

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

ASP.NET WEB PAGES USING C#

ASP.NET Web Pages Using C#

Student Guide

Revision 1.0

ASP.NET Web Pages Using C#

Rev. 1.0

Student Guide

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Object Innovations.

Product and company names mentioned herein are the trademarks or registered trademarks of their respective owners.



™ is a trademark of Object Innovations.

Author: Robert J. Oberg

Copyright ©2016 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations
877-558-7246
www.objectinnovations.com

Printed in the United States of America.

Table of Contents (Overview)

Chapter 1	Introduction to Web Pages and WebMatrix
Chapter 2	Programming with C# and Razor
Chapter 3	Form Processing and HTTP Basics
Chapter 4	State Management and ASP.NET Infrastructure
Chapter 5	Web Database Applications
Chapter 6	ASP.NET and Azure
Appendix A	Learning Resources
Appendix B	HTTP and Fiddler

Directory Structure

- **The course software installs to the root directory *C:\OIC\AspWpCs*.**
 - Example programs for each chapter are in named subdirectories of chapter directories **Chap01**, **Chap02**, and so on.
 - The **Labs** directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.
- **Data files install to the directory *C:\OIC\Data*.**

Table of Contents (Detailed)

Chapter 1 Introduction to Web Pages and WebMatrix.....	1
Web Application Fundamentals.....	3
WebMatrix	4
A Simple HTML Page	5
Template Gallery	6
WebMatrix Main Window	7
An HTML File	8
HTML Page in Browser.....	10
Adding a New File.....	11
Styles.....	13
Cascading Style Sheet.....	14
Dynamic Web Sites.....	16
Dynamic Web Site Example.....	17
Starter Files	18
Welcome Page	19
Testing the Page.....	20
C# and Razor.....	21
IIS Express	22
favicon.ico.....	23
ASP.NET History	24
ASP.NET Web Forms.....	25
ASP.NET MVC	26
ASP.NET Web Pages.....	27
Lab 1	28
Summary	29
Chapter 2 Programming with C# and Razor	33
Dynamic Websites	35
Razor View Engine.....	36
Demo: Dynamic Output with ASPX.....	37
ASPX Embedded Content.....	39
Razor Embedded Content	40
Intermixed HTML and C# Code.....	41
Intermixed Code Example	42
C# Code Block.....	43
Code Block and Intermixed HTML.....	44
Lab 2A	45
Reusable Code in Web Pages	46
Leap Year Function	47
Test Page.....	48
Another Leap Year Function	49
Helper for an Unordered List.....	50

Lab 2B.....	51
Using .NET Classes	52
Class Example.....	53
Using .NET Class Libraries	55
Class Library Example.....	56
Summary	59
Chapter 3 Form Processing and HTTP Basics	67
HTML Forms	69
A Simple Form.....	70
HTML Markup for the Form	71
Razor Code in the Form.....	72
Query String.....	73
HTTP.....	74
HTTP Requests and Responses	75
HTTP Request Line	76
Request Header Fields	77
Internet Media Types.....	78
Media Type in HTTP.....	79
HTTP Methods.....	80
POST Example.....	81
Request Object.....	82
Lab 3A	83
Using JavaScript	84
Addition Page with JavaScript.....	85
More about Forms.....	86
Labels.....	87
Radio Buttons	88
Checkboxes	89
Checkboxes in a Group.....	90
Fieldset.....	91
Razor Code for Input Example	92
Styling	93
Select.....	94
HTML Markup for Select	95
Razor Code for Select Example.....	96
Lab 3B.....	97
A Form with Multiple Buttons.....	98
Validation.....	99
Validation Helper.....	100
Validation Example	101
Example without Validation	102
Razor Code.....	103
Basic Validation – Step 1.....	104
Running Step 1.....	106
Remembering Data Input.....	107

Checked Helper.....	108
Running Step 2.....	109
Field Error Messages	110
Styling the Error Messages	111
Lab 3C.....	112
Summary	113
Chapter 4 State Management and ASP.NET Infrastructure	127
ASP.NET Infrastructure.....	129
Session and Application State.....	131
Session State Demo.....	132
Global.asax	136
Session Object.....	137
Logging In.....	138
Default Page – UserName.....	139
Default Page – ItemList	140
Session Variable Issues.....	141
Session State and Cookies	142
Session State Timeout.....	143
Session State Store.....	144
Application State.....	145
Implementing Application State.....	146
Global.asax	147
Users.cshtml.....	148
Login.cshtml	149
Bouncing the Web Server	150
Cookies	151
Cookies and ASP.NET.....	152
HttpCookie Properties.....	153
Example – Exposing Cookies	154
Delete Cookie – Code	160
Caching	161
Output Caching	162
Simple Output Caching Example.....	163
Lab 4	165
Summary	166
Chapter 5 Web Database Applications	173
Creating a Database	175
Table Designer.....	177
Entering Data	178
Files Workspace.....	179
Displaying the Data	180
Database Class	181
DynamicRecord Class.....	182
WebGrid Helper.....	183

WebGrid.GetHtml Method	185
Grid Columns.....	186
Table Style	187
Lab 5A	188
Entering Data	189
Input Form for a New Employee	190
Inserting Data into Database.....	191
Validation and Error Handling	192
Database Schema Errors	193
Validation of Input Data	194
Network Errors	195
Database Errors.....	196
Exception Handling	197
Running the Application.....	198
Updating a Record	199
Querying for a Single Row	200
Update Link in WebGrid	201
Markup for the Table.....	202
WebGrid Columns	203
Custom Display for a Grid Column.....	204
Database Update Code.....	205
Error Handling	206
Remembering the Employee Id.....	207
Deleting a Record	208
Delete Page	209
Initializing Employee Data	210
Running the Application.....	211
Database Delete Code.....	212
Lab 5B.....	213
Connecting to Other Databases.....	214
SmallPub Database	215
DataDemo Example	217
Performing a Join.....	218
List of Books.....	219
Summary	220
Chapter 6 ASP.NET and Azure.....	235
What Is Windows Azure?	237
A Windows Azure Testbed.....	238
Windows Azure Demo.....	239
Home Page	241
Publishing to Azure	242
Signing In.....	243
Updating a Site to Azure.....	247
Configuration File Change.....	249
Uploading the Database	250

Book Site on Azure	251
Lab 6	252
Summary	253
Appendix A Learning Resources	257
Appendix B HTTP and Fiddler	261
Using Fiddler	263
Examining GET Requests	264
Examining POST in Fiddler.....	268
Request Body for POST.....	269
Summary	270

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Chapter 1

Introduction to Web Pages and WebMatrix

Introduction to Web Pages and WebMatrix

Objectives

After completing this unit you will be able to:

- **Review the fundamentals of Web applications and set up a testbed using WebMatrix.**
- **Develop static websites using HTML and CSS with WebMatrix as an editor.**
- **Test your websites using the Web server that is integrated with WebMatrix.**
- **Describe dynamic Web applications with content generated on the server.**
- **Explain the benefits of ASP.NET.**
- **Describe the programming models provided by ASP.NET: Web Forms, MVC and Web Pages.**
- **Discuss the advantages and drawbacks of these different ASP.NET programming models.**
- **Create a simple ASP.NET Web Pages dynamic website using WebMatrix.**

Web Application Fundamentals

- **A Web application consists of document and code pages in various formats.**
- **The simplest kind of document is a static HTML page, which contains information that will be formatted and displayed by a Web browser.**
 - An HTML page may also contain hyperlinks to other HTML pages.
 - A hyperlink (or just “link”) contains an address, or a Uniform Resource Locator (URL), specifying where the target document is located.
- **The resulting combination of content and links is sometimes called “hypertext” and provides easy navigation to a vast amount of information on the World Wide Web.**

WebMatrix

- **WebMatrix 3 is the latest version of a free Microsoft tool for creating, testing and publishing websites¹.**
- **You can download and install WebMatrix from this link:**

<https://www.microsoft.com/web/webmatrix/wmx3features.aspx>

- **WebMatrix includes many features:**
 - An editor for many different types of Web pages
 - IIS Express for testing websites
 - Microsoft SQL Server Compact Edition (SQL CE) for database development
 - NuGet to obtain and install packages for added functionality
 - Support for many programming languages, including C#, Visual Basic, PHP, and others
 - Easy-to-use publishing to the Internet, including to the Azure cloud.

¹ Microsoft has recently introduced a free cross-platform tool Visual Studio Code that runs on OS X, Linux and Windows. This course uses the classic WebMatrix tool.

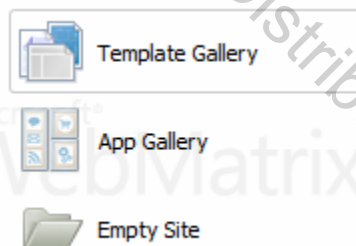
A Simple HTML Page

- **Let's begin by creating a very simple HTML personal home page.**

1. Start WebMatrix.



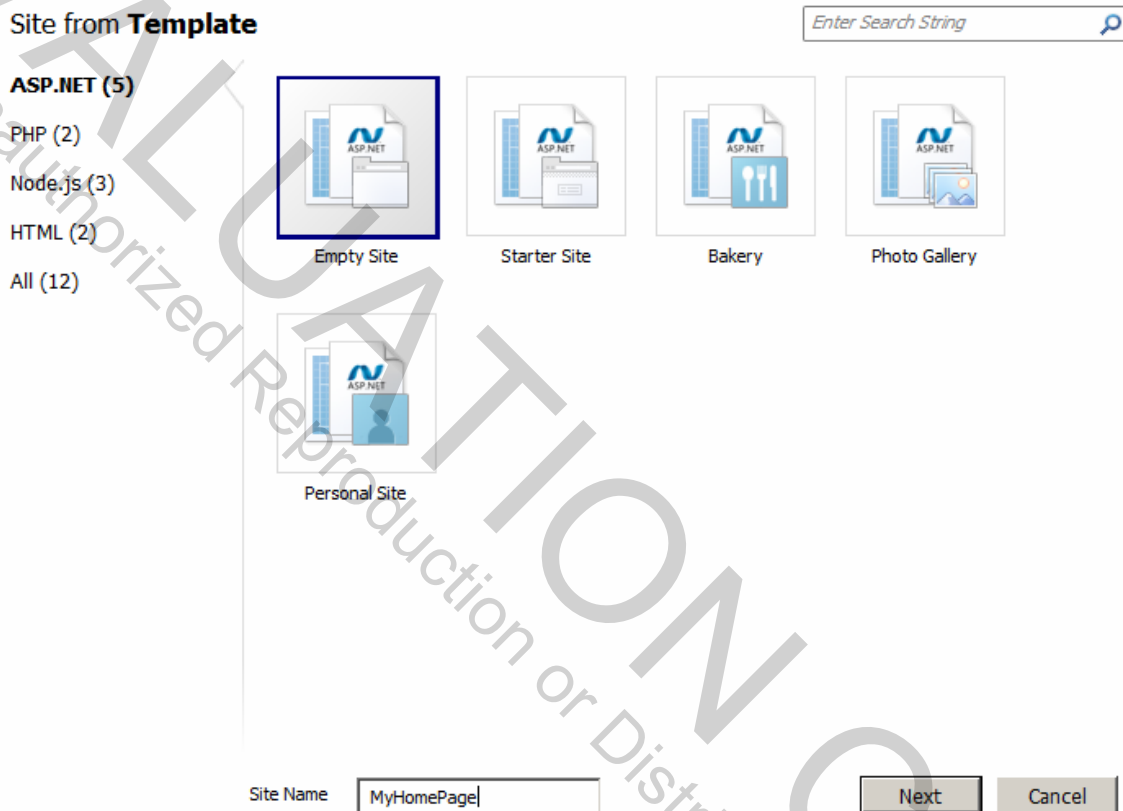
2. Click the icon for New.



3. Select Template Gallery.

Template Gallery

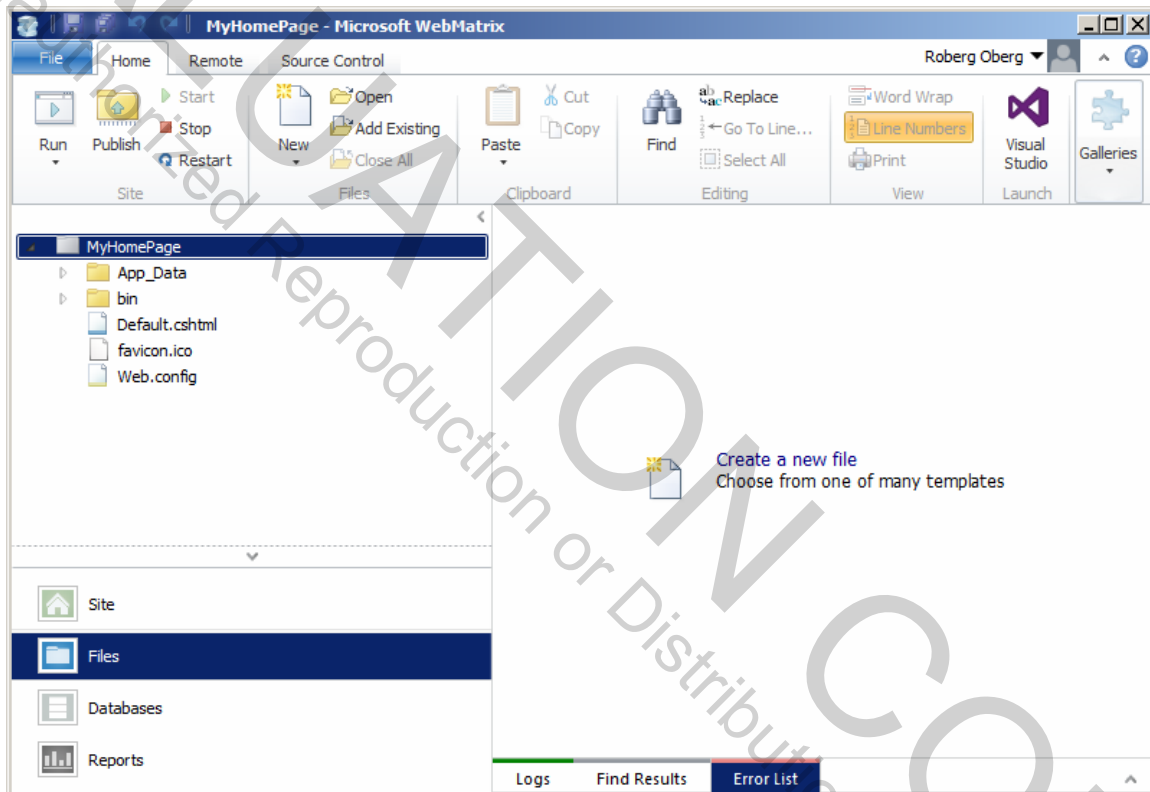
4. A list of templates will be displayed. By default only the ASP.NET templates will be shown.



5. Choose the Empty Site and enter **MyHomePage** as the Site Name. Click Next.
6. You will be offered the opportunity to create a site on Windows Azure. Click Skip. (We will discuss deployment to Azure later.)

WebMatrix Main Window

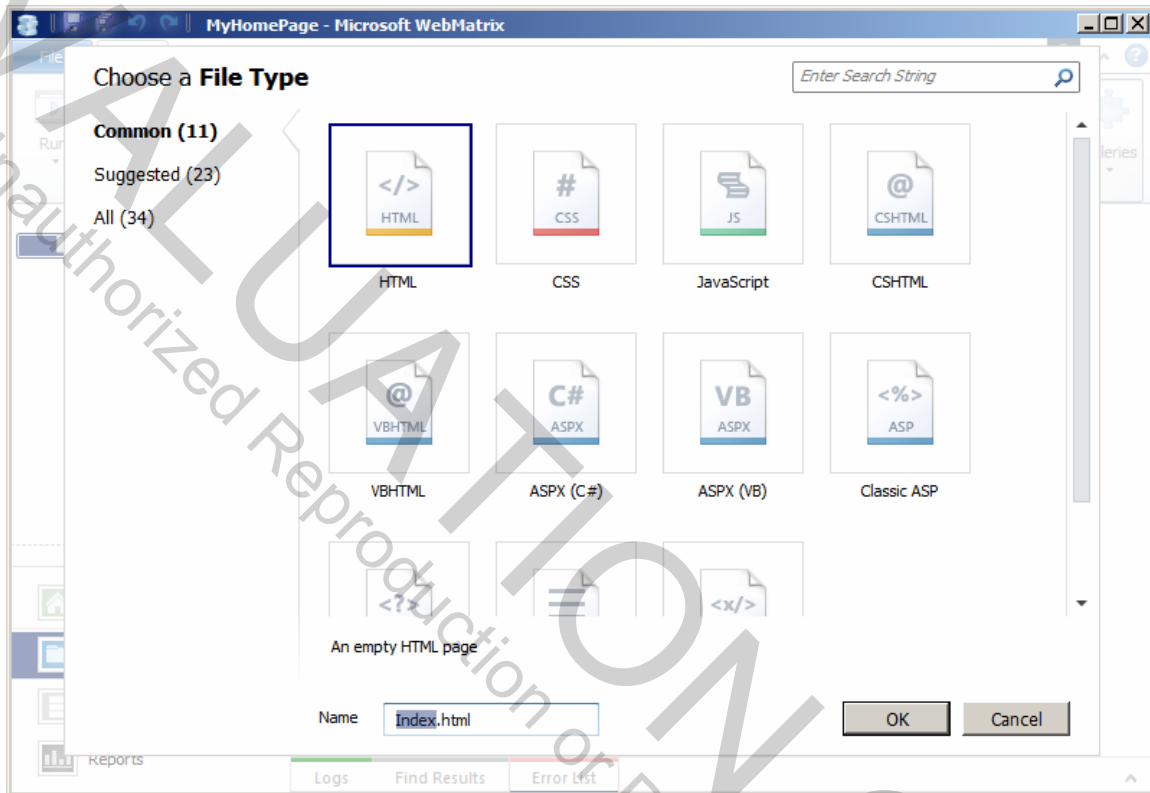
7. The WebMatrix main window will now be displayed you're your **MyHomePage** site. By default WebMatrix opens with the Files options and corresponding workspace, with a ribbon at the top. The user interface is quite intuitive and should be easy to navigate with a little practice.



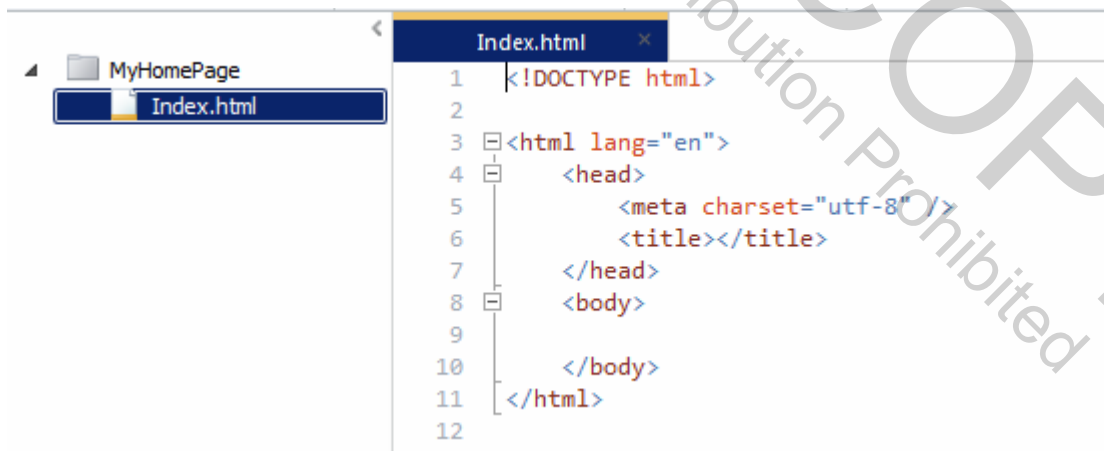
8. Delete the starting folders and files, so that you will begin with a completely empty site.
9. Click Create a new file. You could also use the New button on the ribbon.

An HTML File

10. Select HTML file type and assign the name **Index.html**.



11. Click OK. You will be shown a basic HTML starter page.



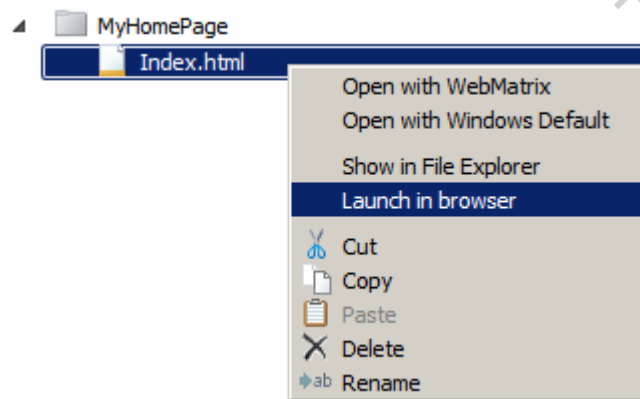
An HTML File (Cont'd)

12. Type in the following simple HTML. Notice that the editor provides IntelliSense and does automatic indenting.

```
<!DOCTYPE html>

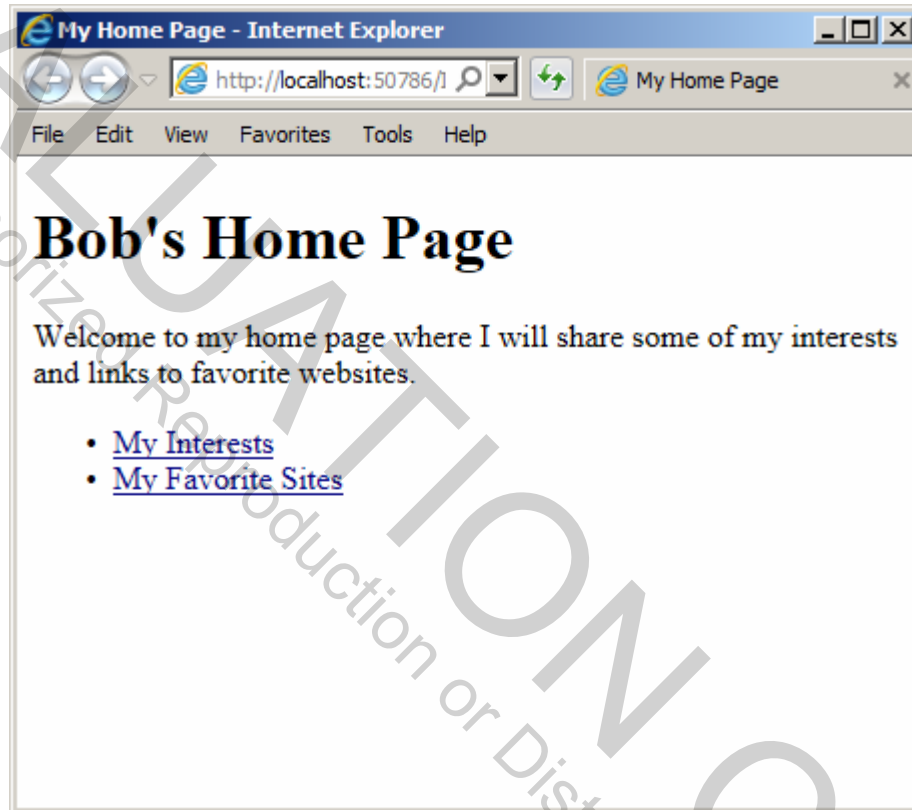
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>My Home Page</title>
  </head>
  <body>
    <h1>Bob's Home Page</h1>
    <p>Welcome to my home page where I will
      share some of my interests and links
      to favorite websites.
    </p>
    <ul>
      <li><a href="Interests.html">
        My Interests</a></li>
      <li><a href="Favorites.html">
        My Favorite Sites</a></li>
    </ul>
  </body>
</html>
```

13. Right-click over Index.html to launch in browser.



HTML Page in Browser

14. Your default browser will launch and display the selected page. This process will automatically save the files in your site.



15. If you click on the links, naturally you will get HTTP Not Found errors, because the corresponding pages have not yet been created. Notice that we used relative paths in specifying the links.

```
<a href="Interests.html">My Interests</a>
```

Adding a New File

16. Close the browser window. Back in WebMatrix click New on the ribbon and select New File.

17. Again select HTML as the file type. Enter **Interests.html** as the name of the new file.

18. Enter this simple HTML.

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Interests</title>
  </head>
  <body>
    <h1>My Interests</h1>
    <ul>
      <li>Programming</li>
      <li>Movies</li>
      <li>Reading</li>
      <li>Hiking</li>
      <li>Environment</li>
    </ul>
  </body>
</html>
```

19. You can test the new page by launching it directly in the browser. You can test that the link on the home page works by launching **Index.html** in the browser.

Another New File

20. In a similar manner create the file **Favorites.html** and enter HTML establishing a list of links to some websites. In this case use absolute path names for the URLs. (We have removed some of the indenting to more clearly show the HTML text.)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Favorites</title>
</head>
<body>
<h1>My Favorite Websites</h1>
<ul>
  <li><a href= "http://www.objectinnovations.com">
    Object Innovations</a></li>
  <li><a href="http://www.netflix.com">
    Netflix</a></li>
  <li><a href="http://www.thinkglobalgreen.org">
    Think Global Green</a></li>
</ul>
</body>
</html>
```

21. Test the whole site, beginning with **Index.html**. A copy of the site at this point is saved in **MyHomePage\Step1** in the **Chap01** folder.

- **By default WebMatrix will save a new website you create in the folder *My Web Sites* in *My Documents* in the *Documents* library.**
 - We will refer to this folder simply as **My Web Sites**.

Styles

- **An important aspect of modern website design is to separate specification of the structure of a web page from its visual appearance.**

- The structure is concerned with headers, paragraphs, lists, and so on. It is specified by HTML tags.
- The visual appearance is concerned with things like fonts, colors, layout, and so on. It is specified by styles.

- **Styles can be applied at three levels:**

- Inline styles apply to a single tag. For example:

```
<li style="color: green">Environment</li>
```

- Embedded styles apply to a single page, specified by a `<style>` tag within the `<head>` section of the document.

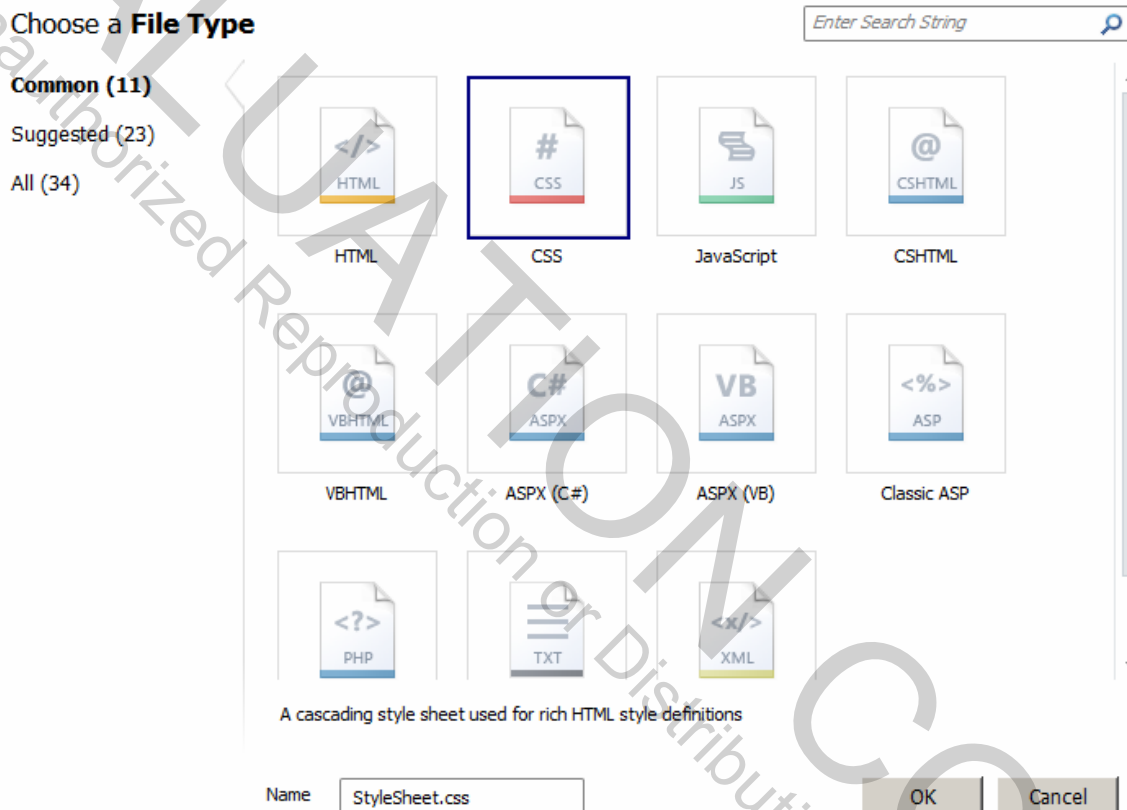
```
<style>  
  li {font-family: Arial, sans-serif}  
</style>
```

- External styles apply to a whole site (or a section of a site). They are specified in a **cascading style sheet**, which is a file with the extension **.css**.

- **We applied the inline and embedded illustrated styles to *Interests.html*.**
- **We will create an external style sheet.**

Cascading Style Sheet

- **Let's create a style sheet for our simple website.**
 - In WebMatrix add a new file. This time select the CSS style type. Go with the default file name **StyleSheet.css**.



- Simple starter code is supplied for styling the body. Specify a background color of lightgray.

```
body { background-color: lightgray
}
```

- Finally, apply a <link> element in the <head> section of each page to link to the style sheet.

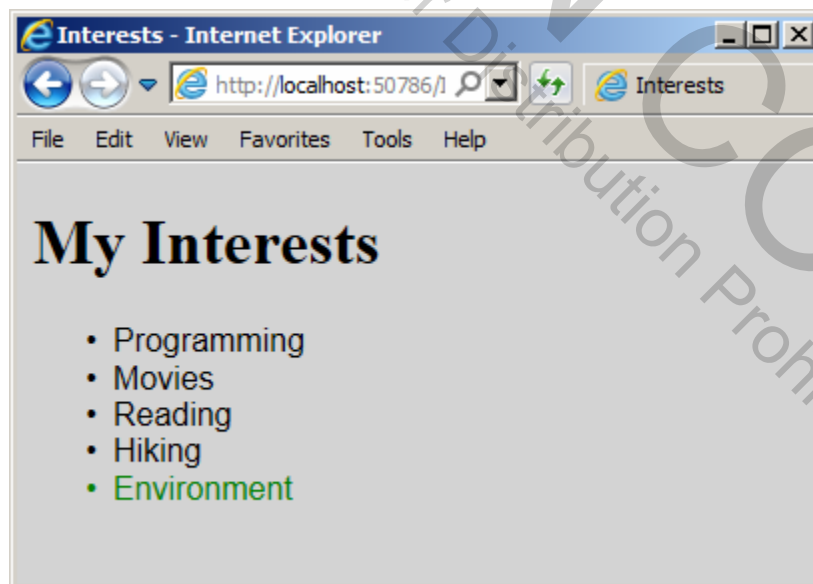
```
<link href="StyleSheet.css" rel="stylesheet"
      type="text/css" />
```

Testing the Styles

- To see the results of specifying these styles, launch the browser for the index page.



- Click on the first link.



- The site at this point is saved in **MyHomePage\Step2** in the chapter folder.

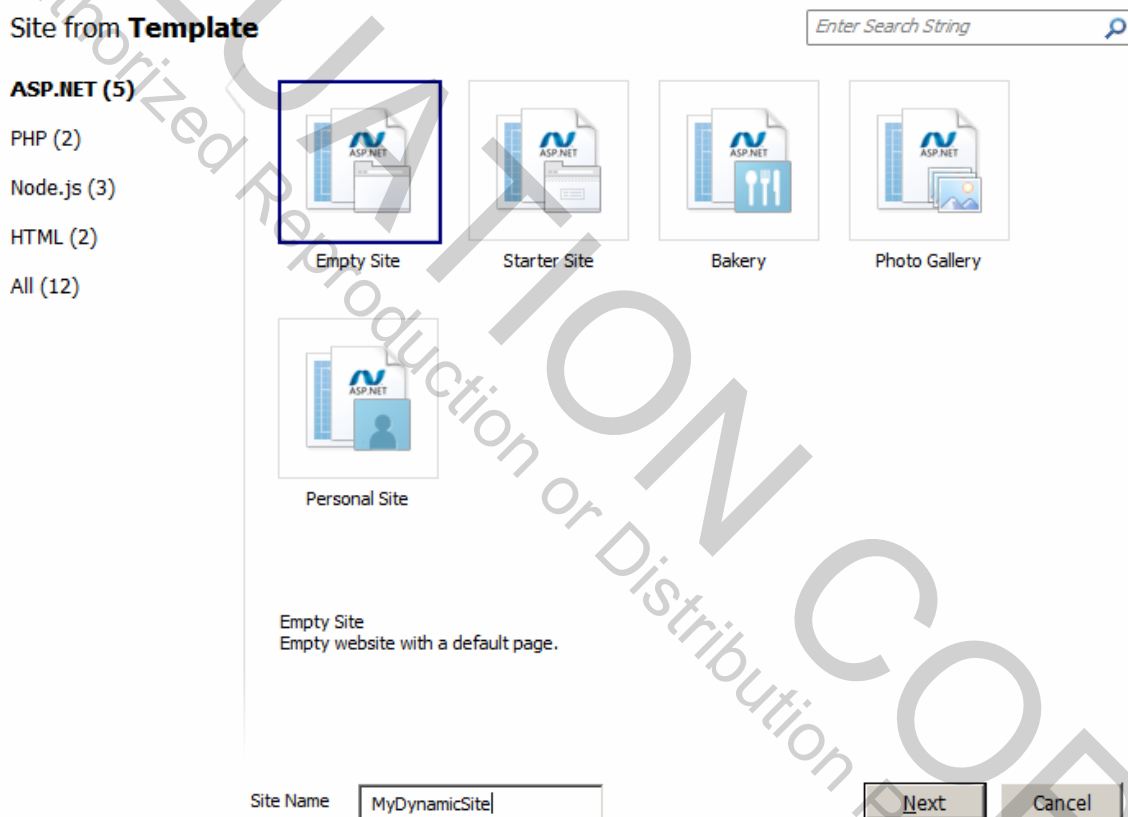
Dynamic Web Sites

- **In a dynamic web site some of the content is generated at the time a page is requested.**
- **There are two approaches to adding dynamic content to websites.**
- **One approach is via client-side technologies, such as the DOM (Document Object Model) and a scripting language such as JavaScript.**
 - WebMatrix supports this approach. You may have noticed JavaScript (.js) as the third type of file that can be chosen when adding a new file to your site.
- **The other approach is via server-side technologies, such as:**
 - PHP
 - ASP (Active Server Pages)
 - ASP.NET
- **WebMatrix supports all of these.**
- **In this course we will focus on server-side dynamic web sites using ASP.NET Web Pages.**
 - We will also briefly compare Web Pages with two other ASP.NET technologies: Web Forms and MVC.

Dynamic Web Site Example

- **Let's create a brand new website to provide a simple illustration of a dynamic website.**

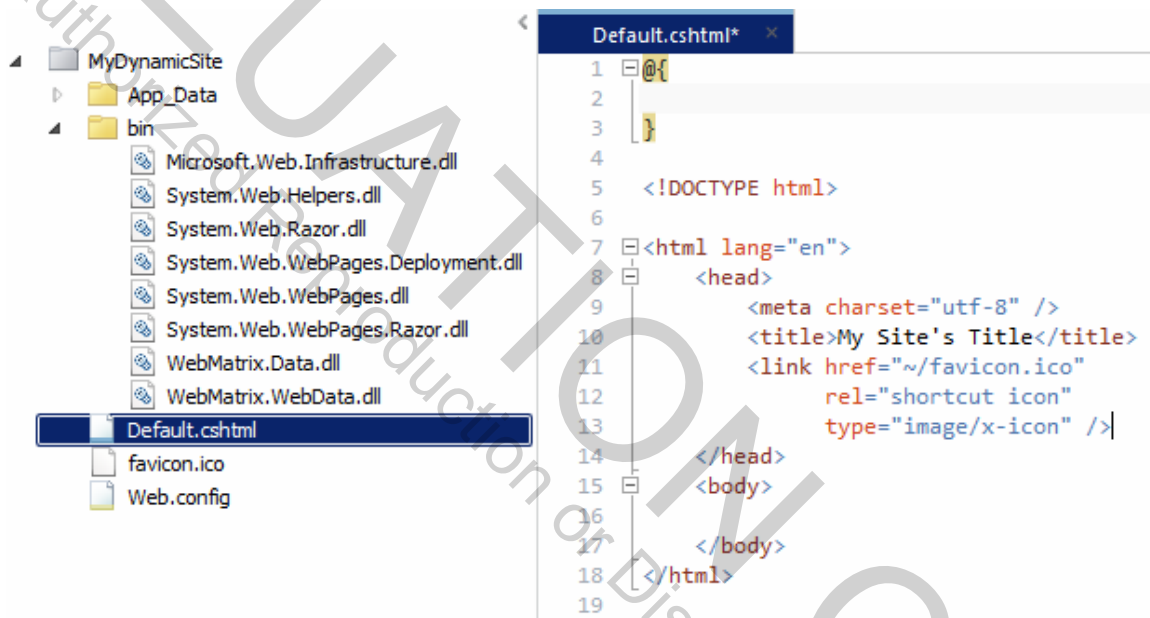
1. In WebMatrix create a new site. As before use the Empty Site template from the ASP.NET category in the Template Gallery. For site name use **MyDynamicSite**.



2. Click Next, and again skip over creating a Windows Azure site.
3. The main window for WebMatrix will open up. This time don't delete the starter files.

Starter Files

4. Expand the **bin** folder. This contains DLLs that are provided by default to support dynamic content.
5. Examine the starter file **Default.cshtml**. This is like a new HTML file, except there is space for a code block at the top of the file.

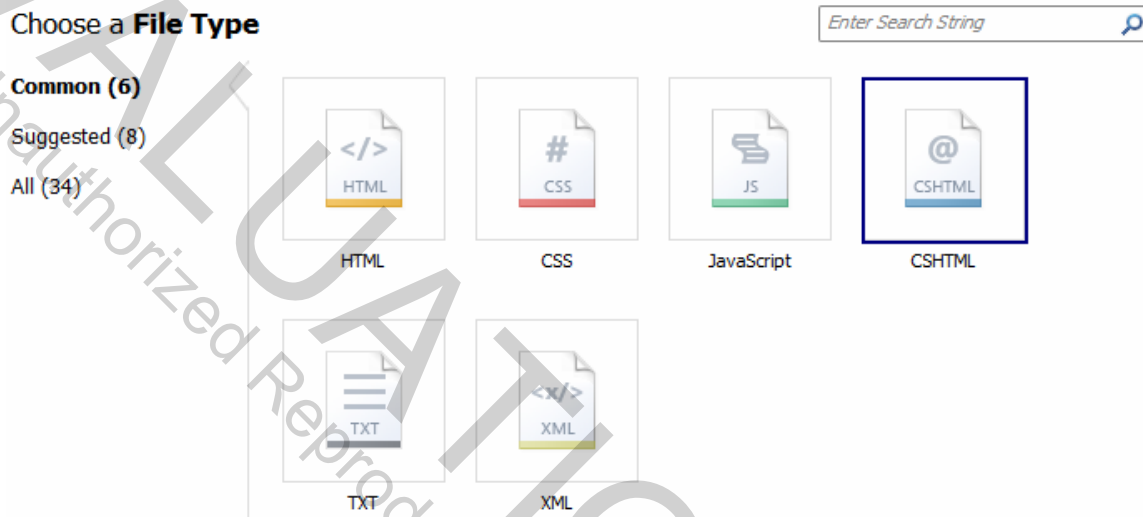


6. In this example, provide only one link on the home page.

```
...
<html lang="en">
...
  <body>
    <h1>My Dynamic Site</h1>
    <ul>
      <li><a href="Welcome">Welcome</a></li>
    </ul>
  </body>
</html>
```

Welcome Page

7. Add a new file to your site. This time choose CSHTML as the file type. Assign the name **Welcome.cshtml** to the new file.



8. Provide the following code and markup on the new page.

```
@{
    string name = "Bob";
    var now = DateTime.Now;
}

<!DOCTYPE html>

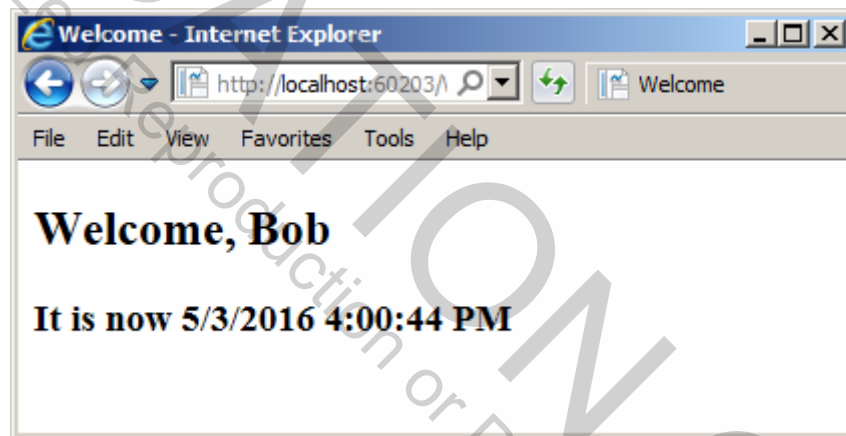
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Welcome</title>
  </head>
  <body>
    <h2>Welcome, @name</h2>
    <h3>It is now @now</h3>
  </body>
</html>
```

Testing the Page

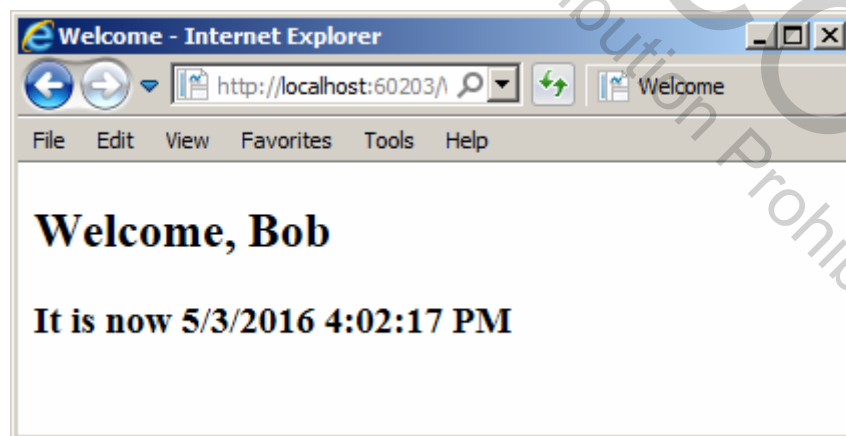
9. You can either launch this page directly in the browser, or you can go through **Default.cshtml**. Let's do the latter. Notice that the **href** for the link does not have a file extension. The WebMatrix runtime will automatically resolve the extension to recognize **Welcome.cshtml**.

```
<a href="Welcome">Welcome</a>
```

10. Click on the first link. That will bring up the welcome page.



11. Click the Refresh button in the browser. The time changes!



C# and Razor

- At the top of the file there is some C# code delimited by a pair of braces preceded by the @ symbol. Within the braces you can have ordinary C# code.


```
@{  
    string name = "Bob";  
    var now = DateTime.Now;  
}
```

- Within the HTML markup you can reference the current value of these variable by simply using the variable name prefixed again by the @ symbol.

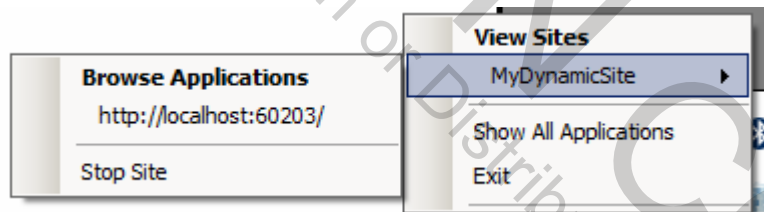
```
<h2>Welcome, @name</h2>  
<h3>It is now @now</h3>
```

- This terse syntax is called “Razor” and is also used in ASP.NET MVC.
- We will discuss Razor in more detail in the next chapter.

IIS Express

- **WebMatrix will automatically launch IIS Express as a Web server to enable you to test your pages.**
 - IIS Express is also the development Web server used in recent versions of Visual Studio.
- **You show the hidden icons in the task bar by clicking the up “arrow” , the first button in the group at the right bottom of your screen in Windows 7.**

- **Right-click the  button to bring up the menu for IIS Express.**



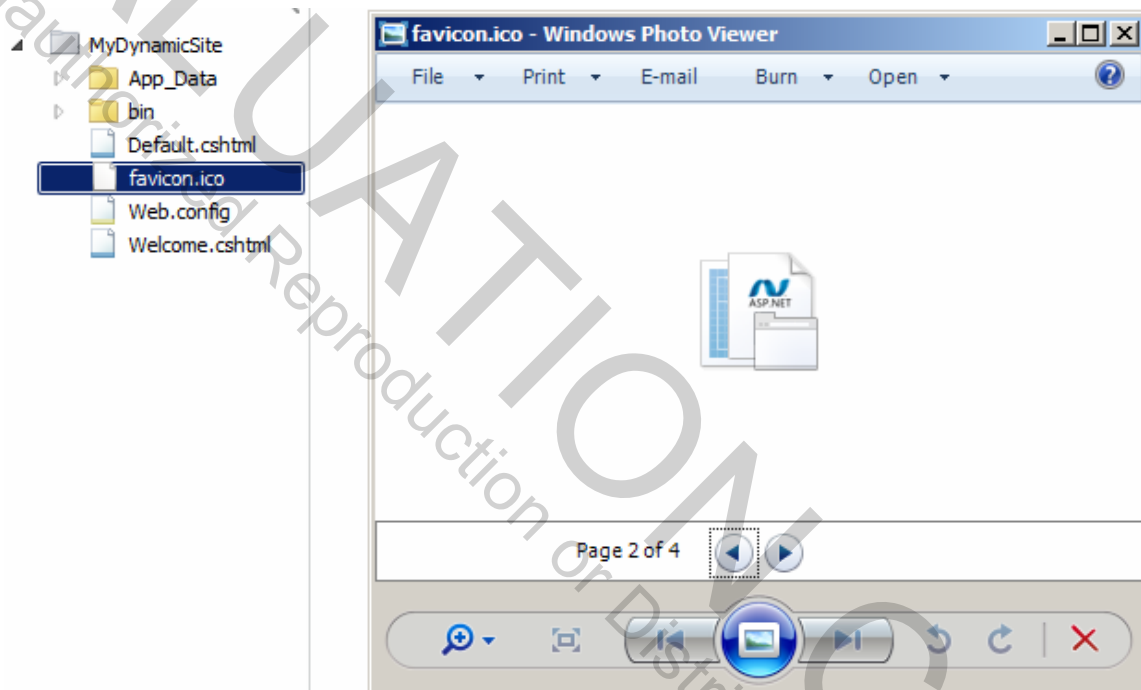
- **Notice that IIS Express will use a random port number assigned by WebMatrix. (You will also see this port number in the browser’s address bar.)**

`http://localhost:60203/`

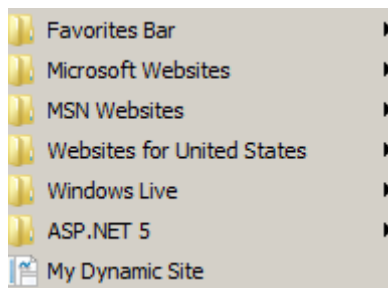
- **You can do experiments, such as stopping the site in the middle of testing your site.**

favicon.ico

- **One feature of the starter site is a *favicon* icon file.**
 - If you open the file, you can see the different size icons displayed in Windows Photo Viewer. The largest of the four icons is 48 pixels square.



- **You can edit this file by an icon editing tool.**
- **If you add your site to Favorites in your browser, the icon will be displayed in your Favorites list.**



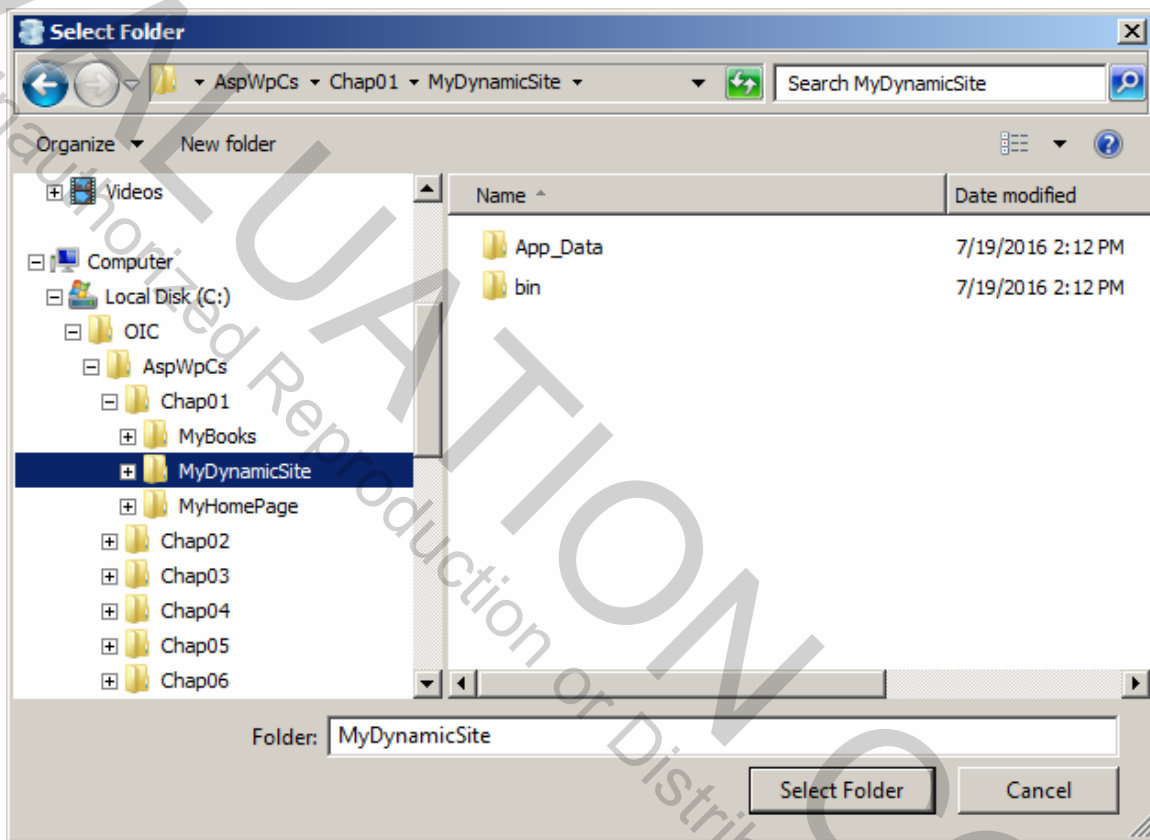
Opening an Existing Website

- When you create a new website in WebMatrix, by default it will be created in a folder in *My Web Sites*.
- When you start WebMatrix you have a choice of **My Sites, New and Open**.
 - My Sites will allow you to open any of your sites in **My Web Sites**.
 - With Open you have a wide variety of choices. Folder will allow you to open a site anywhere on your local file system.

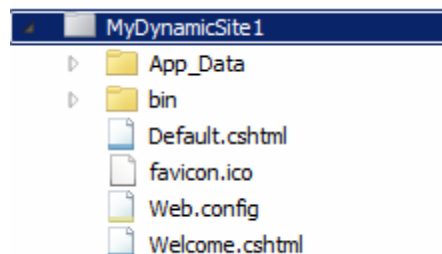


Opening an Existing Website (Cont'd)

- You can then navigate in Windows Explorer to the folder containing the site you wish to open.



- After skipping Azure, your site will then be opened, possibly with a new name to make it unique.



ASP.NET History

- **Microsoft's original server side technology was Active Server Pages or ASP (or, now, "Classic ASP").**
 - Dynamic content is generated via a scripting language such as VBScript or Jscript, which must be compiled each time the page is accessed.
- **In 2002 Microsoft introduced ASP.NET, built on top of the .NET Framework.**
 - You can use compiled, object-oriented languages with ASP.NET, including C# and Visual Basic.
 - All the power of the .NET Framework is available to you, including the extensive class library.
 - Two types of Web applications were supported: Web Forms and Web Services (on top of SOAP).
- **In 2009 Microsoft introduced ASP.NET MVC 1.0 as a sophisticated alternative to Web Forms, based on the Model-View-Controller design pattern.**
- **In 2010 in connection with MVC 2.0 the succinct Razor syntax was introduced. Also the easy-to-use WebMatrix and ASP.NET Web Pages came out.**
- **In 2012 ASP.NET Web API was introduced as a simpler programmatic interface built directly on HTTP.**
- **Next is the open source ASP.NET 5.**

ASP.NET Web Forms

- **Web Forms helps you build form-based Web pages. A WYSIWYG development environment enables you to drag controls onto Web pages.**
 - Special “server-side” controls present the programmer with an event model similar to what is provided by controls in ordinary Windows programming.
- **This drag and drop approach to UI development was pioneered by Visual Basic for Windows.**
- **Web Forms applied the model to Web applications, making it very easy to create form-based Web applications without requiring extensive knowledge of HTML.**
 - The development environment and runtime take care of that for you.
- **However, to make this work, a lot had to happen behind the scenes.**
 - For example, extensive “view state” would have to be maintained for each web page.
 - The developer does not have easy access to the HTML to achieve customized visual appearance.
 - Automated testing of applications is difficult.

ASP.NET MVC

- **Key advantages of ASP.NET MVC include:**
 - The MVC pattern promotes **separation of concerns** into input logic (controller), business logic (model) and UI (view). This aids in managing complexity.
 - These components are loosely coupled, promoting parallel development.
 - This loose coupling also facilitates automated testing.
 - Views are created using standard HTML and cascading style sheets, giving the developer a high degree of control over the user interface.
 - There is no view state, reducing the load on the browser in rendering a page.
- **Separation of Concerns:**
 - Each component has one responsibility
 - SRP – Single Responsibility Principle
 - DRY – Don't Repeat Yourself
 - More easily testable
 - Helps with concurrent development

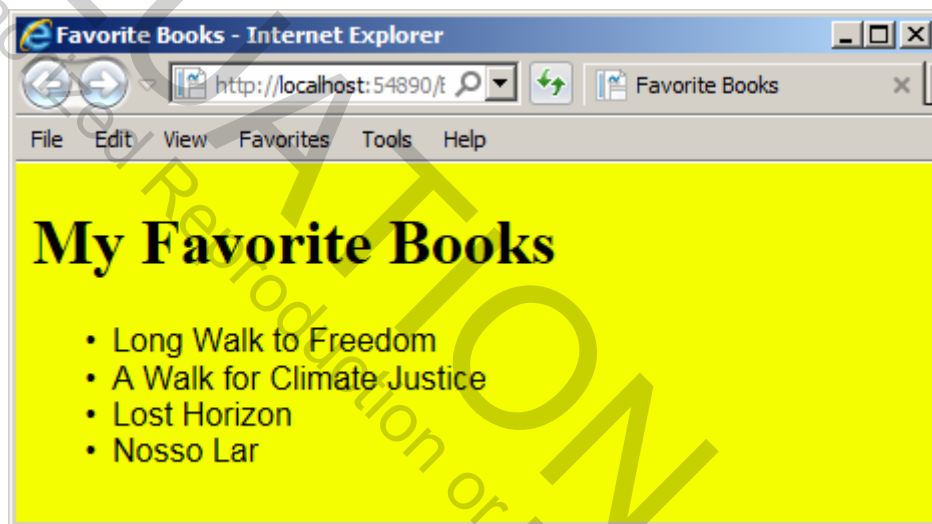
ASP.NET Web Pages

- **A key advantage of ASP.NET MVC is that it does not hide the underlying HTML.**
 - Thus the developer has a high degree of control over the user interface.
- **But ASP.NET MVC is quite a sophisticated framework, useful for large applications but likely overkill for smaller websites.**
- **ASP.NET Web Pages provides a simple approach to creating web sites, while still allowing a high degree of control over the user interface through the use of HTML and CSS.**
- **The preferred development platform for both Web Forms and MVC work is Visual Studio, a highly capable tool but also a large one.**
- **WebMatrix provides a simpler development platform for creating Web Pages applications.**
- **Visual Studio can also be used in the development of Web Pages applications.**
 - If you are experienced with Visual Studio, you will find it easy to use with Web Pages. You can then take advantage of debug capabilities not found in WebMatrix.

Lab 1

A Static Web Page for Displaying a List

In this lab you will use WebMatrix to implement a simple static website for maintaining a list of book titles. The book titles are displayed in an unordered list. You will also do some simple styling of the output.



Detailed instructions are contained in the Lab 1 write-up at the end of the chapter.

Suggested time: 30 minutes

Summary

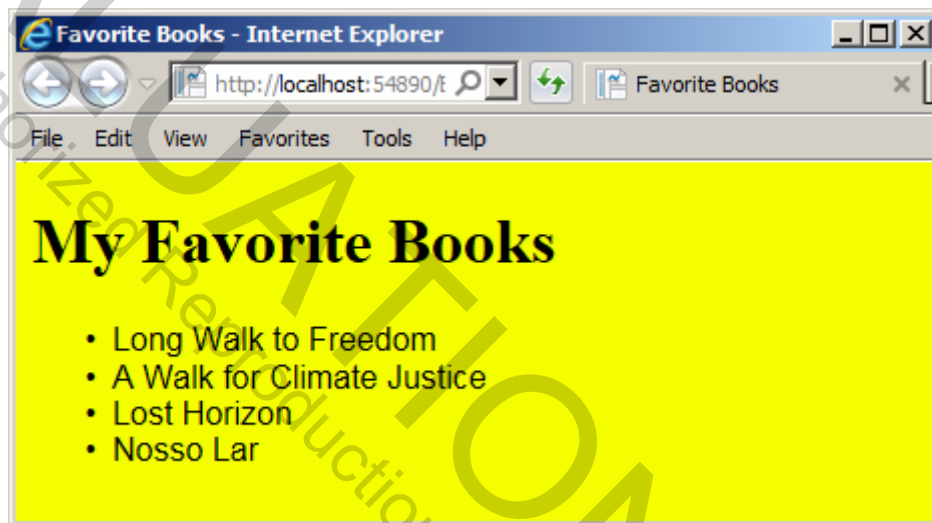
- **You can develop static websites using HTML and CSS with WebMatrix as an editor.**
- **You can test your websites using IIS Express, a Web server that is integrated with WebMatrix.**
- **Dynamic Web applications contain content generated on the server.**
- **ASP.NET is built on top of the .NET Framework.**
- **You can use compiled, object-oriented languages with ASP.NET, including C# and Visual Basic.**
- **ASP.NET programming models include Web Forms, MVC and Web Pages.**
 - Web Forms provides a WYSIWYG drag and drop approach to designing the user interface, hiding the underlying HTML.
 - MVC gives direct access to HTML and CSS but is a rather sophisticated environment.
 - Web Pages is an easy-to-use framework also allowing direct to HTML and CSS.

Lab 1

A Static Web Page for Displaying a List

Introduction

In this lab you will use WebMatrix to implement a simple static website for maintaining a list of book titles. The book titles are displayed in an unordered list. You will also do some simple styling of the output.



Suggested Time: 30 minutes

Root Directory: OIC\AspWpCs

Directories: My Web Sites (create your website here)
 Chap01\MyBooks\Step1 (solution to Part 1)
 Chap01\MyBooks\Step2 (solution to Part 2)

Part 1. Create Web Page with List of Books

1. Use WebMatrix to create a new website **MyBooks** based on the Empty Site template. Skip creating a site on Windows Azure.
2. Add a new CSHTML file **Books.cshtml** to your site
3. Edit the HTML to provide the title "Favorite Books" and an h1 header "My Favorite Books".

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```

<head>
  <meta charset="utf-8" />
  <title>Favorite Books</title>
</head>
<body>
  <h1>My Favorite Books</h1>
</body>
</html>

```

4. Launch the page in the browser.



5. Edit the HTML to display some book titles (of your own choosing) as an unordered list.

```

<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Favorite Books</title>
    <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <h1>My Favorite Books</h1>
    <ul>
      <li>Long Walk to Freedom</li>
      <li>A Walk for Climate Justice</li>
      <li>Lost Horizon</li>
      <li>Nosso Lar</li>
    </ul>
  </body>
</html>

```

6. Launch in the browser. You are now at Step 1.

Part 2. Provide Styling with a Cascading Style Sheet

1. Add a new CSS file **StyleSheet.css** to your site.
2. Edit the CSS file to specify a font family. It is a good practice to use alternative names to help the browser find a match. For example, **Arial** on a PC would be

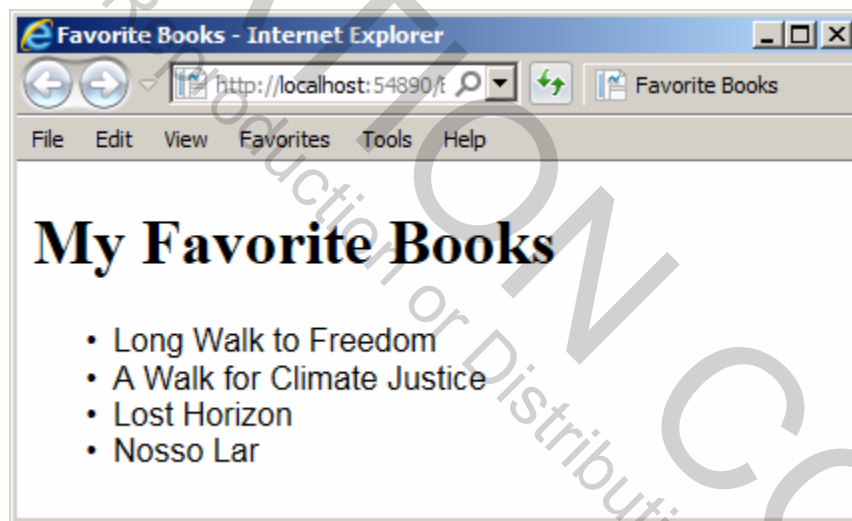
Helvetica on a Mac. It is also a good idea to provide a generic font family as a backup, such as **sans-serif**.

```
body {
}
li {
    font-family: Arial, Helvetica, sans-serif;
}
```

3. Edit the CSHTML file to link to the style sheet. This should be specified in the <head> section.

```
<head>
<meta charset="utf-8" />
<title>Favorite Books</title>
<link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
```

4. Launch in the browser. Note that the font family of the elements has been changed, but the font family of the <h1> element remains defaulted.



5. Now specify a different background-color for the body. Note that WebMatrix will bring up a color picker tool for you. The result will be a color specified in hex.

```
body {
    background-color:#f4ff00;
}
li {
    font-family: Arial, Helvetica, sans-serif;
}
```

6. Launch in browser. You are now at Step 2.

Chapter 5

Web Database Applications

Web Database Applications

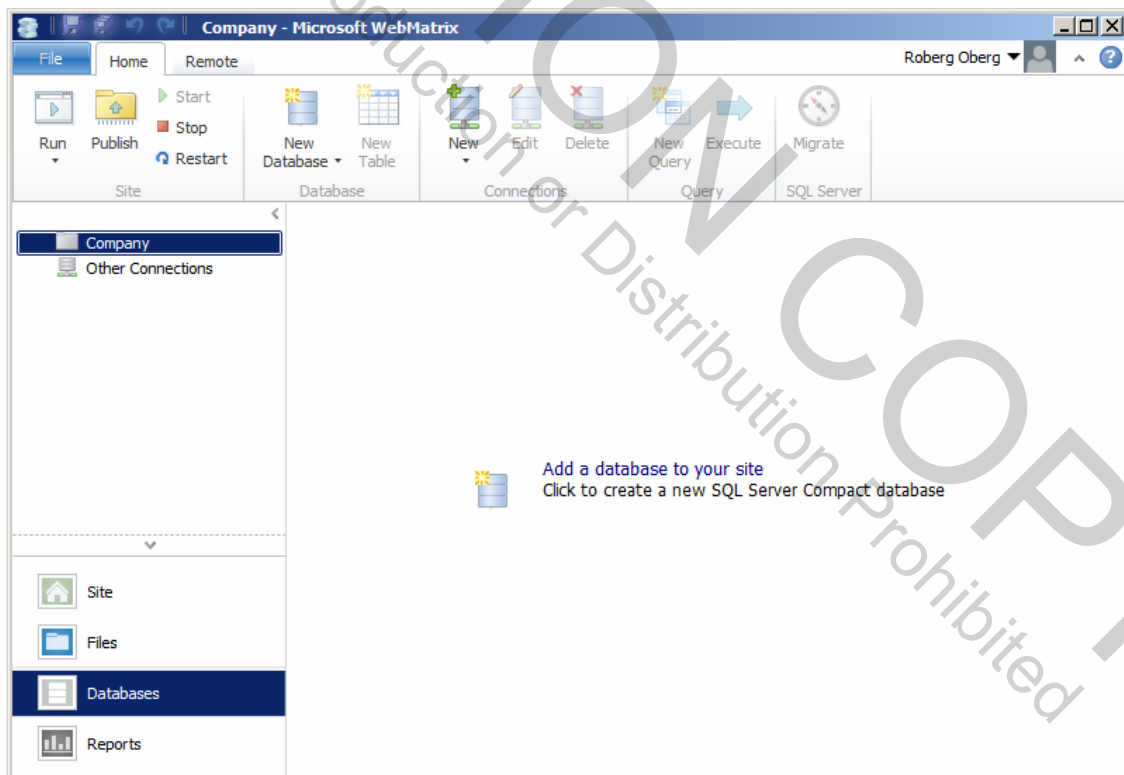
Objectives

After completing this unit you will be able to:

- Use WebMatrix to create a SQL Server CE database.
- Use the Database helper to open a database and execute SQL statements on the database.
- Use the WebGrid helper to display tabular data, such as data from a database.
- Customize the WebGrid display of data through suitable HTML styling.
- Implement a Web application to insert, update and delete records in a database.
- Protect your website from SQL injection attacks through the use of parameters.
- Access existing databases other than SQL Server CE ones.

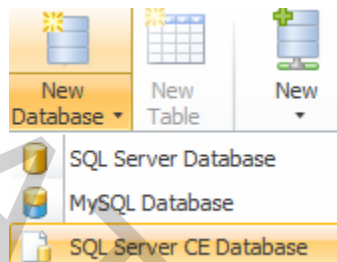
Creating a Database

- **WebMatrix integrates database functionality in a very seamless manner.**
 - It provides a great platform for creating Web database applications using ASP.NET Web Pages.
- **Let's begin by creating a new website and a simple database from scratch.**
 1. Create a new site **Company** using the Empty template. Skip Azure.
 2. In the left pane, select the Databases workspace.

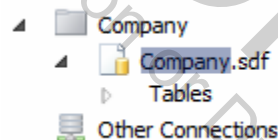


Creating a Database (Cont'd)

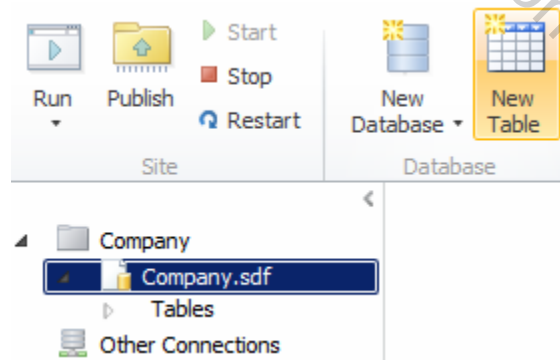
3. In the main window click the link to add a new SQL Server Compact database to your new site. You may also use the ribbon, which gives you a choice of a SQL Server, MySQL or SQL Server CE database. For now we are sticking with SQL Server CE.



4. This will create a SQL Server CE database (in an .sdf file), with suggested name the same as your site. You can change the name to whatever you wish. We will stick with **Company.sdf**.



5. Press Enter to accept this name and select the database. You will then see a New Table button in the ribbon.



6. Click the New Table button. The Table Designer will now come up.

Table Designer

7. Enter three names, ID, FirstName, and LastName. ID will be the primary key and also has the IsIdentity option selected (this will automatically generate sequential values). Null are not allowed in any of these fields. The data types are, respectively, **int** and **nvarchar** of maximum length 10 for first name and 15 for last name.

(Company.sdf).NewTable_1*

Employee

Name	Data Type	Default Value	Is Primary Key?	Is Identity?	Allow Nulls
ID	int	Null	Yes	Yes	No
FirstName	nvarchar (10)	Null	No	No	No
LastName	nvarchar (15)	Null	No	No	No

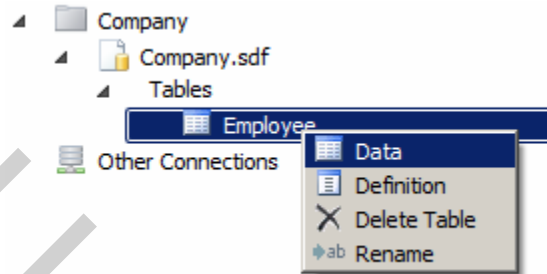
8. Save your design by Ctrl+S or menu File | Save. You could also use the Quick Access Toolbar. This small toolbar can be brought up by right-clicking over an empty area of the menu bar or the ribbon. There are little buttons for Save, Save All, Undo and Redo.



9. The next step will be to enter some data.

Entering Data

10. With the Database workspace still selected, right-click over the Employee table and choose Data from the context menu.

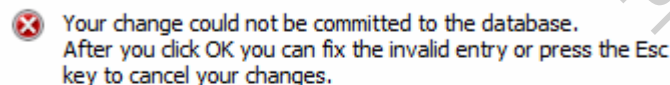


11. Enter a few first and last names. Notice that the ID field will be generated for you automatically.

ID	FirstName	LastName
1	Amy	Jones
2	Robert	Smith
3	Carl	Pike

12. Every time you hit Enter after the last column, the row will be entered into the database. If you attempt to enter invalid data, for example a name that is too long or leave a field NULL, you will get an error message.

Invalid data



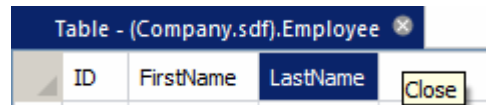
The column cannot contain null values. [Column name = LastName, Table name = Employee]

[Copy details](#)

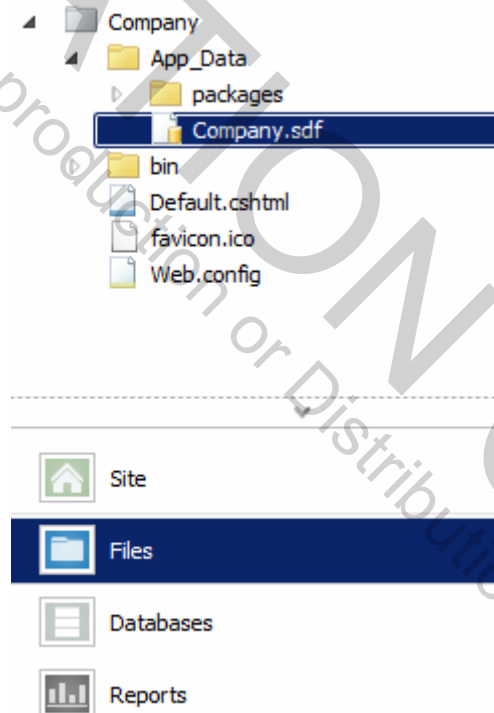
OK

Files Workspace

13. When finished with a window in WebMatrix, you can close it by clicking the little X at the top right.



14. Select the Files workspace. Now you can see the new database file **Company.sdf** has been created in the **App_Data** folder, and you can start working with **.cshtml** files for HTML markup and Razor code.



Displaying the Data

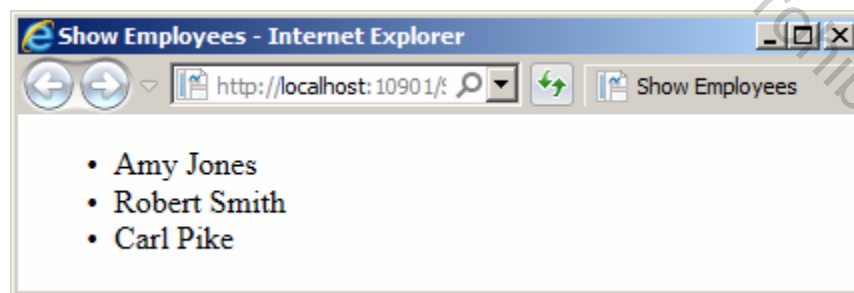
1. Add a new CSHTML file **Show.cshtml**.
2. Provide the following initial Razor code block.

```
@{  
    var db = Database.Open("Company");  
    var data = db.Query("select * from Employee");  
}
```

3. Provide the following HTML markup with Razor.

```
<html lang="en">  
  <head>  
    <meta charset="utf-8" />  
    <title>Show Employees</title>  
  </head>  
  <body>  
    <ul>  
      @foreach (var item in data)  
      {  
        <li>@item.FirstName @item.LastName</li>  
      }  
    </ul>  
  </body>  
</html>
```

4. Launch the page in the browser.



5. The site is saved in **Company\Step1** in the chapter folder.

Database Class

- The *Database* helper gives access to the .NET Database class in the *WebMatrix.Data* namespace.

- Important methods are the static method **Open()** and the method **Query()**.

- **Database.Open** method

- Opens a connection to a database using a file name or a named connection string.

```
public static Database Open(  
    string name  
)
```

- The parameter **name** is either a filename, without extension, of a **.sdf** or **.mdf** file stored in the **App_Data** folder or else a connection string, which is defined in **Web.config**.

- **Database.Query** method

- Executes a SQL query, returning a list of rows.

```
public IEnumerable<Object> Query(  
    string commandText,  
    params Object[] parameters  
)
```

- The parameter **commandText** specifies a SQL query.
- The second (optional) parameter is used for parameterized queries, discussed later.

DynamicRecord Class

- When the SQL statement is a **SELECT** statement, *Query()* returns a collection of *DynamicRecord* objects.
 - The properties are generated dynamically at runtime from the column names used in the SELECT query.
 - Thus you can use the property dot notation to access them.

```
<li>@item.FirstName @item.LastName</li>
```

- You could also use indexer notation.

```
<li>@item["FirstName"] @item["LastName"]</li>
```

- If per chance the SQL table had a column whose name has a space, you would *have to* use indexer notation.

```
Item["First Name"]
```

WebGrid Helper

- When displaying results of a **SELECT** query it is often convenient to use a table.
- You could directly use the **HTML** table tags and suitable **Razor** code.
- But it is much easier to use the **WebGrid** helper, which will generate the **HTML** for a table automatically.
 - See Step 2 of the **Company** example.

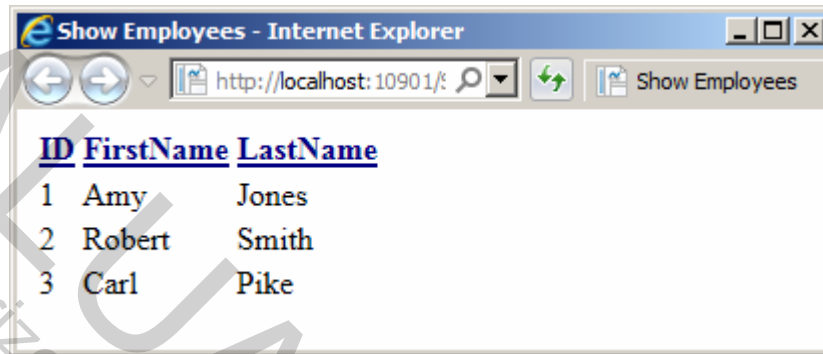
```
@{
    var db = Database.Open("Company");
    var data = db.Query("select * from Employee");
    var grid = new WebGrid(data);
}

<!DOCTYPE html>

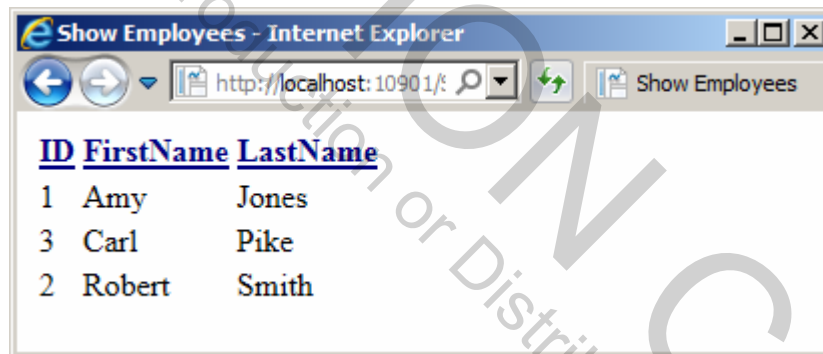
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Show Employees</title>
    </head>
    <body>
        @grid.GetHtml()
    </ul>
    </body>
</html>
```


WebGrid Helper (Cont'd)

- Launch the page in the browser.



- The column names are represented by links. If you click on a column, the display will be sorted.



- Notice that since the SELECT query was SELECT *, all the columns are shown, including the automatic ID, which might not be of interest to the user.
 - The `GetHtml()` method has a parameter that can be used to control which columns are displayed.

WebGrid.GetHtml Method

- **This method returns the HTML markup that is used to render the WebGrid instance.**
 - There are a great many parameters to control the rendering!

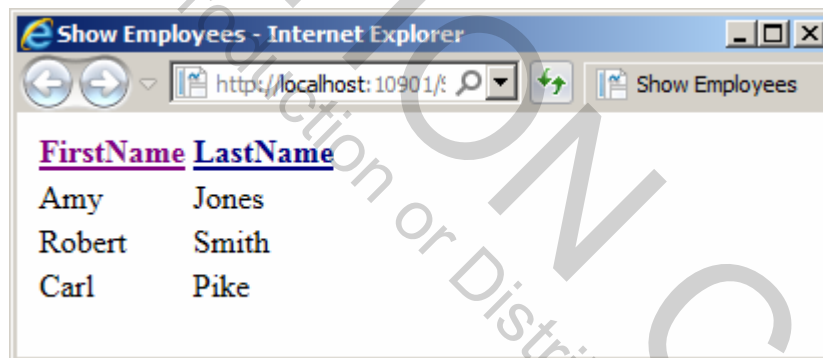
```
public IHtmlString GetHtml(  
    string tableStyle,  
    string headerStyle,  
    string footerStyle,  
    string rowStyle,  
    string alternatingRowStyle,  
    string selectedRowStyle,  
    string caption,  
    bool displayHeader,  
    bool fillEmptyRows,  
    string emptyRowCellValue,  
    IEnumerable<WebGridColumn> columns,  
    IEnumerable<string> exclusions,  
    WebGridPagerModes mode,  
    string firstText,  
    string previousText,  
    string nextText,  
    string lastText,  
    int numericLinksCount,  
    Object htmlAttributes  
)
```

Grid Columns

- **Here is code to control which columns are rendered.**
 - Notice that we used a named argument rather than positional notation!

```
@grid.GetHtml(  
    columns: grid.Columns(  
        grid.Column("FirstName"),  
        grid.Column("LastName")  
    )  
)
```

- **Here is the output:**



<u>FirstName</u>	<u>LastName</u>
Amy	Jones
Robert	Smith
Carl	Pike

Table Style

- Let's define some CSS styles to control the appearance of the grid.

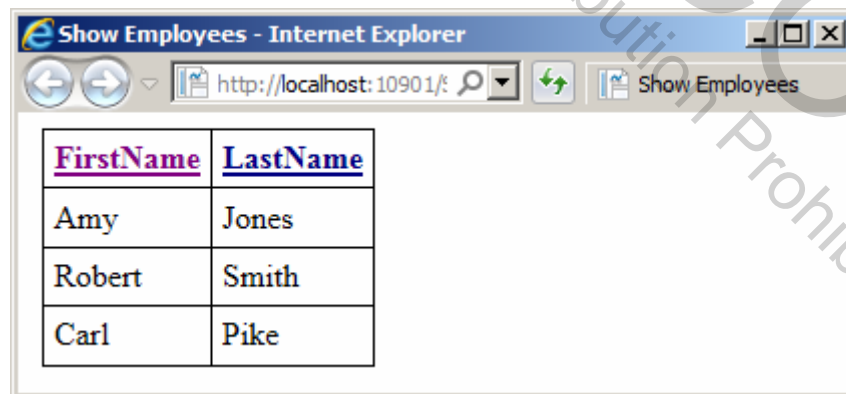
- Basically, we want to supply borders.

```
<style>
.grid { margin: 4px; border-collapse: collapse;
       width: auto; }
.grid th, .grid td { border: 1px solid black;
                     padding: 5px; }
</style>
```

- Use this style when invoking *GetHtml()*:

```
@grid.GetHtml(
    tableStyle: "grid",
    columns: grid.Columns(
        grid.Column("FirstName"),
        grid.Column("LastName")
    )
)
```

- Here is the output:



The screenshot shows a web browser window titled "Show Employees - Internet Explorer". The address bar displays "http://localhost:10901/". The page content shows a table with two columns, "FirstName" and "LastName", and three rows of data: Amy Jones, Robert Smith, and Carl Pike. The table has a simple border and the headers are underlined.

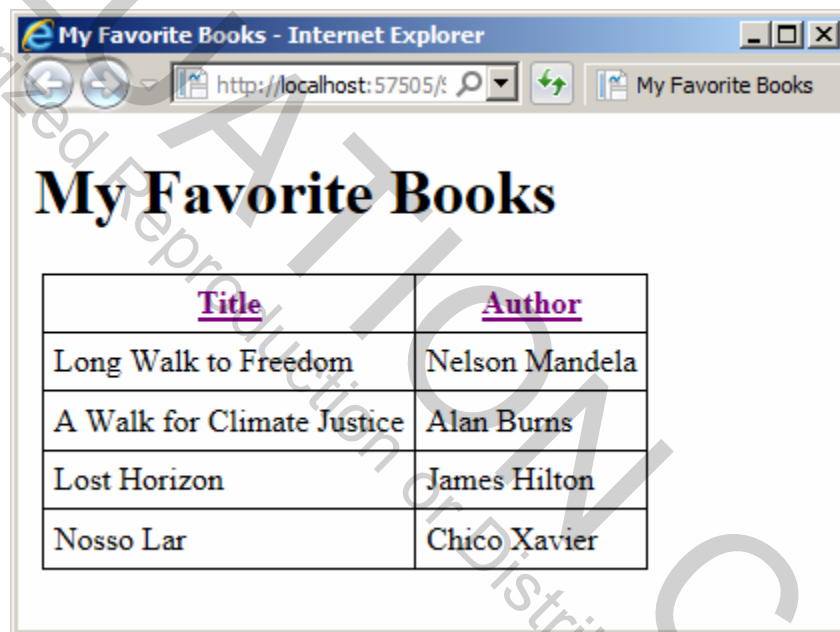
<u>FirstName</u>	<u>LastName</u>
Amy	Jones
Robert	Smith
Carl	Pike

- This completes Step 2 of the *Company* example.

Lab 5A

List of Books in a Database

In this lab you will implement a Web Pages application to maintain a list of books, stored in a database. The books are shown in a grid. (Later you will add code allowing a user to insert, delete and update the table of books.)



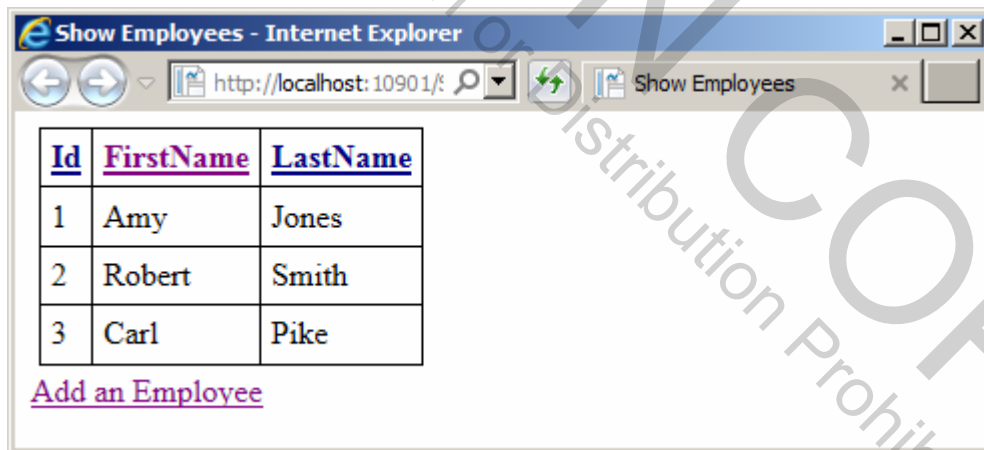
<u>Title</u>	<u>Author</u>
Long Walk to Freedom	Nelson Mandela
A Walk for Climate Justice	Alan Burns
Lost Horizon	James Hilton
Nosso Lar	Chico Xavier

Detailed instructions are contained in the Lab 5A write-up at the end of the chapter.

Suggested time: 30 minutes

Entering Data

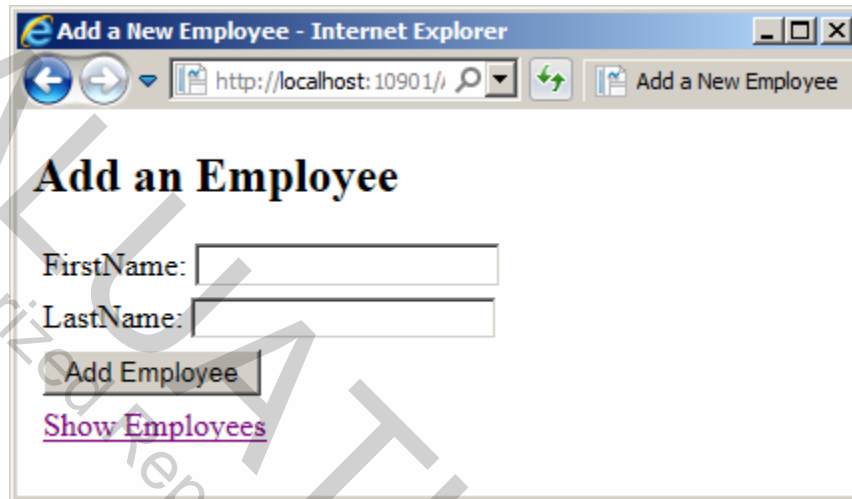
- In a Web database application you will frequently want to enable your users to enter new data into a database.
- You can implement this functionality doing the following:
 - Provide an input form for entering data.
 - Respond to a submit of the form by executing a SQL INSERT statement to insert the new data into the database. It will be important for the security of our application to use a parameterized query in constructing the SQL statement.
 - Provide a link to give the user access to the form for entering data. We will put a link on the **Show.cshtml** page.



- We show the Id column in this version of the application. You will then see how Id's get generated automatically when new names are inserted.

Input Form for a New Employee

- We illustrate with Step 3 of our *Company* application.



- The page for adding a company is **Add.cshtml**. Here is the markup for the form and a link back to **Show.cshtml**.

```
<h2>Add an Employee</h2>
<form method="post">
  <div>
    <label for="first">FirstName:</label>
    <input id="first" name="first"
      value="@Request.Form["first"]" />
  </div>
  <div>
    <label for="last">LastName:</label>
    <input id="last" name="last"
      value="@Request.Form["last"]" />
  </div>
  <div>
    <input type="submit" value="Add Employee" />
  </div>
</form>
<div>
  <a href="Show.cshtml">Show Employees</a>
</div>
```

Inserting Data into Database

- **Inserting the data is done in the Razor code block at the top, which includes constructing a SQL INSERT statement.**

```
@{
    string first = "";
    string last = "";

    if(IsPost)
    {
        first = Request.Form["first"];
        last = Request.Form["last"];
        var db = Database.Open("Company");
        string cmd = "INSERT INTO Employee
            (FirstName, LastName) VALUES(@0, @1)";
        db.Execute(cmd, first, last);
    }
}
```

- **The database code is highlighted.**
- **Note use of the parameterized query for the INSERT.**
 - You should *always* make use of parameterized queries rather than build a SQL command by concatenation.
 - This will prevent a SQL injection attack.
 - You can read about SQL injection attacks on MSDN:

[https://msdn.microsoft.com/en-us/library/ms161953\(SQL.105\).aspx](https://msdn.microsoft.com/en-us/library/ms161953(SQL.105).aspx)

Validation and Error Handling

- **This example works fine for inserting good data, but there are various problems that could occur.**
 - The user could submit blank data.
 - The user could submit invalid data, e.g. too long a name.
 - There could be a network or database problem.
- **Let's see how Step 3 deals with these various situations.**
 - A blank record could get into the database!

<u>ID</u>	<u>FirstName</u>	<u>LastName</u>
1	Amy	Jones
2	Robert	Smith
3	Carl	Pike
12	David	O'Malley
13		

- Try submitting a first name longer than 10 characters.

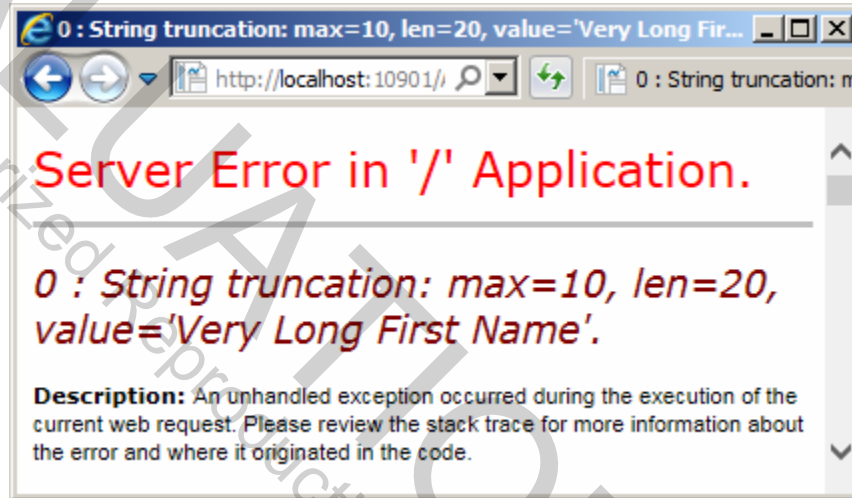
Add an Employee

FirstName:
 LastName:

[Show Employees](#)

Database Schema Errors

- **The long first name violates the database schema.**
 - We get an error message that is not user friendly (although in this case clear).



- **A good way to deal with such errors is by validating the data before submitting it to the database.**
 - We discussed validation in Chapter 3.

```
Validation.RequireField("first",  
    "First Name required");  
Validation.Add("first",  
    Validator.StringLength(10));  
Validation.RequireField("last",  
    "Last Name required");  
Validation.Add("last",  
    Validator.StringLength(15));
```

Validation of Input Data

- Before processing the input data, we check that it is valid.

```
if (IsPost && Validation.IsValid())  
{  
    ...  
}
```

- In the markup we display validation errors.

```
<input id="first" name="first" ...  
@Html.ValidationMessage("first")
```

- We provide a style for error messages.

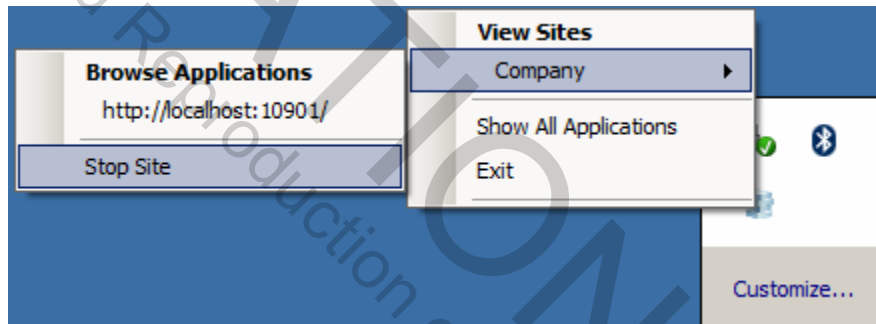
```
<style>  
    div { margin: 5px; }  
    .field-validation-error{ color:red; }  
</style>
```

- Now the user will be presented with good error messages.

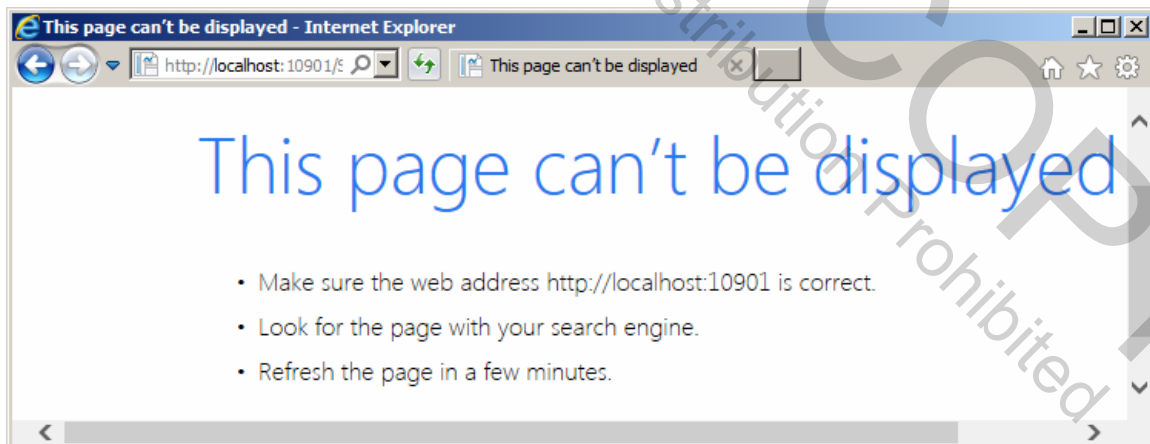


Network Errors

- Even if we have thorough validation code in our application, other error conditions can also crop up unrelated to user input.
- For example, the network or the website itself could go down.
- We can simulate such an outage by stopping our site in IIS Express.



- Then we would get an error message like this:

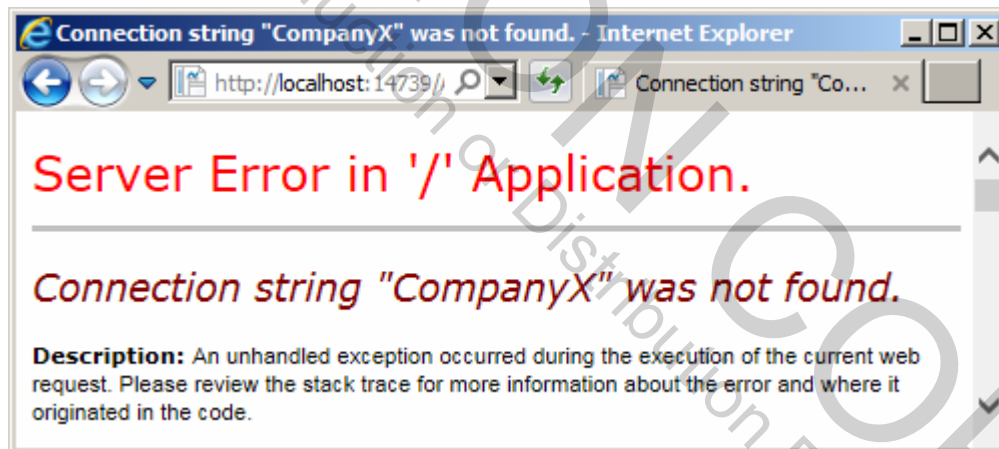


Database Errors

- Another class of errors is errors pertaining to the database.
- For example, suppose you try to open a database using the wrong name.
 - We could simulate such an error by changing the name used in the **DB.Open()** statement.

```
var db = Database.Open( "CompanyX" );
```

- Launch in the browser and try adding a good name.
- We get an unhandled exception.



Exception Handling

- You can make your application more robust by placing your database code in a *try* block with a *catch* handler.

- A string variable holds a message.

```
...
string message = "";
if (IsPost && Validation.IsValid())
{
    first = Request.Form["first"];
    last = Request.Form["last"];
    try
    {
        var db = Database.Open("Company");
        string cmd = "INSERT INTO Employee
            (FirstName, LastName) VALUES(@0, @1)";
        db.Execute(cmd, first, last);
        message =
            "Employee has been added to database";
    }
    catch (Exception ex)
    {
        message = ex.Message;
    }
}
```

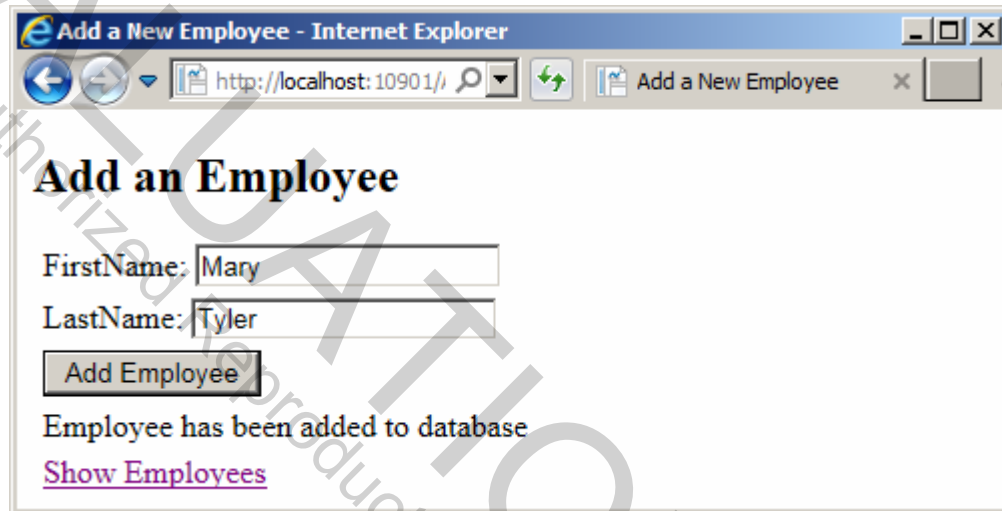
- The message is displayed after the form.

```
...
</form>
<div>
    @message
</div>
```

- The application is now at Step 4.

Running the Application

- If we entered good data and the database operations were successful, a message is displayed indicating that a record has been added to the database.



- If there is a database error, such as using the wrong name for the database (to test, modify source code), we will get an error message on our form.



Updating a Record

- **When adding a new record we start with a blank form.**
- **To update a record we should populate the form with the existing record, which is identified by its primary key.**
 - In our application we will pass the Id in the query string.
 - We can then obtain the row data by code like this.

```
string first = "";
string last = "";
string empId = "";
string message = "";

if (!IsPost)
{
    empId = Request.QueryString["id"];
    if (!empId.IsNullOrEmpty())
    {
        var db = Database.Open("Company");
        string cmd =
            "select * from Employee where Id = @0";
        var row = db.QuerySingle(cmd, empId);
        first = row.FirstName;
        last = row.LastName;
    }
    else
    {
        message = "No Employee ID supplied";
    }
}
```

- This code should be executed only the first time the page is shown, i.e. not in response to a POST.

Querying for a Single Row

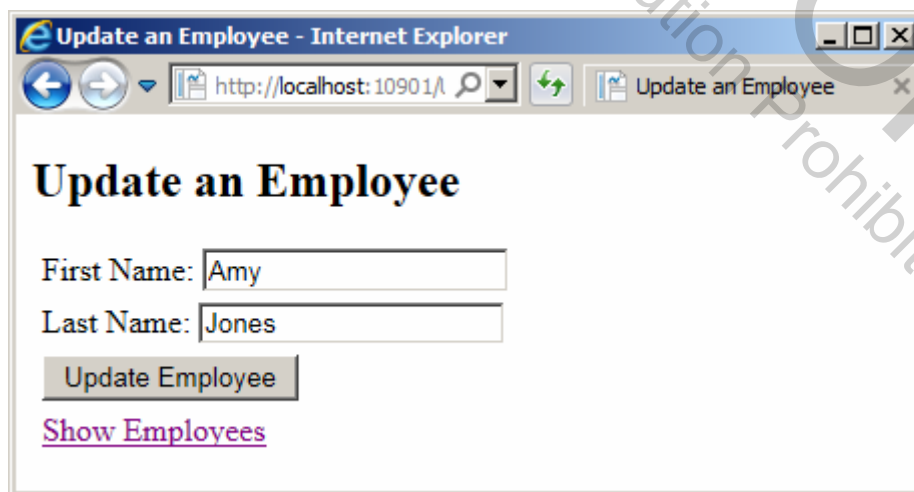
- We used the *Query()* method to obtain a collection of rows.
- We use *QuerySingle()* to obtain a single row, using a **SELECT** command that returns only a single row.

```
string cmd =  
    "select * from Employee where Id = @0";  
var row = db.QuerySingle(cmd, empId);  
first = row.FirstName;  
last = row.LastName;
```

- Again we perform a parameterized query to prevent a SQL injection attack.

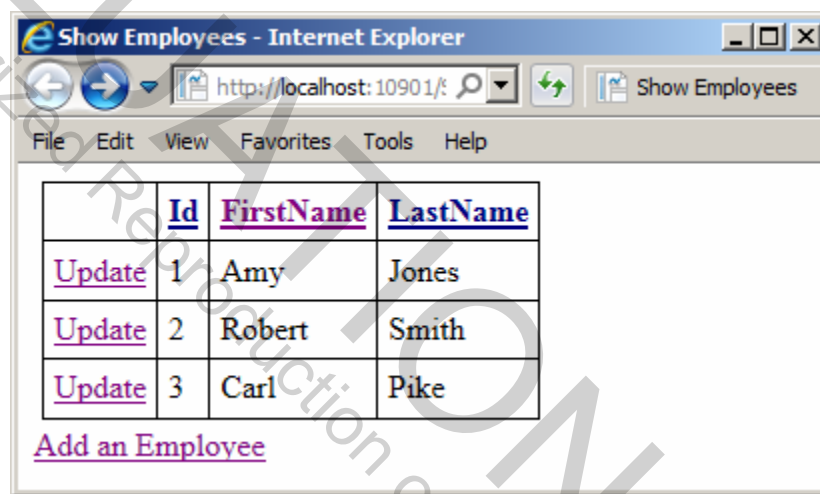
- **Step 5A provides this code (no update in DB yet).**
 - Launch **Update.cshtml** in the browser. Append the query string “?id=1” to obtain the first record. The complete URL will then look like this:

`http://localhost:10901/Update.cshtml?id=1`

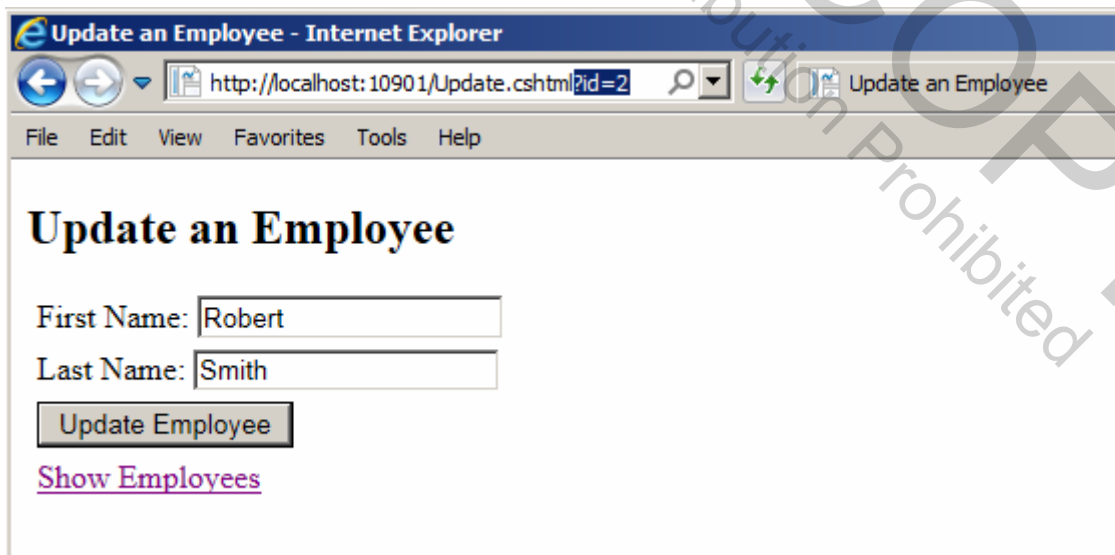


Update Link in WebGrid

- The next step in crafting our update functionality will be to create a new link column in the WebGrid displaying the data, so that we can conveniently select the row to be updated.
 - Step 5B illustrates (still no actual update in database).



- Click on one of the Update links, and you will be brought to the Update page with a query string passing the Id.



Markup for the Table

- **Back on the Show page, examine the source in the browser. Look at the rows of the HTML table.**

```
<tr>
  <td><a href="/Update.cshtml?id=1">Update</a></td>
  <td>1</td>
  <td>Amy</td>
  <td>Jones</td>
</tr>
<tr>
  <td><a href="/Update.cshtml?id=2">Update</a></td>
  <td>2</td>
  <td>Robert</td>
  <td>Smith</td>
</tr>
<tr>
  <td><a href="/Update.cshtml?id=3">Update</a></td>
  <td>3</td>
  <td>Carl</td>
  <td>Pike</td>
</tr>
```

WebGrid Columns

- To understand how such markup is created, let's look more closely at WebGrid columns.
- Here is the code creating the markup we just saw:

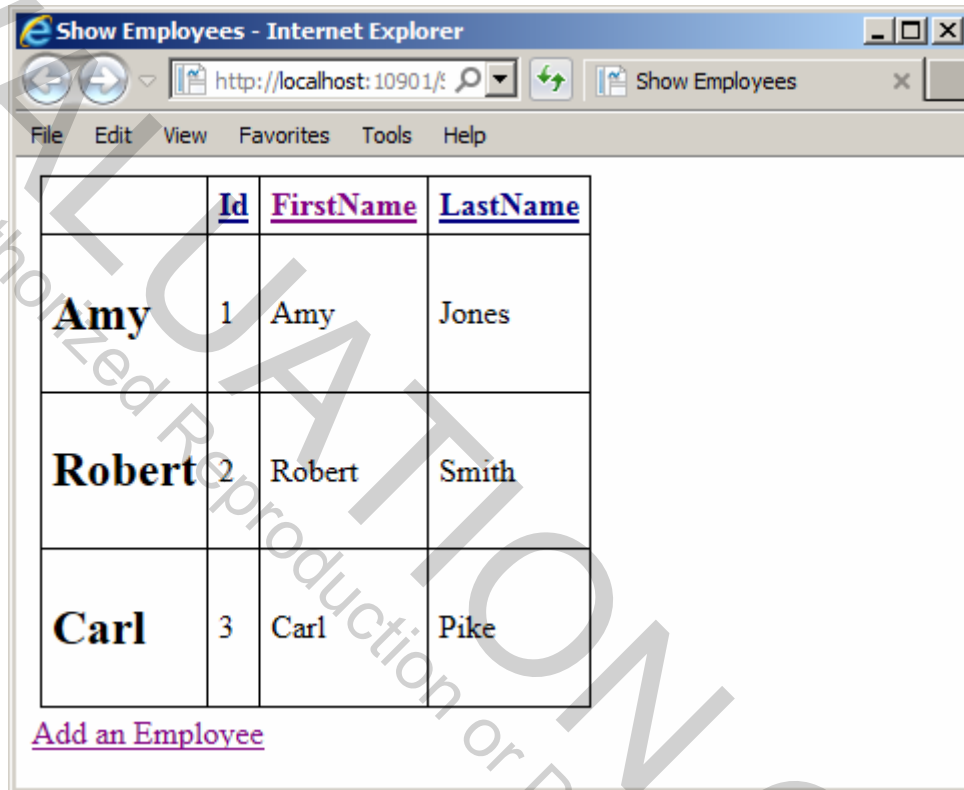
```
@grid.GetHtml(  
    tableStyle: "grid",  
    columns: grid.Columns(  
        grid.Column(format:  
@<a href="~/Update.cshtml?id=@item.Id">Update</a>),  
        grid.Column("Id"),  
        grid.Column("FirstName"),  
        grid.Column("LastName")  
    )  
)
```

- The last three columns simply use the column name.
- The first column in the grid uses **format** to enable specifying HTML markup.
- The **item** object refers to the database record for a row.
- In Step5B you can see more clearly how this works by removing the comment for a simpler markup and commenting out the markup setting up the link.

```
@grid.GetHtml(  
    tableStyle: "grid",  
    columns: grid.Columns(  
grid.Column(format:@<h2>@item.FirstName</h2>),  
        grid.Column("Id"),  
        grid.Column("FirstName"),  
        grid.Column("LastName")  
    )  
)
```

Custom Display for a Grid Column

- Launch the page in the browser.



- Now examine the source in the browser.

```
<tr>
  <td><h2>Amy</h2></td>
  <td>1</td>
  <td>Amy</td>
  <td>Jones</td>
</tr>
<tr>
  <td><h2>Robert</h2></td>
  <td>2</td>
  <td>Robert</td>
  <td>Smith</td>
</tr>
...
```

Database Update Code

- **The final step in implementing the update functionality is to provide the appropriate SQL.**
 - This will be done when the form is submitted via POST.

```
if (IsPost)
{
    Validation.RequireField("first",
        "First Name required");
    ...

    empId = Request.Form["id"];
    first = Request.Form["first"];
    last = Request.Form["last"];
    if (Validation.IsValid())
    {
        try
        {
            var db = Database.Open("Company");
            string cmd =
                "UPDATE Employee SET FirstName=@0, LastName=@1
                WHERE Id=@2";
            int numrow = db.Execute(cmd, first, last,
                                    empId);

            message = string.Format(
                "{0} row(s) updated", numrow);
        }
        catch (Exception ex)
        {
            message = ex.Message;
        }
    }
}
```

- **Complete code is in *Company\Step5C*. At last the database actually gets updated!**

Error Handling

- **As usual, the trickiest part of the code is the error handling.**
 - You can review the complete code.
- **There are some nuances with *Update.cshtml* compared to *Add.cshtml*.**
 - For example, we don't need to set up the validation rules until we respond to a POST request.
 - We need to call **Validation.IsValid()** after the variables **empId**, **first** and **last** have been assigned, because we use the value of these variables to remember the contents of the textboxes.

```
<input id="id" name="id" value="@empId" />
...
<input id="first" name="first" value="@first" />
```

- We illustrate the use of the **AddFormError()** method of the **Validation** helper. This can be used for defining additional error message.

```
if (row != null)
{
    first = row.FirstName;
    last = row.LastName;
}
else
{
    Validation.AddFormError(
        "No Employee ID supplied");
}
```

Remembering the Employee Id

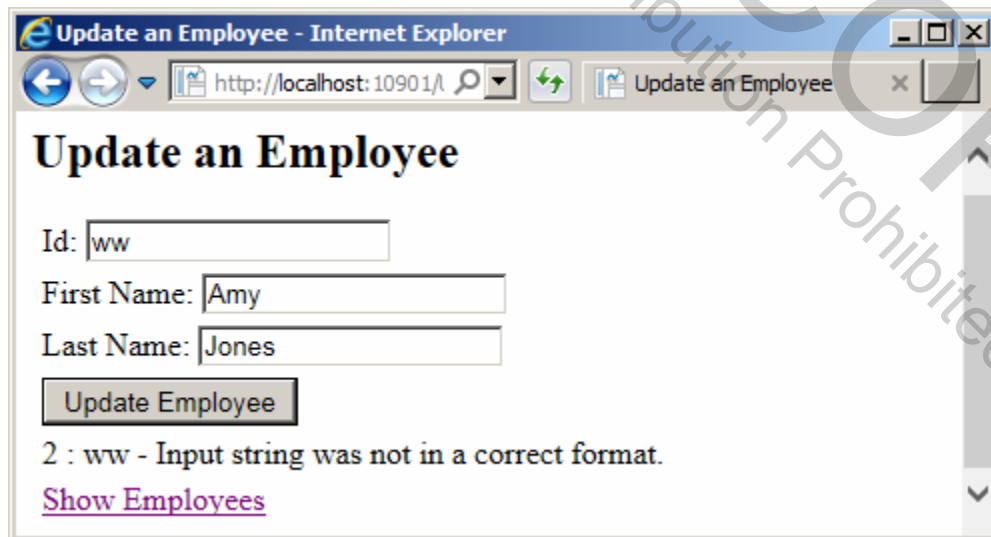
- The Employee Id is originally passed to the Update page in a query string.
- Normally the user would not enter the Id. Hence it would not need to be shown. You could hide it by using the *hidden* attribute on the `<input>` element.

```
<input type="hidden" name="id" value="@empId" />
```

- When it is hidden you won't need an associated label, hence no need for an `id` attribute.
- But even though it is hidden, you can still retrieve a value from form data.

```
empId = Request.Form["id"];
```

- We chose to use an ordinary `<input>` element so that we could test error conditions, e.g. a non-numerical value for the Id.



Update an Employee - Internet Explorer

http://localhost:10901/

Update an Employee

Update an Employee

Id: ww

First Name: Amy

Last Name: Jones

Update Employee

2 : ww - Input string was not in a correct format.

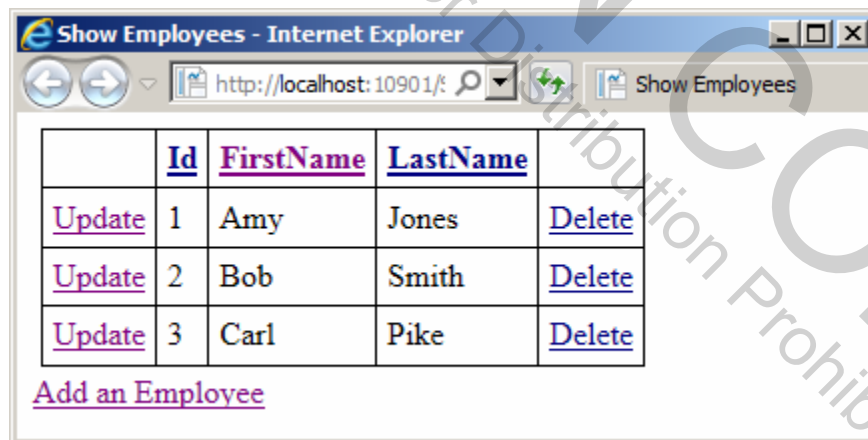
[Show Employees](#)

Deleting a Record

- Finally, let's add code to our Web application to support deleting records from the database.
- Let's begin by providing a column of *Delete* links in the WebGrid.

```
columns: grid.Columns(
    grid.Column(format:
        @<a href="~/Update.cshtml?id=@item.Id">Update</a>),
    grid.Column("Id"),
    grid.Column("FirstName"),
    grid.Column("LastName"),
    grid.Column(format:
        @<a href="~/Delete.cshtml?id=@item.Id">Delete</a>
    )
)
```

- This is done in exactly the same manner as for the Update links.



Delete Page

- **The HTML markup for the Delete page is very similar to that of the Update page.**
 - But we use `` elements rather than textboxes to show the employee details. The user is not editing this information.
 - However, we supply a textbox for the Id as before to facilitate testing error conditions.

```
<h2>Delete an Employee</h2>
@Html.ValidationSummary()
<form method="post">
  <div>
    <label for="id">Id:</label>
    <input id="id" name="id" value="@empId" />
  </div>
  <fieldset>
    <legend>Employee Data</legend>
    <div>
      <span>FirstName</span>
      <span>@first</span>
    </div>
    <div>
      <span>LastName</span>
      <span>@last</span>
    </div>
  </fieldset>
  <div>
    <input type="submit" value="Delete Employee" />
  </div>
  <div>
    @message
  </div>
</form>
...
```

Initializing Employee Data

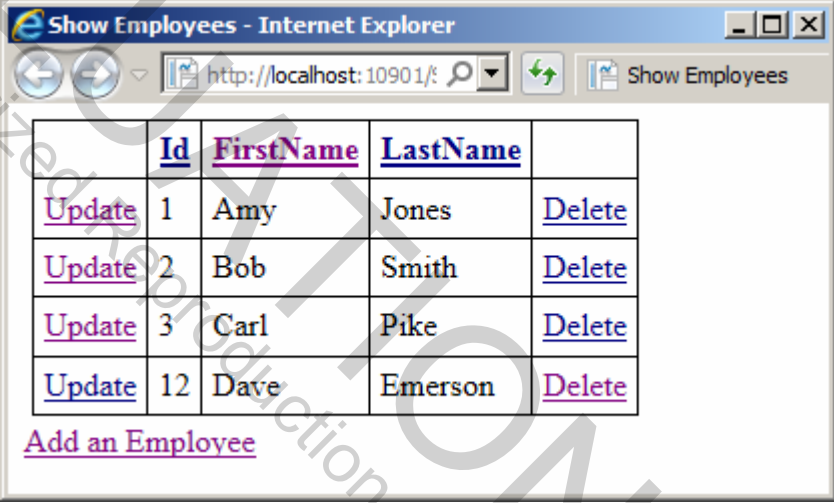
- Likewise the code to retrieve a single record from the database corresponding to the Id passed in the query string is the same.

```
string first = "";
string last = "";
string empId = "";
string message = "";

if (!IsPost)
{
    empId = Request.QueryString["id"];
    if (!empId.IsNullOrEmpty())
    {
        var db = Database.Open("Company");
        string cmd =
            "select * from Employee where Id = @0";
        var row = db.QuerySingle(cmd, empId);
        if (row != null)
        {
            first = row.FirstName;
            last = row.LastName;
        }
        else
        {
            Validation.AddFormError(
                "No Employee ID supplied");
        }
    }
    else
    {
        Validation.AddFormError(
            "No Employee ID supplied");
    }
}
```

Running the Application

- If you have not added any records to the database, you might want to do that now, so you can test deleting one of them.
- Launch *Show.cshtml* in the browser.

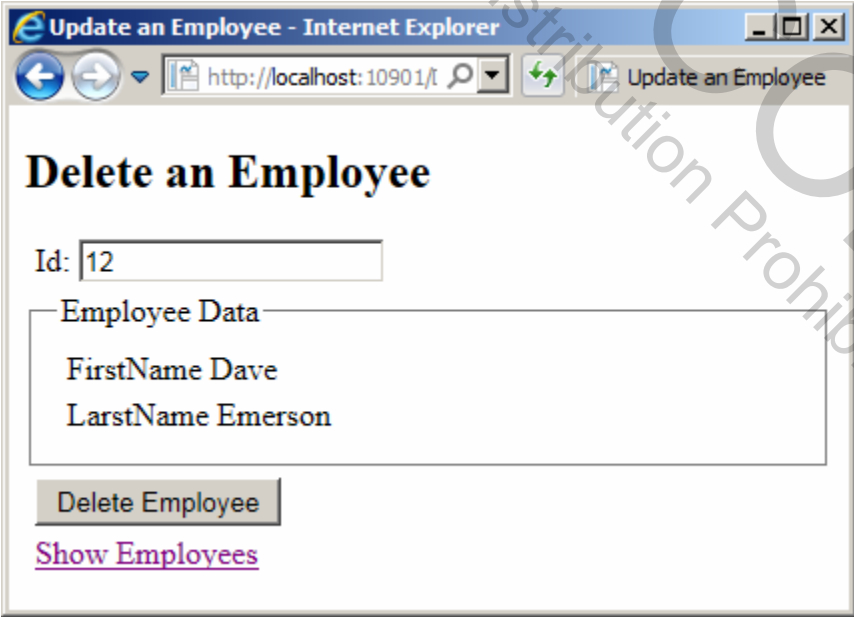


Internet Explorer window titled "Show Employees - Internet Explorer" showing the URL `http://localhost:10901/`. The page displays a table with employee records and links to update or delete each record.

	<u>Id</u>	<u>FirstName</u>	<u>LastName</u>	
<u>Update</u>	1	Amy	Jones	<u>Delete</u>
<u>Update</u>	2	Bob	Smith	<u>Delete</u>
<u>Update</u>	3	Carl	Pike	<u>Delete</u>
<u>Update</u>	12	Dave	Emerson	<u>Delete</u>

[Add an Employee](#)

- Click on a Delete link.



Internet Explorer window titled "Update an Employee - Internet Explorer" showing the URL `http://localhost:10901/`. The page displays a form to delete an employee.

Delete an Employee

Id:

Employee Data

FirstName Dave
LarstName Emerson

[Show Employees](#)

Database Delete Code

- **The final step in implementing the delete functionality is to provide the appropriate SQL.**
 - This will be done when the form is submitted via POST.

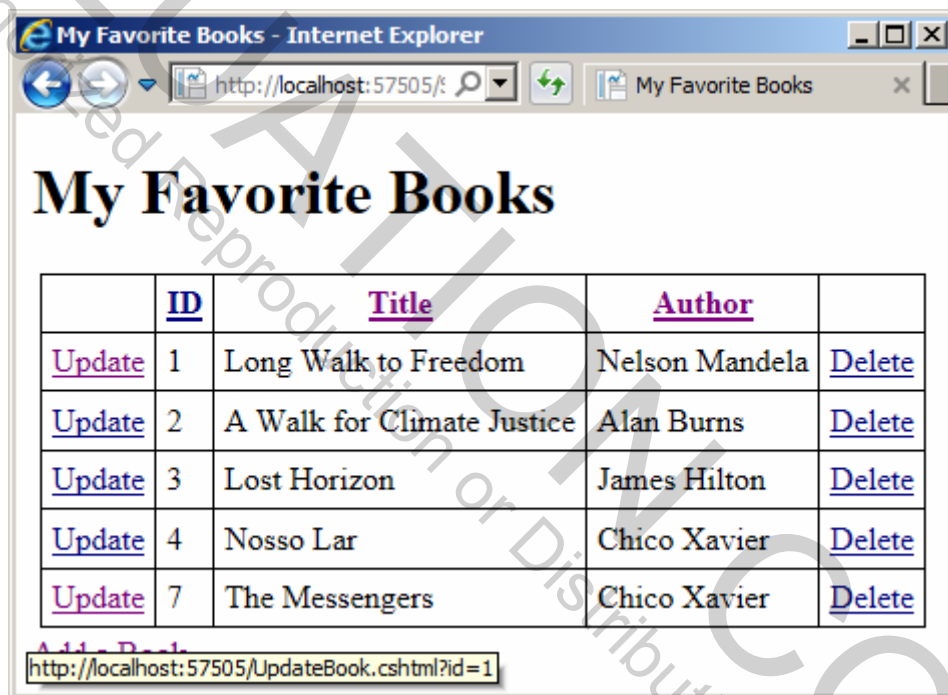
```
if (IsPost)
{
    empId = Request.Form["id"];
    try
    {
        var db = Database.Open("Company");
        string cmd =
            "DELETE FROM Employee WHERE Id=@0";
        int numrow = db.Execute(cmd, empId);
        message = string.Format("{0} row(s) deleted",
                                numrow);
    }
    catch (Exception ex)
    {
        message = ex.Message;
    }
}
```

- **Complete code is in *Company\Step6*.**
- **In both UPDATE and DELETE the WHERE clause is very important.**
 - With the wrong WHERE clause you could delete or update many records, even all of them!

Lab 5B

Updating the Books Database

In this lab you will add code to your Book database application allowing a user to insert, delete and update books in the Book table of your database. You will also provide suitable links for navigating between the pages of your application.



Detailed instructions are contained in the Lab 5B write-up at the end of the chapter.

Suggested time: 90 minutes

Connecting to Other Databases

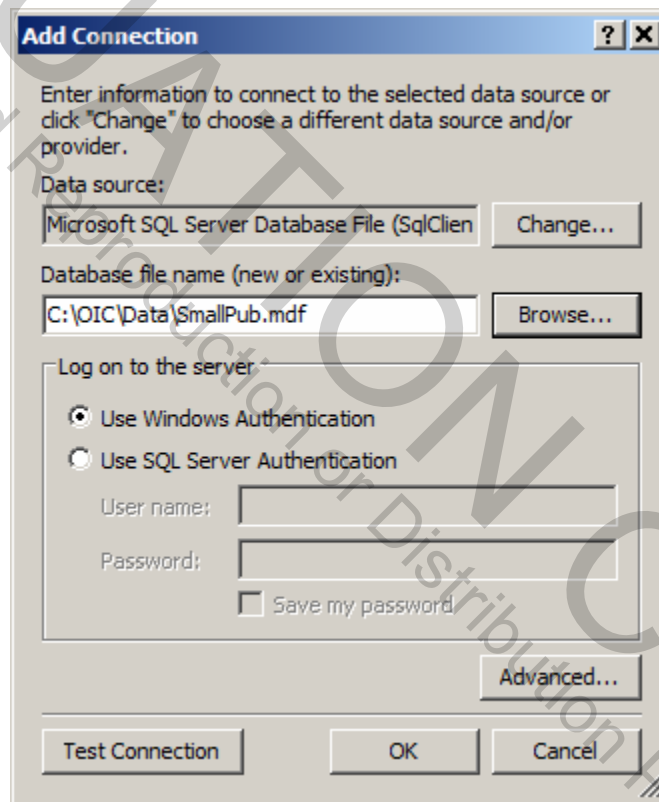
- WebMatrix provides seamless support for creating Web database applications using SQL Server CE.
- If you have another database installed on your system and a .NET provider for it, you can connect to it from within an ASP.NET Web Pages application.
- Use the *OpenConnectionString()* method of the Database helper.
 - Required arguments are a connection string and a provider.

```
string connStr =  
    @"Data Source=(LocalDB)\MSSQLLocalDB;"  
    + @"AttachDbFilename=C:\OIC\Data\SmallPub.mdf;"  
    + "Integrated Security=True";  
string provider="System.Data.SqlClient";  
var db = Database.OpenConnectionString(connStr,  
                                       provider);
```

- For a complete example, see *DataDemo* in the chapter folder.
 - This example assumes you have SQL Server 2014 Express LocalDB.
 - It comes automatically with Visual Studio 2015, and it is a free download from Microsoft.
 - You can conveniently examine the **SmallPub.mdf** database in Server Explorer of Visual Studio 2015.

SmallPub Database

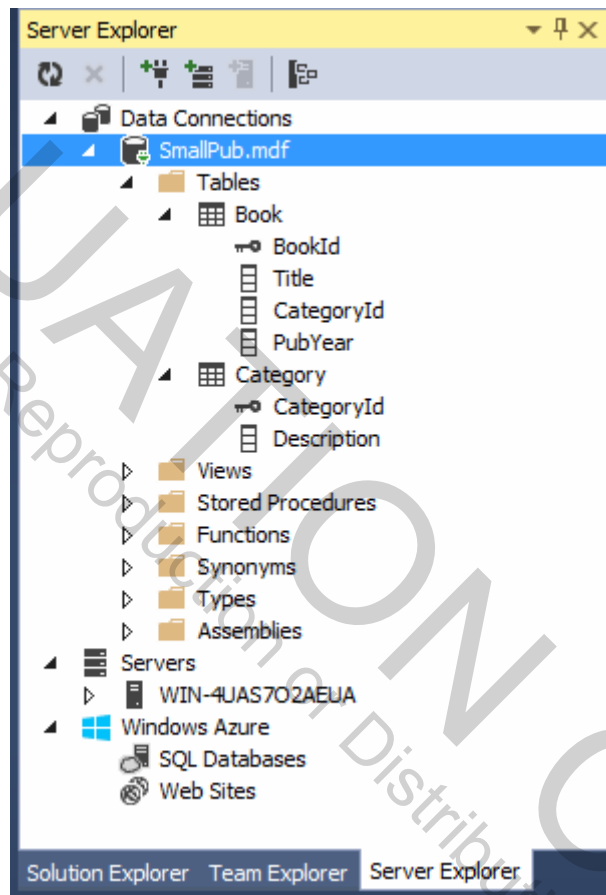
- The SmallPub database contains information about books classified according to categories.
 - It is stored in the file **SmallPub.mdf** in the **C:\OIC\Data** folder.
 - Set up a connection to it in Server Explorer in Visual Studio.



- Click Test Connection. It should work!

SmallPub Database (Cont'd)

- When the connection is set up, you can examine the database in Server Explorer.



- We will be using the SmallPub database later in our examples.

DataDemo Example

- Open the *DataDemo* site in WebMatrix.
- Launch the page *ShowCategories.cshtml* in the browser.



- A very simple interface is provided for adding, deleting, and updating records in the *Category* table.
 - You can examine the complete code.

Performing a Join

- **The SmallPub database contains two tables, so our example can illustrate performing a more complex SQL query, such as an inner join.**

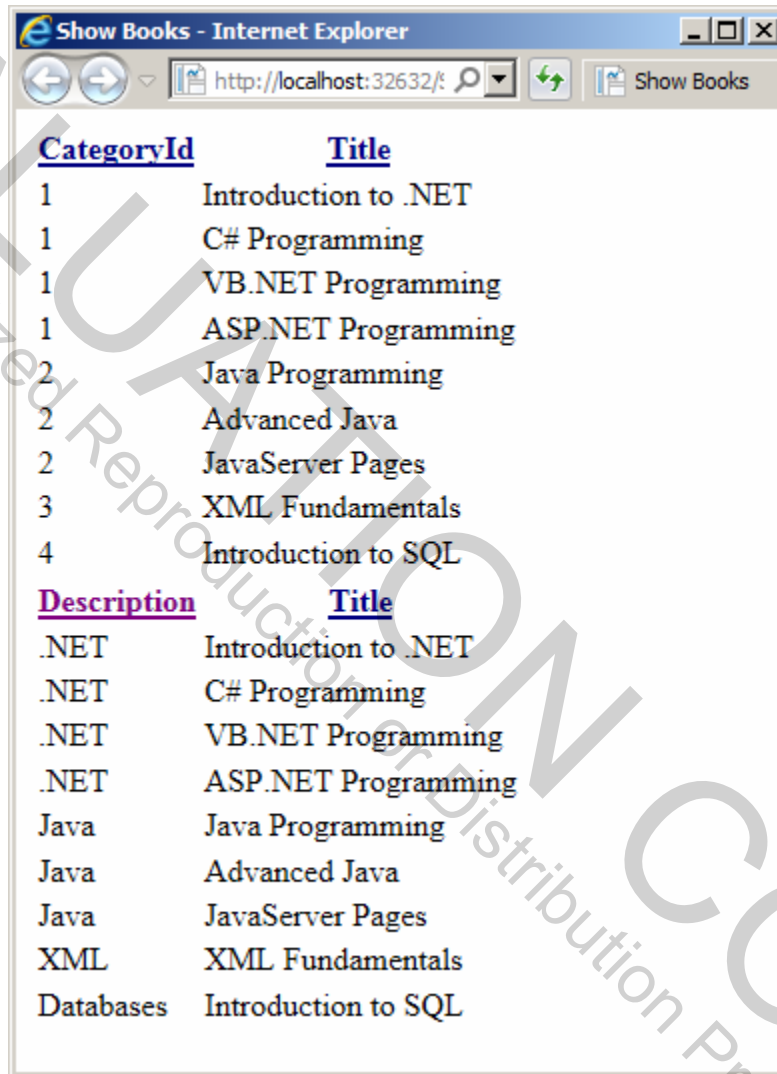
- See **ShowBooks.cshtml**.

```
string connStr =  
    @"Data Source=(LocalDB)\MSSQLLocalDB;"  
    + @"AttachDbFilename=C:\OIC\Data\SmallPub.mdf;"  
    + "Integrated Security=True";  
string provider="System.Data.SqlClient";  
var db = Database.OpenConnectionString(connStr,  
provider);  
var data = db.Query(  
    "select CategoryId, Title from Book");  
var grid = new WebGrid(source: data);  
string query =  
    "select Category.Description, Book.Title " +  
    "from Book inner join Category " +  
    "on Category.CategoryId = Book.CategoryId";  
var data2 = db.Query(query);  
var grid2 = new WebGrid(source: data2);
```

- By means of this join you can display a list of books with a category description used rather than a category ID.

List of Books

- Launch the page *ShowBooks.cshtml* in the browser.



<u>CategoryId</u>	<u>Title</u>
1	Introduction to .NET
1	C# Programming
1	VB.NET Programming
1	ASP.NET Programming
2	Java Programming
2	Advanced Java
2	JavaServer Pages
3	XML Fundamentals
4	Introduction to SQL

<u>Description</u>	<u>Title</u>
.NET	Introduction to .NET
.NET	C# Programming
.NET	VB.NET Programming
.NET	ASP.NET Programming
Java	Java Programming
Java	Advanced Java
Java	JavaServer Pages
XML	XML Fundamentals
Databases	Introduction to SQL

Summary

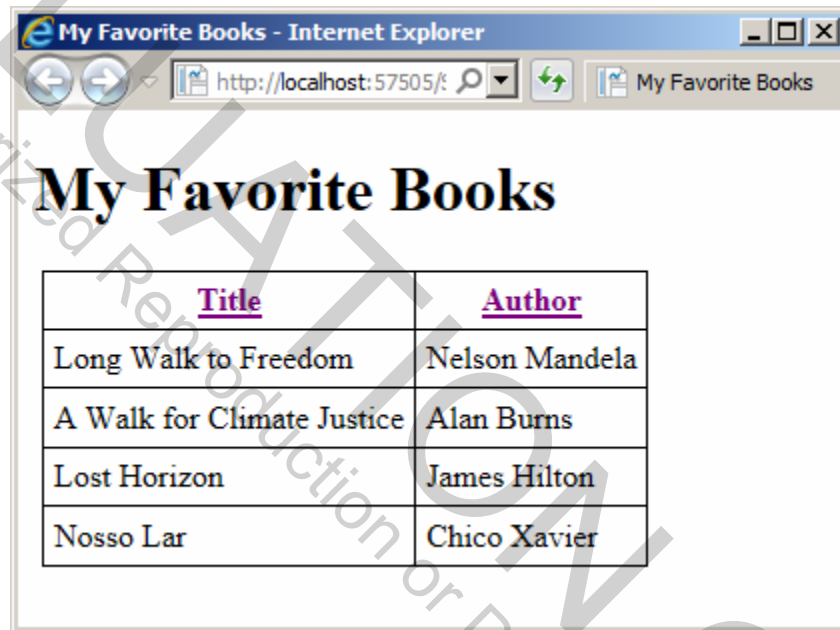
- **WebMatrix provides tools to create and manage SQL Server CE databases.**
- **You can use the Database helper to open a database and execute SQL statements on the database.**
- **With the WebGrid helper you can display tabular data, such as data from a database.**
- **You can customize the WebGrid display of data through suitable HTML styling.**
- **With the Database helper you can insert, update and delete records in a database.**
- **You should *always* use parameterized queries to protect your website from SQL injection attacks.**
- **You access existing databases other than SQL Server CE ones by specifying an appropriate connection string.**

Lab 5A

List of Books in a Database

Introduction

In this lab you will implement a Web Pages application to maintain a list of books, stored in a database. The books are shown in a grid. (Later you will add code allowing a user to insert, delete and update the table of books.)



Suggested Time: 30 minutes

Root Directory: OIC\AspWpCs

Directories: My Web Sites (create your website here)
 Chap05\BookData\Step1 (solution)

Instructions

1. Use WebMatrix to create a new website **BookData** from the Empty Site template in the Template Gallery. Skip Azure.
2. Add a new SQL Server CE database **BookData.sdf** to your site.
3. Create a new table **Book**. Add three fields:
 - a. **BookId**, of type **int**, primary key, identity

b. **Title** of type **nvarchar(30)**, no nulls

c. **Author** of type **nvarchar(15)**, no nulls

Book

Name	Data Type	Default Value	Is Primary Key?	Is Identity?	Allow Nulls
ID	int	Null	Yes	Yes	No
Title	nvarchar (30)	Null	No	No	No
Author	nvarchar (15)	Null	No	No	No

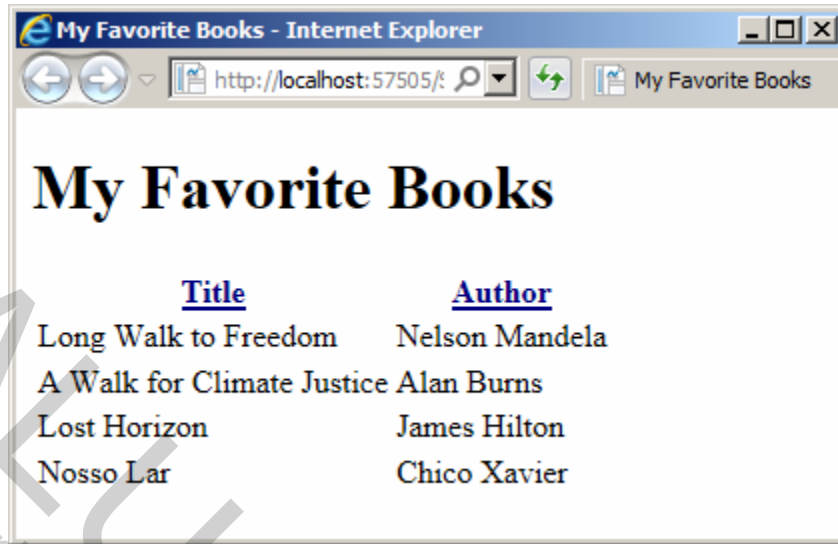
- Add some sample book data.
- Switch to the File workspace and add a file **ShowBooks.cshtml**.
- Provide a title “My Favorite Books” for the page and also display it in a <h1> header.
- Provide an initial Razor code block to open the database and obtain the data from a SELECT query. Get only the Title and Author columns. Initialize a new WebGrid object to display this data.

```
@{
    var db = Database.Open("BookData");
    var data = db.Query("select Title, Author from Book");
    var grid = new WebGrid(data);
}
```

- Add to the HTML markup the Razor code to render the HTML for this grid.

```
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>My Favorite Books</title>
  </head>
  <body>
    <h1>My Favorite Books</h1>
    @grid.GetHtml()
  </body>
</html>
```

- Launch the file in the browser to make sure everything is working so far. We have the bare bones display of the grid without any styling.



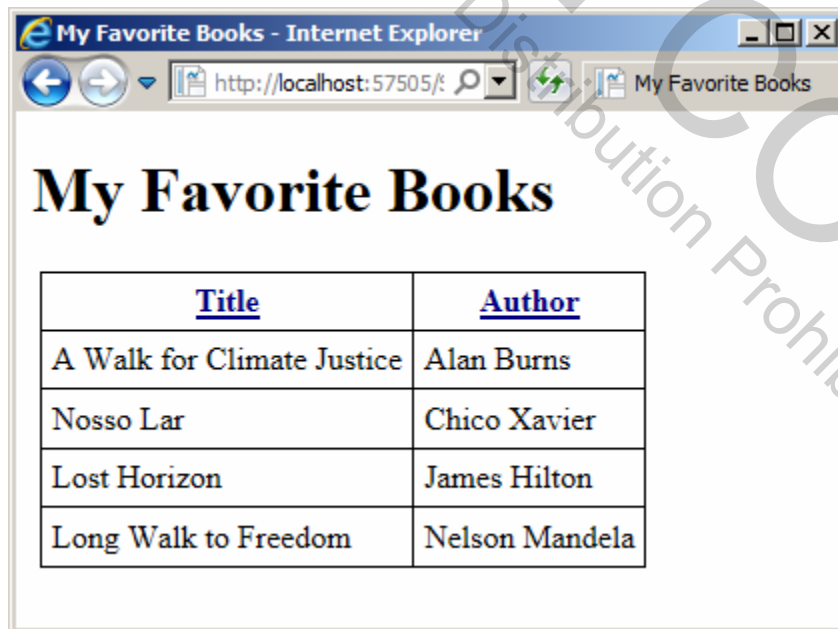
10. Finally, put in some styling for the grid. You use the style code from that given in the **Company** example.

```
<style>
.grid { margin: 4px; border-collapse: collapse; width: auto; }
.grid th, .grid td { border: 1px solid black; padding: 5px; }
</style>
```

11. Use this table style in the Razor code to display the list.

```
@grid.GetHtml(tableStyle: "grid")
```

12. Launch in the browser. Sort alphabetically by Author name.



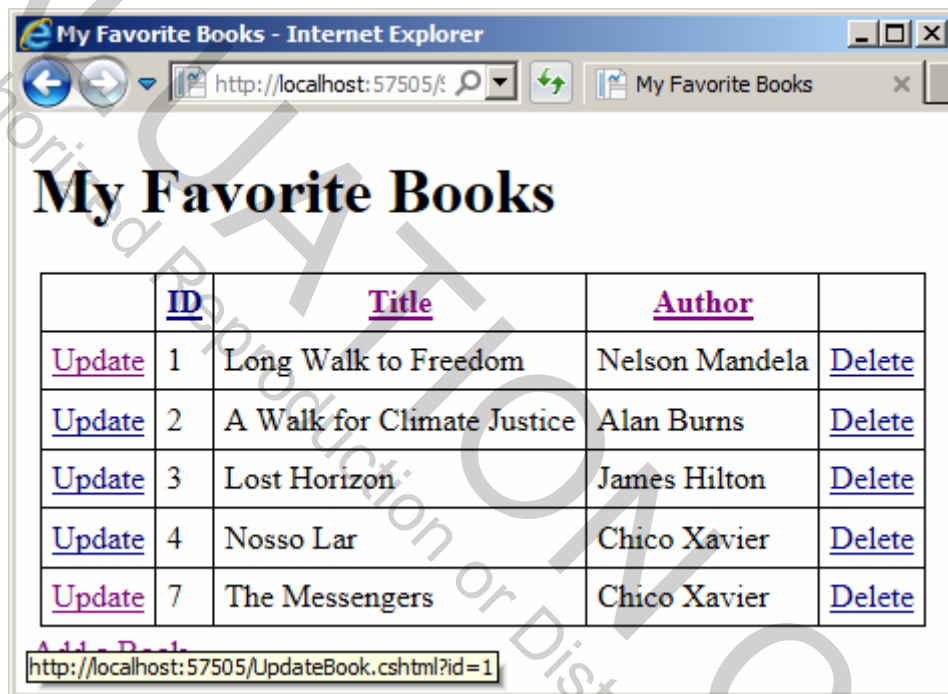
13. You are done!

Lab 5B

Updating the Books Database

Introduction

In this lab you will add code to your Book database application allowing a user to insert, delete and update books in the Book table of your database. You will also provide suitable links for navigating between the pages of your application.



Suggested Time: 90 minutes

Root Directory: OIC\AspWpCs

Directories:

My Web Sites\BookData	(continue your work here)
Labs\Lab5B\BookData	(alternate starter website)
Chap05\BookData\Step1	(backup of starter site)
Chap05\BookData\Step2	(solution to Part 1)
Chap05\BookData\Step3	(solution to Part 2)
Chap05\BookData\Step4	(solution to Part 3)

Part 1: Add and Update Records in the Database

1. If you finished Lab 5A, open the **BookData** site in **My Web Sites**, otherwise open it in **Lab5B**.

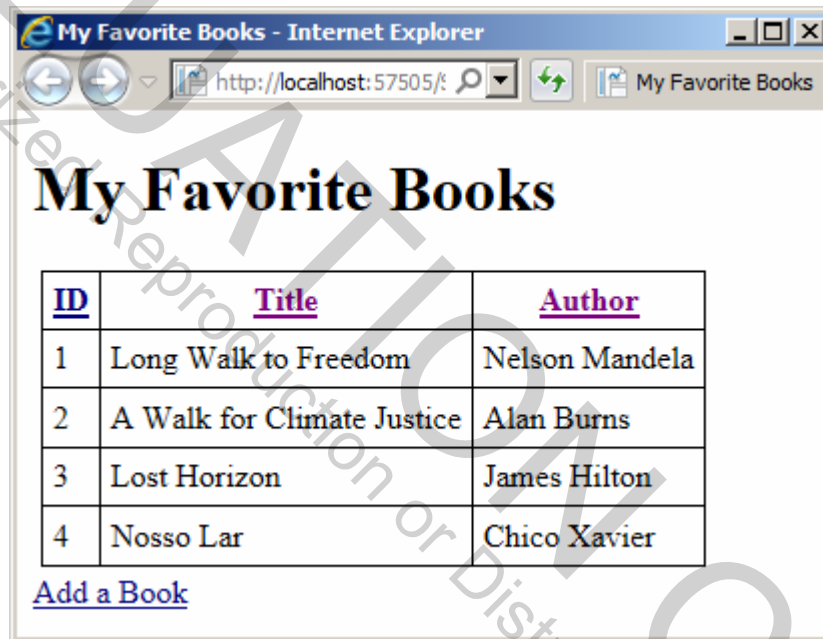
2. Modify the initial Razor code block so that the SELECT query obtains the ID column as well as Title and Author.

```
var data = db.Query("select ID, Title, Author from Book");
```

3. At the bottom of the markup add a link to a new page for adding a book.

```
<div>
  <a href="AddBook.cshtml">Add a Book</a>
</div>
```

4. Launch the file **ShowBooks.cshtml** in the browser to make sure your code to display the books is working.



5. Add a new page **AddBook.cshtml** to you site.
6. Provide a title “Add a New Book” for the page and also display it in a <h1> header.
7. Provide follow code in initial Razor code block.

```
@{
    string title = "";
    string author = "";
    string message = "";
    if (IsPost)
    {
        title = Request.Form["title"];
        author = Request.Form["author"];
        var db = Database.Open("BookData");
        string cmd = "INSERT INTO Book (Title, Author) VALUES(@0, @1)";
        db.Execute(cmd, title, author);
        message = "Book has been added to database";
    }
}
```

```
}
}
```

8. Provide suitable markup to display a form for entering title and author, a message showing result of the database operation, and a link to the Show Books page.

```
<head>
  <meta charset="utf-8" />
  <title>Add a New Book</title>
  <style>
    div { margin: 5px; }
  </style>
</head>
<body>
  <h1>Add a New Book</h1>
  <form method="post">
    <div>
      <label for="title">Title:</label>
      <input id="title" name="title"
        value="@Request.Form["title"]" />
    </div>
    <div>
      <label for="author">Author:</label>
      <input id="author" name="author"
        value="@Request.Form["author"]" />
    </div>
    <div>
      <input type="submit" value="Add Book" />
    </div>
  </form>
  <div>
    @message
  </div>
  <div>
    <a href="ShowBooks.cshtml">Show Books</a>
  </div>
</body>
```

9. Test your application at this point. You should be able to add new books and see the result.
10. Add a new page **UpdateBook.cshtml**.
11. Provide a title “Update a Book” for the page and also display it in a <h1> header.
12. Provide the following code in initial Razor code block to obtain the book ID from the query string and store the title and author in variables so that they can be displayed.

```
@{
  string title = "";
  string author = "";
  string bookID = "";
  string message = "";

  if (!IsPost)
```

```

{
    bookID = Request.QueryString["id"];
    if (!bookID.IsNullOrEmpty())
    {
        var db = Database.Open("BookData");
        string cmd = "select * from Book where ID = @0";
        var row = db.QuerySingle(cmd, bookID);
        if (row != null)
        {
            title = row.Title;
            author = row.Author;
        }
        else
        {
            message = "No Book ID supplied";
        }
    }
    else
    {
        message = "No Book ID supplied";
    }
}
}

```

13. Provide the following markup.

```

<head>
    <meta charset="utf-8" />
    <title>Update a Book</title>
    <style>
        div { margin: 5px; }
    </style>
</head>
<body>
    <h1>Update a Book</h1>
    <form method="post">
        <div>
            <label for="id">ID:</label>
            <input id="id" name="id" value="@bookID" />
        </div>
        <div>
            <label for="title">Title:</label>
            <input id="title" name="title" value="@title" />
        </div>
        <div>
            <label for="author">Author:</label>
            <input id="author" name="author" value="@author" />
        </div>
        <div>
            <input type="submit" value="Update Book" />
        </div>
    </form>
    <div>
        @message
    </div>
</div>

```

```

        <a href="ShowBooks.cshtml">Show Books</a>
    </div>
</body>

```

14. Launch the page **Update.cshtml** in the browser. You will get an error, since you have not passed the book ID in the query string.

15. Now append ?id=1 to the URL of the website in the browser. The URL will look like this:

`http://localhost:57505/UpdateBook.cshtml?id=1`

16. You should then see the data displayed for the first book in the table.

Update a Book

17. Add code in the initial Razor code block to respond to a POST and update the selected record in the database.

```

if (IsPost)
{
    bookID = Request.Form["id"];
    title = Request.Form["title"];
    author = Request.Form["author"];
    var db = Database.Open("BookData");
    string cmd = "UPDATE Book SET Title=@0, Author=@1 WHERE ID=@2";
    int numrow = db.Execute(cmd, title, author, bookID);
    message = string.Format("{0} row(s) updated", numrow);
}

```

18. Test your application by updating some of the records you've added. You already have quite a bit of functionality in your application, but it is rather inconvenient for the user, who has to manually supply a book ID in the query string. Your application is now at Step 2 and you have completed Part 1.

Part 2: Finish Update and Provide Delete Functionality

1. In **ShowBooks.cshtml** modify the call to **GetHtml()** to specify the columns in the WebGrid. Provide links to Update and Delete pages in the grid.

```

@grid.GetHtml(
    tableStyle: "grid",

```

```

columns: grid.Columns(
    grid.Column(format:
        @<a href=~ /UpdateBook.cshtml?id=@item.ID>Update</a>),
    grid.Column("ID"),
    grid.Column("Title"),
    grid.Column("Author"),
    grid.Column(format:
        @<a href=~ /DeleteBook.cshtml?id=@item.ID>Delete</a>)
)
)

```

2. Launch in the browser and test that the Update link works. The book ID in the query string is now provided automatically.
3. Add a new page **DeleteBook.cshtml**.
4. Provide a title “Delete a Book” for the page and also display it in a <h1> header.
5. Provide code in initial Razor code block to obtain the book ID from the query string and store the title and author in variables so that they can be displayed. (It will be identical code to what you used in **UpdateBook.cshtml**.)
6. Provide markup to display the data for the book to be deleted, allowing the user to make sure that the right book will be deleted. We don’t use textboxes, since the data is not being changed, but elements.

```

<head>
    <meta charset="utf-8" />
    <title>Delete a Book</title>
    <style>
        div { margin: 5px; }
    </style>
</head>
<body>
    <h2>Delete a Book</h2>
    <form method="post">
        <div>
            <label for="id">ID:</label>
            <input id="id" name="id" value="@bookID" />
        </div>
        <fieldset>
            <legend>Book Data</legend>
            <div>
                <span>Title</span>
                <span>@title</span>
            </div>
            <div>
                <span>Author</span>
                <span>@author</span>
            </div>
        </fieldset>
        <div>
            <input type="submit" value="Delete Book" />
        </div>
    </form>

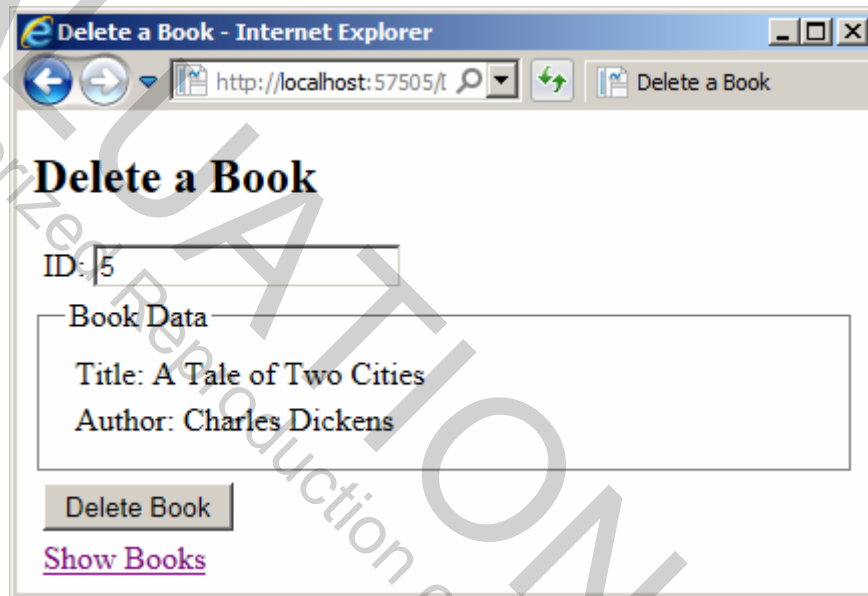
```

```

        <div>
            @message
        </div>
    </form>
    <div>
        <a href="ShowBooks.cshtml">Show Books</a>
    </div>
</body>

```

7. Test your new functionality so far by launching **ShowBooks.cshtml** and trying out a delete link.



8. Finally, add code in the initial Razor block to respond to a POST request to delete the selected row from the database, using an appropriate SQL query.

```

if (IsPost)
{
    bookID = Request.Form["id"];
    var db = Database.Open("BookData");
    string cmd = "DELETE FROM Book WHERE ID=@0";
    int numrow = db.Execute(cmd, bookID);
    message = string.Format("{0} row(s) deleted", numrow);
}

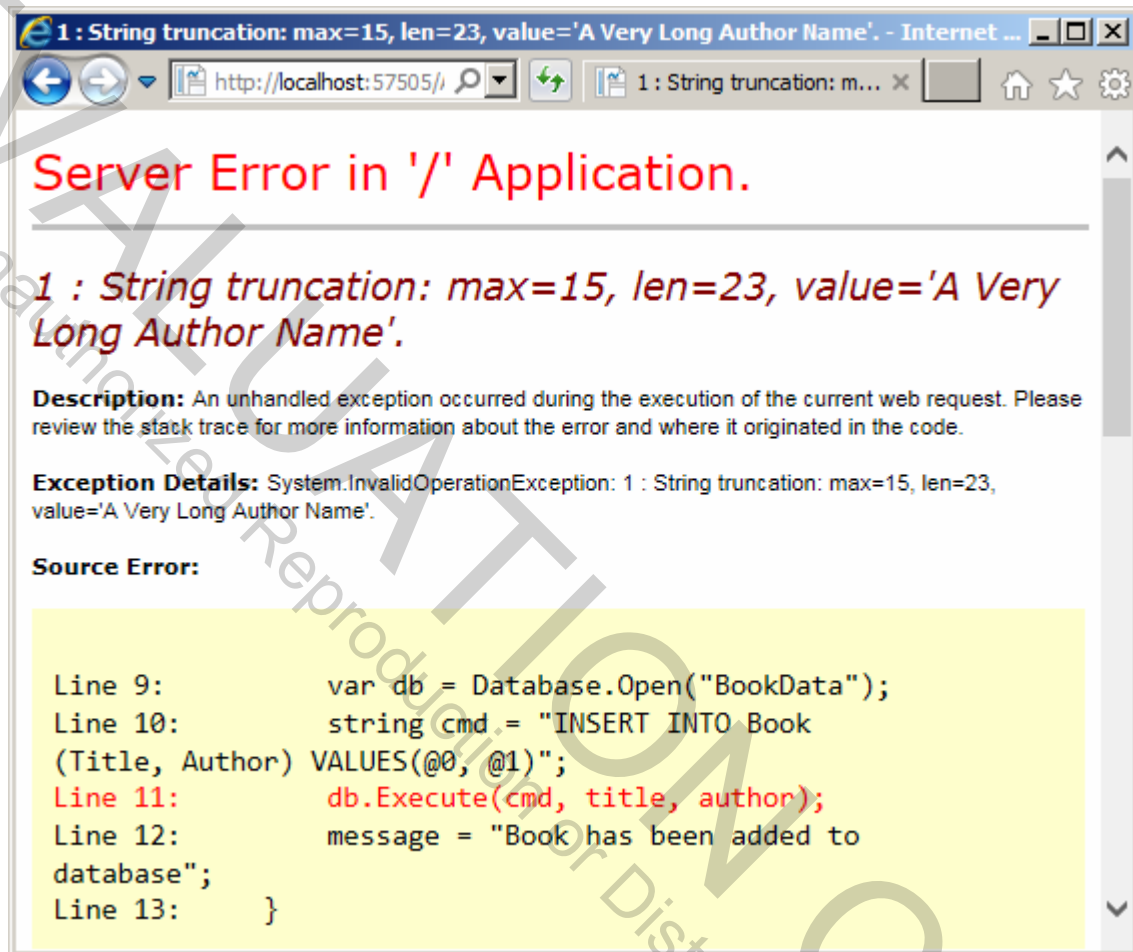
```

9. Test your application. You should be able to add, update and delete records from the database. Try some error conditions, such as leaving out a field or having a title our author name longer than allowed in the database. You will make your application more robust in Part 3.

Part 3: Exception Handling and Validation

1. In this final part of the lab we will make the application more robust. First, let's look at an example of entering bad data, causing an error. Try adding a new book with an

author name that is too long (more than 15 characters). You will hit a runtime error with a message that is not user friendly, and there is no opportunity provided to correct the error.



2. In **AddBook.cshtml** provide exception handling code. If there is an exception (which for this example would be thrown by the database engine), assign the error message to the variable **message**.

```

try
{
    var db = Database.Open("BookData");
    string cmd = "INSERT INTO Book (Title, Author) VALUES(@0, @1)";
    db.Execute(cmd, title, author);
    message = "Book has been added to database";
}
catch (Exception ex)
{
    message = ex.Message;
}

```


- Try running again and assign a name that is too long in the Author field. Now the exception is caught, and the user will have a chance to enter corrected data. See screen capture.

Add a New Book

Title:

Author:

1 : String truncation: max=15, len=23, value='A Very Long Author Name'.

[Show Books](#)

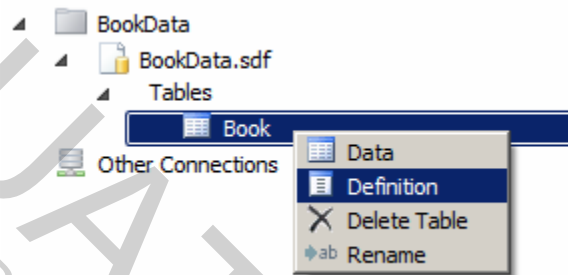
- Enter a shorter name for the author, and try again. This time it will work!

My Favorite Books

	<u>ID</u>	<u>Title</u>	<u>Author</u>		
Update	1	Long Walk to Freedom	Nelson Mandela	Delete	
Update	2	A Walk for Climate Justice	Alan Burns	Delete	
Update	3	Lost Horizon	James Hilton	Delete	
Update	4	Nosso Lar	Chico Xavier	Delete	
Update	7	The Messengers	Chico Xavier	Delete	
Update	8	My Wonderful Book	Shorter Name	Delete	

[Add a Book](#)

5. Provide similar exception handling code in **UpdateBook.cshtml** and in **DeleteBook.cshtml**. Note that there are two places in each file for exception handling, depending on whether there is a POST or not.
6. Test your application at this point for various error conditions.
7. Finally, let's supply validation code of the kind we've done before. Begin with the file **AddBook.cshtml**. Add validation rules at the top of the initial Razor code block. You can obtain the proper maximum lengths for Title and Author from the schema of the database. In WebMatrix examination the Definition of the Book table.



```
Validation.RequireField("title", "Title required");
Validation.Add("title", Validator.StringLength(30));
Validation.RequireField("author", "Author required");
Validation.Add("author", Validator.StringLength(15));
```

8. Before processing posted form data, check that it is valid.

```
if (IsPost && Validation.IsValid())
{
    ...
}
```

9. In the markup provide appropriate field validation messages.

```
<div>
    <label for="title">Title:</label>
    <input id="title" name="title" value="@Request.Form["title"]" />
    @Html.ValidationMessage("title")
</div>
<div>
    <label for="author">Author:</label>
    <input id="author" name="author" value="@Request.Form["author"]" />
    @Html.ValidationMessage("author")
</div>
```

10. Provide the styling you want for field validation messages.

```
<style>
    div { margin: 5px; }
    .field-validation-error { color:red; }
</style>
```

11. Provide validation in **UpdateBook.cshtml**. This is handled a little differently than in **AddBook.cshtml** in the initial code block. We don't need to supply the validation rules unless we are responding to a POST. We test for validity before using the retrieved form data.

```

if (IsPost)
{
    Validation.RequireField("title", "Title required");
    Validation.Add("title", Validator.StringLength(30));
    Validation.RequireField("author", "Author required");
    Validation.Add("author", Validator.StringLength(15));

    bookID = Request.Form["id"];
    title = Request.Form["title"];
    author = Request.Form["author"];

    if (Validation.IsValid())
    {
        try
        {
            var db = Database.Open("BookData");
            string cmd =
                "UPDATE Book SET Title=@0, Author=@1 WHERE ID=@2";
            int numrow = db.Execute(cmd, title, author, bookID);
            message = string.Format("{0} row(s) updated", numrow);
        }
        catch (Exception ex)
        {
            message = ex.Message;
        }
    }
}

```

12. The rest of the validation code is the same as in **AddBook.cshtml**. Provide field validation messages.

```

<div>
    <label for="title">Title:</label>
    <input id="title" name="title" value="@title" />
    @Html.ValidationMessage("title")
</div>
<div>
    <label for="author">Author:</label>
    <input id="author" name="author" value="@author" />
    @Html.ValidationMessage("author")
</div>

```

13. And an appropriate style.

```

<style>
    div { margin: 5px; }
    .field-validation-error { color:red; }
</style>

```

14. Test thoroughly. You are all done!



7400 E. Orchard Road, Suite 1450 N
Greenwood Village, Colorado 80111
Ph: 303-302-5280
www.ITCourseware.com