folder in which our Web application project is located. In our application, it is C:\Chapter 8\OracleWebInsert.
2. Make sure that the startup page is LogIn. To confirm this, right-click on our project icon from the Solution Explorer window, select the Start Options item from the pop-up menu. On the opened dialog box, be sure that the Specific page radio button is selected and the page LogIn.aspx is in that box. Click on the OK button to close this dialog box.
3. Make sure that a faculty named Ali Mhamed is not located at the Faculty table in our sample database because we will use this faculty as an example to insert it into our sample database. To confirm that, open the Faculty table from our sample database and delete this record if it is in there. The reason for us to do this is because the database does not allow us to insert the same record more than once, so we must first delete that record before we can insert the same data into the database. If you want to insert other faculty data other than Ali Mhamed, you don't need to take this step.

Now click on the Start Debugging button to run the project. Enter the suitable username and password to the LogIn page, and select the Faculty Information from the Selection page to open the Faculty page. First, select any faculty member from the combobox control ComboName and click on the Select button to retrieve back and display all information related to that selected faculty. Then enter the following data as the information for a new faculty member:

- Mhamed.jpg — Faculty Photo textbox
- M56789 — Faculty ID textbox
- Ali Mhamed — Faculty Name textbox
- Professor — Title textbox
- MTC-353 — Office textbox
- 750-378-3355 — Phone textbox
- University of Main — College textbox
- amhamed@college.edu — Email textbox

Click on the Insert button to insert this new record into the database.

To validate this data insertion, go to the ComboName combobox control, and you can find that the new inserted faculty member Ali Mhamed is already there. Click on it to select this faculty and then click on the Select button to retrieve this new inserted record from the database and display it on this page. The inserted record is displayed on this page, which is shown in Figure 8.71.

Our data insertion to the Oracle database is successful. Click on the Back button and then on the Exit button to close our project. A complete Web application project named

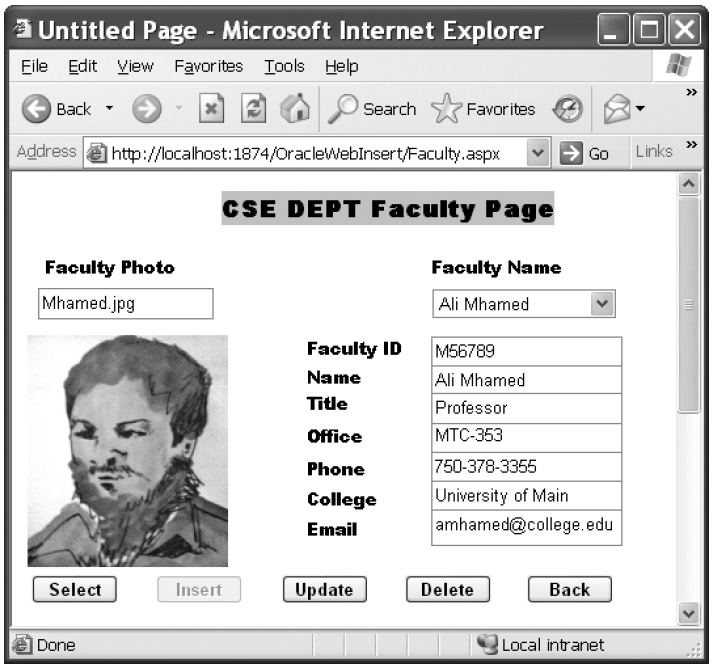

Figure 8.71 Data validation process.

OracleWebInsert can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

In the next section, we will discuss how to perform data updating and deleting in the Oracle database via the website.

8.9 DEVELOP ASP.NET WEB APPLICATION TO UPDATE AND DELETE DATA IN ORACLE DATABASES

Because of the coding similarity between the SQL Server and the Oracle databases, we only emphasize the important differences on the coding for these two databases. To save time and space, we want to modify an existing Web application project OracleWebInsert we developed in the previous section to make it as our new project OracleWebUpdateDelete. To do that, open Windows Explorer and create a new folder such as Chapter 8 if you have not created it. Copy the project OracleWebInsert from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1) and paste it in our folder C:\Chapter 8. Rename the project to OracleWebUpdateDelete.

We divide this section into two parts in terms of the coding function:

1. The first part is to modify the new project to perform data updating actions in the Oracle database.
2. The second part is to develop stored procedures to perform data deleting actions in the Oracle database.

Now let's start from the first part—modify the new project so it performs data updating in the Oracle database.

8.9.1 Modify Project to Perform Data Updating

Open the Visual Studio.NET and go to the File|Open Web Site menu item and browse to the folder C:\Chapter 8. Select our new project OracleWebUpdateDelete and then click on the Open button to open it.

The coding process to this page can be divided into the following two operations:

1. Create codes to the Update button's Click method.
2. Create a user-defined UpdateParameters() method.

Let's begin with the first modification.

8.9.1.1 Create Codes to Update Button Click Method

Open the Update button's Click method by double-clicking on the Update button from the Faculty Web form and enter the codes shown in Figure 8.72 to this method.

Let's take a closer look at this piece of code to see how it works.

A. An updating query string is declared first with the oldName as the name of the dynamic parameter. This is because when you want to update the faculty name, the original name stored in the combobox control ComboName becomes the old name, and we need to

Faculty ▼ cmdUpdate_Click() ▼

```
protected void cmdUpdate_Click(object sender, EventArgs e)
{
A       string cmdString = "UPDATE Faculty SET faculty_name=:name, office=:office, phone=:phone, " +
                           "college=:college, title=:title, email=:email WHERE (faculty_name =: oldName)";
B       OracleCommand oraCommand = new OracleCommand();
        string FacultyImage;
        int intUpdate = 0;
C       txtID.Text = string.Empty;
D       FacultyImage = txtPhoto.Text;                    //reserve the new inserted faculty photo location
E       if (FacultyImage == string.Empty)
            FacultyImage = "Default.jpg";
F       Application["FacultyImage"] = FacultyImage;      //reserve faculty image for validation
G       oraCommand.Connection = (OracleConnection)Application["oraConnection"];
        oraCommand.CommandType = CommandType.Text;
        oraCommand.CommandText = cmdString;
H       UpdateParameters(ref oraCommand);
I       intUpdate = oraCommand.ExecuteNonQuery();
J       oraCommand.Dispose();
K       ComboName.Items.Clear();
        UpdateFaculty();
L       txtPhoto.Text = string.Empty;                    //clean up the faculty photo box
M       if (intUpdate == 0)
            Response.Write("<script>alert('The data updating is failed')</script>");
}
```

Figure 8.72 Coding for the Update button's Click method.

distinguish this old name with the updated faculty name stored in the textbox control txtName.

B. All data objects used in this method are created here, and a local integer variable intUpdate is also created, which is used as a holder to keep the returned data value from the execution of the ExecutNonQuery() method later.

C. Before we can perform data updating, we need to clean up the Faculty ID textbox since we don't want to update this piece of information.

D. We need to reserve the faculty image file, that is, the location of this image file, and save it to a local variable since we need it in the data validation process later.

E. If the content of this faculty image file is empty, which means that the user did not want to insert a new faculty image with this data insertion, a default faculty image file is used since we do not want to keep our Faculty Photo box empty during the validation process.

F. This faculty image file is stored into an Application state function as a global variable, and it will be used later in the validation process.

G. The Command object is initialized with the Connection object, Command type, and Command text.

H. The user-defined UpdateParameters() method, whose detailed coding is shown in Figure 8.73, is called to assign all input parameters to the Command object.

I. The ExecuteNonQuery() method of the command class is called to execute the data updating operation. This method returns a feedback data to indicate whether this data updating is successful or not, and this returned datum is stored to the local integer variable intUpdate.

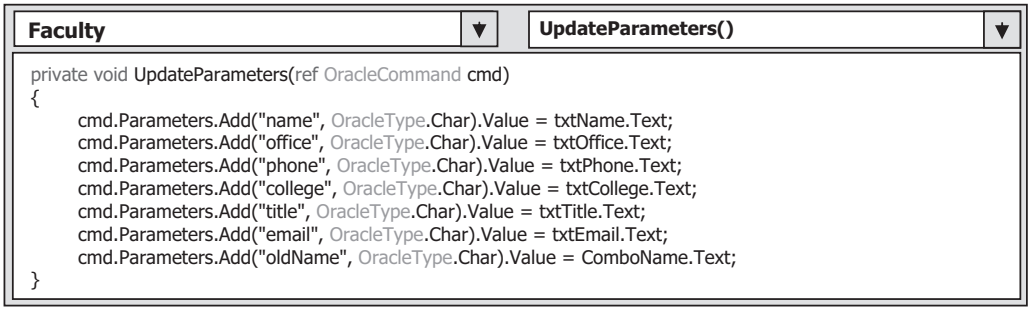

Figure 8.73 Coding for the user-defined UpdateParameters method.

J. A cleaning job is performed to release all data objects used in this method.

K. The combobox control ComboName is cleaned up and the user-defined UpdateFaculty() method is executed to refill that control to update the faculty members stored in that combobox.

L. The content of the Faculty Photo textbox is cleaned up to make it ready for that next data insertion action.

M. The data value returned from calling the ExecuteNonQuery() is exactly equal to the number of rows that have been successfully updated in the database. If this value is zero, which means that no row has been updated and this data updating has failed, a warning message is displayed for that situation. Otherwise if this value is nonzero, this data updating is successful.

The codes for the user-defined UpdateParameters() method are shown in Figure 8.73. The function of this piece of code is: Seven input parameters are assigned to the Parameters collection property of the Command object using the Add() method.

At this point we have finished all coding development for the data updating action in the Oracle database in the Faculty page. Before we can run the project to test this data updating function, make sure that the starting page is the LogIn page and a default faculty image file Default.jpg has been stored in our default folder. To check the starting page, right click on our project icon from the Solution Explorer window, select the Start Options item from the popup menu, and then check the Specific page radio button and select the LogIn.aspx as the starting page.

Now we can start to run the project to test the data updating function in the Faculty page in our sample Oracle database.

8.9.2 Develop Stored Procedures to Perform Data Deleting

As we discussed at the beginning of this section, to delete a record from a relational database, one must follow the correct sequence. In other words, one must first delete the records that are related to the record to be deleted in the parent table from the child tables. For example, in our application, to delete a record from the Faculty table, one must first delete the related records from the LogIn and the Course tables, and then one can delete the desired record from the Faculty table. The reason for that is because the `faculty_id` is a primary key in the Faculty table, but it is a foreign key in other tables.

Based on the analysis above, it can be seen that to delete one record from a parent table such as the Faculty table in our sample database, many deleting queries will be executed to first delete related records from the child tables such the LogIn and the Course, and then delete the target record from the parent table. Fortunately, we have set the Cascade mode for our data updating and deleting actions when we built our sample Oracle database CSE_DEPT in Chapter 2. In that case, we did not need to handle those multiple updating and deleting jobs ourselves. Instead the Oracle database engine can do that cascaded updating and deleting for us automatically. However, an easy and flexible way to perform these multiple deleting queries is to use the stored procedure to perform this data deleting.

8.9.2.1 Delete Existing Record from Faculty Table

Recall that in Section 7.8.3.2 in Chapter 7, we discussed how to develop a stored procedure in the Oracle database and use that stored procedure to perform the data deleting operation in the Oracle database. In this section we still want to use our Faculty table as an example to show how to delete an existing record from related tables.

In our sample database, there are two child tables related to our Faculty table, the LogIn table and the Course table. Two child tables are connected with the Faculty table by using the `faculty_id`, which is a primary key in the Faculty table and a foreign key in two child tables. To delete a faculty member from the parent table, or the Faculty table, one must first delete those records that are related to that faculty member in the parent table from the child table, such as from the LogIn and the Course tables, and then one can delete that faculty member from the Faculty table. Basically, this deleting can be divided into the following three steps or three queries:

1. Delete all records related to the faculty member to be deleted in the Faculty table from the LogIn table. In our sample database, only one row is related to each faculty member in the LogIn table.
2. Delete all records that are related to the faculty member to be deleted in the Faculty table from the Course table. In our sample database, there are about four to six records related to each faculty member in the Course table since each faculty can teach four to six courses.
3. Delete the faculty member from the parent or the Faculty table.

These three steps are exactly equivalent to three deleting queries, and we can combine these three queries into a single stored procedure. However, since we have set the Cascade mode for our data updating and deleting actions when we built our Oracle sample database in Chapter 2, we can use only one deleting query to perform this data deleting action. By calling and executing this stored procedure we can easily complete this data deleting operation. The complete data deleting operation can be divided into the following three steps:

1. Develop the stored procedure in the Oracle database to perform the data deleting function.
2. Call the stored procedure from the ASP.NET Web application to perform the data deleting in the Oracle database.
3. Validate the data deleting action after the data deleting operation.

To save time and space, we will not discuss how to create a stored procedure in the Oracle database environment to perform this data deleting operation in this section since we discussed this topic in detail in Section 7.8.3.2 in Chapter 7. Refer to that section to get more detailed materials about this issue. We will use the stored procedure DeleteFaculty_SP, which was developed in Section 7.8.3.2 in Chapter 7, to perform the data deleting action in this section.

Therefore, in the following part, we assume that we have finished developing the stored procedure DeleteFaculty_SP and we only take care of the coding for steps 2 and 3.

8.9.2.2 Develop Codes for Delete Button Click Method

Open the Delete button's Click method by double-clicking on the Delete button from the Faculty Web form window, and enter the codes shown in Figure 8.74 into this method.

Let's take a closer look at this piece of code to see how it works.

A. The content of the query string is now equal to the name of the stored procedure DeleteFaculty_SP that we created in the Oracle database in Section 7.8.3.2. Refer to that section to get the detailed codes for this stored procedure. When calling a stored procedure, the content of the query string must be equal to the name of the stored procedure.

B. The data object and the local variable used in this method are declared here. The integer variable intDelete is used to hold the returned value of executing the data deleting method ExecuteNonQuery() of the Command class later.

C. The Command object is initialized with the associated objects. The first object is the Connection object oraConnection that is stored in the Application state function. The second object is the Command type, and the StoredProcedure property must be assigned to this Command type property to make sure that the application will call a stored procedure as a query during the project runs.

D. The dynamic parameter is initialized with the real parameter, faculty name, which is stored in the combobox control ComboName. Note that you must use the faculty name stored in this combobox control, not the faculty name stored in the faculty name textbox control for this dynamic parameter since the latter is an updated faculty name, not an original faculty name.

Faculty | cmdDelete_Click()

```
protected void cmdDelete_Click(object sender, EventArgs e)
{
A       string cmdString = "DeleteFaculty_SP";
B       OracleCommand oraCommand = new OracleCommand();
        int intDelete = 0;
C       oraCommand.Connection = (OracleConnection)Application["oraConnection"];
        oraCommand.CommandType = CommandType.StoredProcedure;
        oraCommand.CommandText = cmdString;
D       oraCommand.Parameters.Add("FacultyName", OracleType.Char).Value = ComboName.Text;
E       intDelete = oraCommand.ExecuteNonQuery();
F       oraCommand.Dispose();
G       if (intDelete == 0)
           Response.Write("<script>alert('The data deleting is failed')</script>");
}
```

Figure 8.74 Coding for the Delete button's Click method.

E. The ExecuteNonQuery() method of the Command class is called to run the stored procedure to perform the data deleting operation. This method will return an integer to indicate whether this calling is successful or not.

F. A cleaning job is performed to release all objects used in this method.

G. The integer value returned from the calling of the ExecuteNonQuery() method is equal to the number of rows that have been successfully deleted from the database. If this value is zero, which means that no row has been deleted from the database and this data deleting has failed, a warning message is displayed to indicate this situation. Otherwise if this value is nonzero, at least one row has been deleted from the database and this data deleting is successful.

Now we have finished all coding developments for the data deleting action in the Oracle database for the Faculty page. Before we can run the project to test the data deleting function, make sure that:

- The starting page is the LogIn page.
- A default faculty photo file Default.jpg has been stored in our default folder in which our Web application project is located.

Now we can run the project to test the data deleting function. Click on the Start Debugging button to run the project. Enter the suitable username and password to the LogIn page, and select the Faculty Information from the Selection page to open the Faculty page. Then select the faculty member `Ying Bai` from the combobox control ComboName and click on the Select button to retrieve and display this faculty information in the Faculty page. Now click on the Delete button from this page to try to delete this record from the Faculty table in our sample database. Immediately you can find that all seven textboxes that contain the selected faculty information are cleaned up. Does that mean our data deleting successful? To answer this question, let's perform the following steps to confirm this data deleting action.

8.9.2.3 Validate Data Deleting Action

There are two ways to confirm data deleting. The first way is to try to retrieve this deleted faculty record from the database. Our data deleting would be successful if no such faculty information can be found and retrieved from the database. The second way is to open our sample database to check the associated tables to confirm this data deleting.

Let's do this confirmation using the first way. Still in the Faculty page, keep the faculty name `Ying Bai` selected in the combobox control ComboName and click on the Select button to retrieve this faculty information back from the database and display it in the Faculty page. A warning message "No matched faculty found!" is displayed, which means that this faculty record has been successfully deleted from the database.

Next let's open the Oracle database to check the associated tables to confirm this data deleting. Open the Oracle Database 10g XE home page by going to Start|All Programs| Oracle Database 10g Express Edition|Go To Database Home Page items. On the opened Login page, enter the username and password such as `CSE_DEPT` and `reback`, and then click on the Login button to open the Home page. Click on the drop-down arrow on the Object Browser icon and select the item Browse|Tables to open the Table page.

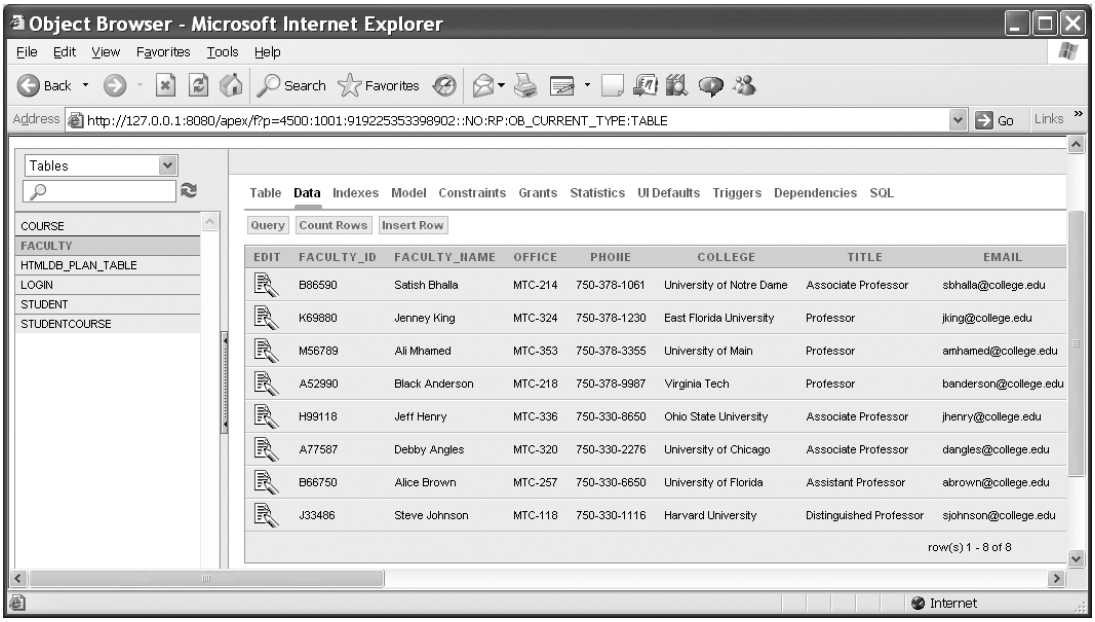

Figure 8.75 Faculty table after the data deleting.

On the opened Table page, click on the FACULTY table from the left pane, and then click on the Data tab to open this table, which is shown in Figure 8.75. You can find that the faculty member `Ying Bai` with the `faculty_id` = B78880 has been deleted from this Faculty table.

As we mentioned before, our sample database is a relational database, and the Faculty table has some relationships with other tables such as LogIn and the Course; that is, the Faculty table has some relationships with all four tables in our sample database, which include the Student and the StudentCourse tables. But at this moment, we only take care of the LogIn and the Course tables, and we will discuss the other two tables in the next section.

Open the LogIn and the Course tables by clicking on each of them one by one from the left pane, and you can find that those records related to the faculty member `Ying Bai` with a `faculty_id` = B78880 have been deleted from the LogIn and the Course tables. The relationship between the Faculty and the LogIn as well as between the Faculty and the Course is set up by the `faculty_id`, which is a primary key in the Faculty table and a foreign key in both LogIn and Course tables. This confirms that our data deleting is successful.

But the story is not finished. As you know, the Faculty table has some relationships with the other four tables in our sample database, which include the Student and the StudentCourse tables. To check this relationship, open the StudentCourse table. You can find that all courses related to (taught by) the faculty member `Ying Bai` have been deleted from this table, too! These courses include CSC-132B, CSC-234A, CSE-434, and CSE-438. That is not enough. Take a closer look at records in this table. You can find that the students who are identified by the `student_id` and took any of the four courses taught by the deleted faculty member `Ying Bai` have also been deleted from the

StudentCourse table! Why are those records deleted and who did it? To answer this question, we need to review the building process of our sample Oracle database. Recall that when we built our sample database in Chapter 2, we set up the relationships between four tables by using the foreign and the primary keys. Now let's take a closer look at those elements to try and answer this question in the next section.

8.9.2.4 Constraint Property—On Delete Cascade in Data Table

Recall that in Section 2.11.3 in Chapter 2, we used the constraint property to set up foreign keys and create relationships between tables. When we add a foreign key to a table, we need to indicate the Constraint Name and the Constraint Type. For example, to create a foreign key for the StudentCourse table and set up a relationship between the Course and the StudentCourse table, we selected the `course_id` as the primary key for the Course table and used it as a foreign key for the StudentCourse table. To create this foreign key to the StudentCourse table, the Constraint Name and the Constraint Type are:

- STUDENTCOURSE_COURSE_FK
- Foreign Key

The important point is that there is a checkbox named **On Delete Cascade**, which is located at the right of the Constraint Type textbox. We checked this checkbox when we created this foreign key for the StudentCourse table. To make this issue clear, we redisplay Figure 2.65 in Chapter 2, as Figure 8.76 in this section.

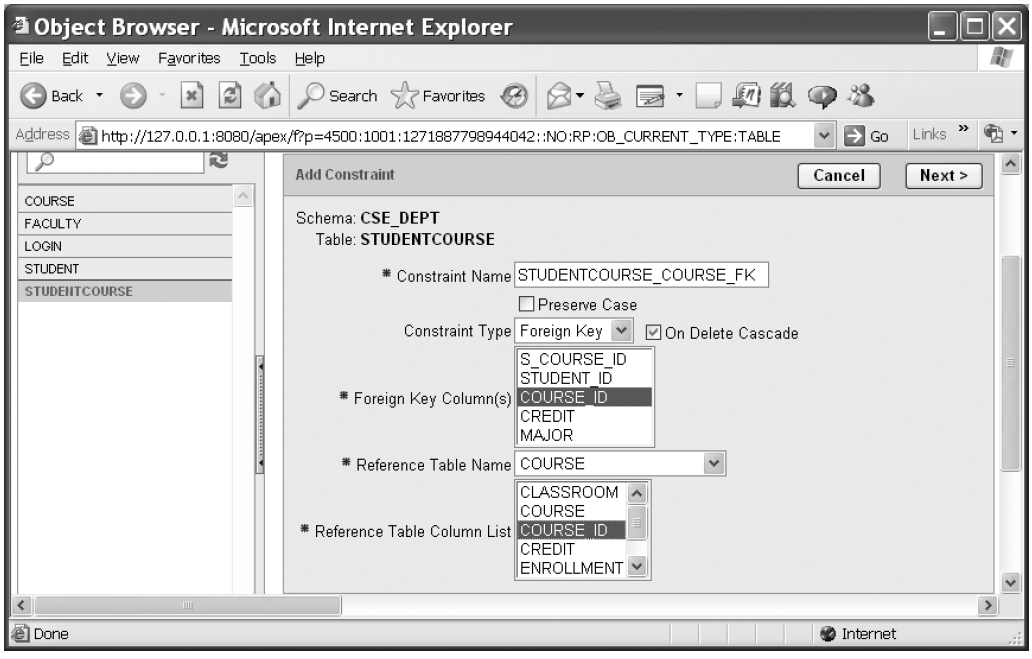

Figure 8.76 Create the foreign key between the StudentCourse and the Course table.

It can be found from this figure that the **On Delete Cascade** checkbox is checked. This means that all records related to this foreign key `course_id` in this StudentCourse table will be deleted if the primary key, which is the course_id in the Course table, is deleted. This is the meaning of the so-called cascaded deleting or **On Delete Cascade**. The word *cascade* means series, and *cascaded deleting* means if records that contain a primary key in a table (parent table) are deleted, all related records that have the same foreign key in all other tables would also be serially deleted.

Now we can answer the question we asked in the last section. All students who are identified by the associated `student_id` and took any of the four courses taught by the deleted faculty member `Ying Bai` have also been deleted from the StudentCourse table. The reason for that is because of the `course_id`, which is a primary key in the Course table but a foreign key in the StudentCourse table. Since the checkbox **On Delete Cascade** was checked when we set up the relationship between these two tables, all records related to this foreign key `course_id` in the StudentCourse table will be serially deleted by the database engine if the records that contain the primary key `course_id` in the Course table are deleted. The `faculty_id` in the Course table is a foreign key, and when any of the four courses that are identified by their `course_id` and taught by the faculty member `Ying Bai` are to be deleted from the Course table, all records related to that `course_id`, which is a foreign key in the StudentCourse table, will also be deleted since the `course_id` is a primary key in the Course table. It is the Oracle database engine that performed this cascaded or series data deleting if this checkbox **On Delete Cascade** was checked when the relationship was set up among the tables. Similar things happened to the `student_id`, which is also a foreign key in the StudentCourse table.

Before we can close the Oracle database 10g XE, it is highly recommended to recover all deleted records to the associated tables. Refer to Tables 8.13 to 8.16 to add those deleted records back to the associated tables. Now you can close the Oracle Database 10g XE.

Table 8.13 Data to Be Added in the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu

Table 8.14 Data to Be Added in the LogIn Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	

Table 8.15 Data to Be Added in the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880

Table 8.16 Data to Be Added in the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1005	J77896	CSC-234A	3	CS/IS
1009	A78835	CSE-434	3	CE
1014	A78835	CSE-438	3	CE
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE

A complete Web application project OracleWebUpdateDelete can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

8.10 CHAPTER SUMMARY

A detailed and completed introduction to the ASP.NET and the .NET Framework, including the ASP.NET 3.5, is provided at the beginning of this chapter. This part is especially useful and important for readers who do not have any knowledge or background in the Web application developments and implementations.

Following the introduction section, a detailed discussion on how to install and configure the environment to develop the ASP.NET Web applications is provided. Some essential tools such as the Web server, IIS, and FrontPage Server Extension 2000, as well as the installation process of these tools, are introduced and discussed in detail.

Starting in Section 8.3, detailed developments and building processes of ASP.NET Web applications to access databases are discussed with seven real Web application projects. Two popular databases, SQL Server and Oracle, are utilized as the target databases for those projects. The seven real ASP.NET Web application projects include:

1. ASP.NET Web application to select and display data from the Microsoft SQL Server database
2. ASP.NET Web application to insert data into the Microsoft SQL Server database
3. ASP.NET Web application to update and delete data in the Microsoft SQL Server database
4. ASP.NET Web application to perform data actions with LINQ to SQL query
5. ASP.NET Web application to select and display data from the Oracle database
6. ASP.NET Web application to insert data into the Oracle database
7. ASP.NET Web application to update and delete data in the Oracle database

The stored procedures are utilized in the last project to help readers to perform the data deleting action against our sample databases more efficiently and conveniently. The detailed discussion on the data deleting order is provided to help readers to understand the integrity constraint built in the relational database. It is a tough topic to update or delete data from related tables in a relational database, and a clear and deep discussion on this topic will significantly benefit readers and improve their knowledge and hands-on experience on these issues.

HOMEWORK

I. True/False Selections

____**1.** The actual language used in the communication between the client and the server is HTML.

____**2.** ASP.NET and .NET Framework are two different models that provide the development environments to the Web programming.

____**3.** The .NET Framework is composed of the Common Language Runtime (called runtime) and a collection of class libraries.

____**4.** You access the .NET Framework by using the class libraries provided by the .NET Framework, and you implement the .NET Framework by using the tools such as Visual Studio.NET provided by the .NET Framework, too.

____**5.** ASP.NET 3.5 is a programming framework built on the .NET Framework 3.5 and it is used to build Web applications.

____**6.** The fundamental component of ASP.NET is the Web Form. A Web Form is the Web page that users view in a browser, and an ASP.NET Web application can contain one or more Web Forms.

____**7.** A Web Form is a dynamic page that runs on the server side, and it can access server resources when it is viewed by users via the client browser.

____**8.** Similar to traditional Web pages, an ASP.NET 3.5 Web page can only run scripts on the client side.

____**9.** The controls you added to the Web form will run on the Web server when this Web page is requested by the user through a client browser.

___**10.** To allow a listbox control to response to a user click as the Web page runs, the AutoPostBack property of that listbox must be set to False.

II. Multiple Choices

1. When the user sends a request from the user's client browser to request a Web page, the server needs to build that form and sends it back to the user's browser in the _____ language format.

a. ASP.NET
b. .NET Framework
c. XML
d. HTML

2. Once a requested Web page is received by the client's browser, the connection between the client and the server is _____.

a. Still active
b. Terminated
c. Not active
d. Either active or inactive

3. As a Web application runs, the programs developed in any .NET-based language are converted into the _____ codes that can be recognized by the CLR, and the CLR can compile and execute the MSIL codes by using the Just-In-Time compiler.

a. Visual Studio.NET
b. Visual Basic.NET

c. Microsoft Intermediate Language (MSIL)

d. C#

4. The terminal file of an ASP.NET Web application is a _____ file.

a. Dynamic Linked Library (dll)

b. MSIL

c. XML

d. HTML

5. Because Web pages are frequently refreshed by the server, one must use the _____ to store the global variable.

a. Global.asax file

b. Defaulty.aspx file

c. Config file

d. Application state function

6. One needs to use the _____ method to display a message box in Web applications.

a. MessageBox.Show()

b. MessageBox.Display

c. Java script alert()

d. Response.Write()

7. Unlike the Windows-based applications that use the Form_Load as the first event method, a Web-based application uses the _____ as the first event method.

a. Start_Page

b. Page_Load

c. First_Page

d. Web_Start

8. To delete data from a relational database, one must first delete the data from the _____ tables, and then one can delete the target data from the _____ table.

a. Major, minor

b. Parent, child

c. Parent, parent

d. Child, parent

9. To allow the SQL Server database engine to delete all related records from the child tables, the Delete Rule item in the INSERT And UPDATE Specifications box of the Foreign Key Relationship dialog box must be set to _____.

a. No action

b. Cascade

c. Default

d. Null

10. To display any message on a running Web page, one must use the _____ method.

a. MessageBox.Show()

b. Response()

c. Response.Redirect()

d. Response.Write()

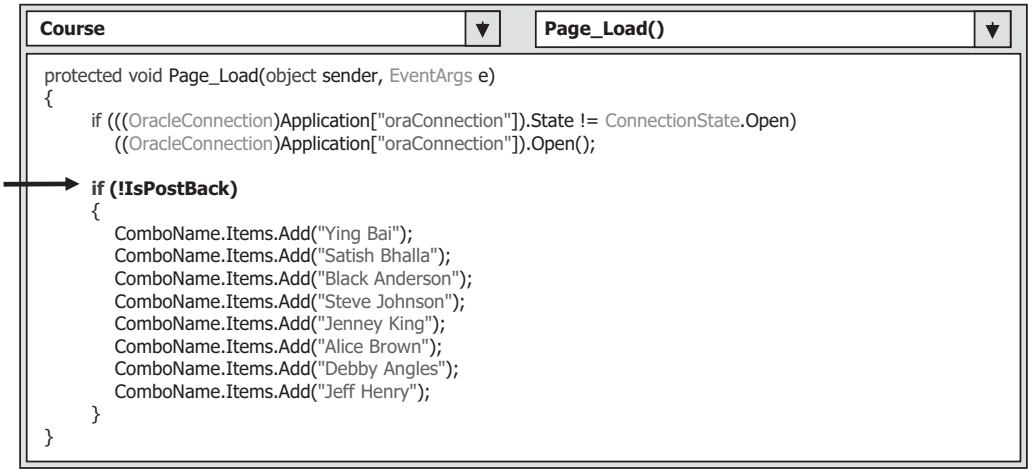

Figure 8.77 Codes for the Page_Load method.

III. Exercises?

1. Write a paragraph to answer and explain the following questions:

- **a.** What is ASP.NET?
- **b.** What is the main component of the ASP.NET Web application?
- **c.** How is an ASP.NET Web application executed?

2. Suppose we want to delete one record from the Student table in our sample database CSE_DEPT based on one student_id = "H10210". List all deleting steps and deleting queries including data deleting from the child and the parent tables.

3. Figure 8.77 shows a piece of code developed in the Page_Load() method. Explain the functionality of the statement **if (!IsPostBack)** block.

4. Add a Web page and develop codes to perform data deleting for the Student form page in the SQLWebUpdateDelete project (the project file can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1)).

5. Add a Web page UpdateCourse and develop the codes to perform the course updating actions for the Course Form page in the project OracleWebUpdateDelete (the project can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1)).

6. Develop a Web page called Student.aspx and create a stored procedure to delete one record from the Student table by using the project OracleWebUpdateDelete (the project can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1)). It is highly recommended to recover those deleted records after they are deleted.