MX



macromedia®
COLDFUSION® MX 7

Getting Started Building ColdFusion MX Applications

Trademarks

1 Step RoboPDF, ActiveEdit, ActiveTest, Authorware, Blue Sky Software, Blue Sky, Breeze, Breezo, Captivate, Central, ColdFusion, Contribute, Database Explorer, Director, Dreamweaver, Fireworks, Flash, FlashCast, FlashHelp, Flash Lite, FlashPaper, Flex, Flex Builder, Fontographer, FreeHand, Generator, HomeSite, JRun, MacRecorder, Macromedia, MXML, RoboEngine, RoboHelp, RoboInfo, RoboPDF, Roundtrip, Roundtrip HTML, Shockwave, SoundEdit, Studio MX, UltraDev, and WebHelp are either registered trademarks or trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words, or phrases mentioned within this publication may be trademarks, service marks, or trade names of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

This product includes code licensed from RSA Data Security.

Third-Party Information

This guide contains links to third-party websites that are not under the control of Macromedia, and Macromedia is not responsible for the content on any linked site. If you access a third-party website mentioned in this guide, then you do so at your own risk. Macromedia provides these links only as a convenience, and the inclusion of the link does not imply that Macromedia endorses or accepts any responsibility for the content on those third-party sites.

Copyright © 1999–2005 Macromedia, Inc. All rights reserved. U.S. Patents Pending. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without written approval from Macromedia, Inc. Notwithstanding the foregoing, the owner or authorized user of a valid copy of the software with which this manual was provided may print out one copy of this manual from an electronic version of this manual for the sole purpose of such owner or authorized user learning to use such software, provided that no part of this manual may be printed out, reproduced, distributed, resold, or transmitted for any other purposes, including, without limitation, commercial purposes, such as selling copies of this documentation or providing paid-for support services.
Part Number ZCF70M200

Acknowledgments

Project Management: Randy Nielsen

Writer: Anne Sandstrom

Editors: Linda Adler, Noreen Maher

Production Management: Patrice O'Neill,

Media Design and Production: John Francis, Adam Barnett

Special thanks to Sawako Gensure, Seungmin Lee, Takashi Koto, Nozomi Kugita, Masayo Noda, Hiroshi Okugawa, Bowne Global Solutions

First Edition: January 2005

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

INTRODUCTION 7

PART I: Welcome to ColdFusion

CHAPTER 1: Introducing ColdFusion MX 11

About ColdFusion MX 11
How ColdFusion processes pages 12
Building applications with CFML 13

CHAPTER 2: CFML Basics 15

Working with ColdFusion pages 15
About CFML elements 18
Working with CFML expressions 23
About conditional processing 27
Processing form data 28
Commenting your code 30

CHAPTER 3: Database Fundamentals 31

About database basics 31
About SQL 34
Using SQL with ColdFusion 36

PART II: Building a ColdFusion Application

CHAPTER 4: Lesson 1: Preparing to Build the Sample Application 41

Application development steps 41
Exercise 1: Determining the application functional requirements 42
Exercise 2: Determining the data requirements 43
Exercise 3: Designing the database for your application 43
Exercise 4: Developing the sample application 46

CHAPTER 5: Lesson 2: Configuring Your Development Environment 49
Exercise 1: Verifying the tutorial file structure 49
Exercise 2: Setting up your development environment 50
Exercise 3: Configuring a database connection 52
Exercise 4: Configuring debugging options 55
Summary 57

CHAPTER 6: Lesson 3: Retrieving Data 59
Exercise 1: About the trip list 60
Exercise 2: Retrieving the information for the trip list 61
Exercise 3: Displaying the query result using the cfoutput tag 62
Exercise 4: Writing structured, reusable code 63
Exercise 5: Creating additional queries 66
Summary 69

CHAPTER 7: Lesson 4: Building Dynamic Queries 71
Exercise 1: Creating the trip search form 71
Exercise 2: Building a query that uses dynamic SQL 75
Exercise 3: Constructing the initial Trip Search Results page 79
Exercise 4: Testing the Trip Search Results page 81
Exercise 5: Enabling the departure and price criteria on the Trip Search form 82
Exercise 6: Testing the revised Trip Search Results page 83
Summary 83

CHAPTER 8: Lesson 5: Creating a Trip Detail Page 85
Exercise 1: Creating a Trip Detail page 86
Exercise 2: Avoiding the potential security risk when using dynamic SQL 91
Exercise 3: Linking the Trip Search Results page with the Trip Detail page 92
Exercise 4: Enhancing the look of the Trip Search Results and Trip Detail pages . . 93
Summary 95

CHAPTER 9: Lesson 6: Creating a Main Application Page 97
Exercise 1: Creating the main application page from the Trip Detail page 98
Exercise 2: Adding database maintenance buttons 100
Summary 101

CHAPTER 10: Lesson 7: Validating Data to Enforce Business Rules 103
Exercise 1: Using an HTML form to collect data 104
Exercise 2: Creating a simple action page 107
Exercise 3: About data validation 108
Exercise 4: Providing server-side validation 109
Exercise 5: Validating data on the client using ColdFusion form tags 114
Exercise 6: Dynamically populating the list of event types 119
Exercise 7: Validating the existence of the trip photo file 120
Summary 122

CHAPTER 11: Lesson 8: Implementing Browsing and Searching 123
Exercise 1: Enabling users to browse trip details 123
Exercise 2: Determining actions based on which button a user clicks. 127
Exercise 3: Enabling searching from the Trip Detail page 127
Summary. 128

CHAPTER 12: Lesson 9: Enabling Database Maintenance 129
Exercise 1: Deleting the current trip record shown on the Trip Detail page 129
Exercise 2: Adding trips with SQL INSERT statements 131
Exercise 3: Adding data using the cfinsert tag 135
Exercise 4: Updating a SQL row using the cfupdate tag. 137
Exercise 5: Linking the Trip Edit page to the main page 138
Exercise 6: Inserting values in the edit form. 139

CHAPTER 13: Lesson 10: Restricting Access to ColdFusion Applications 143
Exercise 1: About the Login Wizard 143
Exercise 2: Requiring users to log in to the Compass Travel application. 147
Summary. 148

INDEX 149

INTRODUCTION

Getting Started Building ColdFusion MX Applications is intended for anyone who needs to begin programming in the Macromedia ColdFusion MX 7 development environment. This manual includes a tutorial, which uses supporting files that are installed if you chose to install sample applications. If you did not install the ColdFusion MX sample applications, you can download the tutorial files from the ColdFusion Support center on the Macromedia website.

About Macromedia ColdFusion MX 7 documentation

The ColdFusion MX 7 documentation is designed to provide support for the complete spectrum of participants.

Documentation set

The ColdFusion MX 7 documentation set includes the following titles:

Book	Description
Installing and Using ColdFusion MX	Describes system installation and basic configuration for Microsoft Windows, Solaris, and Linux. To see this manual, go to http://www.macromedia.com/go/livedocs_cfmx7docs_installing.
Configuring and Administering ColdFusion MX	Part I describes how to manage the ColdFusion environment, including connecting to your data sources and configuring security for your applications. Part II describes Verity search tools and utilities that you can use for configuring the Verity K2 Server search engine, as well as creating, managing, and troubleshooting Verity collections. To see this manual, go to www.macromedia.com/go/livedocs_cfmx7docs_configadmin.
ColdFusion MX Developer's Guide	Describes how to develop your dynamic web applications, including retrieving and updating your data, and using structures and forms. This manual includes two volumes. To see this manual, go to www.macromedia.com/go/livedocs_cfmx7docs_dev.
Getting Started Building ColdFusion MX Applications	Contains an overview of ColdFusion features and application development procedures. This manual includes a tutorial that guides you through the process of developing a sample ColdFusion application. To see this manual online, go to www.macromedia.com/go/livedocs_cfmx7docs_gs.

Book	Description
CFML Reference	Provides descriptions, syntax, usage, and code examples for all ColdFusion tags, functions, and variables. This manual includes two volumes. To see this manual, go to www.macromedia.com/go/livedocs_cfmx7docs__cfml_reference.
CFML Quick Reference	Shows the syntax of ColdFusion tags, functions, and variables in a brief guide.

Viewing online documentation

All ColdFusion MX documentation is available online in HTML and Adobe Acrobat Portable Document Format (PDF) files. Go to the documentation home page for ColdFusion MX on the Macromedia website: www.macromedia.com. In addition, you can view the documentation in LiveDocs, which lets you add comments to pages and view the latest comments added by Macromedia, by going to www.macromedia.com/go/livedocs_cfmx7docs.

PART I
Welcome to ColdFusion

Part I provides an introduction to Macromedia ColdFusion MX 7. It defines ColdFusion MX 7 and provides an overview of the ColdFusion Markup Language (CFML). It also provides generic database concepts.

The following chapters are included in Part I:

Chapter 1: Introducing ColdFusion MX 11

Chapter 2: CFML Basics 15

Chapter 3: Database Fundamentals 31

CHAPTER 1
Introducing ColdFusion MX

This chapter introduces the core technologies that are the foundation for Macromedia ColdFusion MX. In addition, it introduces the basic concepts about ColdFusion MX, how it works, and the various components that comprise it.

Contents

About ColdFusion MX 11
How ColdFusion processes pages 12
Building applications with CFML 13

About ColdFusion MX

ColdFusion MX is a powerful web application server that lets you create robust sites and applications without a long learning curve. ColdFusion MX does not require coding in traditional programming languages (for example, C, C++, Java, XML), although it supports these traditional programming languages.

ColdFusion MX consists of the following core components:

- ColdFusion application server
- ColdFusion Markup Language (CFML)
- ColdFusion MX Administrator

The following sections describe these core components in more detail. In addition, the Getting Started Experience, which you can view by opening the Macromedia ColdFusion MX Administrator and clicking Getting Started, provides introductory information about ColdFusion and code snippets that highlight the new features of ColdFusion MX 7.

The ColdFusion application server

The ColdFusion application server is itself a web application that typically resides on the same computer as your web server software. It is the program that parses (reads and interprets) and processes supplied instructions. These instructions are passed to ColdFusion through ColdFusion pages, which use a .cfm or .cfc filename extension. A ColdFusion page looks like an HTML page, but contains special tags that instruct the ColdFusion server to perform specific operations.

The ColdFusion Markup Language

ColdFusion Markup Language (CFML) is a tag-based language similar to HTML that uses special tags and functions. With CFML, you can enhance your standard HTML files with database commands, conditional operators, and high-level formatting functions, and rapidly produce easy-to-maintain web applications.

CFML looks similar to HTML: it includes start and end tags, and each tag is enclosed in angle brackets. All ending tags are preceded with a forward slash (/) and all tag names are preceded with *cf*.

The ColdFusion MX Administrator

You use the ColdFusion MX Administrator to configure and maintain the ColdFusion application server. It is a web-based application that you can access using any web browser, from any computer with an Internet connection.

You can manage the following configuration options with the ColdFusion MX Administrator:

- ColdFusion data sources
- Debugging output
- Server settings
- Application security

For further details about the ColdFusion MX Administrator, see *Installing and Using ColdFusion MX* or *Configuring and Administering ColdFusion MX*.

How ColdFusion processes pages

The following steps explain how the ColdFusion server processes a ColdFusion page:

1. The ColdFusion server looks at the content of the page and searches for the following ColdFusion instructions:
   - Tag names that begin with *cf*.
   - Variables and functions that are always surrounded by number signs (#).
2. If the ColdFusion server finds any HTML or plain text in the page, the ColdFusion server returns it to the web server unchanged.
3. The ColdFusion server processes all the ColdFusion instructions found, and returns any remaining results to the web server. The web server then sends the entire output to the browser.

Building applications with CFML

You build ColdFusion applications as a series of pages that use CFML. Developers can extend this language by creating their own custom tags or user-defined functions (UDF), or by integrating COM, C++, and Java components, such as Java Server Page (JSP) tag libraries.

Interacting with data sources

ColdFusion applications can interact with any database that supports a JDBC technology-based driver. A JDBC technology-based driver uses an Application Programming Interface (API) to execute SQL statements to databases on most platforms. However, ColdFusion is not limited to JDBC data sources. You can also interact with existing Open Database Connectivity (ODBC) data sources by using ODBC Socket, a driver that interacts with an existing ODBC driver.

Interacting with mobile devices

ColdFusion MX 7 lets you create applications that can respond to events and messages from diverse sources, including instant messaging, mobile phone Short Message Service (SMS) text messages, Internet socket requests, and system events. For information, see Chapter 42, "Using Event Gateways" in *ColdFusion MX Developer's Guide*.

Development tools

You can code your ColdFusion application with NotePad or any HTML editor; however, Macromedia recommends that you build your applications using Macromedia Dreamweaver MX. Dreamweaver MX offers features and wizards that enhance ColdFusion development. For more information about Dreamweaver MX, see Chapter 5, "Lesson 2: Configuring Your Development Environment," on page 49.

CHAPTER 2
CFML Basics

This chapter introduces the basic elements of CFML, including how to create Macromedia ColdFusion pages, and use variables, functions, conditional processing, and form processing.

Contents

Working with ColdFusion pages 15

About CFML elements 18

Working with CFML expressions 23

About conditional processing 27

Processing form data 28

Commenting your code 30

Working with ColdFusion pages

ColdFusion pages are plain text files that you use to create web applications. You can create your ColdFusion applications by writing all the code manually or by using wizards (provided with some editors) to generate the majority of the code for you.

You can use the following editors to create your ColdFusion pages:

- Macromedia Dreamweaver MX
- Macromedia HomeSite+
- Macromedia ColdFusion Studio
- Any HTML editor
- Windows Notepad
- Vi or Emacs (UNIX systems)

The best choice for creating ColdFusion pages is Macromedia Dreamweaver MX. Dreamweaver MX includes many CFML features for building applications, such as rapid visual development, robust CFML editing, and integrated debugging. Dreamweaver MX also includes a copy of HomeSite+ for users who are familiar with developing their application code using ColdFusion Studio or HomeSite 5. HomeSite+ combines all the features of ColdFusion Studio and HomeSite 5, along with support for the latest ColdFusion MX tags. For more information, see Chapter 5, "Lesson 2: Configuring Your Development Environment," on page 49.

Note: This manual describes how to create ColdFusion applications by writing your code manually. It does not address how to create ColdFusion pages by generating code with wizards. For information about using wizards to generate CFML code, see the product documentation for Dreamweaver MX and HomeSite+.

Creating a ColdFusion page

Creating a ColdFusion page involves using tags and functions. The best way to understand this process is to create a ColdFusion page.

In the following procedure, you will create a simple ColdFusion page by using HTML tags, one ColdFusion tag, and two ColdFusion functions. The following table briefly explains the ColdFusion tags and functions:

Element	Description
`Now()`	A function supported in CFML that you can use to retrieve information from your system. You will use the `Now()` function in the following procedure to return the current date that is retrieved from your system.
`DateFormat()`	A function that instructs ColdFusion to format the date returned by the `Now()` function.
`cfoutput`	A ColdFusion tag that you use to return dynamic data (data retrieved from a database) to a web page. You will use the `cfoutput` tag in the following procedure to display the current date retrieved from your system.

Note: ColdFusion tags and functions are considered primary elements of CFML. You will learn more about these elements and others later in this manual.

To create a ColdFusion page:

1. Open your editor and create a blank file.
2. Enter the following code on the page:

```
<html>
<head>
<title>A ColdFusion Page</title>
</head>
<body>
<strong>Hello world, this is a ColdFusion page.</strong>
<br>
<cfoutput> Today's date is #DateFormat(Now())# </cfoutput>
</body>
</html>
```

Saving your ColdFusion page

In order for the ColdFusion server to process the page, you must save the ColdFusion page on a computer where the ColdFusion MX is installed. If you are creating your pages on a local server (on which ColdFusion is running), you can save the pages locally; if you are using a remote server, you must save your pages on that server.

If you are using the ColdFusion MX J2EE configuration, you typically save ColdFusion pages under the ColdFusion *web application root*. For example, in the default directory structure when you use the J2EE configuration with JRun, you save pages under *jrun_root*/servers/cfusion/cfusion-ear/cfusion-war. When you use JRun, you can also run the Web Server Configuration Tool and save pages under your web root directory.

Tip: ColdFusion MX on Macintosh OS X uses the J2EE configuration.

To save the code that you typed to create a ColdFusion page:

1. Create a directory called test under the *web_root* or *web_application_root* directory.
2. In the test directory, save the file as cfpage.cfm.

Browsing your code

To ensure that the code you wrote is working as expected, you must view the ColdFusion page in a browser. The following procedure describes how to view the ColdFusion page that you created earlier.

To view the ColdFusion page:

1. Open a web browser and go to the following URL:

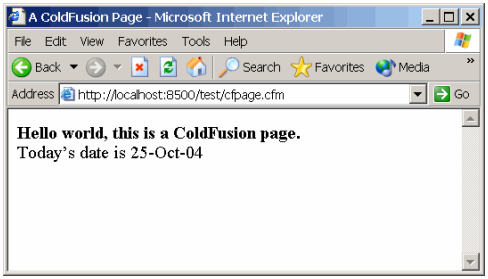
   http://localhost/test/cfpage.cfm

   Note: If you are using the built-in web server, specify the port to use in the URL as follows: `http://localhost:8500/test/cfpage.cfm`.

   The address localhost is only valid when you view pages locally. The URL for a remote site would include the server name or IP address of the server where ColdFusion is installed; for example, http://<serveripaddress>/test/cfpage.cfm.

   If you are using the ColdFusion MX J2EE configuration, you may also need to include a context root in the URL; for example, http://<server>/<context-root>/<page>.cfm. For example, if you deploy an EAR file and use the default context root of cfmx, you specify http://localhost/cfmx/test/cfpage.cfm.

The following figure shows the cfpage.cfm page in the browser:



2. Do the following tasks:

   a. View the source code that was returned to the browser. In most browsers, you can view the source by right-clicking on the page, and then selecting View Source.

   b. Compare the browser source code with the source code that appears in your editor. Notice that the CFML tags were processed on the page but did not appear in the source that was returned to your browser.

      As described in Chapter 1, "Introducing ColdFusion MX," on page 11, ColdFusion processes all the instructions (CFML tags and functions) it receives on a page, and then returns the results of the instructions that your browser can interpret and display.

About CFML elements

CFML consists of two primary language elements: tags and functions. *Tags* let you perform operations such as accessing a database. *Functions* can return data and do other operations like retrieving the system date. Almost everything that you want to accomplish with ColdFusion involves using tags and functions.

You use another important element known as a variable. Variables are an important part of most programming languages and are equally important with CFML. *Variables* let you store information in memory and enable you to pass data.

The following sections describe how to use these three elements.

Tags

You can think of tags as commands that you use to instruct the ColdFusion server to perform operations. These operations might include selecting data from a database, reading a file that resides on the server, or showing the results of processing.

Tag syntax

As discussed in Chapter 1, "Introducing ColdFusion MX," on page 11, ColdFusion tags are similar to HTML tags. ColdFusion tags are enclosed in angle brackets and often have a start and end tag. The start tag encloses the tag name in brackets, like this:

```
<tagname>
```

Most often the end tag encloses the tag name in brackets and includes a forward slash (/), like this:

```
</tagname>
```

The information processed by ColdFusion is placed between the start and end tag, like this:

```
<tagname>
  info to be processed ...
</tagname>
```

ColdFusion tags, for the most part, share these common characteristics:

- All start with *cf*.
- Include a start and end tag.
- Use attributes (like HTML tags), and most attributes have values.

Some ColdFusion tags, such as `cfset`, omit the ending tag. This type of tag uses one set of angle brackets and places all the required information between the left (<) and right (>) angle brackets, like this:

```
<cfset name="bob">
```

For a complete list of tags and their syntax, see *CFML Reference*.

Tag attributes

Tag attributes instruct ColdFusion about the details of an operation. For example, to update a database table, ColdFusion requires specific information about the database, such as the database name and the table name. The code required to write this type of statement might look like this:

```
<cfupdate datasource="mydb" tablename="mytable">
```

where `datasource` and `tablename` are attributes of the `cfupdate` tag and `"mydb"` and `"mytable"` are attribute values.

For a complete list of tags and their attributes, see *CFML Reference*.

Functions

Typically, a function acts on data. It can generate a value or a set of values, usually from some input. You can perform the following operations (actions) with functions:

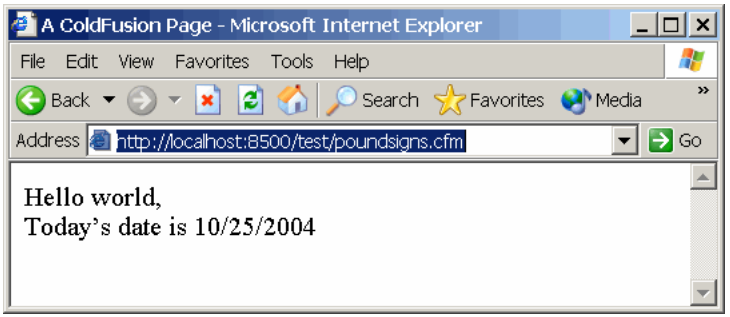
- Manipulate data and time values
- Examine a value or variable
- Display and format information
- Manipulate string data
- Retrieve system information and resources
- Perform mathematical operations

Functions and number signs

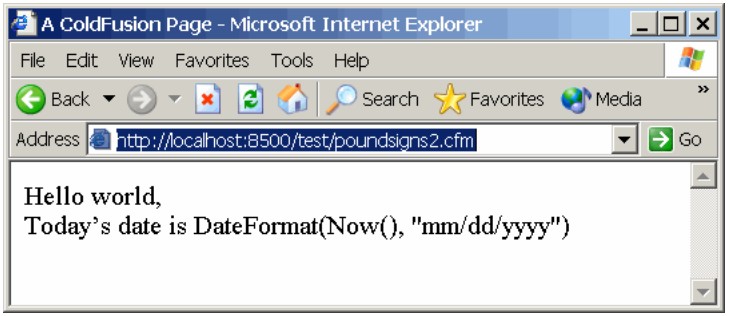
You use number signs (#) with functions to display the results of a function on the page. Number signs tell the ColdFusion server to evaluate the content between the number signs and display the value, for example:

```
<cfoutput>
  Hello world, <br>
  Today's date is #DateFormat(Now(), "mm/dd/yyyy")#
</cfoutput>
```

The following figure shows the output of this example:



If you did not include the number signs around the `DateFormat(Now(), "mm/dd/yyyy")` function, ColdFusion would not evaluate the function and the previous example would display your source code, as follows:



For more information about how to use number signs with functions, see *ColdFusion MX Developer's Guide*.

Functions and parentheses

All functions have parentheses, regardless of whether the function acts on data. Consider the following function:

```
#Now()#
```

If you put anything inside the parentheses of the `Now()` function, an error would occur. The `Now()` function returns an unformatted date and time. However, you can format the results of this function with other functions, such as the `DateFormat()` or `TimeFormat()` functions.

Using functions on values

Usually, a function performs an operation on a value, and the value can include the value of a variable. For example, to format the value of a variable that contains a value in dollars, the code to write this statement might look like this:

```
#DollarFormat(price)#
```

The `DollarFormat` function returns a value as a string and formats that value with two decimal places, a thousands separator, and a dollar sign. The number signs (#) around the function instruct ColdFusion to evaluate the content between the number signs and display the value.

Nesting functions

Functions can generate data, as well as act on data. Consider the following example:

```
#DateFormat(Now(), "mm/dd/yyyy")#
```

In this example, the `Now()` function generates the date, and then the `DateFormat` function formats the date.

Variables

Variables let you store data in memory on the server. Variables always have a name and a value. You can assign a value to a variable, or you can instruct ColdFusion to assign variable values based on data that it retrieves from a data source, such as a database table.

Naming variables

You must use the following rules for naming ColdFusion variables:

- Names are case-insensitive (uppercase, lowercase, or mixed case).
- Names can contain only letters, numbers, and underscore characters.
- Each name must begin with a letter.
- Special characters (such as double-quotation marks ("), reserved names (such as functions and tags), and spaces are not allowed.

Ways to use variables

You can use variables for the following purposes:

- Store data collected from a form.
- Store results of a calculation (such as the number of database records returned).
- Use as input to a function.

Creating variables with the cfset tag

ColdFusion lets you create variables as you need them. You create the variable (name and value) using the `cfset` tag. This tag has the following syntax:

```
<cfset variable_name = value>
```

In the following examples, the variables are assigned a string literal value. All string literal values are surrounded by double-quotation marks.

```
<cfset my_first_name = "Kaleigh">
<cfset my_last_name = "Smith">
```

In the next example, ColdFusion uses the values of the `my_first_name` and `my_last_name` variables to set the value for the `my_full_name` variable in the last line of code. The ampersand (`&`) string operator joins the variables, and the space surrounded by double-quotation marks (`" "`) adds a space between the variables.

```
<cfset my_first_name = "Kaleigh">
<cfset my_last_name = "Smith">
<cfset my_full_name = variables.my_first_name & " " & variables.my_last_name>
```

Tip: String values assigned to a variable must be enclosed in single-quotation marks (') or double-quotation marks ("). Numeric or Boolean values assigned to a variable do not require single- or double-quotation marks.

So far, all the variable examples have shown local variables. *Local variables* are variables that you can use only on the current ColdFusion page. The previous example used a variables prefix to reference an existing variable on the page. Using a prefix when referencing a variable is important because ColdFusion supports many types of variables. Use the following syntax to reference a local variable:

```
variables.variablename
```

Because ColdFusion lets you use the same name with variables of more than one type, ColdFusion relies on *scope referencing*. In scope referencing, you preface the variable's name with the scope when you refer to that variable.

Other variables and their scope

Each type of variable that ColdFusion supports has it own scope, or where it can be referenced, and its own way of referencing that variable type. The following table identifies some of the more common types of variables and their prefixes:

Scope	Prefix	Description
Variables (local variable)	Variables	Variables created using a `cfset` or `cfparam` tag. Most often you define the variable on the current page or on a page that you include using the `cfinclude` tag.
Form	Form	Data entered in tags in an HTML form or ColdFusion form and processed on the action page.
URL	URL	Variables passed to a page as URL string parameters.
Query	QueryName	Variables that have names based on the column names that you select in the database table. The values are created when you execute the query that selects data from the database.

You will use these other types of variables in Part II of this manual. For additional information about variables, see *CFML Reference*.

Displaying variable output

Output is what remains after the ColdFusion server processes the CFML tags on a page. Usually the output has two parts:

- Information that the user sees (for example, a confirmation message)
- Information that is stored by the server as a result of processing (for example, user input collected from a form)

One of the tags that ColdFusion provides to display output is the `cfoutput` tag. The `cfoutput` tag instructs ColdFusion to process all the code between the `cfoutput` start and end tags. The syntax for the `cfoutput` tag looks like this:

```
<cfoutput>
  {normal html, text, and ColdFusion processing instructions}
</cfoutput>
```

To return the value of a variable, you must always surround the variable name with number signs (#) and place the variable name between the `cfoutput` start and end tags. For example, the following code creates a variable and instructs the ColdFusion server to return the value of the variable:

```
<cfset my_first_name = "Kaleigh">
<cfset my_last_name = "Smith">
<cfset my_full_name = variables.my_first_name & " " & variables.my_last_name>
<cfoutput>
  #variables.my_full_name#
</cfoutput>
```

The following text is the output:

Kaleigh Smith

Working with CFML expressions

Expressions are an important part of the ColdFusion language. *Expressions* are a collection of different elements, ColdFusion variables, functions, and operators. You can think of them as strings of text that consist of one or more of the following elements:

- Literal text (string), numbers, dates, and other values
- Variables
- Functions
- Operators (`&` for joining statements, + for addition, and so on)

This chapter includes many examples of expressions; for example:

- `#variables.my_full_name#`
- `DateFormat(Now())`
- `my_first_name= "Kaleigh"`

When you build expressions in ColdFusion, you can include simple and complex elements; how you represent these elements determines how ColdFusion processes your application.

Building expressions

In ColdFusion, you build expressions as you need them. The expressions can include simple elements, such as the expressions shown previously, or they can include complex elements, such as arithmetic functions, strings, and decision operators. (You build some complex expressions in Part II of this manual.)

As mentioned, it is important that elements are identified properly in your expression so that ColdFusion processes them as expected, and you avoid unnecessary errors. When you write expressions, consider the following coding practices:

- Use of consistent character case
- When to use number signs (#)
- When quotation marks are needed

Specifying a consistent character case

Because the ColdFusion server is case-insensitive, you can write expressions using all uppercase, all lowercase, or mixed case. However, for code readability and consistency, you should use the same character case in all your programs. If you write your programs using the same case rules, you might prevent errors from occurring when you combine CFML on a page with case-sensitive languages, such as JavaScript.

Specifying number signs to denote functions or variables

In ColdFusion, you specify number signs (#) to denote functions and variables within a string of text. You use number signs to show the results of the function or variable on the page. Number signs instruct the ColdFusion server to evaluate the function (or variable) between the number signs and display the value. The value of the function (or variable) appears in the browser as a result.

The following are some common ways to use number signs:

- In the following example, you include the number signs to return the value to a page:

  ```
  <cfoutput> Hello #variables.my_first_name# </cfoutput>
  ```

  If you omit the number signs, the text, not the value, appears on the page.
- In the following example, you do not include the number signs because you are using the `cfset` tag to assign one variable's value to another value:

  ```
  <cfset my_full_name = variables.my_first_name & " " &
    variables.my_last_name>
  ```

- To display a number sign on a page, you must designate the number sign as a literal character. You do this by using two number signs (##); for example:

  ```
  <cfoutput>
    ##1: Your name.
  </cfoutput>
  ```

  The result is the following output:

  #1. Your name.

For more information and examples on using number signs in expressions, see *ColdFusion MX Developer's Guide*.

Specifying quotation marks around values

When you assign literal values to variables, you must surround the literal value with single- or double-quotation marks. ColdFusion interprets the content between the quotation marks as a literal value and assigns that value to the variable; for example:

```
<cfset my_first_name = "Kaleigh">
<cfset my_last_name = "Smith">
<cfset my_age = 5>
```

ColdFusion instantiates the variable `my_first_name` to the string literal `Kaleigh`. Further, `Smith` is assigned to the variable `my_last_name` and `5` is assigned to age.

When referencing a variable by its name, you do not surround the name with quotation marks. In the following example, when you concatenate literal text and variables using the & operator, you don't surround the variable references with quotation marks:

```
<cfset the_string = "My name is " & variables.my_first_name &
  " and my age is " & variables.my_age>
```

`My name is` is literal text, and you, therefore, surround it with quotation marks. The variable references `variables.my_first_name` and `variables.my_age` are not surrounded by quotation marks. ColdFusion uses the values of the referenced variables (`Kaleigh` and `5`, respectively) when assigning the value to the variable `the_string`.

To display quotation marks on a page as literal characters, you must use two consecutive quotation marks; for example:

```
<cfset mystring = "We all shouted ""Happy Birthday"" when he entered the
  room.">
<cfoutput>
  #mystring#
</cfoutput>
```

The result is the following output:

We all shouted "Happy Birthday" when he entered the room.

Specifying operators in expressions

In ColdFusion, you use operators to test conditions; for example, you use the `IS` operator to test for equality. When you use operators in expressions, you must only use supported logical operators that ColdFusion can interpret properly. For example, if you use the greater than operator (>) or the less than operator (<), ColdFusion interprets them as the start or end of a tag rather than as an operator.

The following table lists the unsupported logical operators and their equivalent ColdFusion operators:

Unsupported logical operator	Equivalent ColdFusion decision operator	Description
=	IS, EQUAL, EQ	Tests for equality.
<	LT, LESS THAN	Tests for less than.
<=	LTE, LE, LESS THAN OR EQUAL TO	Tests for less than or equal to.
>	GT GREATER THAN	Tests for greater than.
>=	GTE, GREATER THAN OR EQUAL TO	Tests for greater than or equal to.
<>	IS NOT, NEQ, NOT EQUAL	Tests for nonequality.
	CONTAINS	Tests whether a value is contained within a second value.
	DOES NOT CONTAIN	Tests whether a value is not contained within a second value.

Arithmetic operators

The following table lists the arithmetic operators that ColdFusion supports:

Operators	Description
+, -, *, /	The basic arithmetic operators: addition, subtraction, multiplication, and division. In the case of division, the right operand cannot be zero.
+, -	Unary arithmetic operators for setting the sign of a number as either positive or negative (+ or -).
Mod	Returns the remainder (modulus) after a number is divided by a divisor. The result has the same sign as the divisor. The right operand cannot be zero. For example, 11 MOD 4 is 3.
\	Divides two integer values. Use the \ (backslash character) to separate the integers. The right operand cannot be zero. For example, 9 \ 4 is 2.
^	Returns the result of a number raised to a power (exponent). Use the ^ (caret) to separate the number from the power. The left operand cannot be zero. For example, 2 ^ 3 is 8.

String operator

The following table describes the one ColdFusion string operator that is a concatenation operator:

Operator	Description
&	Concatenates strings.

About conditional processing

So far, all the coding examples shown in this chapter are considered linear coding examples. *Linear code* is when ColdFusion executes code starting with the first line on the page, and processes every line in order. Although you will use linear code in your applications, you will often write code that performs various actions based on conditions, such as the following:

- Determine whether a user entered a value in a form field.
- Display results based on user input.
- Display messages based on the time of day.

You use conditional processing to customize the behavior of your application. *Conditional processing* facilitates decision making and lets you control how the code on a page is processed.

In ColdFusion, you implement conditional processing with flow control tags. These tags are similar to other programming language control elements, such as `if`, `then`, and `else`. When using these tags, you can facilitate decision making in your code. The most fundamental tags used to control code execution are the `cfif`, `cfelse`, and `cfelseif` tags. Because you will see and use these tags in Part II of this manual, the following sections provide a basic introduction on how to use these tags. For more information about other conditional processing tags, including tags for looping, see *ColdFusion MX Developer's Guide*.

Using the cfif tag to evaluate True or False conditions

To create statements that let you evaluate conditions and perform an action based on the result, you use the `cfif` tag to create a `cfif` statement. The basic syntax is as follows:

```
<cfif expression>
  HTML and CFML tags executed if expression is True.
</cfif>
```

In this example, ColdFusion only executes the code inside the `cfif` statement if the expression evaluates to True. To perform actions if the expression is False, you can use the `cfelse` tag. For example, if the following `cfif` expression evaluates to False, the code between the `cfelse` tag and the ending `cfif` tag is processed:

```
<cfif expression>
  HTML and CFML tags executed if expression is True.
<cfelse>
  HTML and CFML tags executed if expression is False.
</cfif>
```

Using the cfelseif tag to evaluate multiple expressions

To evaluate multiple expressions in a `cfif` statement, you can use `cfelseif` and `cfelse` tags in your statement; for example:

```
<cfif expression 1>
  HTML and CFML tags executed if expression 1 is True.
<cfelseif expression 2>
  HTML and CFML tags executed if expression 2 is True.
<cfelse>
  HTML and CFML tags executed for expression(s) that is False.
</cfif>
```

The following example shows you how you can evaluate multiple expressions using these tags. This example uses a form in which users can enter their state to determine their state tax:

```
<cfoutput>
<cfif form.state IS "MA">
  #form.state# State Tax: 8.5%
<cfelseif form.state IS "VA">
  #form.state# State Tax: 8.2%
<cfelse>
  #form.state# State Tax Unknown
</cfif>
</cfoutput>
```

The output of this `cfif` statement is based on the value entered by the user. If the user enters MA in the state form field, the state tax results returned is 8.5%. If the user enters VA in the state form field, the state tax results returned is 8.2%. If the user enters any other state in the state form field, State Tax Unknown is returned.

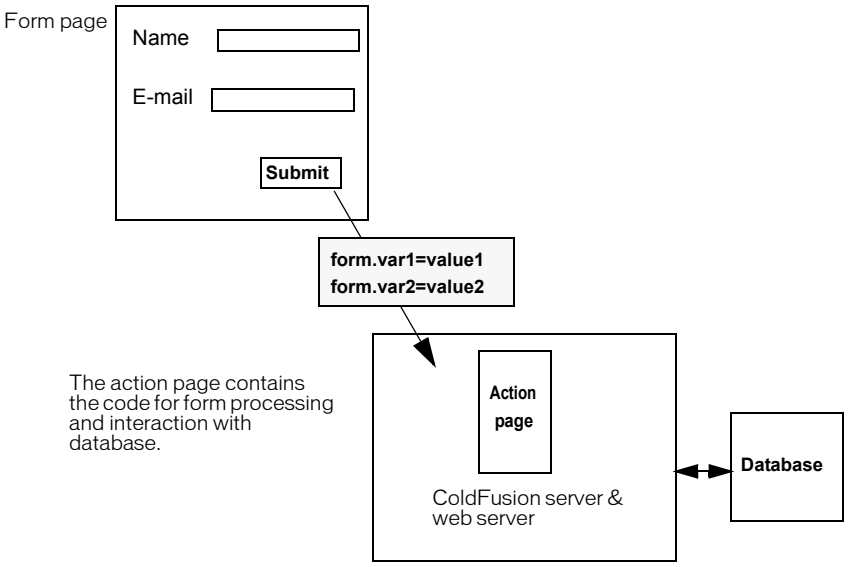
Processing form data

Virtually all web applications that gather and write information to a database use a form to accomplish that task. Forms let you collect information from a user (using an order form, registration form, and so on) and write that information to a database. Like HTML, there are two independent steps for creating a form in ColdFusion:

1. Creating the layout for the form itself.
2. Writing the code to process the submitted information.

Every form that you create in ColdFusion consist of two parts: the form page and the action page. These two pages work together to process user input. The *form page* contains the user interface elements, such as input fields and radio buttons. The *action page* handles the processing of the form page data.

When a user submits a form, the form values are stored in form variables and sent to the action page for processing. The following figure shows the relationship between the form page and action page:



In order for the form page to find its corresponding action page, the action statement in the form tag must be correct. The form tag includes the information that tells the server where to send the data that it collects. It also tells the server how to send it. To process these instructions to the server, the form tag uses the following syntax:

```
<form action="actionpagename.cfm" method="Post">
  HTML and CFML form tags
</form>
```

The first attribute (`action`) in the form tag lets you specify where to send the data. The page that you specify where to send the data is the name of the action page. The second attribute in the form tag is `method`. The only method that ColdFusion supports is `post`. All ColdFusion forms must set the method attribute to `post`.

In Part II of this manual, you use ColdFusion form tags to create forms and write collected values to a database.

Commenting your code

As in other programming languages, it is important to include comments in your code. You should comment your code for the following reasons:

- Commented code is easier to debug than code that is not commented.
- If you describe the code on the page, it is easier to make modifications.
- Commented code tends to be better organized.

Comment tag

The ColdFusion comment tag is similar to the HTML comment tag, except that it has three dashes instead of two:

```
<!--- This is a CFML comment --->
```

ColdFuions comments can wrap to more than one line. ColdFusion comments are not returned to the browser because the ColdFusion server processes and omits the comments from the page. The user will never be able to read your comments.

CHAPTER 3
Database Fundamentals

This chapter provides a quick overview of relational database concepts and terms. It describes what a database is and how it is organized. It also discusses the Structured Query Language (SQL) that you use to interact with databases.

Contents

About database basics 31

About SQL 34

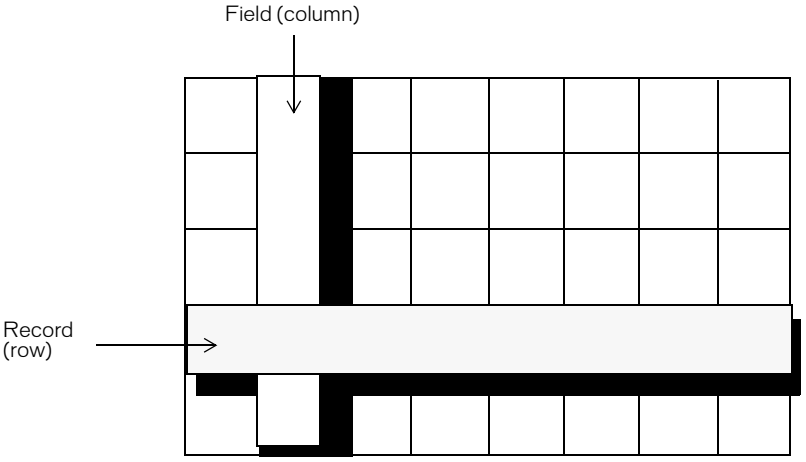
Using SQL with ColdFusion 36

About database basics

Even though you do not need a thorough understanding of database management systems to create ColdFusion applications, you must understand some basic concepts and techniques about databases. The information in this chapter will help you get started using ColdFusion.

What is a relational database?

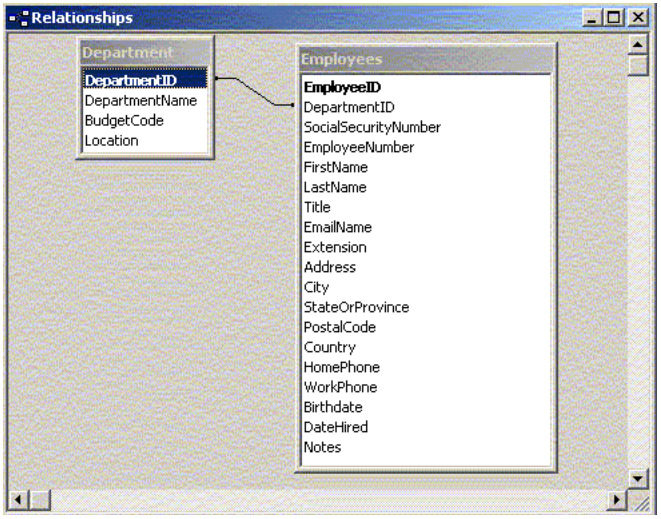
A *relational database* is a structured collection of information that is related to a particular subject or purpose, such as an inventory database or a human resources database. You use databases to manage information. Information, such as product name, cost, and on-hand inventory, is stored in a database. Within the database, you organize the data into storage containers called tables. *Tables* are made up of columns and rows. Columns represent individual *fields* in a table. Rows represent *records* of data in a table. You can think of database tables as grids, as in the following example:



Each field in the table contains one piece of information. In an employee table, for example, one column contains the employee name, another contains the employee phone number, and the other columns each store one piece of information, such as the address, city, state, zip, and salary information. Each record represents one set of related information. For example, an employee table might store information about one employee per row. The number of rows in a table represents the total number of table records.

About relational tables

In a database, you can organize data in multiple tables. For example, if you manage a database for the Human Resources department, you might have one table that lists all the information about employees and another table that lists all the departments:



You have multiple departments for employees, but you would not store the information about the departments in every employee row for the following reasons:

- The department information is the same for each employee in a given department; however, repeating the department information for each employee is redundant. Storing redundant data takes up more disk space.
- If the department information changes, you can update one occurrence. All references to that department are updated automatically.

Storing multiple occurrences of the same data is rarely a good thing. Good relational database design separates application entities into their own tables. Key values from one table are often stored in a related table rather than repeating the information. The key value is used to join the data between the tables to return the complete set of data required.

About SQL

SQL (Structured Query Language) is a language that lets you communicate with databases. For example, you can use SQL to retrieve data from a database, add data to a database, delete or update records in a database, change columns in multiple rows, add columns to tables, and add and delete tables.

Unlike other computer languages, SQL is made up of a small number of language elements that let you interact efficiently with a database. Some of the more frequently used elements include the following SQL commands:

Command	Use
SELECT	Retrieve (query) information in a database.
INSERT	Add records to a database.
UPDATE	Update information in a database.
DELETE	Delete information in a database.

Understanding basic SQL SELECT statements

One of the most widely used SQL statements is the SELECT statement. The SQL SELECT statement retrieves columns of data from a database. The tabular result is stored in a *result* table (called the record set).

You use the following SELECT statement to retrieve information from a table:

```
SELECT column_name(s) FROM table_name
```

Consider a table named Clients that contains the following rows:

LastName	FirstName	Address	City
Brown	Marie	12 State St	Boston
Adams	Russell	521 Beacon St	Boston
Carter	Joan	1 Broadway	New York

To select the columns named LastName and FirstName, use the following SELECT statement:

```
SELECT LastName, FirstName FROM Clients
```

The result of this SQL statement contains the following data:

LastName	FirstName
Brown	Marie
Adams	Russell
Carter	Joan

Using the SQL WHERE clause to limit the rows returned

To conditionally select data from a table, you can add a WHERE clause to the SELECT statement, which results in the following syntax:

```
SELECT column_name FROM table_name WHERE column condition value
```

With the WHERE clause, you can use any of the following operators:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
AND	Joins one or more conditions
OR	Joins one or more conditions
LIKE	Specifies a search for a pattern in a column. You can use a percent sign (%) to define wildcards (missing letters in the pattern) before and after the pattern.

For example, to select the columns named LastName and FirstName for Clients whose City is Boston, use the following SELECT statement:

```
SELECT LastName, FirstName FROM Clients Where City = 'Boston'
```

The result of the preceding SQL statement contains the following data:

LastName	FirstName
Brown	Marie
Adams	Russell

You can compose a WHERE clause with one or more conditions; these are called subclauses. You join subclauses using the operators AND and OR. The AND operator displays a row if *all* conditions that are listed are True. The OR operator displays a row if *any* of the conditions listed are True. The following statement shows an example of a WHERE clause with multiple subclauses:

```
SELECT LastName, FirstName FROM Clients Where City = 'Boston' AND FirstName =
  'Marie'
```

The result of the preceding SQL statement contains the following data:

LastName	FirstName
Brown	Marie

Note: The preceding SQL SELECT examples use single-quotation marks around the value. SQL uses single-quotation marks around text values. Most database systems also accept double-quotation marks. Do not enclose numeric values in quotation marks.

Sorting the results

You use the ORDER BY clause to sort the result rows. The following SQL statement returns an alphabetic list of people sorted by last name, and then first name, from the Clients table:

```
SELECT * FROM Clients Order By LastName, FirstName
```

The default is to return the results in ascending order (top to bottom). If you include the DESC keyword in the ORDER BY clause, the rows are returned in descending order (bottom to top).

The following statement returns a reverse alphabetic list of the Clients table:

```
SELECT * FROM Clients Order By LastName, FirstName DESC
```

Note: The SQL SELECT statement is quite powerful. There are several other options for retrieving data from a SQL database using the SELECT statement, which are not described in this manual. For more information, consult a SQL reference.

Using SQL with ColdFusion

ColdFusion communicates with your data source through a database interface called Java Database Connectivity (JDBC). JDBC is a standard application programming interface (API) for accessing information from different database systems and different storage formats.

About data sources

A *data source* is a complete database configuration that uses a JDBC driver to communicate with a specific database. In ColdFusion, you must configure a data source for each database file that you want to use. After you configure a data source, the ColdFusion server is then capable of communicating with that data source through the JDBC driver.

You configure data sources in ColdFusion by using the ColdFusion MX Administrator. For information about how to configure the sample data source file that is supplied for use with Part II of this manual, see "Lesson 2: Configuring Your Development Environment" on page 49. For more information about configuring a data source in ColdFusion, see *Installing and Using ColdFusion MX* or *ColdFusion MX Developer's Guide*.

Writing SQL and CFML statements to interact with a data source

After ColdFusion makes a connection to the data source, you can interact with that database by using SQL and ColdFusion.

To interact with an established data source, you must include SQL statements in your CFML statements; for example:

```
<cfquery name="queryname" datasource="namedbfile">
  SELECT FirstName, LastName, DepartmentID
  From Employee
</cfquery>
```

In this example, the first attribute of the `cfquery` tag is the name of the query. The second attribute of the `cfquery` tag defines the name of the data source. The `SELECT` statement defines the fields (columns) to be retrieved from a table named Employee.

CFML tags that interact with a database

The following table lists the CFML tags that you can use to interact with a database:

Tag	Description
`cfquery`	To retrieve (query) information in a database.
`cfinsert`	To add records to a database.
`cfupdate`	To update information in a database.

Part II of this manual introduces these tags when you use them to interact with the sample database. For more information about interacting with a database, see *ColdFusion MX Developer's Guide* or *CFML Reference.*

PART II
Building a ColdFusion Application

Part II provides a tutorial that steps you through building a sample Macromedia ColdFusion application. It consists of the following lessons:

Lesson 1: Preparing to Build the Sample Application 41
Lesson 2: Configuring Your Development Environment 49
Lesson 3: Retrieving Data 59
Lesson 4: Building Dynamic Queries 71
Lesson 5: Creating a Trip Detail Page 85
Lesson 6: Creating a Main Application Page 97
Lesson 7: Validating Data to Enforce Business Rules 103
Lesson 8: Implementing Browsing and Searching 123
Lesson 9: Enabling Database Maintenance 129
Lesson 10: Restricting Access to ColdFusion Applications 143

CHAPTER 4
Lesson 1: Preparing to Build the Sample Application

This tutorial guides you through the process of building a simple Macromedia ColdFusion web application. By the end of the tutorial, you will be familiar with how to use Macromedia ColdFusion MX 7 to query and update a database, validate data, generate a report, and require user authentication.

The application that you will build in this tutorial is for a fictitious travel company called Compass Travel. Compass Travel markets a wide range of adventure trips to the public through its website. Trip coordinators at Compass Travel are responsible for maintaining the trip information. The sample tutorial application assists the trip coordinators in maintaining trip information in the Compass Travel database.

ColdFusion development is the emphasis of the tutorial; therefore, you will not need to design or build the Compass Travel database. It is important, however, for you to be familiar with the layout of the database. Additionally, you must understand the functional requirements that will help determine the application design. This lesson provides an overview of these application design steps, while the remainder of this manual guides you through the lessons on constructing the sample application.

Application development steps

Most software applications contain three major functions:

- A user interface to capture data
- Logic to validate the captured data
- A database to store the validated data

The steps to develop these major functions vary from project to project. In this tutorial, you will review or participate in the following application development steps to build the Compass Travel Trip Maintenance application:

Step	Description
1	Determine the application functional requirements.
2	Determine the data requirements by identifying the information required for the Trip Maintenance application.

Step	Description
3	Design the database for your application by exploring the database tables that will store the trip information.
4	Develop the ColdFusion application pages.

The following sections explore an overview of each of these application development steps.

Exercise 1: Determining the application functional requirements

Before you can build the sample application, you must understand the functional requirements that influence its design. The design of the sample application centers around the daily tasks performed by Compass Travel's trip coordinators. The following table describes these tasks:

Trip coordinator task	Description
Produce current trip listing	To help Compass Travel agents take trip reservations on the telephone and in person, the trip coordinator maintains a list of current trip offerings.
Provide trip information	On an ad hoc basis, Compass Travel management asks the trip coordinator to develop lists of trips that meet specific criteria.
Maintain trip information	The trip coordinator is responsible for keeping all trip information up to date. To do this, the coordinator needs to locate a trip to edit it or delete it. Additionally, the coordinator must be able to add a new trip.
Ensure the quality of trip information	The trip coordinator is responsible for periodically browsing the current trip offerings to ensure that all the information is accurate. Additionally, when adding a new trip or editing an existing one, the trip coordinator must ensure that the data adheres to the Compass Travel business rules.

You can derive several functional requirements for the new application from the preceding table. For example, the sample application must provide the ability to do the following:

- Generate trip listings.
- Find trips based on user-supplied criteria.
- Browse trips.
- Add a new trip.
- Delete an existing trip.
- Edit an existing trip.
- Validate new or updated trips against Compass business rules.

In the lessons that follow, you will build ColdFusion pages to address each of these functional requirements. The basis for every requirement is the idea of a trip. Before you can build code to address any of these requirements, you must understand which attributes of a trip are important to Compass Travel. For this you must determine the data requirements for the application. Understanding the data requirements is essential to building the proper database to hold the application data.

Exercise 2: Determining the data requirements

Prior to creating the application pages to capture trip information, you must determine what type of data is required about each trip. For the example in this tutorial, the Compass Travel trip coordinator must maintain the following information about each trip:

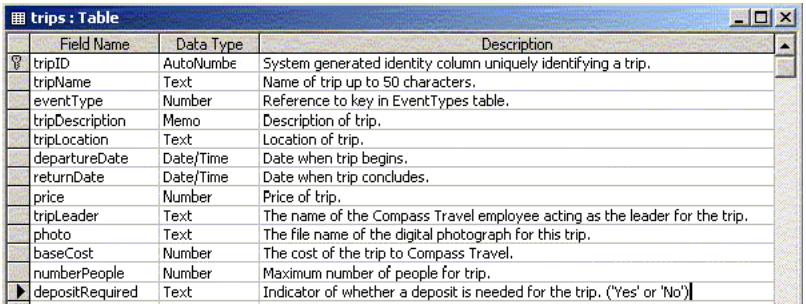
- Trip name
- Type of event (surfing, mountain climbing, kayaking, and so on)
- Trip description
- Trip location
- Departure date
- Return date
- Price
- Base cost
- Assigned trip leader
- Total number of people who can attend the trip
- An indicator of whether a deposit is required
- Trip photograph

By collecting the preceding information about each trip, the Compass Travel website can market its trips online to the general public. Customers who are booking a trip need to know the trip name, when the trip begins and ends, the price, and a description. Additionally, the trip coordinator must identify the filename for a photograph of each trip. The Compass Travel website displays the photograph to further entice prospective customers into booking the trip. Finally, Compass Travel considers it important to store the base cost for each trip to help determine trip profitability. The cost must be captured, but it is for Compass Travel internal use only and is not shown on the public website.

Exercise 3: Designing the database for your application

After you identify the information to collect, you must consider where to store the data. Prior to creating the data collection form and instructing ColdFusion where to store the form data, you must have a database ready to accept the data.

If you had to create the Compass Travel database, you would create a table named trips to store the information that you plan to collect about each trip. The table would look something like the following table:



trips : Table

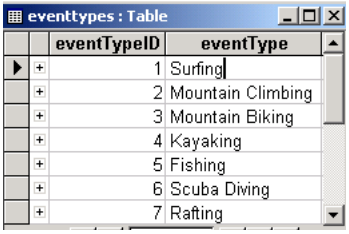
Field Name	Data Type	Description
tripID	AutoNumbe	System generated identity column uniquely identifying a trip.
tripName	Text	Name of trip up to 50 characters.
eventType	Number	Reference to key in EventTypes table.
tripDescription	Memo	Description of trip.
tripLocation	Text	Location of trip.
departureDate	Date/Time	Date when trip begins.
returnDate	Date/Time	Date when trip concludes.
price	Number	Price of trip.
tripLeader	Text	The name of the Compass Travel employee acting as the leader for the trip.
photo	Text	The file name of the digital photograph for this trip.
baseCost	Number	The cost of the trip to Compass Travel.
numberPeople	Number	Maximum number of people for trip.
depositRequired	Text	Indicator of whether a deposit is needed for the trip. ('Yes' or 'No')

Recognizing the data types

Each field in the Trips table has a data type attribute that describes the type of data that can be stored in the column. For instance, the tripName column can contain text data, while the price column can only contain numeric data. It is important to know what type of data is valid for each column so that your data collection forms can validate against incorrect values entered by the user.

Looking closer, you might wonder why the eventType column is a number and not a text data type column. Recall that data requirements analysis identified the need for a type of event (surfing, mountain climbing, kayaking, and so on). The purpose of this column is to classify trips into various categories based on the trip activity. It is essential that the application classify the trips consistently. Therefore, it is important to offer a list of event types for the user to select, rather than to accept free text input.

To present a list of event types for user selection, the event types are stored in a separate table, the eventtypes table. This table is already populated and contains the following rows:



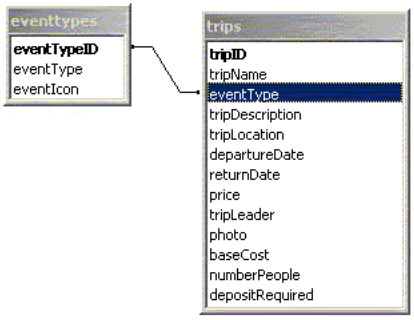
eventtypes : Table

eventTypeID	eventType
1	Surfing
2	Mountain Climbing
3	Mountain Biking
4	Kayaking
5	Fishing
6	Scuba Diving
7	Rafting

Establishing a relationship between the two tables

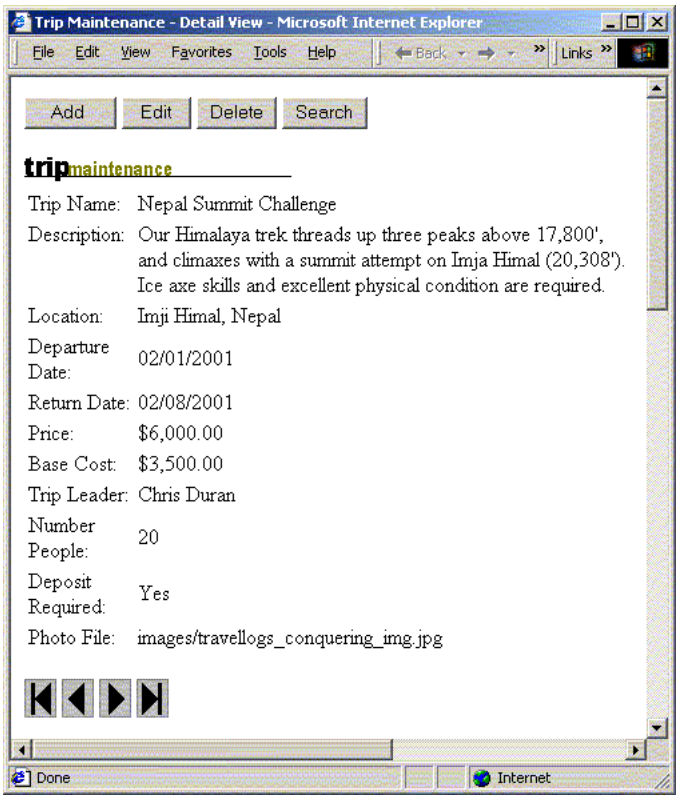
When the user selects an event type from the list obtained from reading the eventtypes table, the correct event type must be saved to the trips table with all the other trip-related data. The application could store the eventType (for example, mountain climbing) itself into the eventType column in the trips table. But if the name Mountain Climbing were later shortened to Climbing in the eventtypes table, new mountain climbing trips would be classified differently than ones saved before the change. For this reason and to save space in the database, the key to the eventtypes row (eventTypeID) is stored in the trips table instead.

The two tables are said to have a *relationship*. This relationship works by matching data in ***key*** fields. In this case, the matching fields consist of a *primary key* (eventTypeID) from the eventtypes table, which provides a unique identifier for each record, and a *foreign key* (eventType) in the trips table. The foreign key contains the same value as the primary key, pointing to a unique event type. The following figure shows this relationship:



Exercise 4: Developing the sample application

Using the application functional requirements and the database that is provided for this tutorial, you are ready to use ColdFusion to develop the Trip Maintenance application. The remaining lessons in the tutorial step you through the process of constructing this application. When you are done, the main page for the Trip Maintenance application will appear as follows:



The main application page is where users will come to view information about trips and to navigate to other ColdFusion pages to add, edit, or search for new trips.

The following lessons explain how to do these tasks:

- Configure your development environment ("Lesson 2: Configuring Your Development Environment" on page 49).
- Retrieve data from a database ("Lesson 3: Retrieving Data" on page 59).
- Build dynamic queries ("Lesson 4: Building Dynamic Queries" on page 71).
- Build the trip detail page ("Lesson 5: Creating a Trip Detail Page" on page 85).
- Build the main application page ("Lesson 6: Creating a Main Application Page" on page 97).
- Write code to enforce Compass Travel business rules ("Lesson 7: Validating Data to Enforce Business Rules" on page 103).

- Implement browsing and searching ("Lesson 8: Implementing Browsing and Searching" on page 123).
- Enable database maintenance ("Lesson 9: Enabling Database Maintenance" on page 129).
- Restrict access to the Compass Travel application by requiring user authentication ("Lesson 10: Restricting Access to ColdFusion Applications" on page 143).

How to proceed

Each lesson in the tutorial is designed to let you proceed at your own pace. At any time, you can stop and later return to that place in a lesson so that you can complete all the sections in the lesson.

Each lesson guides you through a scenario to enhance the Compass Travel Trip Maintenance application. Sections within a lesson present basic programming concepts that you need to understand before completing the section exercises.

Depending on your programming experience, you can read the entire lesson and then proceed to the hands-on exercises, or you can skip some information in the lesson and proceed directly to the exercises. For each exercise, there is a completed version of each file created or modified in that lesson. The completed solution files are in the directory *cf_root*/cfdocs/getting_started/solutions. Appended to each filename is the lesson and exercise number. For example, the version of the file tripdetail.cfm that you would have after performing the steps in Lesson 5, Exercise 4 is named tripdetail_lesson5_ex4.cfm. To use the solution file and continue with Lesson 5, Exercise 5, copy the tripdetail_lesson5_ex4.cfm file to the *cf_root*/cfdocs/my_app directory, and overwrite the tripdetail.cfm file if it already exists in the my_app directory.

Working directories

The following table describes the working directories for this tutorial:

Directory	Description
my_app	You save all your source code in this directory.
solutions	You can find solutions to all the exercises in this directory.
db	You use this directory as the working directory for the Compass Travel database.
photos	You use this directory to access existing trip photographs.
images	You use this directory to access application image files.

Locating the working directories

The working directories for this tutorial are under your web root directory. For example, the directory path on your computer might look like the following:

- (On Windows) *web_root*\cfdocs\getting_started
- (On UNIX) *web_root*/cfdocs/getting_started

Note: When using the J2EE configuration, the cfdocs directory is under the ColdFusion web application root. For more information, see "Saving your ColdFusion page" on page 17.

You can view ColdFusion application pages on your local computer by opening a web browser and entering one of the following URLs:

Configuration	URL
For local third-party web server configuration	http://localhost/cfdocs/getting_started/my_app/<pagename>.cfm
For built-in ColdFusion web server configuration	http://localhost:8500/cfdocs/getting_started/my_app/<pagename>.cfm

For more information about the tutorial file structure and the location of the getting_started subdirectories, see "Exercise 1: Verifying the tutorial file structure" on page 49.

Requirements

To use this tutorial, you must have the following components installed:

ColdFusion MX 7 For information on how to install ColdFusion MX 7, see *Installing and Using ColdFusion MX*.

Database Management System A database management system can be installed on the same computer as ColdFusion MX or on a separate computer. For the purpose of this tutorial, a Microsoft Access database file for Microsoft Windows users and a PointBase database file for UNIX users have been provided. For information about how to configure the Compass Travel data source, see "Lesson 2: Configuring Your Development Environment" on page 49. For additional information about configuring a data source, see *Installing and Using ColdFusion MX*.

Web browser You can use Microsoft Internet Explorer (4.0 or later) or Netscape Navigator (6.0 or later).

Text editor or IDE (Interactive Development Environment) In order to use the Macromedia Dreamweaver Extensions described in this tutorial, you must use Dreamweaver MX 2004. However, you can use Macromedia HomeSite+, ColdFusion Studio, any text editor, or IDE, and copy and paste the code that the Dreamweaver Extensions create. In the exercises in this tutorial, the term *editor* means Dreamweaver MX 2004, HomeSite+, ColdFusion Studio, or any text editor or IDE of your choice.

CHAPTER 5
Lesson 2: Configuring Your Development Environment

In this lesson, you set up your development environment for the tutorial. This lesson describes the tutorial file structure, and how to configure the database connection and debugging options. Additionally, it provides an overview of using Macromedia Dreamweaver MX for Macromedia ColdFusion development.

This lesson explains how to do the following tasks:

Exercise 1: Verifying the tutorial file structure 49

Exercise 2: Setting up your development environment 50

Exercise 3: Configuring a database connection 52

Exercise 4: Configuring debugging options 55

Exercise 1: Verifying the tutorial file structure

Before you begin the tutorial, verify that the configuration of the computer where ColdFusion MX 7 is installed matches the file structure described in the following sections.

The files required to complete the Compass Travel tutorial are installed under the web server root directory. The location of this directory varies, depending on whether you chose to configure a local third-party web server (such as IIS) or the ColdFusion stand-alone web server during installation, as follows:

- When you use a third-party web server (such as IIS or Apache), the files are installed in: *web_root*/cfdocs/getting_started.
- When you use the ColdFusion built-in web server, the files are installed in: *cf_root*\webroot\cfdocs\getting_started.

The following is the getting_started directory structure:

```
getting_started
   -db
   -my_app
      -images
   -new_user_database
   -photos
   -solutions
```

Note: When you use the J2EE configuration, the cfdocs directory is under the ColdFusion web application root. For more information, see "Saving your ColdFusion page" on page 17.

ColdFusion MX installs two copies of the sample CompassTravel database file. The working copy is located in the db directory; a backup copy of the file is in the new_user_database directory.

To ensure that you are working with the original database file, verify that the file in the db directory has the same date as the backup file in the new_user_database directory. If the date of the file in the db directory is later than the backup file, replace the file in the db directory with a copy of the backup database.

Caution: Do not write to the database file in the new_user_database directory. The backup file lets multiple users perform the tutorial.

In each of the database subdirectories, the tutorial provides one sample database file for Microsoft Windows users and one sample database file for UNIX users. Windows users use a Microsoft Access file, and UNIX users use a PointBase file.

Note: The sample PointBase file consists of two files: compasstravel.dbn and compasstavel$1.wal. ColdFusion MX uses both of these files to work with the content in the sample PointBase database.

Save all the files that you create for the tutorial application in the my_app directory. This directory contains one subdirectory for images. The image subdirectory contains the required image files for the tutorial application.

The photos directory contains the required photo files for the tutorial application. The solutions directory provides sample application files that you can use when building the tutorial application.

Exercise 2: Setting up your development environment

If you are running ColdFusion in a Windows environment, Macromedia recommends that you use Macromedia Dreamweaver MX for ColdFusion development; however, if you are running ColdFusion in a UNIX environment, you should use the text editor that you prefer. Although Dreamweaver MX offers features that help you create ColdFusion applications quickly and easily, you can still follow the steps of the tutorial by entering or copying the code examples into your editor.

Note: To use the Dreamweaver Extensions included with ColdFusion MX 7, you must use Macromedia Dreamweaver MX 2004 version 7.0.1.

About Macromedia Dreamweaver MX

Macromedia Dreamweaver MX is the preferred development environment for building ColdFusion MX 7 applications. It supports the latest ColdFusion MX 7 features, and it combines superior code editing features with the visual design features of Dreamweaver. You can build ColdFusion MX 7 applications by writing the code manually or generating the code by using one of the code-generating tools provided with Dreamweaver MX. In addition, ColdFusion MX 7 provides extensions to Dreamweaver that let you perform some administrative tasks without leaving the Dreamweaver environment and generate code for some of the most common tasks you encounter when creating a ColdFusion application.

With Dreamweaver MX, you can author and test your application code from a local or remote client. You can save your code directly to the server computer where ColdFusion is installed. The following sections provide an overview of Dreamweaver MX, and information on how to configure Dreamweaver MX for ColdFusion development.

Features for ColdFusion developers

Dreamweaver MX provides a wide variety of code editing features for ColdFusion developers, including the following:

- Extensions that generate code for common ColdFusion application tasks.
- Extensions that let you perform administrative tasks without leaving the Dreamweaver environment.
- Rich tag editors for quickly setting attributes and values for every CFML tag.
- Code hints for writing CFML tag attributes.
- A code validator for validating code readiness against other ColdFusion versions.
- A tag chooser with integrated reference material for inserting ColdFusion tags.
- A snippets panel for reusing code.
- An integrated debugging display for quickly pinpointing problem areas in the code.
- A remote ColdFusion server connection for browsing remote data sources and files.
- ColdFusion MX documentation included in the Dreamweaver MX online Help.

To use Dreamweaver MX to build the sample ColdFusion application in this tutorial, see the following sections for information about configuring Dreamweaver for ColdFusion development.

Configuring Dreamweaver MX for ColdFusion development

Before you use Dreamweaver MX to create the sample application in this tutorial, you must configure Dreamweaver to use the ColdFusion MX 7 Extensions for Dreamweaver, which are included with ColdFusion MX 7. You must also define the connection to the sample database file. The Extensions simplify the process of configuring database connections.

To configure Dreamweaver MX to create the sample application:

1. Install the ColdFusion MX 7 Extensions for Dreamweaver by double-clicking the cfmx7dreamweaverextensions.mxp file, which is located in the cf_root/wwwroot/CFIDE/installers directory.
2. Create a site that contains the tutorial files.
3. Specify ColdFusion as the application document type.
4. Specify ColdFusion as the site application server technology. For information about how to perform these steps, see the Dreamweaver MX online Help or *Using Dreamweaver MX*.
5. Specify the RDS login information for the site.

Tip: If you are a new Dreamweaver MX user, you can perform the Dreamweaver MX tutorial before using Dreamweaver MX to build the sample application. The tutorial in this book does not describe how to use Dreamweaver. The purpose of this tutorial is to teach you how to build ColdFusion applications using ColdFusion Markup Language (CFML).

Exercise 3: Configuring a database connection

One of the most commonly used and most powerful features of ColdFusion is the ability to connect to and manipulate data from a wide variety of databases. Like most ColdFusion developers, you will probably use this capability often. To be able to connect to a database, you must create a a data source in ColdFusion. A data source contains the information that ColdFusion needs to be able to recognize and communicate with a database.

ColdFusion MX 7 includes a Dreamweaver extension that lets you create a data source without having to leave the Dreamweaver environment. If you are not using Dreamweaver, you use the ColdFusion Administrator to create data sources; see"Configuring a data source in the Administrator" on page 52 for more information.

Using Dreamweaver to configure a data source

To create a ColdFusion data source without leaving the Dreamweaver environment, you need the following information:

- The type of connection to use
- The name to call the data source
- The location of the database file(s)

To configure a database connection in Dreamweaver:

1. Ensure that the site that contains the tutorial files is the current site.
2. Select Create New > ColdFusion.
3. On the Databases tab, click the + button.
4. Select Microsoft Access Connection.
5. In the CF Data Source Name text box, enter **CompassTravel**.
6. Specify the Database File as c:\CFusionMX7\wwwroot\cfdocs\getting_started\db\CompassTravel.mdb.
7. Select the Use Default Username check box.
8. When prompted for the password for this site, enter the ColdFusion Administrator password you specified when you installed ColdFusion MX 7.
9. Click OK.
10. Continue with "Exercise 4: Configuring debugging options" on page 55.

Configuring a data source in the Administrator

You can use the ColdFusion MX 7 Administrator to configure a data source. If you are running ColdFusion in a UNIX environment, you must use the ColdFusion MX 7 Administrator to connect to the PointBase database in the data source.

To access the ColdFusion MX 7 Administrator:

- Open a browser and go to one of the following URLs:
  - External web server users: http://localhost/CFIDE/administrator
  - Built-in web server users: http://localhost:8500/CFIDE/administrator
  - J2EE server users: http://localhost:*portnumber*/CFIDE/administrator. If you installed ColdFusion Enterprise Edition with JRun, the default HTTP port is 8300.

Note: If you are accessing the ColdFusion MX Administrator from a remote client, you must replace localhost with the host name or IP address of the computer where ColdFusion MX is installed.

The following sections describe how to establish a connection to the sample tutorial database file and how to enable optional debugging settings.

Configuring the connection to the sample database file

The following procedures describe how to configure a connection to the sample database file (CompassTravel) using the ColdFusion MX Administrator. Prior to building the sample application, you must configure the Compass Travel database connection.

Perform one of the following procedures. The Microsoft Access procedure is for Windows users. The PointBase procedure is for UNIX and Macintosh OS X users.

To define the connection to the sample Microsoft Access database:

1. In the ColdFusion MX Administrator, select Data & Services > Data Sources.
2. In the Data Source Name box, enter **CompassTravel**.
3. In the Driver list, select Microsoft Access with Unicode.
4. Click Add.
5. Click Browse Server and navigate to the *cf_root*/cfdocs/getting_started/db/CompassTravel.mdb file.
6. Click Apply.
7. Click Show Advanced Settings and ensure that the settings for CLOB and BLOB are enabled (checked).
8. Click Submit to complete the data source configuration.

   The ColdFusion MX Administrator verifies the data source connection and the name CompassTravel appears in the Connected Data Sources table.
9. Ensure that OK appears in the Status column for CompassTravel.

   If the connection to the CompassTravel data source fails, do the following:

   a. Verify that the name of the data source file does not contain a space. If it does contain a space, delete the data source from the Connected Data Source dialog box. To do this, click the Delete action button associated with the CompassTravel data source name, and then repeat the steps in this procedure to reconfigure this data source.

   b. Verify that the path specified for the Compass Travel database file is correct.

To define the sample PointBase database file:

1. In the ColdFusion MX Administrator, select Data & Services > Data Sources.

   The Add New Data Source dialog box appears.

2. Specify the following:

Field	Action
Data Source name text box	Specify the name CompassTravel. ***Note***: Ensure that the name of the data source file does not contain any spaces. If the name contains a space, the data source connection fails.
Driver drop-down selection box	Select Other.

3. Click Add to configure the data source name and driver.

   The Data Source dialog box appears.

4. Specify the following:

Field	Action
JDBC URL	Enter the following JDBC URL for the Compass Travel PointBase files: jdbc:pointbase:compasstravel,database.home=/<home location>/wwwroot/cfdocs/getting_started/db The following is the default home location for stand-alone ColdFusion web server configurations: /opt/coldfusionmx7/wwwroot/cfdocs/getting_started/db
Driver Class	Enter the following driver class: com.pointbase.jdbc.jdbcUniversalDriver
Driver Name	Specify PointBase.
Username	Specify PBPUBLIC.
Password	Specify PBPUBLIC.
Description	Enter the following: Database file for Compass Travel tutorial

5. Click Show Advanced Settings to make the following settings:

Field	Action
Maintain Connection	Disabled (clear)
CLOB	Enabled (checked)
BLOB	Enabled (checked)

6. Click Submit to complete the data source configuration.

   The name CompassTravel appears in the Connected Data Sources dialog box.

7. Click Verify All Connections to ensure that ColdFusion can access this file.

   OK appears in the Status column for successful connections.

   If the connection to the compass travel data source fails, do the following:

   a Verify that the name of the data source file does not contain a space. If it does contain a space, delete the data source from the Connected Data Source dialog box. To do this, click the Delete action button associated with the CompassTravel data source name, and then repeat the steps in this procedure to reconfigure this data source.

   b Verify that the JDBC URL is correct.

Exercise 4: Configuring debugging options

ColdFusion provides debugging information to make it easier to fix problems in your application. You can view debugging information for individual ColdFusion pages while editing them in Dreamweaver, or enable debugging throughout your application by using the ColdFusion MX 7 Administrator to set debugging options.

Viewing debugging information within Dreamweaver

Instead of enabling debugging for an application, you can view debugging information for any ColdFusion page in Dreamweaver.

To view debugging information for a ColdFusion page in Dreamweaver:

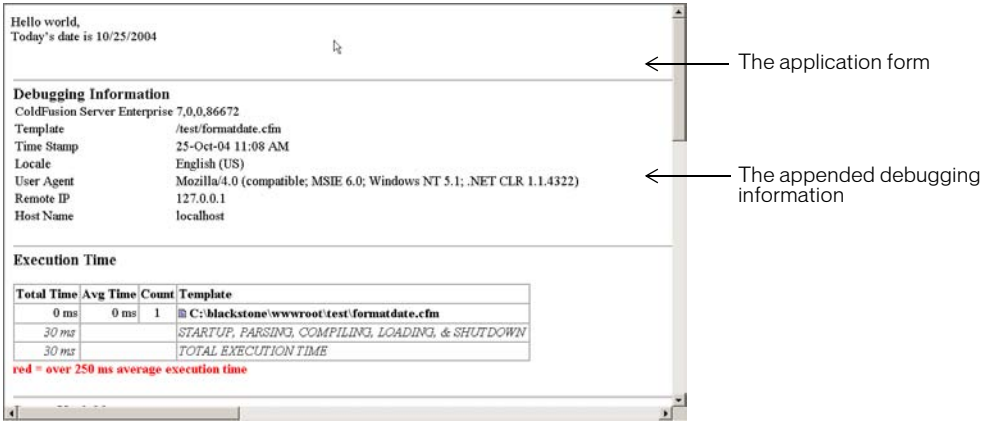
1. Open the page to debug in Dreamweaver.
2. On the Documents tab, click the Server Debug icon.
3. If an Exceptions category appears in the Results panel, click the + button to expand the category.
4. Fix the error and save the file, and click Browse.
5. To leave debug mode, switch to either Code view or Design view.

Enabling debugging in the ColdFusion MX Administrator

The ColdFusion MX Administrator provides a variety of debugging settings that let you enable debugging information on a server-wide basis. If you are working on a development system, you can have these options turned on all the time. However, if you are working on a production system, you most likely will not want to have these options turned on, because the debugging information can appear on the bottom of an application page or in a dockable tree in your browser.

The location of the debugging information or the type of debugging data shown varies, depending on the options that you enable on the Debugging page in the ColdFusion MX Administrator. In the following example, the debugging output includes general information about the ColdFusion server, the execution time of the application, and variable information:

The following figure shows an example of how debugging information can appear when appended to the bottom of a page in a browser:



If you are using a development server to build the sample application in this tutorial, you can enable some of these settings to help debug any unexpected problems.

Use the following steps to enable debugging options in the ColdFusion MX Administrator.

To enable debugging options:

1. In the ColdFusion MX Administrator, select Debugging & Logging > Debugging Settings.

   A list of debugging options appears on the Debugging Settings page.

2. Select the Enable Debugging check box.

   The debugging service is enabled for all options already selected on the page.

3. On the Debugging Settings page, view the description of each option that is enabled. If you do not want to append debugging information for a specific option, clear the check box.

   For the purpose of this tutorial, enable the following debugging options:

Option	Description
Database Activity	Identifies database activity related to SQL query events.
Exception Information	Identifies ColdFusion exceptions raised in the debugging output.
Tracing Information	Lets you trace event information reported in the debugging output.
Form, URL and Session Variables	Displays variable information in the debugging output.

4. Click Submit Changes when you are done.

Sending debugging information to remote clients

If you are using a remote client to perform the tutorial in this book, you must specify your IP address to receive debugging information. If you are working on a local client (the computer where ColdFusion is installed), you do not need to do this.

To receive debugging information when using a remote client:

1. In the ColdFusion MX Administrator, select Debugging & Logging > Debugging IP Addresses.

   The Debugging IP Address page appears.
2. In the IP Address text box, enter the IP address of your remote client.
3. Click Add.

Summary

In this lesson, you set up your development environment, configured data sources, and set up debugging options.

In the next lesson

In the next lesson, you learn to retrieve data from a database, display the results of querying a database, and use ColdFusion components to write structured, reusable code.

CHAPTER 6
Lesson 3: Retrieving Data

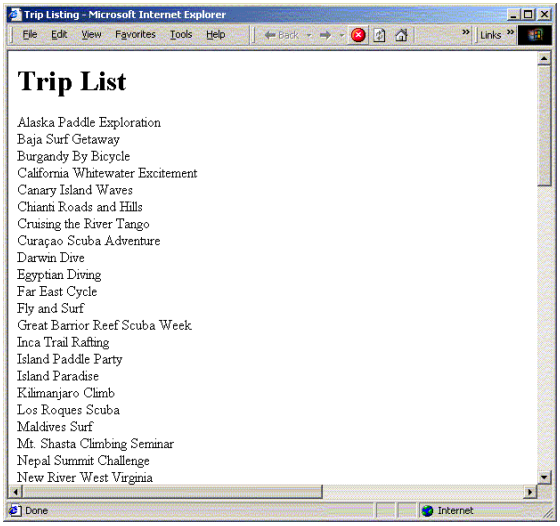
In this lesson, you begin the construction of a Macromedia ColdFusion web application for the fictitious company, Compass Travel. The exercises in this lesson introduce you to the process of retrieving data from the Compass Travel relational database.

This lesson contains the following exercises:

Exercise 1: About the trip list 60
Exercise 2: Retrieving the information for the trip list 61
Exercise 3: Displaying the query result using the cfoutput tag 62
Exercise 4: Writing structured, reusable code 63
Exercise 5: Creating additional queries 66

Exercise 1: About the trip list

As you recall from "Lesson 1: Preparing to Build the Sample Application" on page 41, one of the requirements for the Trip Maintenance application is the ability to generate trip listings. To help Compass Travel agents take trip reservations by telephone and in person, the trip coordinator maintains a list of current trip offerings. Years ago, the coordinator would type the list and fax it to the various Compass Travel offices in an effort to keep everyone informed. When Compass Travel built an intranet that was accessible by all offices, the trip coordinator added the following HTML web page to the site:



Each time the Trip List HTML page is rendered in a browser, it displays the same web page. Since the page always shows an identical trip list, it is considered a *static* web page. You should only use static web pages when you are creating a page that is not likely to change often.

Using the static web page approach, the Trip Coordinator needs to modify all the web pages that reference trip lists when trips are added, deleted, or trip names are changed. This manual process of updating each web page can lead to inaccurate or untimely information. Luckily, Compass Travel has built a database that contains a list of trips, so you can build a more accurate and timely solution for the trip coordinator.

In this lesson, you will create a dynamic Trip List page to present an up-to-date list of trips on the Compass Travel website. The primary users of this component are the Compass Travel coordinators and agents, not the general public.

To display the information on the page, you do the following:

1. Retrieve the data that you want to display.
2. Display the data on the page.

Exercise 2: Retrieving the information for the trip list

Relational database management systems process SQL instructions sent to them from various applications. ColdFusion sends SQL statements to database managers to manipulate data. ColdFusion needs a way to know to which database manager to send a specific SQL string for evaluation. In CFML, the `cfquery` tag serves this purpose. You will use the SQL SELECT statement and the `cfquery` tag to create a dynamic version of the Trip List page described earlier in this lesson. In this example, you use the `cfquery` tag to return all the trip names found in the tripName column in the Compass Travel Trips table. To use the SQL SELECT statement to dynamically retrieve this information, you must execute the SQL SELECT statement between the `cfquery` start and end tags.

To retrieve the trip list:

1. If you are working in Dreamweaver, ensure that the site that you created in Lesson 1 is the current site. For more information, see "Configuring Dreamweaver MX for ColdFusion development" on page 51.
2. Open a new file and save it as triplisting.cfm in the my_app directory.

   Note: If you are working in Dreamweaver, ensure that you select ColdFusion Templates in the Save As Type list.
3. Enter the following code, or do the steps listed in the "Let Dreamweaver do it" section.

```
<cfquery name="TripList" datasource="CompassTravel">
  SELECT trips.tripName FROM trips
</cfquery>
```

4. Save the file.

Let Dreamweaver do it

Dreamweaver lets you create a query without having to enter the code.

To create a query in Dreamweaver:

1. Click the Bindings tab in the application panel.
2. Click the + button.
3. Select Recordset (Query).
4. In the Name text box, enter **TripList**.
5. From the Data source list, select CompassTravel.
6. From the Tables list, select trips.
7. Next to Columns, click Selected.
8. Select tripName.
9. Click OK.
10. Save the file.

Your ColdFusion application page retrieves the information for the trip list. Next, you need to display the information.

Reviewing the code

The following table describes the code used to build the query:

Code	Explanation
`<cfquery name="TripList" datasource="CompassTravel">`	ColdFusion query named TripList. Submits any SQL statement between the `cfquery` start and end tags to the data source specified in the `datasource` attribute.
`SELECT trips.tripName FROM trips`	SQL SELECT statement to retrieve all tripName(s) from the trips table.

Exercise 3: Displaying the query result using the cfoutput tag

In Chapter 2, "CFML Basics," on page 15, you learned that the ColdFusion `cfoutput` tag is an easy mechanism to display literal text and the contents of variables. Additionally, the `cfoutput` tag significantly simplifies displaying the results of queries. When used to display the results from a query, the `cfoutput` tag automatically loops through the record set for you. You simply specify the name of the query in the `query` attribute of the `cfoutput` tag:

```
<cfoutput query="TripResult">
```

All the code between the `cfoutput` start and end tags is the *output code block*. The output code block executes repeatedly, once for each row in the record set. However, if the query returns no rows, ColdFusion skips the code contained in the output code block.

```
<cfoutput query = "xxx">
  ...output code block...
</cfoutput>
```

Displaying the column contents from the SQL statement

In CFML you surround variables with number signs (#) to display their contents using the `cfoutput` tag. You also use this approach with column names specified in the SELECT statement of a `cfquery`. For example, when you want to display the trip names from the SQL query, you use `#tripName#` within the output code block:

```
<cfoutput query="TripResult">
  #tripname#
</cfoutput>
```

For additional information about using SQL with `cfquery` and `cfoutput`, see *ColdFusion MX Developer's Guide*.

To display the query results:

1. Modify the triplisting.cfm file by adding the highlighted code so that it appears as follows:

```
<cfquery name="TripList" datasource="compasstravel">
    SELECT trips.tripName FROM trips
</cfquery>
<html>
  <head>
    <title>Trip Listing</title>
  </head>
```

```
  <body>
    <h1>Trip List</h1>
    <cfoutput query="TripList">#tripName#<br></cfoutput>
  </body>
</html>
```

2. Save the file as triplisting.cfm in the my_app directory.
3. View the triplisting.cfm page in a browser. The page lists all the trip names retrieved from the Compass Travel database.

Reviewing the code

The following table describes the code used to display the query result set:

Code	Explanation
`<cfoutput query="TripResult"> #tripName#<br</cfoutput>`	Output code block. Displays the value of the tripName column for each row in the result set from the "TripResult" query.

Exercise 4: Writing structured, reusable code

Generally, it is good coding practice to separate business logic from the display. The ColdFusion page that you just created contains both business logic (the database query) and presentation (the output code block). To separate them, you put the query in a ColdFusion component (CFC). Doing so separates business logic from presentation; it also makes it easy to reuse the query anywhere in your application. For more information, see Chapter 10, "Building and Using ColdFusion Components" in *ColdFusion MX Developer's Guide*.

To move the query to a CFC:

1. Create the CFC file.
2. Copy the query to the CFC.
3. Call the method that contains the query.

Creating the CFC file

ColdFusion components (CFCs) are special files saved with the filename extension .cfc. They can contain data and functions. Within CFCs, functions are referred to as *methods*. Actions that you want ColdFusion to perform, such as querying a database, are contained in component methods. One CFC can contain many methods. Each method in a CFC can return only one variable. The following is the general syntax of a CFC:

```
<cfcomponent>
  <cffunction name="firstMethod">
     <!--- CFML code for this method goes here. --->
  </cffunction>
  <cffunction name="secondMethod">
     <!--- CFML code for this method goes here. --->
  </cffunction>
</cfcomponent>
```

To create the CFC file:

1. Create a directory named components as a subdirectory of the my_app directory.
2. Open a new blank file.

   Note: If you are using Dreamweaver, select Dynamic page in the Category list, and select ColdFusion Component in the Dynamic Page list.

3. Enter the following code, or do the steps listed in the "Let Dreamweaver do it" section.

```
<cfcomponent displayName="Get Trips" hint="Get trip information">
  <cffunction name="basicList" displayName="List all trips"
  hint="List trips in same order as in table" access="public"
  returnType="query" output="false">

    <cfreturn>
  </cffunction>
</cfcomponent>
```

4. Save the file as gettrips.cfc in the components directory.

Let Dreamweaver do it

Dreamweaver lets you create a ColdFusion component without having to enter the code.

To create a CFC in Dreamweaver:

1. Click the Components tab.
2. Click the + button.
3. In the Display name text box, enter **Get Trips**.
4. In the Hint text box, enter **Get trip information**.
5. In the Name text box, enter **gettrips**.
6. Click the Browse button, and make the components directory the current directory.
7. In the Section list, click Functions.
8. Click the + button.
9. In the Name text box, enter **basicList**.
10. In the Display name text box, enter **List all trips**.
11. In the Hint text box, enter **List trips in same order as in the table**.
12. Select query as the Return type.
13. Click OK.
14. Save the file as gettrips.cfc in the components directory.

Copying the query to the CFC

To copy the query to the CFC, you copy the CFML to the CFC, between the opening and closing `cffunction` tags.

To copy the query to the CFC:

1. Highlight the following code on the triplisting.cfm page:

```
<cfquery name="TripList" datasource="CompassTravel">
  SELECT trips.tripName FROM trips
</cfquery>
```

2. Cut the highlighted code and copy it to the gettrips.cfc page so that it appears as follows:

```
<cfcomponent displayName="Get Trips" hint="Get trip information">
  <cffunction name="basicList"
    displayName="List all trips" hint="List trips in same order as in table"
    access="public" returnType="query" output="false">
    <cfquery name="TripList" datasource="CompassTravel">
      SELECT trips.tripName FROM trips
    </cfquery>
    <cfreturn>
  </cffunction>
</cfcomponent>
```

3. Modify the code by adding the following text so that the method returns the results of the query to the triplisting.cfm page:

```
<cfreturn TripList>
```

4. Save the gettrips.cfc file.

Calling the query method

To perform the query that is now in a method in a ColdFusion component, you have to call (invoke) the method. To do so, you can use the `cfinvoke` tag. Within the `cfinvoke` tag, you specify the name of the ColdFusion component, the method to call, and the query to return to the calling page. The name of the component includes the package, "cfdocs.getting_started.my_app.components." The package looks very similar to the path, except that it contains periods instead of slashes. Like a path, it specifies the location of the component.

To invoke the method:

1. Go to the top of the triplisting.cfm file.
2. Enter the following code, or do the steps listed in the "Let Dreamweaver do it" section.

```
<cfinvoke
 component="cfdocs.getting_started.my_app.components.gettrips"
 method="basicList"
 returnvariable="TripList">
</cfinvoke>
```

3. Save the file.
4. View the triplisting.cfm page in a browser and notice that the page lists the trip names, just as it did previously.

Let Dreamweaver do it

Dreamweaver lets you invoke a method in a CFC without having to write the code.

To invoke the method using Dreamweaver:

1. Click the Components tab in the Application panel.
2. Click the + button next to cfdocs.getting_started.my_app.components.
3. Click the + button next to gettrips.
4. Select query basicList() and drag it to the top of the triplisting.cfm file.
5. Change the value of the `returnvariable` to be the name of the query, TripList, as follows:

   ```
   returnvariable="TripList"
   ```

6. Return to Step 3 in the "To invoke the method:" procedure.

Exercise 5: Creating additional queries

In this exercise, you will improve the Trip List page to make it easier for the Compass Travel agents to locate trips. You must make the following improvements:

- Sort the trip names in alphabetical order.
- Display the departure date, return date, and price for each trip.
- Develop a Budget Trip List report that identifies trips that are priced $1500 or less.

You could modify the existing query; however, you may need to use that query in the future. Instead, you can create an additional method in the gettrips.cfc component with a query that meets the preceding requirements. You can then call the new methods with the enhanced query from the triplisting.cfm page. Display the triplisting.cfm page in the browser after each step to ensure that the corresponding requirement was met.

To modify the application, you must:

- Create the query that meets the requirements.
- Invoke the method that contains the query.
- Display the results.

Creating the query

To continue writing structured, reusable code, you create the query in a method in the existing CFC.

To create a method with enhanced query results:

1. Open the gettrips.cfc file.
2. Enter the following code just before the closing `cfcomponent` tag, or do the steps in the "Let Dreamweaver do it" section.

   ```
   <cffunction name="getBudgetTrips" displayName="Budget trip list"
     hint="List trips under $1500 alphabetically"
     access="public"
     returnType="query"
   ```

```
output="false">
<cfquery name="budgetTrips" datasource="CompassTravel">
  SELECT tripName, departureDate, returnDate, price
  FROM trips
  WHERE price <= 1500
  ORDER BY tripName
</cfquery>
<cfreturn budgetTrips>
</cffunction>
```

3. Save the file.

Let Dreamweaver do it

You can use the Dreamweaver Extensions to create a CFC query and invoke the method that contains the query from the triplisting.cfm page.

To create the enhanced CFC query in Dreamweaver:

1. Open the gettrips.cfc file and position the pointer before the closing `cfcomponent` tag.
2. Click the Bindings tab.
3. Click the + button.
4. Select Recordset (Query).
5. In the Name text box, enter **budgetTrips**.
6. Click the New Function button.
7. In the New function name text box, enter **getBudgetTrips**.
8. Click OK.
9. In the Data Source list, select CompassTravel.
10. In the Table list, select trips.
11. Click the Selected button.
12. Select tripName, departureDate, returnDate, and price from the list of columns.
13. In the Filter section, select price, select < from the list of operators, select Entered value, and enter **1500** in the text box.
14. In the Sort section, select tripName, and select Ascending.
15. Click OK.
16. Save the file.

Invoking the new method

To use the new query, you invoke the method that contains the query.

To invoke the new method with the enhanced query:

1. Open the triplisting.cfm file.
2. Modify the following code, or do the steps listed in the "Let Dreamweaver do it" section.

```
<cfinvoke
 component="cfdocs.getting_started.my_app.components.gettrips"
 method="getBudgetTrips"
 returnvariable="budgetTrips">
</cfinvoke>
```

3. Save the file.

Let Dreamweaver do it

You can use the Dreamweaver Extensions to create the CFC query and invoke the method that contains the query from the triplisting.cfm page.

To invoke the enhanced CFC query in Dreamweaver:

1. Delete the `cfinvoke` code block.
2. Click the Components tab.
3. Click the + button next to cfdocs.getting_started.my_app.components, and click the + button next to gettrips.
4. Select query getBudgetTrips and drag it to the top of the file.
5. Change the value of the `returnvariable` to be the name of the query, budgetTrips, as follows:

```
returnvariable="budgetTrips"
```

6. Save the file.

Displaying the results

To display the results of the new query, you refer to the new query in the `cfoutput` block and include all the columns that you want to display.

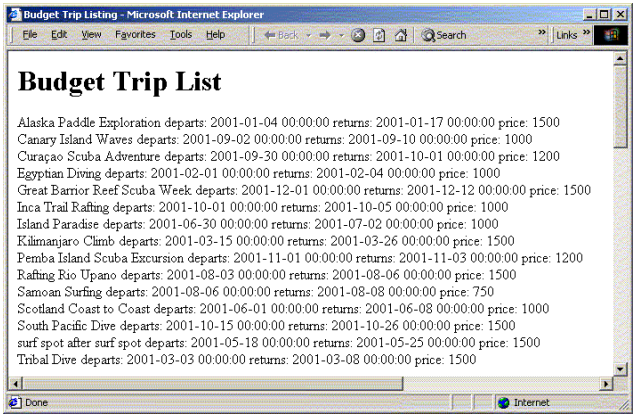
To display the results of the enhanced query

1. Change the output block in the triplisting.cfm file to output all three selected fields from the enhanced query, as follows:

```
<cfoutput query="budgetTrips">
  #tripName#
  departs: #departureDate#
  returns: #returnDate#
  price: #price#<BR>
</cfoutput>
```

2. Change the heading tag in the triplisting.cfm file from `<h1>Trip List</h1>` to `<h1>Budget Trip List</h1>`.

3. Save both the triplisting.cfm file and the gettrips.cfc file.
4. View the triplisting.cfm page in a browser and verify that all the new requirements were met. The revised TripListing.cfm page looks like this:



The dates and prices in the preceding listing are not formatted. In "Lesson 6: Creating a Main Application Page" on page 97, you will enhance the look of this page.

Summary

This lesson described how to access a relational database using ColdFusion. You used the SQL SELECT statement and the `cfquery` and `cfoutput` tags to display trip lists. You also learned to use ColdFusion components to write structured, reusable code.

In the next lesson

In the next lesson, you will build the search form to let users select which information to retrieve. To build the query based on what the user enters in the search form, you will use dynamic SQL. You will then display the results of the search on an action page.

CHAPTER 7
Lesson 4: Building Dynamic Queries

In this lesson, you develop a search form to allow users to select the information to retrieve, and then display, the search results.

This lesson contains the following exercises:

Exercise 1: Creating the trip search form 71
Exercise 2: Building a query that uses dynamic SQL 75
Exercise 3: Constructing the initial Trip Search Results page 79
Exercise 4: Testing the Trip Search Results page 81
Exercise 5: Enabling the departure and price criteria on the Trip Search form 82
Exercise 6: Testing the revised Trip Search Results page 83

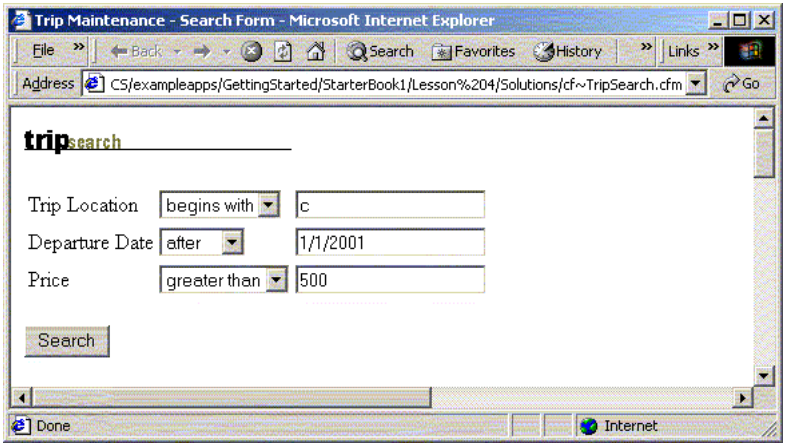
Exercise 1: Creating the trip search form

As you recall from "Lesson 1: Preparing to Build the Sample Application" on page 41, two of the requirements for the Trip Maintenance application are the ability to generate trip listings and a trip query facility. You will create a search interface that meets both of these requirements in this exercise.

About the trip search form

The dynamic listings that you developed in "Lesson 3: Retrieving Data" meet many of Compass Travel's requirements for locating trips. However, what if the number of trips were in the thousands or tens of thousands? Locating the right trip for a customer might be difficult and certainly time consuming. Moreover, it is very hard, if not impossible, to anticipate all the ways that a user might want to search for trips.

A better solution is to provide an interface for the user to specify the search criteria. The purpose of the Trip search form is to enable Compass Travel employees to search and view brief details about existing trips on their website. The completed form should appear as follows:



Designing the search criteria page

When designing the search criteria page, it is a good idea to develop a list of possible queries that the user might issue when searching for the records. Most Compass Travel customers are primarily concerned with trip locations, departure dates, and price; the following is a list of the types of queries that the agents are likely to require at Compass Travel:

- List the trips located in Hawaii.
- Identify the trips with a price greater than $3,000.
- Show the trips departing after 11/11/2005 that are priced less than $2,000.

There are a number of considerations to take into account when you design a search page to capture the user's search criteria. The following are two of the most important considerations:

- For which database columns will the user be allowed to specify a search condition?
- Should the user be allowed to identify which database columns to include in the record set?

In this exercise, the Compass Travel trip coordinator searches the trips based on tripLocation, departureDate, and price. Because these columns are the only ones that users can query, they are the only ones contained in the WHERE clause of the generated SQL statement. In addition, the coordinator has no control over which columns are returned in the record set. The query always returns the same columns to identify a trip:

- tripName
- tripLocation
- departureDate
- returnDate
- price
- tripID

In later exercises, you will reference these columns when you build the SQL SELECT statement for the `cfquery` in the search action page.

Understanding search query operators

Now that you have decided which columns can be queried (tripLocation, departureDate, and price), you can build a simple form that lets the user enter values for each of these fields. If the user enters a value (for example, Boston) for the tripLocation field and leaves the other two fields blank, the search results page constructs the following SQL statement:

```
SELECT tripName, tripLocation, departureDate,
  returnDate, price, tripID
FROM trips
WHERE tripLocation = 'Boston'
```

When you design the Search Criteria page, you must decide which operators to support for each of the columns you can query. The operators that you use depend on the data type of the SQL column.

For example, suppose the user wants a list of all the trips where the trip location begins with a "B." SQL is well-suited for this type of query. Typical SQL string operators are *equals*, *starts with*, *contains*, and *ends with*.

However, price is a numeric data type. The user can specify any of the following:

- Price is 5000
- Price less than 600
- Price greater than 1500

Although many more operators are permissible, for simplification, you can use the following operators for the Compass Travel columns that you can query:

Column to query	Query operators
tripLocation	is, begins with
departureDate	is, before, after
price	is, greater than, less than

Creating the Trip Search page

A simple design for a search criteria page presents an operator list and data entry field for each of the columns that you can query. To create the search criteria page, you create an HTML form.

To create the search page:

1. Create a blank file.

   Note: If you are using Dreamweaver, select Dynamic page in the Category list and ColdFusion in the Dynamic Page list.

2. Enter, or copy and paste, the following code in the blank file:

```
<html>
<head>
```

```
<title>Trip Maintenance - Search Form</title>
</head>
<body>
<img src="images/tripsearch.gif">
<!--- Search form --->
<form action="tripsearchresult.cfm" method="post">

<table>
  <!--- Field: tripLocation --->
  <tr>
    <td>Trip Location
    </td>
    <td>
      <select name="tripLocationOperator">
        <option value="EQUALS">is
        <option value="BEGINS_WITH">begins with
      </select>
    </td>
    <td>
      <input type="text" name="tripLocationValue">
    </td>
  </tr>
  <!--- Field: departureDate --->
  <tr>
    <td>Departure Date
    </td>
    <td>
      <select name="departureOperator">
        <option value="EQUALS">is
        <option value="BEFORE">before
        <option value="AFTER">after
      </select>
    </td>
    <td>
      <input type="text" name="departureValue">
      </td>
  </tr>
  <!--- Field: price --->
  <tr>
    <td>Price
    </td>
    <td>
      <select name="priceOperator">
        <option value="EQUALS">is
        <option value="GREATER">greater than
        <option value="SMALLER">smaller than
      </select>
    </td>
    <td>
      <input type="text" name="priceValue">
    </td>
  </tr>
</table>
<p>
<input type="submit" value="Search">
```

```
</form>
</body>
</html>
```

3. Save the file as tripsearchform.cfm in the my_app directory.

Reviewing the code

The following table describes the search criteria code and its function:

Code	Explanation
`<form action="tripsearchresult.cfm" method="post">`	Identifies tripsearchresult.cfm as the search action page. Results of user entry are passed to the search action page.
`<select name="tripLocationOperator"> <option value="EQUALS">is <option value="BEGINS_WITH">begins with </select>`	Builds a drop-down list offering the query operators for tripLocation. There must one operator list box for each queryable column.
`<input type="text" name="tripLocationValue">`	Captures a value to test. There is one text control for each queryable column.

Exercise 2: Building a query that uses dynamic SQL

Dynamic SQL is a term that refers to SQL code that your program generates using variables before the SQL is executed. You can use dynamic SQL to accomplish tasks such as adding WHERE clauses to a search based on the fields that the user filled out on a search criteria page.

Based on the columns that you can query in this tutorial, the SQL query to display the search results would look like this:

```
SELECT tripName,  tripLocation, departureDate, returnDate, price, tripID
FROM trips
```

The purpose of the Trip Search form is to supply the data needed to build the WHERE clause to finish this SQL SELECT statement and constrain the query according to the user's input.

When the user enters the search criteria on the Trip Search form and clicks the Search button, the form fields are posted to the Trip Search Results page. The posted field values compose the WHERE clause in the SQL SELECT statement. The following example lists the WHERE clauses that you can generate depending on the criteria set on the search page:

```
WHERE tripLocation = 'Aruba'
WHERE tripLocation Like 'C%'
WHERE tripLocation = 'China'
  AND departureDate > 1/1/2001
  AND price < 1500
```

In this example, the SQL AND operator joins the search condition clauses. To simplify the trip search example, you will use the SQL AND operator to combine all the search condition clauses. A more sophisticated search criteria page might present the user a choice of using AND or OR to connect one search criterion with the others.

The action page invokes a method that builds the WHERE clause so that the SQL SELECT statement retrieves the information that the user requests. Then, the action page displays an HTML table with the results of the user query using the `cfoutput` block.

Building the WHERE Clause with the cfif and cfset tags

The WHERE clause in a SQL SELECT statement is a string. You use the CFML `cfset` and `cfif` tags to conditionally build the WHERE clause depending on values passed to the search action page. The `cfset` statement creates a variable or changes the value of an existing variable. For example, to create a variable named color and initialize its value to red, you use the following statement:

```
<cfset color = "red">
```

The `cfif` tag instructs the program to branch to different parts of the code depending on whether a test evaluates to True or False. For example, to have some code execute if the color variable is equal to red, and other code execute if it is not, you use the following pseudocode:

```
<cfif color EQ "red">
... statements for color red
<cfelse>
... statements for other than red
</cfif>
```

Building a SQL WHERE clause in code is largely an exercise in string concatenation. The `&` operator combines two strings in ColdFusion. For example, the following code snippet:

```
<cfset FirstName = "Wilson">
<cfset LastName = "Gato">
<cfset FullName = FirstName & " " & LastName>
<cfoutput>My name is #FullName#.</cfoutput>
```

results in the following text:

```
My name is Wilson Gato.
```

For each search criterion on the Trip Search form, the code within the Trip Search Results page must do the following:

- Verify that the user entered data in the search criterion's value field. To do so, you use the `cfif` tag; for example, `<cfif Form.tripLocationValue GT "">`.
- If data was entered, construct a WHERE subclause by concatenating the following:
  - The SQL keyword AND
  - The corresponding SQL column name (in the Trip Search example, tripLocation) for the search criterion
  - The SQL operator equivalent of the search query operator
  - The test value entered by the user

The following code shows the creation of the WHERE subclause:

```
<cfif Form.tripLocationOperator EQ "EQUALS">
  <cfset WhereClause = WhereClause & " AND tripLocation = '" &
    form.tripLocationValue & "'" >
<cfelse>
```

```
  <cfset WhereClause = WhereClause & " AND tripLocation like '" &
    form.tripLocationValue & "%'" >
</cfif>
```

When you test for a string column within the WHERE clause of the SQL SELECT statement, you must enclose the test value in quotation marks.

When you use a variable to construct a WHERE clause, you must preserve the quotation marks so that the database server does not return an error. To preserve the quotation marks, you must use the ColdFusion `PreserveSingleQuotes` function. The `PreserveSingleQuotes` function prevents ColdFusion from automatically escaping single-quotation marks contained in the variable string passed to the function.

Note: The `cfqueryparam` tag also escapes single-quotation marks. For more information, see *CFML Reference*.

Creating the CFC query

The following code shows how to construct the tripLocation SQL WHERE subclause. Specifically, it uses a dynamic SQL SELECT statement built from parameters from the Trip Search page to display the search results. To continue the good coding practice of separating business logic and presentation, you put the code to build the query using dynamic SQL in a function in the CFC that you have been working with.

To add the new query to the CFC:

1. Open the file gettrips.cfc file and position the pointer before the closing `cfcomponent` tag.
2. Enter the following code, or do the steps in the "Let Dreamweaver do it" section:

```
<cffunction name="getTripsFromForm" access="public" returntype="query">
  <cfquery name="TripResult" datasource="CompassTravel">
    SELECT tripID, tripName, tripLocation, departureDate,
      returnDate, price FROM trips
  </cfquery>
  <cfreturn TripResult>
</cffunction>
```

3. Add the logic for creating the WHERE clause dynamically by entering the highlighted code.

```
<cffunction name="getTripsFromForm" access="public" returntype="query">
  <!--- Create WHERE clause from data entered via search form --->
  <cfset WhereClause = " 0=0 ">
  <!--- Build subclause for trip location --->
  <cfif Form.tripLocationValue GT "">
    <cfif Form.tripLocationOperator EQ "EQUALS">
      <cfset WhereClause = WhereClause & " and tripLocation = '" &
form.tripLocationValue & "'" >
    <cfelse>
      <cfset WhereClause = WhereClause & " and tripLocation like '" &
form.tripLocationValue & "%'" >
    </cfif>
  </cfif>
  <cfquery name="TripResult" datasource="CompassTravel">
    SELECT tripID, tripName, tripLocation, departureDate,
      returnDate, price FROM trips
```

```
    </cfquery>
    <cfreturn TripResult>
</cffunction>
```

4. Add the highlighted code to the `cfquery` block to use the dynamically built WHERE clause in the query:

```
<cfquery name="TripResult" datasource="CompassTravel">
  SELECT tripID, tripName, tripLocation, departureDate,
    returnDate, price FROM trips
    WHERE #PreserveSingleQuotes(WhereClause)#
</cfquery>
```

5. Save the file.

Let Dreamweaver do it

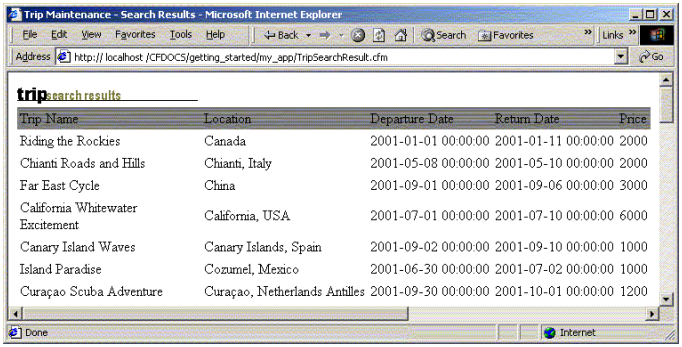
You can use the Dreamweaver Extensions to create the CFC query.

To construct the query with Dreamweaver:

1. Click the Bindings tab.
2. Click the + button.
3. Select Record set (Query).
4. In the Name text box, enter **TripResult**.
5. Click the New Function button.
6. In the New function name text box, enter **getTripsFromForm** and click OK.
7. From the Data source list, select CompassTravel.
8. From the Tables list, select trips.
9. Click the Selected radio button.
10. Select tripName, tripLocation, departureDate, returnDate, price, and tripID.
11. Click OK.
12. Return to Step 3 in the "To add the new query to the CFC:" section.

Exercise 3: Constructing the initial Trip Search Results page

After the user enters the search criteria and submits the form, the results are posted to the Trip Search Results page, as the following figure shows:



The logic contained in the search results page, also known as the action page, invokes the CFC method that builds the SQL SELECT statement contained in a `cfquery` tag by using ColdFusion string manipulation. The action page displays the result using the `cfoutput` tag.

To create the Trip Search Results page:

1. Create a blank document and save it as tripsearchresult.cfm in the my_app directory.

   Note: If you are using Dreamweaver, select Dynamic page in the Category list, and ColdFusion in the Dynamic Page list.

2. Enter the following code, or do the steps in the "Let Dreamweaver do it" section.

```
<cfinvoke
  component="cfdocs.getting_started.my_app.components.gettrips"
  method="getTripsFromForm"
  returnvariable="TripResult">
</cfinvoke>
```

3. Enter or copy and paste the following code after the closing `cfinvoke` tag:

```
<html>
<head>
<title>Trip Maintenance - Search Results</title>
</head>
<body>
  <img src="images/tripsearchresults.gif">
  <table border="0" cellpadding="3" cellspacing="0">
    <tr bgcolor="Gray">
      <td>Trip Name
      </td>
      <td>Location
      </td>
      <td>Departure Date
      </td>
      <td>Return Date
      </td>
      <td>Price
```

```
        </td>
      </tr>
      <cfoutput query="TripResult">
        <tr>
          <td>#tripName#
          </td>
          <td>#tripLocation#
          </td>
          <td>#departureDate#
          </td>
          <td>#returnDate#
          </td>
          <td>#price#
          </td>
        </tr>
      </cfoutput>
  </table>
  </body>
  </html>
```

4. Save the file.

Let Dreamweaver do it

As you have in previous exercises, you can let Dreamweaver generate the code to invoke the method.

To invoke the method using Dreamweaver:

1. Click the Components tab.
2. Click the + button next to cfdocs.getting_started.my_app.components.
3. Click the + button next to gettrips.
4. Select query getTripsFromForm() and drag it to the top of the triplisting.cfm file.
5. Change the value of `returnvariable` to be the name of the query, TripResult, as follows:

   ```
   returnvariable="TripResult"
   ```

6. Return to Step 3 in the "To create the Trip Search Results page:" section.

Reviewing the code

The following table describes the code used to build the tripLocation WHERE subclause:

Code	Explanation
`<cfset WhereClause = " 0=0 ">`	The `cfset` tag initializes the `WhereClause` variable to hold the WHERE clause to be constructed. The initial value is set to `" 0=0 "` so that the WHERE clause has at least one subclause in case the user enters no search criteria.

Code	Explanation
`<cfif Form.tripLocationValue GT "">`	The `cfif` tag tests to see if the user entered anything in the Value input field for tripLocation criterion.
`SELECT tripName, tripLocation, departureDate, returnDate, price, tripID FROM trips WHERE #PreserveSingleQuotes(WhereClause)#`	SQL query to execute. The `PreserveSingleQuotes` function ensures that quotation marks are passed to the database server as intended.

The preceding code only builds the tripLocation subclause. In the following exercise, you will add code for the other two columns that you can query, departureDate and price.

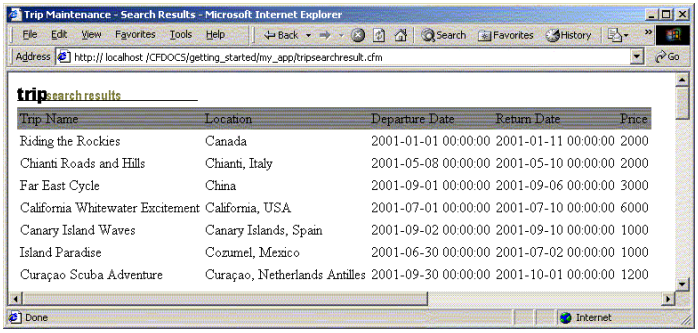
Exercise 4: Testing the Trip Search Results page

In this exercise, you will test the tripsearchresults.cfm page. First, you will test the Trip Search Results page by entering criteria on the Trip Search form and inspecting the results. Next, you will finish the code to construct the complete WHERE clause for all three columns that you can query from the Trip Search form.

To test the Trip Search Results page:

1. View the tripsearchform.cfm page from the my_app directory in your browser and do the following:
   a. In the Trip Location drop-down list, select the Begins With option, and enter the value **C** in the text box.
   b. Click Search.

      The Trip Results page displays several entries, as follows:



   c. Notice in the Trip Results page that only one trip has a trip location of China.
   d. Click the Back button in your browser to return to the Trip Search page.
2. In the Trip Location drop-down list of the Trip Search page, select the Is option, enter the value **China**, and then click Search.

   The Trip Search Results page displays only one entry for the trip to China in the HTML table.

3. Verify that the other criteria (departure date and price) are not taken into consideration yet as follows:
   a Click the Back button in the browser to return to the Trip Search page.
   b In the Departure Date drop-down list box, select Before, enter **1/1/1900** as the date, and select Smaller Than 0 for the price.

      Either of these conditions would produce a results page with no rows.
   c Click the Search button.

      The Trip Search Results page should be identical to the one in Step 2, because the code to build the WHERE clause in the Trip Search Results page does not include departure date and price.

Exercise 5: Enabling the departure and price criteria on the Trip Search form

In this exercise, you will modify the Trip Search Results page to add the criteria needed for the departure and price query.

To enable the departure and price criteria:

1. Open the gettrips.cfc file.
2. In the getTripsFromForm method, position the pointer before the `cfquery` code block.
3. To build the `departureDate` WHERE subclause, enter or copy and paste the code in the following table immediately *before* the `cfquery` code block:

For this database	Enter this code

(Windows users) Microsoft Access database file	`<!--- Build subclause for departureDate --->`
	`<cfif Form.departureValue GT "">`
	`<cfif Form.departureOperator EQ "EQUALS">`
	`<cfset WhereClause = WhereClause & " and departureDate = " &`
	`CreateODBCDate(Form.departureValue)>`
	`<cfelseif Form.departureOperator EQ "AFTER">`
	`<cfset WhereClause = WhereClause & " and departureDate > " &`
	`CreateODBCDate(Form.departureValue)>`
	`<cfelseif Form.departureOperator EQ "BEFORE">`
	`<cfset WhereClause = WhereClause & " and departureDate < " &`
	`CreateODBCDate(Form.departureValue)>`
	`</cfif>`
	`</cfif>`

(UNIX users) PointBase database file	`<!--- Build subclause for departureDate --->`
	`<cfif Form.departureValue GT "">`
	`<cfif Form.departureOperator EQ "EQUALS">`
	`<cfset WhereClause = WhereClause & " and departureDate = Date '" &`
	`Form.departureValue & "'">`
	`<cfelseif Form.departureOperator EQ "AFTER">`
	`<cfset WhereClause = WhereClause & " and departureDate > Date '"`
	`& Form.departureValue & "'">`
	`<cfelseif Form.departureOperator EQ "BEFORE">`
	`<cfset WhereClause = WhereClause & " and departureDate < Date '" &`
	`Form.departureValue & "'">`
	`</cfif>`
	`</cfif>`

4. To build the `price` WHERE subclause, enter the following code after the code you entered in the previous step.

```
<!--- Build subclause for price--->
<cfif Form.priceValue GT "">
  <cfif Form.priceOperator EQ "EQUALS">
    <cfset WhereClause = WhereClause & " and price = " & form.priceValue>
  <cfelseif Form.priceOperator EQ "GREATER">
    <cfset WhereClause = WhereClause & " and price > " & form.priceValue>
  <cfelseif Form.priceOperator EQ "SMALLER">
    <cfset WhereClause = WhereClause & " and price < " & form.priceValue>
  </cfif>
</cfif>
```

5. Save the file.

Exercise 6: Testing the revised Trip Search Results page

In this exercise, you will verify that the `price` and `departureDate` are now considered in the query.

To test the revised Trip Search Results page:

1. Open the tripsearchform.cfm page in the my_app directory in your browser.
2. In the Departure Date drop-down list box, select Before, enter **1/1/1900** as the date (specify **1900-1-1** on UNIX).
3. Select Smaller Than, and enter 0 for the price.
4. Click the Search button.

   Because the departure date is considered in the query, there are no rows returned.

   Note: If you planned to use many more fields as search criteria, the approach used to add departure date and price criteria to the Trip Search form is not the most elegant solution. A generic routine to handle WHERE clause string construction based on specific data types could reduce the code and be a more extensible solution than the one presented here. However, this more extensible approach is beyond the scope of this tutorial.

Summary

This lesson described how to build a search tool that dynamically builds a WHERE clause of the SQL SELECT statement using `cfif` and `cfset` tags. To ensure that the SQL statement remains intact, you used the `PreserveSingleQuotes` CFML function.

In the next lesson

In the next lesson, you will build the Trip Detail page, which will display detailed information about the currently selected trip.

CHAPTER 8
Lesson 5: Creating a Trip Detail Page

In this lesson you will enhance the Compass Travel Trip Maintenance application. This lesson contains the following exercises:

Exercise 1: Creating a Trip Detail page 86
Exercise 2: Avoiding the potential security risk when using dynamic SQL 91
Exercise 3: Linking the Trip Search Results page with the Trip Detail page 92
Exercise 4: Enhancing the look of the Trip Search Results and Trip Detail pages 93

ColdFusion tags and functions introduced in this lesson

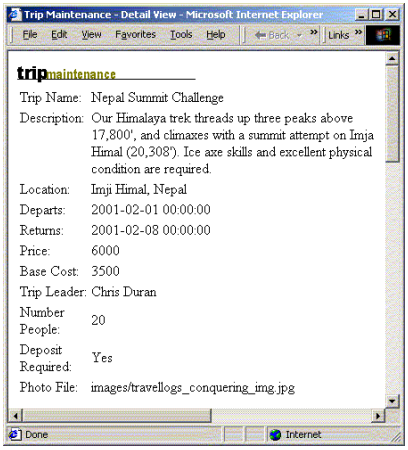
The following table describes the ColdFusion tags and functions that you use in this lesson to enhance the sample ColdFusion application:

Element	Type	Description
`DollarFormat`	Function	Returns a number as a String formatted with two decimal places, a thousands separator, and a dollar sign. If the specified number is negative, parentheses are used.
`DateFormat`	Function	Returns a formatted date-time value. If no mask is specified, the `DateFormat` function returns the date value in the *dd/mmm/yy* format. The `DateFormat` function supports dates that have the U.S. date format.
`IsDefined`	Function	Evaluates a String value to determine whether it represents an existing variable. Returns True if the variable is found, False if not found.
`URLEncodedFormat`	Function	Converts a text string into a String that you can safely use in a URL.

Exercise 1: Creating a Trip Detail page

By design, the Trip Search Results page displays a subset of the information about a trip. To get additional information about any of the trips, the user should be able to click any row to display the detailed trip data.

In this exercise, you build a Trip Detail page to provide all the information about a particular trip that is stored in the Compass Travel trips database. The following figure shows an example of the Trip Detail page that you build:



After you complete the Trip Maintenance application in this tutorial, you will use this Trip Detail page in several ways:

- You can call the Trip Detail page directly by typing in the address of the page with an ID. For example, to view trip information for Rio Cahabon Rafting with tripID 24, you open a browser and enter the following URL:

  ```
  http://localhost/cfdocs/getting_started/my_app/tripdetail.cfm?ID=24
  ```

- You can navigate to the Trip Detail page by creating a hyperlink from the trip name on the Trip Search Results page. This will offer the user drill-down capability when searching for trips. You will link the Trip Search Results page and the Trip Detail page in one of the exercises in this lesson.
- You can use browse buttons on the Trip Detail page to navigate the Trips table row-by-row. You will implement this navigational feature in "Lesson 6: Creating a Main Application Page" on page 97.

Building the Trip Detail page

To build the Trip Detail page, you do the following:

- Create a CFC that queries the database.
- Create the Trip Detail page, which invokes the CFC query and displays the results of the query.

To build the CFC query:

1. Create a blank ColdFusion page.
2. Save the file as tripdetail.cfm in the my_app directory.
3. Perform the following steps, or do the steps in the "Let Dreamweaver do it" section.
   a. Enter the following code:

```
<cfinvoke
  component="cfdocs.getting_started.my_app.components.displaytripdetail"
  method="getTripDetails"
  returnvariable="tripDetails">
</cfinvoke>
```

   b. Save the file.
   c. Create a file named displaytripdetail.cfc in the my_app/components directory.
   d. Enter the following code:

```
<cfcomponent>
  <cffunction name="getTripDetails" access="public" returntype="query">
    <cfset var tripDetails = "" >
    <cfquery name="tripDetails" datasource="CompassTravel" maxrows=1>
    SELECT * FROM trips
       <cfif IsDefined("URL.ID")>
       WHERE tripID = #URL.ID#
       </cfif>
    </cfquery>
    <cfreturn tripDetails>
  </cffunction>
</cfcomponent>
```

   e. Save the file.

Let Dreamweaver do it

You can use the Dreamweaver Extensions to build the CFC query.

To create the CFC query in Dreamweaver:

1. Open a new file in the my_app directory in Dreamweaver.
2. Save the file as tripdetail.cfm.
3. Click the Bindings tab.
4. Click the + button.
5. Select Recordset (Query).
6. In the Name text box, enter **tripDetails**.
7. In the Data sources list, select CompassTravel.
8. Click the CFC Query button.
9. In the Name text box, enter **tripDetails**.
10. Click the Create New Component button.
11. In the Name text box, enter **displaytripdetail**.

12. In the Component Location text box, enter **/components/**.
13. In the Recordset name text box, enter **tripDetails**.
14. In the Function text box, enter **getTripDetails**.
15. In the Data source list, select CompassTravel.
16. In the Table list, select trips.
17. Click OK.
18. When asked whether to include dependent files, click Yes.
19. Click OK.
20. In the displaytripdetail.cfc file, modify the `cfquery` code block by adding the highlighted text:

```
<cfquery name="tripDetails" datasource="CompassTravel" maxrows=1>
    SELECT * FROM trips
       <cfif IsDefined("URL.ID")>
       WHERE tripID = #URL.ID#
       </cfif>
    </cfquery>
```

21. Save the file.

To display the results of the query on the Trip Detail page:

1. Open the tripdetail.cfm file.
2. Enter or copy and paste the following `cfoutput` code after the `cfinvoke` code block:

```
<cfoutput query="TripDetails">
  <img src="images/tripmaintenance.gif">
<table>
  <tr>
    <td valign="top">Trip Name:
    </td>
    <td>#tripName#
    </td>
  </tr>
  <tr>
    <td valign="top">Description:
    </td>
    <td>#tripDescription#
    </td>
  </tr>
  <tr>
    <td valign="top">Location:
    </td>
    <td>#tripLocation#
    </td>
  </tr>
  <tr>
    <td valign="top">Departure Date:
    </td>
    <td>#departureDate#
    </td>
  </tr>
```

```
  <tr>
    <td valign="top">Return Date:
    </td>
    <td>#returnDate#
    </td>
  </tr>
  <tr>
    <td valign="top">Price:
    </td>
    <td>#price#
    </td>
    </tr>
  <tr>
    <td valign="top">Base Cost:
    </td>
    <td>#baseCost#
    </td>
  </tr>
  <tr>
    <td valign="top">Trip Leader:
    </td>
    <td>#tripLeader#
    </td>
  </tr>
  <tr>
    <td valign="top">Number People:
    </td>
    <td>#numberPeople#
    </td>
  </tr>
  <tr>
    <td valign="top">Deposit Required:
    </td>
    <td>#depositRequired#
    </td>
  </tr>
  <tr>
    <td valign="top">Photo File:
    </td>
    <td>#photo#
    </td>
  </tr>
</table>
</cfoutput>
```

3. To provide a title that appears in the browser window, insert the following HTML code before the `<cfoutput query = "TripQuery">` line:

```
<html><head>
  <title> Trip Maintenance - View Record </title>
</head>
<body>
```

4. Add the following code at the end of the file:

```
</body>
</html>
```

5. Save the file.

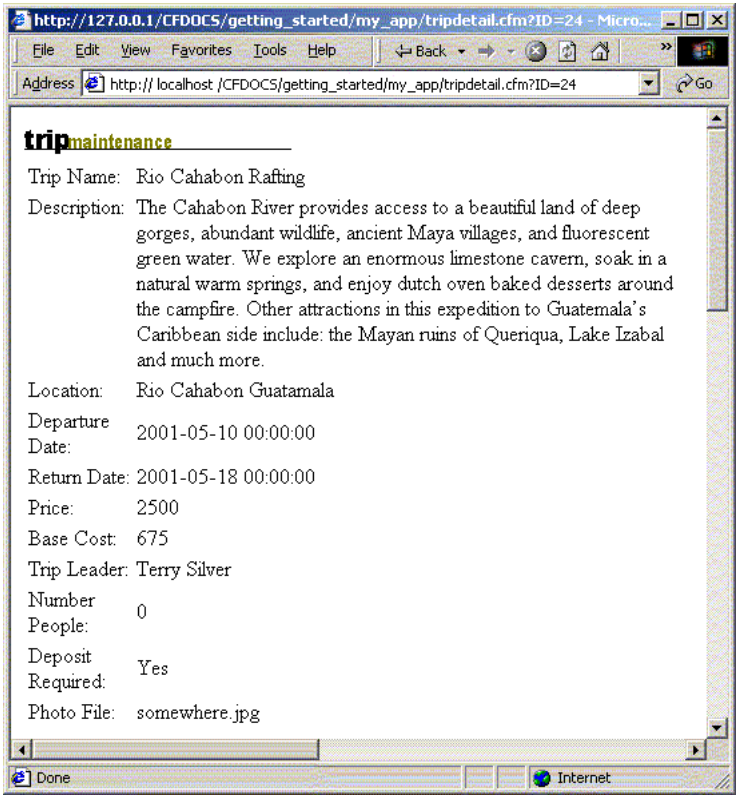
To view the Trip Detail page:

1. Open a browser.
2. Enter one of the following URLs, which use the tripID of 24, which specifies the Rio Cahabon Rafting trip:

   `http://localhost/cfdocs/getting_started/my_app/tripdetail.cfm?ID=24`

   Note: If you are using the built-in ColdFusion server, enter `localhost:8500` instead of `localhost`.

   The following figure shows the expected result:



Reviewing the code

The following table describes the ColdFusion code that you use to build the Trip Detail page:

Code	Explanation

`<cfcomponent>`
`  <cffunction name="getTripDetails"`
`  access="public"`
`  returntype="query">`
`...`
`<cfreturn tripDetails>`
`  </cffunction>`
`</cfcomponent>`
	The `cfcomponent` tag creates a CFC. The `cffunction` tag creates a method named `getTripDetails` in the CFC; the method returns the results of the query to the calling page.

`<cfquery name="TripQuery"`
`  dataSource="CompassTravel"`
`  maxRows=1>`
	The `cfquery` tag includes a `maxRows` attribute. This attribute limits the number of result rows brought back from the database. In the Trip Detail page, you want to show only a single row at a time; therefore, the `maxRows` attribute is set to `1`.

`<cfif IsDefined("URL.ID")>`
`  WHERE tripID = #URL.ID#`
`</cfif>`
	The URL.ID specifies a parameter that you can include in the URL that requests this page. If the ID parameter is passed within the URL, it is used in the SQL query to identify the `tripID` to SELECT. You can use the CFML `IsDefined` function to determine if a parameter is passed within the URL. You can also use `IsDefined` to determine if the user has entered data in form fields prior to the form post action.

As you did in this exercise, you can build comprehensive database query applications using CFML and dynamic SQL. To further test the new Trip Detail page that you created, you will link it to the search facility that you built in Lesson 4: Building Dynamic Queries. However, before you link that search facility, you must understand a potential security risk using dynamic SQL. The following exercise describes this risk and how to avoid it.

Exercise 2: Avoiding the potential security risk when using dynamic SQL

To reduce round trips between the client and the database server, many SQL database servers permit the client to submit multiple SQL statements in a single request, separated by a semicolon (;). For these database managements systems, the following SQL request is valid:

```
DELETE from trips where tripLocation = 'China'; SELECT tripName from trips
```

This request might be an efficient way to list the trips that remain after the database management system removes the China trip. Problems arise when the SQL statement is built dynamically.

In the Trip Maintenance application, when the client program or user passes an ID in the URL that calls the Trip Detail page, the page displays the relevant trip information. The following code builds the correct WHERE clause that supports this behavior:

```
<cfif IsDefined("URL.ID")>
  WHERE tripID = #URL.ID#
</cfif>
```

If a user called the Trip Detail page using the following statement:

```
http://localhost/cfdocs/getting_started/my_app/tripdetail.cfm?ID=24;DROP+trips
```

the SQL database management system executes the proper SQL SELECT statement, and then immediately erases the Trips table from the database.

Protecting your application

To ensure that your application is protected from such an attack, you can exploit the fact that the ID must be a numeric value. The CFML `Val` function returns the numeric value at the beginning of a string expression. You can use the `Val` function as follows:

```
<cfif IsDefined("URL.ID")>
  WHERE tripID = #Val(URL.ID)#
</cfif>
```

If nonnumeric data is passed within the URL ID field, the `Val` function returns 0, and the trip with ID 0 appears (if one exists). If the user enters the previously cited URL (http://localhost/cfdocs/getting_started/my_app/tripdetail.cfm?ID=24;DROP+trips), the application ignores the non-numeric values and displays the trip information of trip ID 24.

Warning: The exercises in this tutorial ignore the dynamic SQL risk from attack. To eliminate this risk, you should use ColdFusion functions (such as `Val`) to perform type checking on all URL parameters. For queries, you can also use the `cfqueryparam` tag, which is explained in *CFML Reference*.

Exercise 3: Linking the Trip Search Results page with the Trip Detail page

In this exercise, you will modify the Trip Search Results page to let the user view the details of any trip. To do this, you will convert each trip name entry in the results page to a link, which will display the trip's detailed information in the detail page.

Use the following steps to link the Trip Search Results page (tripsearchresult.cfm) to the Trip Detail page (tripdetail.cfm).

To create links between the Trip Search Results page and the Trip Detail page:

1. Open the tripsearchresult.cfm file from the my_app directory.
2. Replace `#tripName#` in the `cfoutput` block with the following code:

   ```
   <a href="tripdetail.cfm?ID=#URLEncodedFormat(tripID)#">#tripName# </a>
   ```

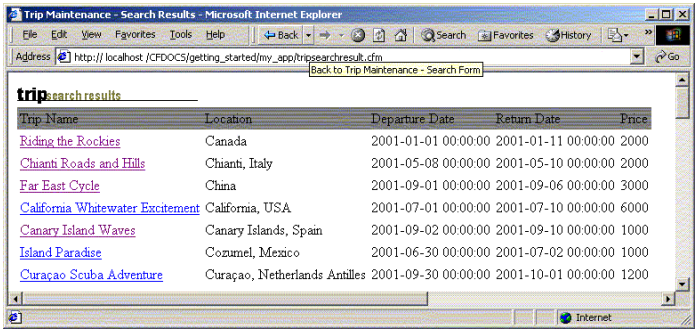
   Note: The `URLEncodedFormat` is a ColdFusion function that returns a URL-encoded string. Spaces are replaced with `%20`, and nonalphanumeric characters with equivalent hexadecimal escape sequences. The function lets you pass arbitrary strings within a URL, because ColdFusion automatically decodes URL parameters that are passed to the page.

3. Save the file.

To test the revised page:

1. View the tripsearchform.cfm page from the my_app directory in your browser.
2. In the Trip Location drop-down list, select Begins With, and enter the value C.
3. Click Search.

   The Trip Search Results page displays a hyperlink for each trip name, as the following figure shows:



4. To view the Trip Detail page for a trip, click the trip name.

You might notice that the dates and prices in both the Trip Detail and Trip Search Results pages are unformatted. You will improve the appearance of the application in "Exercise 4: Enhancing the look of the Trip Search Results and Trip Detail pages" on page 93.

Exercise 4: Enhancing the look of the Trip Search Results and Trip Detail pages

The Trip Maintenance application now provides a useful drill-down mechanism for locating trips. Although this application is functionally sound, its appearance could be improved, including formatting dates and dollar amounts, and making long lists easier to read.

Formatting dates and dollar amounts

ColdFusion provides several functions to improve application appearance. These include the `DateFormat` and `DollarFormat` functions that format dates and currency variables.

To format the dates and dollar amounts:

1. To format the Trip Detail page dollar and date fields, open the tripdetail.cfm file in the my_app directory in your editor and make the following changes:

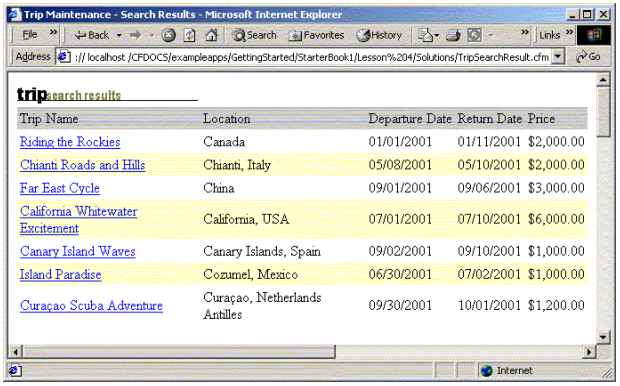
Existing code	Replacement code
`#departureDate#`	`#DateFormat(departureDate, "mm/dd/yyyy")#`
`#returnDate#`	`#DateFormat(returnDate, "mm/dd/yyyy")#`
`#price#`	`#DollarFormat(price)#`
`#baseCost#`	`#DollarFormat(baseCost)#`

2. Save the file.
3. To format the currency and date fields on the Trips Search Results page, open the tripsearchresult.cfm file in your editor and make the changes for `departureDate`, `returnDate`, and `price` indicated in the table.
4. Save the file.

To view the application's new appearance:

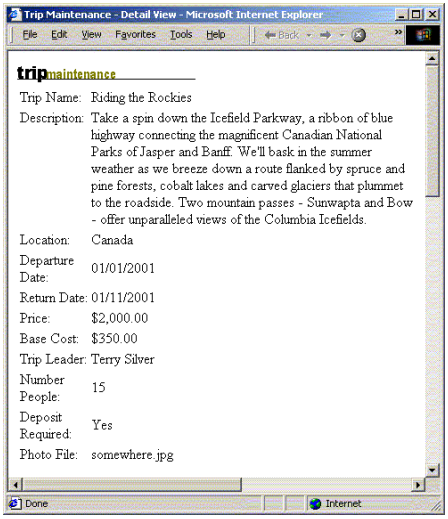
1. View the tripsearchform.cfm page in the my_app directory in your browser.
2. In the Trip Location drop-down list, select Begins With and enter the value **C**.
3. Click Search.

   The Trip Search Results page appears:



4. In the Trip Search Results page, click the link for Riding the Rockies.

   The properly formatted Trip Detail page appears:



Summary

In this lesson, you transformed the search facility that you built in "Lesson 4: Building Dynamic Queries" into a drill-down facility for trip queries. You built a Trip Detail page to show more information about a particular trip. You also formatted the Trip Search Results and Trip Detail pages using the CFML `DateFormat` and `DollarFormat` functions. You linked the Trip Search Results page with the Trip Detail page.

In the next lesson

In the next lesson, you will add navigation and maintenance buttons on the main Trip Maintenance application page.

CHAPTER 9
Lesson 6: Creating a Main Application Page

In this lesson you will enhance the Compass Travel Trip Maintenance application. So far, you created a very useful drill-down query facility. Compass Travel trip coordinators can produce lists required by management and easily locate and display information about any trip. There are several requirements that were identified in "Lesson 1: Preparing to Build the Sample Application" on page 41 that you have not yet addressed:

- The ability to browse through the Trips table.
- The ability to add, delete, and edit trip information.

You will modify the Trip Detail page to accomplish this additional functionality. The Trip Detail page shows information about a single trip. You will convert the Trip Detail page into the main application page by adding the following functionality:

- Navigation buttons to browse the database
- Database maintenance buttons to edit, delete, and add new trips, as well as a button to search

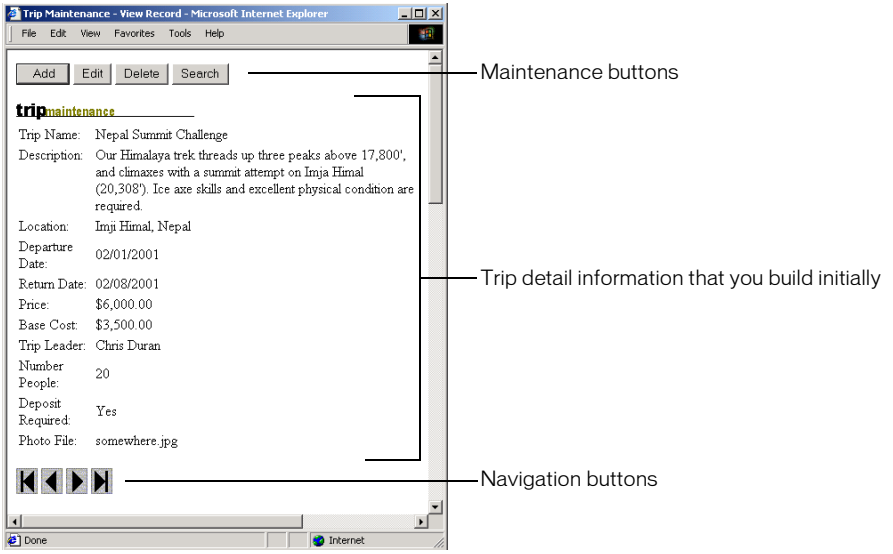
This lesson contains the following exercises:

Exercise 1: Creating the main application page from the Trip Detail page 98

Exercise 2: Adding database maintenance buttons 100

Exercise 1: Creating the main application page from the Trip Detail page

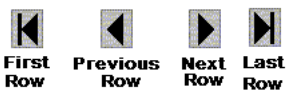
In this exercise, you convert the Trip Detail page into the main Trip Maintenance application page. The main application page includes additional buttons for navigating to other ColdFusion pages and browsing the trip database records.



Adding navigation buttons to browse the database

The drill-down search function developed in "Lesson 5: Creating a Trip Detail Page" on page 85 is very useful when the user knows some search criteria to enter. Flipping back and forth between the results page and the detail page to navigate through a record set can be tedious. Moreover, on occasion the trip coordinator might want to browse the Trips database just to check for anomalies or to become familiar with its contents. In these cases, the trip coordinator does not know the criteria to search for in advance.

The following figure shows the navigation buttons. The label below each button does not appear in the application; it describes which row to display relative to the currently displayed row.



To add navigation buttons to the Trip Detail page:

1. Open the tripdetail.cfm file in the my_app directory.
2. Insert the following code between the `</table>` and `</cfoutput>` tags:

```
<form action="navigationaction.cfm" method="post">
  <input type="hidden" name="RecordID" value="#tripID#">
  <!--- graphical navigation buttons --->
  <input type="image" name="btnFirst" src="images/first.gif">
  <input type="image" name="btnPrev" src="images/prev.gif">
  <input type="image" name="btnNext" src="images/next.gif">
  <input type="image" name="btnLast" src="images/last.gif">
</form>
```

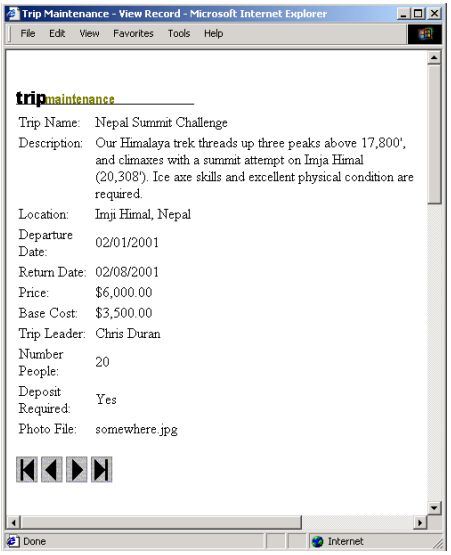
   Note: The current trip record ID (`tripID`) is in a hidden field in the form code. This field provides the action page with current record ID that it must have in order to build the query to access the appropriate record in the Trips database table.

3. Save the file.

To test the updated application:

1. View the updated tripdetail.cfm page in a browser.

   The Trip Search Results page appears:



2. Test the buttons by clicking any navigation button.

   An error occurs because the navigation action page (navigationaction.cfm) does not exist. The navigation action page processes the navigation button requests. You will build the navigation action page in "Lesson 7: Validating Data to Enforce Business Rules" on page 103.

Reviewing the code

The following table describes the navigation code for the Trip Detail page:

Code	Explanation
`<form action="navigationaction.cfm" method="post">`	Form tag that identifies the navigationaction.cfm file to handle record navigation.
`<INPUT type="hidden" name="RecordID" value="#tripID#">`	Hidden `RecordID` field with the value of the current `tripID`.

`<input type="image" name="btnFirst" src="images/first.gif">`
`<input type="image" name="btnPrev" src="images/prev.gif">`
`<input type="image" name="btnNext" src="images/next.gif">`
`<input type="image" name="btnLast" src="images/last.gif">`
	Navigation buttons that are image type HTML input tags.

Exercise 2: Adding database maintenance buttons

The search and sequential navigation capabilities are features for locating Compass Travel trips. After locating a trip, the trip coordinator must be able to modify or delete it. Additionally, when viewing the detail for a trip, the trip coordinator must be allowed to add a new trip or use the search facility. To enable trip coordinators to do this, you add the following buttons to the Trip Detail page:



As described in "Exercise 1: Creating the main application page from the Trip Detail page" on page 98, it is important to pass the current record ID (`tripID`) to the action page to build the proper SQL statement to process the navigation button requests. It is also important to pass the current record ID to the Maintenance Action page. Therefore, you use an HTML `input` tag to hide the current `recordID` and post it to the maintenanceaction.cfm page.

To add maintenance buttons:

1. Open the tripdetail.cfm file from the my_app subdirectory.
2. Enter the following code immediately after the `<cfoutput query="TripQuery">` tag:

```
<form action="maintenanceaction.cfm" method="post">
  <input type="hidden" name="RecordID" value="#tripID#">
  <input type="submit" name="btnAdd" value="   Add    ">
  <input type="submit" name="btnEdit" value="  Edit   ">
  <input type="submit" name="btnDelete" value="Delete">
  <input type="submit" name="btnSearch" value="Search">
</form>
```

   Note: The current trip record ID (`tripID`) is in a hidden field in the form code. This field provides the action page with current record ID that it must have in order to build the query to access the appropriate record in the Trips database table.

3. Save the file.

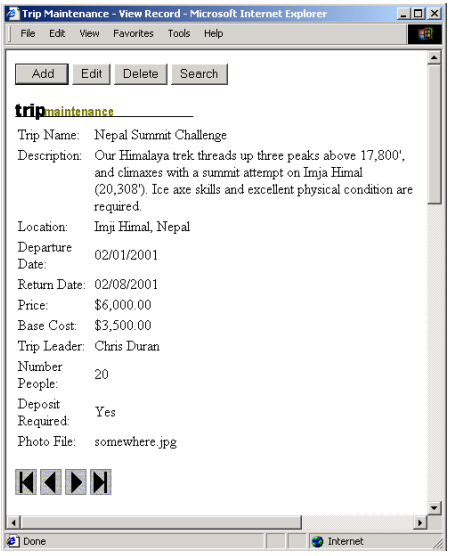
To test the updated application:

1. Open your browser.
2. Enter the following URL to view the updated tripdetail.cfm page

   ```
   http://localhost/cfdocs/getting_started/my_app/tripdetail.cfm?ID=8
   ```

   Note: If you are using the built-in ColdFusion server, enter `localhost:8500` instead of `localhost`.

   The page appears as follows:



3. Click Search or Delete to test the database maintenance buttons.

   An error occurs because the Maintenance Action page does not exist. The Maintenance Action page is required to process the maintenance button requests. You will develop this page in "Lesson 7: Validating Data to Enforce Business Rules" on page 103.

Summary

In this lesson, you added trip maintenance and navigation buttons, which converted the Trip Detail page into the main Trip Maintenance application page.

In the next lesson

In "Lesson 7: Validating Data to Enforce Business Rules" on page 103, you will build the action pages required to implement the navigation and maintenance buttons on the main Trip Maintenance application page.

CHAPTER 10
Lesson 7: Validating Data to Enforce Business Rules

In this lesson, you will create a page for the trip coordinator to add new trip offerings and update existing trips. In addition, you will add logic to validate that data entered so that it complies with Compass Travel business rules.

This lesson contains the following exercises:

Exercise 1: Using an HTML form to collect data 104

Exercise 2: Creating a simple action page 107

Exercise 3: About data validation 108

Exercise 4: Providing server-side validation 109

Exercise 5: Validating data on the client using ColdFusion form tags 114

Exercise 6: Dynamically populating the list of event types 119

Exercise 7: Validating the existence of the trip photo file 120

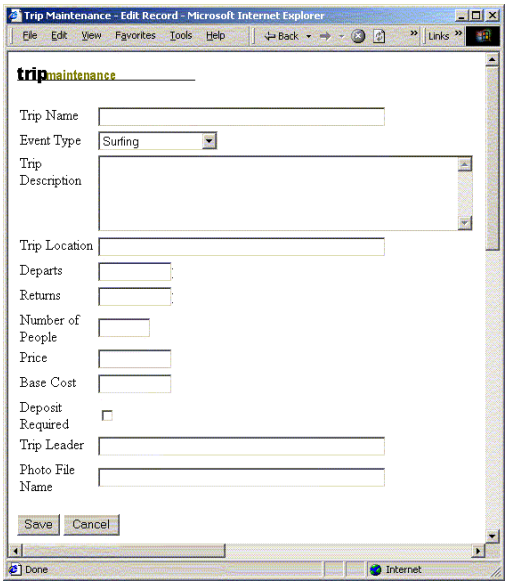
ColdFusion tags and functions introduced in this lesson

The following table describes the ColdFusion tags and functions that you use in this lesson to enhance the ColdFusion application:

Element	Type	Description
`cfform`	Tag	Builds a form with CFML custom control tags that provide more functionality than standard HTML form input elements.
`cfinput`	Tag	Use inside the `cfform` tag to place radio buttons, check boxes, or text boxes. Provides input validation for the specified control type.
`cfselect`	Tag	Use inside the `cfform` tag. Lets you construct a drop-down list form control. You can populate the drop-down list from a query, or use the HTML `option` tag. You also use `option` elements to populate lists. The syntax for the `option` tag is the same as for its HTML counterpart.
`FileExists`	Function	Returns True if the file specified in the argument exists, False if it doesn't exist.

Exercise 1: Using an HTML form to collect data

In this exercise, you develop the Trip Edit page, which provides a data entry form that is used to add new trips and edit existing trips. You validate the data entered against Compass Travel business rules. The fields required to capture trip information are the same as those on the Trip Detail page that you used to display trip information in "Lesson 6: Creating a Main Application Page" on page 97. The following figure shows the Trip Edit page:



The page appears when the user clicks the Add or Edit button on the main Trip Maintenance application page (tripdetail.cfm).

To create the Trip Edit data collection form:

1. Create a blank file.
2. Enter or copy and paste the following code into the file:

```
<html>
<head><title>Compass Travel Trip Maintenance</title></head>
<body>
  <form action="tripeditaction.cfm" method="post">
    <!--- Field: Trip Maintenance Image --->
  <IMG src="images/tripmaintenance.gif">
  <P>
  <table>
      <!--- Field: tripName --->
      <TR>
        <TD valign="top"> Trip Name </TD>
        <TD><INPUT type="text" name=tripName size="50"></TD>
    </TR>
      <!--- Field: eventType --->
    <tr>
      <td valign="top">Type of Event</td>
```

```
  <td><select size="1" name="eventType">
    <option value="1" selected>Surfing</option>
    <option value="2">Mountain Climbing</option>
    <option value="3">Mountain Biking</option>
    </select>
  </td>
</tr>
  <!--- Field: tripDescription --->
  <TR>
    <TD valign="top"> Trip Description </TD>
    <TD>
     <textarea cols="50" rows="5" name="tripDescription"></textarea>
  </TD>
</TR>
  <!--- Field: tripLocation --->
  <TR>
    <TD valign="top">Trip Location</TD>
    <TD><INPUT name=tripLocation size=50></TD>
</TR>
  <!--- Field: departureDate --->
  <TR>
    <TD valign="top"> Departure Date</TD>
    <TD><INPUT name=departureDate size=10></TD>
</TR>
  <!--- Field: returnDate --->
  <TR>
    <TD valign="top"> Return Date</TD>
    <TD><INPUT name=returnDate size=10></TD>
</TR>
  <!--- Field: numberPeople --->
  <TR>
    <TD valign="top">Number of People</TD>
    <TD><INPUT size=6 name=numberPeople></TD>
</TR>
  <!--- Field: Price --->
  <TR>
    <TD valign="top">Price</TD>
    <TD><INPUT size=10 name=price></TD>
</TR>
  <!--- Field: baseCost --->
  <TR>
    <TD valign="top"> Base Cost </TD>
    <TD><INPUT size=10 name=baseCost></TD>
</TR>
  <!--- Field: depositRequired --->
  <TR>
    <TD valign="top"> Deposit Required </TD>
    <TD><input type="checkbox" name="depositRequired" value="Yes"></TD>
</TR>
  <!--- Field: tripLeader --->
  <TR>
    <TD valign="top">Trip Leader</TD>
    <TD><INPUT maxLength=50 size=50 name=tripLeader></TD>
</TR>
  <!--- Field: photo --->
```

```
      <TR>
        <TD valign="top">Photo File Name</TD>
        <TD><INPUT maxLength=50 size=50 name=photo></TD>
      </TR>
    </table>
    <p>
    <input type="submit" value="Save">
    <input type="submit" value="Cancel" name="btnCancel">
    </p>
    </form>
  </body>
  </html>
```

3. Save the file as tripedit.cfm in the my_app directory.

Reviewing the code

The following table explains the use of some of the HTML tags in the Trip Edit page. For more information on HTML, consult any HTML primer.

Tag	Description
Form	You create a data entry form by using the `form` tag. The `form` tag takes two tag attributes; for example: `<form action="tripeditaction.cfm" Method= "Post">` Here, the `action` attribute specifies the name of the ColdFusion file that the web server will navigate to in response to the form's submission. The `method` attribute specifies how data is returned to the web server. Submit all ColdFusion forms using the `Post method` attribute.
Table	You can format a data entry form and display its controls neatly, by using the table tag, `table`; table row tag, `tr`; and the table data tag, `td`.
Form Controls	The form requires controls to collect and submit user input. There are a variety of types of form controls that you can use. For this lesson, you will use the following controls: • `<input>` Accepts text answers, such as Trip Name and Trip Price. • `<input type=checkbox>` Asks yes or no questions, such as Deposit Required? • `<select>`,`<option>` Provides user with a list of possible answers, such as the event type (Mountain Biking, Surfing, and so on). • `<textarea>` Gathers user input on multiple lines, such as for the Trip Description. • `<input type=submit>` Posts the information collected to the server.

To test the Trip Edit page:

1. View the tripedit.cfm page in a browser.
2. Enter a trip name in the Trip Name field.
3. Click Save.

   An error occurs.

If you view the form source (tripedit.cfm) in an editor, you can see that the `<form>` tag has an `action` attribute. This attribute indicates the page that receives the form values posted by the tripedit.cfm page. Because you have not yet created the tripeditaction.cfm page, ColdFusion MX sends an error.

At this point, this form does not store any information in the database and does not enforce any business rules of Compass Travel. In the next exercise, you develop the action page to enforce the business rules.

Exercise 2: Creating a simple action page

In this exercise, you develop an action page that you will eventually use to insert or update trip data passed from the Trip Edit page into the trips table of the Compass Travel database.

To build the Trip Edit action page:

1. Create a blank file.
2. Enter or copy and paste the following code into the file:

```
<html>
  <head>
    <title>Trip Maintenance Confirmation</title>
  </head>
  <body>
      <h1>Trip Added</h1>
      <!--- Database insert logic goes here. --->
      <cfoutput>
        You have added #Form.TripName# to the trips database.
      </cfoutput>
  </body>
</html>
```

3. Save the page as tripeditaction.cfm in the my_app directory.

To test the trip edit page:

1. Open the tripedit.cfm page in the my_app directory in a browser.
2. In the Trip Name field, enter **12345**.
3. Click Save.

   The message "You have added 12345 to the trips database." appears. The tripeditaction.cfm page does not actually update the database yet; it simply displays a message saying that you added the trip to the database. Before you add the code to allow updates to the database, you must add the logic to validate the data entered in the form.

Reviewing the code

The following table describes the code that you use to verify whether a file exists:

Code	Explanation

`<h1>Trip Added</h1>`
`<!--- Database insert logic goes here. --->`
`<cfoutput>You have added #Form.TripName# to the trips database.`
`</cfoutput>`
	Displays the heading "Trip Added." Displays the message "You have added *TripName* to the trips database," where TripName is the trip name you entered in the form's Trip Name field.

Exercise 3: About data validation

To ensure that the data entered in the trip edit form is valid, Compass Travel has identified a set of business rules for capturing and editing trip information. A fundamental concern is ensuring that the captured data is suitable for the column definitions in the Trips table. This type of validation on a single field is often referred to as a *single-field edit*.

Compass Travel has other operating policies that involve evaluating the values from more than one field. These validations, referred to as *cross-field edits*, are usually more difficult to program. To ensure that new trips are uniformly captured, Compass Travel has published cross-field validations and single-field edits in its Compass Travel business rules.

The following table lists the Compass Travel business rules for capturing and editing trip information. This table identifies which rules require single-field or cross-field editing.

	Compass Travel new trip policy	Edit type
1	All trips must be named.	Single-field
2	All trips must be accompanied by a full description.	Single-field
3	Each trip must be categorized by event type. Only valid event types (1-surfing, 2-mountain climbing, and so on) are permissible.	Single-field
4	Trip locations are required.	Single-field
5	The maximum number of people permitted on the trip must be specified.	Single-field
6	The trip departure and return dates must be specified for each trip. All trip dates must be valid future dates. Departure date must precede return date.	Single-field Cross-field
7	The trip's price and base cost are required. Both values are positive numeric values. The trip price must have at least a 20% markup over base cost.	Cross-field
8	Any trip priced over $750 requires a deposit.	Cross-field
9	A trip leader must be identified.	Single-field
10	A photo must accompany all new trips. The photo image file must reside in the images directory of the Compass Travel website.	Single-field

ColdFusion provides numerous ways to validate data. For more information, see Chapter 28, "Validating Data" in *ColdFusion MX Developer's Guide*. In this lesson, you use special tags to simplify the process of enforcing business rules. Using ColdFusion, it is possible to enforce business rules in several places. For example, you can enforce some validation edits on the client. You can enforce other validation edits on the server after the data entry form is submitted. You will explore these options in the following exercises.

Exercise 4: Providing server-side validation

In this exercise, you will learn about the following tasks:

- Creating a local variable to indicate whether data entered in the form conforms to the Compass Travel business rules
- Ensuring that a value was entered
- Evaluating check box and radio button variables
- Adding validation for all Compass Travel business rules

Creating a local variable

The purpose of the tripeditaction.cfm action page is to update the Compass Travel database, so it is important to make certain that any values entered conform to all the business rules before modifying the database. Failure of any one of the rules prevents modification of the database.

One approach to ensuring that the action page considers each business rule is to create a local variable with a `cfset` tag within the action page that tests whether any of the business rules failed.

The `cfset` tag lets you manipulate the value of a variable. For example, the following pseudocode initializes a variable to a specific value and checks the value using the `cfif` statement:

```
<cfset isOk = "Yes">
if rule 1 fails then
<cfset isOK = "No"
...
if Rule n fails then
<cfset isOk = "No">
...
<cfif isOk = "Yes">
  update the database
</cfif>
```

In this example, the `cfset` tag initializes the local variable `isOk` to Yes. If any rule fails, the variable `isOK` is set to No. The code then tests if `isOk` equals Yes, before executing the SQL insert logic.

To create the local variable:

1. Open the tripeditaction.cfm file in an editor.
2. Add the following code at the top of the file:

```
<cfset isOk = "Yes">
```

3. Modify the file by adding the following highlighted code:

```
<cfif isOk EQ "Yes">
  <h1>Trip Added</h1>
  <!--- Database insert logic goes here. --->
  <cfoutput>
    You have added #Form.TripName# to the trips database.
  </cfoutput>
</cfif>
```

4. Save the file.

Because you did not yet add any logic to test whether any values entered in the form are valid, the page works precisely as before, displaying a message that indicates the database was updated, without actually performing the update.

For more information about using the `cfset` and `cfif` tags, see *ColdFusion MX Developer's Guide* or *CFML Reference*.

Ensuring that a value was entered

The first approach that you take to enforce Compass Travel business rules is to enhance the action page to validate the data collected on the data entry form. The action page receives a form variable for every field on the form that contains a value. You use the `cfif` tag to test the values of these fields to ensure that they adhere to Compass Travel business rules.

You can use the `cfif` tag to create conditions that evaluate to either True or False. To use the `cfif` tag to test whether a trip name was entered (business rule 1) on the Trip Edit form, you add the following `cfif` statement:

```
<cfif Form.tripName EQ "">
  <cfoutput> Trip Name cannot be blank. </cfoutput>
</cfif>
```

In this example, the `cfif` statement tests to see if the value of the form variable `tripName` is blank. If the trip name condition evaluates to True, ColdFusion sends the message "Trip name cannot be blank" to the browser.

Note: The keyword `EQ` is an operator that tests for equality. For more information about the `cfif` tag and its operators, see *ColdFusion MX Developer's Guide*.

To ensure that a Trip Name was entered:

1. Open the tripeditaction.cfm file.
2. Enter the following code after the line `<cfset isOk = "Yes">`.

```
<!--- Trip Name is required. --->
<cfif Form.tripName EQ "">
  <CFSET IsOk = "No">
  <cfoutput>Trip name cannot be blank.<br></cfoutput>
</cfif>
```

3. Save the file.

To test the data validation:

1. Open the tripedit.cfm page in a browser.
2. Verify that there is no value in the Trip Name field.
3. Click Save.

   The error message "Trip name cannot be blank" appears.

Evaluating check box and radio button variables

Business rule 8 in the Compass Travel new trip policy requires you to test the value of the `depositRequired` check box form variable. Check box and radio button variables are only passed to the action page when the user selects these options on the form. An error occurs if the action page tries to use a variable that was not passed.

To ensure that an error does not occur, you use the `IsDefined` function in a `cfif` statement to determine whether the user selected the Deposit Required check box on the form.

The `cfif` statement and the `IsDefined` function evaluate the value of the form variable `depositRequired` to determine if a value exists. The statement `not IsDefined` returns True if the specified variable is not found, and the `cfset` statement sets the form variable to No. A value of No indicates that a deposit is not required; a value of Yes indicates that a deposit is required.

To set the value of the Deposit Required check box:

1. Open the tripeditaction.cfm file.
2. Enter the following code after the first `cfif` code block.

```
<cfif not IsDefined("Form.depositRequired")>
  <cfset form.depositRequired = "No">
</cfif>
```

3. Save the file.

Adding validation for all Compass Travel business rules

You add validation for all the form variables.

To validate all the data passed:

1. Open the tripeditaction.cfm file in the my_app directory.
2. Enter the validation code blocks in the following table after the `cfif` code blocks that you already entered.
3. Save the file.

The following table lists the Compass Travel business rules and shows the corresponding code that ensures that the data entered in the form follows the business rules.

Note: You have already entered validation code for business rule 1. Validation code for business rule 10 is described in more detail later in this lesson.

Rule	Description	Validation code

2	All trips must be accompanied by a full description.	`<!--- Trip description is required. --->`
		`<cfif Form.tripDescription EQ "">`
		`  <cfset IsOk = "No">`
		`  <cfoutput>`
		`  Trip description cannot be blank.`
		`  </cfoutput>`
		`</cfif>`

3	Each trip must be categorized by event type. Only valid event types (1-surfing, 2-mountain climbing, and so on) are permissible.	Because event type 1 (surfing) is selected by default, there is always a value for event type.

4	Trip locations are required.	`<!--- Trip location is required. --->`
		`<cfif Form.tripLocation EQ "">`
		`  <cfset IsOk = "No">`
		`  <cfoutput>`
		`  Trip location cannot be blank.`
		`  </cfoutput>`
		`</cfif>`

5	The maximum number of people permitted on the trip must be specified.	`<!--- Number of people is required and must be numeric. --->`
		`<cfif Form.numberPeople EQ "" or IsNumeric(Form.numberPeople) EQ False>`
		`  <cfset IsOk = "No">`
		`  <cfoutput>`
		`  The number of people must be a number and cannot be blank.`
		`  </cfoutput>`
		`</cfif>`

6	The trip departure and return dates must be specified for each trip.
	All trip dates must be valid future dates. Departure date must precede return date.
		`<cfif form.departureDate GT form.returnDate>`
		`  <cfset isOk = "No">`
		`  <cfoutput>`
		`  Departure date cannot precede return date. Please re-enter.`
		`  </cfoutput>`
		`</cfif>`

Rule	Description	Validation code

7	The trip's price and base cost are required. Both values are positive numeric values. The trip price must have at least a 20% markup over base cost.	`<!--- Base Cost is required and must be numeric. --->`
		`<cfif Form.baseCost EQ "" or`
		`IsNumeric(Form.baseCost) EQ False>`
		`<cfset IsOk = "No">`
		`<cfoutput>`
		`Base Cost must be a number and cannot be`
		`blank.`
		`</cfoutput>`
		`<cfelse>`
		`<!--- Price must be 20% greater than Base Cost. --->`
		`<cfif Form.basecost * 1.2 GT #Form.price#>`
		`<cfset IsOk = "No">`
		`<cfoutput>`
		`Price must be marked up at least 20%`
		`above cost.`
		`</cfoutput>`
		`</cfif>`
		`</cfif>`

8	Any trip priced over $750 requires a deposit.	`<!--- Price is required and must be numeric. --->`
		`<cfif Form.price EQ "" or`
		`IsNumeric(Form.baseCost) EQ False>`
		`<cfset IsOk = "No">`
		`<cfoutput>`
		`Price must be a number and cannot be blank.`
		`</cfoutput>`
		`<cfelse>`
		`<!--- A deposit is required when the price is >`
		`$750. --->`
		`<cfif Form.price GT 750 AND`
		`Form.depositRequired EQ "No">`
		`<cfset IsOk = "No">`
		`<cfoutput>`
		`Deposit is required for`
		`trips priced over $750.`
		`</cfoutput>`
		`</cfif>`
		`</cfif>`

9	A trip leader must be identified.	`<!--- Trip Leader is required. --->`
		`<cfif Form.tripLeader EQ "">`
		`<cfset IsOk = "No">`
		`<cfoutput>`
		`A trip leader must be specified.`
		`</cfoutput>`
		`</cfif>`

10	A photo must accompany all new trips.	`<!--- Photo filename is required. --->`
		`<cfif Form.photo EQ "">`
		`<cfset IsOk = "No">`
		`<cfoutput>`
		`Photo filename must be specified.</cfoutput>`
		`</cfif>`

Note: The code for business rules 7 and 8 uses ColdFusion `cfif` and `cfelse` conditional processing tags. The code inside the `cfif` tags only executes when the condition evaluates to True. To perform other actions when the condition evaluates to False, the `cfelse` tag is used. For more information about using conditional processing tags, see *ColdFusion MX Developer's Guide*.

To test the validation code:

1. View the tripedit.cfm page in the browser.
2. In the form, enter **500** in both the Price and Base Cost fields.
3. Click the Save button.

   The trip price error message displays: "Price must be marked up at least 20% above cost."
4. Click the browser Back button to return to the tripedit.cfm page.
5. To avoid the error, enter **800** in the Price field, and click Save.
6. Test various combinations to ensure that all the Compass Travel business rules are enforced by filling out the fields on the form and clicking Save.

   Testing recommendations:

   - Leave out required fields, such as trip name or location.
   - Enter a nonnumeric value in Number of People, such as **one**.
   - Leave the entire form blank and click Save. The following messages appear:

     Trip name cannot be blank. A trip leader must be specified. Photo filename must be specified. The number of people must be a number and cannot be blank. Trip location cannot be blank. Base cost must be a number and cannot be blank. Price must be a number and cannot be blank.

Exercise 5: Validating data on the client using ColdFusion form tags

In this exercise, you will compare server-side and client-side validation. You then modify the testedit.cfm page to use client-side scripting.

Comparing server-side and client-side validation

Validating data on the server-side has two drawbacks. First, the action page is used for validation, so the form page is not in the browser context when the error is trapped. The user, therefore, will not get immediate feedback from the page where the data was entered. Second, because data capture occurs on the client and validation occurs on the server, the number of round-trips to the server is increased. This can cause increased traffic on the network and the server. If the data is validated on the client, only valid data is posted to the server and traffic is reduced.

An alternative approach to server-side editing is to use client-side scripting. Client-side scripting lets you validate the form data entered on the client prior to posting it to the server. CFML provides alternative versions of standard HTML form tags, which provide the advantages of client-side data validation. These data input tags include `cfinput text`, `cfinput radio`, `cfinput checkbox`, `cfselect`, and others. For more information, see Chapter 28, “Validating Data” in *ColdFusion MX Developer’s Guide*.

ColdFusion form tags include the following attributes:

Attribute	Description
validate	The data type that the field tag validates against. Values include: integer, date, time, telephone, zipcode.
message	The error message that appears if validation fails.
range	The range of permissible values for this tag.
required	An indicator of whether data is required for the corresponding tag.

To use the improved form tags, you must replace the HTML form tag with the `cfform` tag. The following code snippets show the use of the improved ColdFusion form tags. The first code snippet shows how the duration field is validated on the server. The second code snippet shows how ColdFusion form tags simplify field validation on the client.

Server-side validation approach (no cfform tag)

The following code is on the server (tripeditaction.cfm page):

```
<!--- Number of people is required and must be numeric. --->
<cfif Form.numberPeople EQ "" or IsNumeric(Form.numberPeople) EQ False>
  <CFSET IsOk = "No">
  <cfoutput>The number of people must be a number and cannot be blank.
  </cfoutput>
</cfif>
```

Code	Explanation
`<cfif Form.numberPeople EQ "" or IsNumeric(Form.numberPeople) EQ False>`	The `cfif` tag evaluates the value of the form variable `numberPeople` to determine if the user entered a value. The `IsNumeric` function checks whether the value entered on the form was a numeric value.

Client-side validation approach using a cfform tag

The following code is on the client (tripedit.cfm page):

```
<cfinput name="duration" message="Duration must be a number and cannot be
  blank."
  validate="integer" required="Yes" size="3" maxlength="3">
```

Code	Explanation
`<cfinput name="duration" message="Duration must be a number and cannot be blank." validate="integer" required="Yes" size="3" maxlength="3">`	Use the `cfinput` tag to create the `duration` input entry field within a `cfform` tag. The `validate` attribute defines the field as an `integer`. The `required` attribute indicates that the field must have an entry. If the data is not entered or data entered is not an integer, the `message` attribute specifies that the message, "Duration must be...." appears.

Modifying the Trip Edit page to use ColdFusion form tags

In this exercise, you use the ColdFusion form tags to move the validation of many business rules from the server to the client. To do this, you change the HTML form tags in the tripedit.cfm page to ColdFusion form tags that validate these fields on the client side. Next, you remove the unneeded server-side single-field validation code from the tripeditaction.cfm page. Finally, you test the form to ensure that the client-side validation is working correctly.

To use the ColdFusion form tags on the Trip Edit page:

1. Open the tripedit.cfm file in the my_app directory in your editor.
2. Locate and change the `<form>` and `</form>` tags to `<cfform>` and `</cfform>` tags, respectively.
3. Change the `<input>` tags to `<cfinput>` tags, `<select>` tags to `<cfselect>` tags, and `<textarea>` tags to `<cftextarea>` tags.

   Note: The input type for the submit button must remain a standard input rather than `cfinput`.

4. For each ColdFusion form tag (`cfinput` and `cfselect`), assign the following appropriate values:

Attribute value	Description
required	Use this attribute for fields that must be filled out or selected.
validate	Use this attribute for fields that require a specific data type for validation. Values include: integer, date, time, telephone, and zip code.
message	Use this attribute for fields that require an error message to appear if validation fails. The message reflects the text that describes the business rule.

The following table contains the revised code blocks:

Rule	Description	Validation code

1	All trips must be named.	`<cfinput name= "tripName"`
		`maxlength = "50" size = "50"`
		`required = "Yes"`
		`message = "Trip name must not be blank">`

2	All trips must be accompanied by a full description.	`<cftextarea name="tripDescription"`
		`required="Yes"`
		`message="Trip description must not be blank.">`
		`</cftextarea>`

3	Each trip must be categorized by event type. Only valid event types (1-surfing, 2-mountain climbing, and so on) are permissible.	`<cfselect size="1" name="eventType"`
		`required="Yes"`
		`message="Type of event must be selected.">`
		`<option value="1" selected>Surfing</option>`
		`<option value="2">Mountain Climbing</option>`
		`<option value="3">Mountain Biking</option>`
		`</cfselect>`

4	Trip locations are required.	`<cfinput size="50" name="tripLocation"`
		`required="Yes"`
		`message="Trip location must not be blank.">`

Rule	Description	Validation code

5	The maximum number of people permitted on the trip must be specified.	`<cfinput name="numberPeople"`
		`size="6"`
		`required="Yes"`
		`validate="integer"`
		`message="The number of people field must be a number and cannot be blank.">`

6	The trip departure and return dates must be specified for each trip. All trip dates must be valid future dates. Departure date must precede return date.	`<cfinput name="departureDate"`
		`size="10"`
		`required="Yes"`
		`validate="date"`
		`message="Departure date must be a valid date.">`
		
		`<cfinput name="returnDate"`
		`size="10"`
		`required="Yes"`
		`validate="date"`
		`message="Return date must be a valid date.">`

7	The trip's price and base cost are required. Both values are positive numeric values. The trip price must have at least a 20% markup over base cost.	`<cfinput name="price"`
		`size="10"`
		`required="Yes"`
		`validate="integer"`
		`message="Price is required and must be numeric.">`
		
		`<cfinput name="baseCost"`
		`size="10"`
		`required="Yes"`
		`validate="integer"`
		`message="Base cost is required and must be numeric.">`

8	Any trip priced over $750 requires a deposit.	`<cfinput name="depositRequired"`
		`type="checkbox"`
		`value="Yes" >`

9	A trip leader must be identified.	`<cfinput name="tripLeader"`
		`maxlength="50" size="50"`
		`required="Yes"`
		`message="A trip leader must be specified.">`

10	A photo must accompany all new trips.	`<cfinput name="photo"`
		`maxlength="50" size="50"`
		`required="Yes"`
		`message="Valid photo file name must be specified.">`

Tip: For additional help, review the completed code in the ttripedit_lesson7_ex5.cfm file within the solutions directory. For more details about using ColdFusion form tags and their attributes, see *ColdFusion MX Developer's Guide*.

Note: The client-side validation code that you added to the tripedit.cfm page does not validate the cross-field rules.

5. Open the tripeditaction.cfm file in the my_app directory and delete the code for the following single-field validation rules:
   - Trip name is required.
   - Trip description is required.
   - Trip leader is required.
   - Photo filename is required.
   - Number of people is required and must be numeric.
   - Trip location is required.
   - Base cost is required and must be numeric.
   - Price is required and must be numeric.
6. Save the file.

   The modified tripeditaction.cfm page appears as follows:

```
<cfset isOk = "Yes">
<cfif not isdefined("Form.depositRequired")>
   <cfset form.depositRequired = "No">
</cfif>
<cfif Form.price GT 750 AND Form.depositRequired EQ "No">
  <cfset IsOk = "No">
  <cfoutput>Deposit is required for trips priced over $750.</cfoutput>
</cfif>
<cfif Form.basecost * 1.2 GT #Form.price#>
  <cfset IsOk = "No">
  <cfoutput>Price must be marked up at least 20% above cost.</cfoutput>
</cfif>
<cfif form.departureDate GT form.returnDate>
  <cfset isOk = "No">
  <cfoutput>Return date cannot precede departure date. Please re-enter.</
  cfoutput>
</cfif><html>
<head>
<title>Trip Maintenance Confirmation</title>
</head>

<body>

<cfif isOk EQ "Yes">
  <h1>Trip Added</h1>
  <cfoutput>You have added #Form.TripName# to the trips database.
  </cfoutput>
</cfif>
</body>
</html>
```

To test the modified code:

1. View the tripedit.cfm page in a browser.
2. Test the client- and server-side field validations by filling out the fields on the form and clicking Save.

   Testing recommendations:

   - Omit required fields such as trip name or location.
   - Make the departure date an invalid date, such as **12/32/2002**.
   - Enter a nonnumeric value in Number of People, such as **one**.

Exercise 6: Dynamically populating the list of event types

Currently, the event types in the tripedit.cfm file are hard-coded. In the preceding exercises, each `option` tag in the event type contained a `value` attribute and option text, such as `Surfing`. These values come from the eventtypes table; they do not have to be hard-coded. The event type in the Trips table is an identifier that is used as a foreign key to the eventtypes table, which holds the actual event names. The eventtypes table column `eventTypeID` is used for the `value` attribute and the `eventType` for the literal value that appears in the select box. To retrieve the data from this table, you must include the following `cfquery`:

```
<cfquery name="GetEvents" datasource="CompassTravel">
  SELECT eventType, eventTypeID
  FROM eventtypes
</cfquery>
```

To exploit the query in the HTML `option` tags, you can replace the HTML `select` tag with a `cfselect` tag.

The `cfselect` tag is an improved version of the HTML `select` tag. Like other ColdFusion form tags, the `cfselect` tag provides the `required` and `message` attributes that validate the data entered. Using the `cfselect` tag and the preceding `cfquery`, you can implement the `eventType` field data entry as follows.

To display a list of event types from the eventtypes table and add validation:

1. View the tripedit.cfm page in a browser. Select the event types drop-down list. Notice that only three event types appear in the list.
2. Open the tripedit.cfm file in the my_app directory.
3. Add the following code before the `<html>` tag:

   ```
   <cfquery name="GetEvents" datasource="CompassTravel">
     SELECT eventType, eventTypeID
     FROM eventtypes
   </cfquery>
   ```

   Note: In previous exercises, you learned to write structured reusable code by placing queries in ColdFusion components. Although this practice is advisable when creating ColdFusion applications, you put the query in the tripedit.cfm file for simplicity.

4. Replace the following eventtypes code lines:

```
<cfselect size="1" name="eventType" required="Yes"
  message="Type of event must be selected.">
  <option value="1" selected>Surfing</option>
  <option value="2">Mountain Climbing</option>
  <option value="3">Mountain Biking</option>
</cfselect>
```

with these lines:

```
<cfselect size="1" name="eventType" required="Yes"
  message="Type of event must be selected.">
  <cfoutput query="GetEvents">
    <option value="#GetEvents.eventTypeID#">
      #GetEvents.eventType#
    </option>
  </cfoutput>
</cfselect>
```

5. Save the file.

To test the modified code:

1. View the tripedit.cfm page in a browser.
2. Select the event types drop-down list. Notice that all seven event types appear in the list.

Exercise 7: Validating the existence of the trip photo file

At this point, you have a more efficient application. The client is handling much of the validation of the Compass Travel new trip business rules. Except for the trip photo file, the server receives only valid data.

The trip photo file business rule does not fit nicely into this design, however. This business rule has two parts:

- A photo filename must accompany all new trips.
- The photo image file must reside in the images directory of the Compass Travel website.

You used the `required` attribute for the photo `cfinput` tag to ensure that a filename is entered. Now you must ensure that the file exists in the right directory so that the application can display it to the user.

Because browser clients are prohibited from doing standard file input and output (I/O) on the web server, the Trip Maintenance application uses server-side validation to ensure the existence of the photo file. You add the business rule for the photo file to the tripeditaction.cfm page.

To verify that a file exists, ColdFusion provides a `FileExists` function. This function has the following syntax:

```
FileExists(absolute_path)
```

This function returns Yes if the file specified in the argument does exist; otherwise, it returns No.

Note: The trip photo images are stored in the following path relative to your web root directory: \cfdocs\getting_ started\photos. Therefore, if your web root is C:\inetpub\wwwroot, the photos are stored in the C:\inetpub\wwwroot\cfdocs\getting_ started\photos directory.

For more information about the `FileExists` function, see *CFML Reference*.

To verify that the photo filename exists:

1. Open the tripeditaction.cfm file in the my_app directory.
2. Add logic to check that the user entered a valid photo filename by copying the following code immediately following the first `<cfset isOk = "Yes">` statement:

```
<!-- Check to see if photo file exists. --->
<cfset PhotoLocation =
"C:\Inetpub\wwwroot\CFDOCS\getting_started\Photos\">
<cfset PhotoLocation = PhotoLocation  & FORM.photo>
<cfif not FileExists(PhotoLocation)>
   <cfset isOk = "No">
   <cfoutput>Trip Photo does not exist</cfoutput>
</cfif>
```

3. Verify that the code you just copied is pointing to the correct PhotoLocation path. The path is specified in the `<cfset PhotoLocation = "C:...">` tag.

   For example, depending on your web server configuration, the PhotoLocation path might be:

   - For MS Windows systems:

```
<cfset PhotoLocation =
"C:\CFusionMX7\wwwroot\cfdocs\getting_started\photos\">
```

     or

```
<cfset PhotoLocation =
"C:\Inetpub\wwwroot\cfdocs\getting_started\photos\">
```

   - For Linux or Solaris systems:

```
<cfset PhotoLocation =
"/opt/cfusionmx7/wwwroot/cfdocs/getting_started/photos/">
```

     or

```
<cfset PhotoLocation =
"/<webserverdocroot>/cfdocs/getting_started/photos/">
```

4. Save the file.

To test the modified code:

1. View the tripedit.cfm page in your browser.
2. Perform tests by doing the following:
   a. In the Trip Edit page, enter valid information in all the required fields except the Photo Filename field.
   b. In the Photo Filename field, enter **nowhere.jpg,** and click Save.

      The following error message appears: Trip photo does not exist.
   c. To avoid the error, replace the invalid photo filename in the Trip Edit page with **somewhere.jpg** and click Save.

      The following message appears: Trip added.

Reviewing the code

The following table describes the code used to verify whether the photo file exists:

Code	Explanation
`<cfset PhotoLocation = "C:\CFusionMX7\wwwroot\cfdocs\ getting_started\photos\">`	The `cfset` tag sets the value of the PhotoLocation path to the appropriate directory. ***Note:*** The code snippet shows one value for Windows systems.
`<cfset PhotoLocation = PhotoLocation & FORM.photo>`	The ColdFusion `&` operator in the `cfset` tag combines the directory name with the name of the photo file entered in the form.
`<cfif not FileExists(PhotoLocation)> <cfset isOk = "No"> <cfoutput> Trip Photo does not exist. </cfoutput> </cfif>`	`FileExists` checks to see if the file indicated by the `fileLocation` variable exists at the specified disk location. If it doesn't, an error message appears.

Summary

As described in this lesson, ColdFusion offers a number of alternatives to validating data. If you are familiar with standard page validation code, ColdFusion supports these development approaches. However, ColdFusion form tags simplify data validation. In this lesson, you also dynamically populated a list.

In the next lesson

Now that you are sure that your application can save valid data, in the next lesson, you will write code to add trips to the database. Additionally, you will add logic to update existing trip data in the Trips table.

CHAPTER 11
Lesson 8: Implementing Browsing and Searching

In this lesson, you will further enhance the Compass Travel Macromedia ColdFusion application by providing code to implement the navigation and searching functions.

This lesson contains the following exercises:

Exercise 1: Enabling users to browse trip details 123

Exercise 2: Determining actions based on which button a user clicks 127

Exercise 3: Enabling searching from the Trip Detail page 127

ColdFusion tags and functions introduced in this lesson

The following table identifies the ColdFusion tag and structure that you use in this lesson to enhance the Trip Maintenance application:

Element	Type	Description
`cflocation`	Tag	Tag opens a ColdFusion page or HTML file.
`URL`	Structure	Structure to hold the variables within a URL.

Exercise 1: Enabling users to browse trip details

In "Lesson 6: Creating a Main Application Page" on page 97, you added buttons to the Trip Detail page to browse records in the database. In this exercise, you build the action pages that implement the actions for these buttons.

To enable browsing, you build the Navigation Action page. This page determines which trip record appears on the Trip Detail page after the user clicks one of the navigation buttons. There is no HTML output displayed from this action page. Instead, this page uses dynamic SQL to identify the `tripID` that must appear on the Trip Detail page. In this dynamic SQL statement, the proper `tripID` is passed as a parameter to the URL, and then redirects the `tripID` to the Trip Detail page.

Using dynamic SQL to browse the Trips table

The `tripID` uniquely identifies a trip in the Trips table. In "Lesson 6: Creating a Main Application Page" on page 97, you displayed the Trip Detail page for a trip by passing the ID as a parameter of the URL. For example, you would use the following URL to display the detail information for a trip with the ID of 20:

```
http://localhost/cfdocs/getting_started/my_app/tripdetail.cfm?ID=20
```

The main objective of the Navigation Action page (navigationaction.cfm) is to navigate to the Trip Detail page using a URL that includes the correct `tripID` based on the navigation button clicked. Because trips are added and later deleted, trips might not be ordered sequentially by ID. There can be missing IDs where trips were deleted. For example, if the current `tripID` is 10 and the user clicks the Next navigation button, the `tripID` of the next trip might not be 11; it could be 14.

To retrieve the proper `tripID`, you must query the database to find out what the next (or previous, first, or last) ID is, based on the current `tripID`. The navigation action page uses dynamic SQL to build a query to find the appropriate ID to use.

In "Lesson 4: Building Dynamic Queries" on page 71, you used ColdFusion string manipulation to construct the proper SQL SELECT WHERE clause. In this lesson, you use a similar approach to build the WHERE clause for navigation. Additionally, you use the proper ORDER BY clause to select the correct trip row from the trips table.

For example, if the current `tripID` equals 6, the following table identifies the proper SQL statement based on the navigation button that the user clicked:

Navigation button	SQL statement to navigate to correct trip ID	SQL statement description
First Row	SELECT tripID FROM trips ORDER BY tripID	Returns the list of all `tripIDs` in ascending (1,2,3...) order.
Previous Row	SELECT tripID FROM trips WHERE tripID < 6 ORDER BY tripID DESC	Returns the list of all `tripIDs` less than 6 in descending (5,4,3...) order.
Next Row	SELECT tripID FROM trips WHERE tripID > 6 ORDER BY tripID	Returns the list of all `tripIDs` greater than 6 in ascending (7,8,9...) order.
Last Row	SELECT tripID FROM trips ORDER BY tripID DESC	Returns the list of all `tripIDs` in descending (99,98,97...) order.

To properly build the SQL SELECT statement for previous and next row navigation, you must know the current `tripID`. This is the reason for using the hidden input tag `RecordID` on the Trip Detail page. You can use the form variable `#Form.RecordID#` in the Navigation Action page for building the proper test in the WHERE clause of the SQL SELECT statement.

Limiting the number of result rows

Each of the SQL statements in the preceding table returns a result set of trips rows. The result set can range from zero to any number of rows. The Navigation Action page must limit the result set count to 1, because only the initial row in the result set appears on the Trip Detail page.

ColdFusion provides the `maxRows` attribute for the `cfquery` tag for this purpose. This attribute limits the number of result rows returned from the database. To show only a single row at a time in the Trip Detail page, set `maxRows` to 1.

To build the Navigation Action page:

1. Create a blank file.
2. Enter the following code in the blank file:

```
<!---  NAVIGATION BUTTONS --->
  <cfquery name="TripQuery" datasource="CompassTravel" maxrows="1">
    SELECT tripID FROM trips
    <cfif IsDefined("Form.btnPrev.X")>
      WHERE tripID < #Form.RecordID#
      ORDER BY tripID DESC
    <cfelseif IsDefined("Form.btnNext.X")>
      WHERE tripID > #Form.RecordID#
      ORDER BY tripID
    <cfelseif IsDefined("Form.btnFirst.X")>
      ORDER BY tripID
    <cfelseif IsDefined("Form.btnLast.X")>
      WHERE tripID > #Form.RecordID#
      ORDER BY tripID DESC
    </cfif>
  </cfquery>
  <cfif TripQuery.RecordCount is 1>
    <cflocation url="tripdetail.cfm?ID=#TripQuery.tripID#">
  <cfelse>
    <cflocation url="tripdetail.cfm?ID=#Form.RecordID#">
  </cfif>
```

3. Save the file as navigationaction.cfm in the my_app directory.

Note: In previous lessons, you adhered to good coding practices by putting queries in ColdFusion components. To optimize performance, and because the Navigation Action page contains only a query, the page is a ColdFusion page rather than a CFC. For more information about code reuse, see Chapter 8, "Creating ColdFusion Elements" in *ColdFusion MX Developer's Guide*.

Reviewing the code

The following table describes the code used to process the navigation button requests:

Code	Explanation

`<cfquery`
`  name="TripQuery"`
`  dataSource="CompassTravel"`
`  maxRows=1>`
	The `cfquery` tag identifies that a query named TripQuery is executed against the CompassTravel data source. The number of rows returned cannot exceed 1 (`maxRows`=1).

`SELECT tripID FROM trips`	The SQL SELECT statement will always start with "SELECT tripID FROM trips."

`<cfif IsDefined("Form.btnPrev.X")>`
`  WHERE tripID < #Form.RecordID#`
`  ORDER BY tripID DESC`
 	
`<cfelseif IsDefined("Form.btnNext.X")>`
`  WHERE tripID > #Form.RecordID#`
`  ORDER BY tripID`
 	
`<cfelseif IsDefined("Form.btnFirst.X")>`
`  ORDER BY tripID`
 	
`<cfelseif IsDefined("Form.btnLast.X")>`
`  WHERE tripID > #Form.RecordID#`
`  ORDER BY tripID DESC`
 	
`</cfif>`
	The `cfif` tag checks whether the user pressed a navigation button on the browse page. The `X` property is checked because the buttons on the detail page use image type HTML input tags. The `X` property is a mouse offset that gets sent when the user clicks a graphic button.
	
	The WHERE and ORDER BY clauses vary depending on the navigation button that the user clicks.

`</cfquery>`	

`<cfif TripQuery.RecordCount is 1>`
`<cflocation`
`  url="tripdetail.cfm?RecordID=#TripQuery.tripID#">`
`<cfelse>`
`<cflocation`
`  url="tripdetail.cfm?RecordID=#Form.RecordID#">`
`</cfif>`
	The `cfif` tag checks to see whether the query returned a row to display. If it did, that `tripID` is used to form a URL to navigate to using the `cflocation` tag. If the query returned no rows, it navigates back to the detail page with the current record ID, which passed in the hidden form variable `RecordID`.

To test the navigation:

1. View the tripdetail.cfm page from the my_app directory in a browser.
2. Click the Next Row button.

   The Trip Detail page shows information about the second trip.
3. Click the Previous Row button.

   The Trip Detail page shows information about the first trip.
4. Click the Last Row button.

   The Trip Detail page shows information about the last trip.
5. Click the First Row button.

   The Trip Detail page shows information about the first trip.

Exercise 2: Determining actions based on which button a user clicks

In "Lesson 6: Creating a Main Application Page" on page 97, you added buttons to the Trip Detail page to let users search and modify the trips database; however, because you had not yet written the code to implement these capabilities, ColdFusion displayed an error when you clicked the buttons. Clicking these button sends the user to the maintenanceaction.cfm page. In this exercise, you will create the Maintenance Action page (maintenanceaction.cfm).

ColdFusion creates a variable only for the button that the user clicked. You use the `IsDefined` function to test for the existence of the variable, which determines which action the application takes.

To create the Maintenance Action page:

1. Create a blank file.
2. Enter the following code in the blank file:

```
<cfif IsDefined("Form.btnSearch")>
  <!--- Code to execute if the user clicked Search. --->
<cfelseif IsDefined("Form.btnDelete")>
  <!--- Code to execute if the user clicked Delete. --->
<cfelseif IsDefined("Form.btnEdit")>
  <!--- Code to execute if the user clicked Edit. --->
<cfelseif IsDefined("Form.btnAdd")>
  <!--- Code to execute if the user clicked Add. --->
</cfif>
```

3. Save the file as maintenanceaction.cfm in the my_app directory.

Exercise 3: Enabling searching from the Trip Detail page

You already created a search capability when you created the tripsearchform.cfm page in "Lesson 4: Building Dynamic Queries" on page 71. When the user clicks the Search button, you want to navigate to the tripsearchform.cfm page. You use the `cflocation` tag to do so.

To enable searching from the Trip Detail page:

1. Open the maintenanceaction.cfm file in the my_app directory in your editor.
2. Add the highlighted code in the file.

```
<cfif IsDefined("Form.btnSearch")>
  <!--- Code to execute if the user clicked Search. --->
  <cflocation url="tripsearchform.cfm">
<cfelseif IsDefined("Form.btnDelete")>
  <!--- Code to execute if the user clicked Delete. --->
<cfelseif IsDefined("Form.btnEdit")>
  <!--- Code to execute if the user clicked Edit. --->
<cfelseif IsDefined("Form.btnAdd")>
  <!--- Code to execute if the user clicked Add. --->
</cfif>
```

3. Save the file.

Reviewing the code

The following table describes the code that executes when the user clicks the Search button:

Code	Explanation
`<cfif IsDefined("Form.btnSearch")>` `  <cflocation url="tripsearchform.cfm">`	If the Form.btnSearch variable exists because the user clicked the Search button, go to the page tripsearchform.cfm.

Summary

In this lesson, you added the capability of browsing trips sequentially to the Trip Detail page. You also limited the result set of the dynamic query using the `MaxRows` attribute of the `cfquery` tag.

By implementing the Maintenance Action page, you enabled users to take action on the current trip. You used the ColdFusion `cflocation` tag to link the search page to the Trip Detail page.

In the next lesson

You have implemented the navigation buttons and the Search button on the Trip Detail page. In "Lesson 9: Enabling Database Maintenance" on page 129, you will implement the Delete, Add and Edit buttons.

CHAPTER 12
Lesson 9: Enabling Database Maintenance

In this lesson, you will enable maintenance of the trips database. The exercises will guide you through the steps of adding the database update logic to delete trips, add new trip offerings, and update existing trips in the Compass Travel database.

This lesson contains the following exercises:

Exercise 1: Deleting the current trip record shown on the Trip Detail page 129
Exercise 2: Adding trips with SQL INSERT statements 131
Exercise 3: Adding data using the cfinsert tag 135
Exercise 4: Updating a SQL row using the cfupdate tag 137
Exercise 5: Linking the Trip Edit page to the main page 138
Exercise 6: Inserting values in the edit form 139

ColdFusion tags introduced in this lesson

The following table identifies the ColdFusion tags that you use in this lesson to enhance the ColdFusion application:

Element	Type	Description
`cfinsert`	Tag	Inserts records in a JDBC data source.
`cfupdate`	Tag	Updates records in a JDBC data source.

Exercise 1: Deleting the current trip record shown on the Trip Detail page

In "Lesson 8: Implementing Browsing and Searching" on page 123, you created the tripeditaction.cfm page to contain server side edits for the trip edit data entry form. In this lesson, you complete the tripeditaction.cfm page.

Before you can write the code to delete a trip, you must understand the underlying SQL statement to delete rows from the trips table.

The SQL DELETE statement removes existing rows in a relational table. The DELETE statement has the following format:

```
DELETE FROM table_name WHERE column_name = some_value
```

For example, the database table named Clients contains holds information about people, in the following rows:

LastName	FirstName	Address	City
Jones	Tom	50 Main St	New York
Adamson	Anita	521 Beacon St	Boston
Green	Peter	1 Broadway	New York

To delete everyone from New York from the table, use the following statement:

```
DELETE FROM Clients WHERE City = 'New York'
```

After the database management system processes the preceding statement, the table contains the following row only:

LastName	FirstName	Address	City
Adamson	Anita	521 Beacon St	Boston

To ensure that the Trip Maintenance application deletes only the proper trip, you must use the unique `tripID` key when issuing the SQL DELETE statement. The `RecordID` field holds the `tripID`. Using the hidden `RecordID input` tag from the Trip Detail page, the following SQL statement deletes a row from the Trips table:

```
DELETE FROM trips WHERE tripID = #Form.RecordID#
```

To enable users to delete trips:

1. Open the maintenanceaction.cfm file in the my_app directory in your editor.
2. Add the highlighted code in the file.

```
<cfif IsDefined("Form.btnSearch")>
  <!--- Code to execute if the user clicked Search. --->
  <cflocation url=”tripsearchform.cfm”>
<cfelseif IsDefined("Form.btnDelete")>
  <!--- Code to execute if the user clicked Delete. --->
  <cfquery name="DeleteRecord" datasource="CompassTravel">
    DELETE FROM trips WHERE tripID = #Form.RecordID#
  </cfquery>
<cflocation url="tripdetail.cfm">
<cfelseif IsDefined("Form.btnEdit")>
  <!--- Code to execute if the user clicked Edit. --->
<cfelseif IsDefined("Form.btnAdd")>
  <!--- Code to execute if the user clicked Add. --->
</cfif>
```

3. Save the page.

To test the delete capability:

1. View the tripdetail.cfm page in a browser.

   The current trip is Nepal. Notice that the destination for the Nepal Summit Challenge trip is Imji Himal, Nepal.

2. Click Search.

   The Trip Search page appears.

3. In the Trip Search page, select Begins With in the selection box for Trip Location.
4. Enter **Imji** in the Trip Location field.
5. Click Search.
6. Verify that only one trip is found whose location begins with Imji.
7. To return to the Trip Detail page for this trip, click Nepal Summit Challenge
8. In the Trip Detail page, click Delete.
9. Click Search.
10. Select Beings With in the selection box for Trip Location.
11. Enter **Imji** in the Trip Location field.
12. Click Search.

    There should be no records, because you deleted the trip.

Exercise 2: Adding trips with SQL INSERT statements

In "Lesson 8: Implementing Browsing and Searching" on page 123, you built a Trip Edit page to collect the data. Now you can modify the Trip Edit action page to insert the data into the database. There are two approaches to inserting data into a SQL database:

- Build a SQL INSERT statement and execute it using the `cfquery` tag.
- Use the ColdFusion `cfinsert` tag. This approach eliminates the need for you to learn SQL syntax.

In this exercise, you use a SQL INSERT statement.

In previous lessons, you used the SQL SELECT statement to retrieve data and the SQL DELETE statement to delete data from the Trips table in the Compass Travel database. To add new trips to the database using SQL, you must understand the syntax of the SQL INSERT statement.

The SQL INSERT statement inserts new rows into a relational table. The SQL INSERT statement has the following format:

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

For example, the database table named Clients contains information about people in the following rows:

LastName	FirstName	Address	City
Tom	Jones	12 State St	Boston
Peter	Green	1 Broadway	New York

To add a record to the table, use the following statement:

```
INSERT INTO Clients
VALUES ('Smith', 'Kaleigh', '14 Greenway', 'Windham')
```

After the database management system processes the preceding statement, the table contains the following rows:

LastName	FirstName	Address	City
Tom	Jones	12 State St	Boston
Peter	Green	1 Broadway	New York
Smith	Kaleigh	14 Greenway	Windham

Notice that the values inserted in the table were surrounded by single-quotation marks in the statement. In SQL, you must surround any text or date values with single-quotation marks; however, you don't use single-quotation marks with numeric values.

Alternatively, you can specify the columns for which you want to insert data. This approach lets you insert data into some columns while omitting others. For this approach, you use the following syntax:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

For example, to add Kaleigh Smith of Windham, with an unknown address, you use the named column approach:

```
INSERT INTO Clients (LastName, FirstName, City)
VALUES ('Smith', 'Kaleigh', 'Windham')
```

You use the `cfquery` tag to execute SQL from ColdFusion. The `cfquery` tag passes SQL statements to your data source. As described in "Lesson 2: Configuring Your Development Environment" on page 49, a data source stores information about how to connect to an indicated data provider, such as a relational database management system. The data source that you established in that chapter stored information on how to access the Compass Travel database. The data source name was "CompassTravel."

To add data using a SQL INSERT statement and a cfquery tag:

1. Open the tripeditaction.cfm file in the my_app directory.
2. Locate the `<cfif isOk EQ "Yes">` tag near the end of the file. After the `<H1>Trip Added </H1>` line, add the following code in the following table to insert the data from the Form variables into the trips table:

For	Code

Windows users, using MS Access	`<!--- Insert the new trip record into the Compass Travel database. --->`
	`<cfquery name="AddTrip" datasource="compasstravel">`
	`INSERT INTO Trips (tripName, eventType, tripDescription, tripLocation,departureDate, returnDate, price, tripLeader, photo, baseCost, numberPeople, depositRequired)`
	`VALUES ('#Form.tripName#', #Form.eventType#, '#Form.tripDescription#', '#Form.tripLocation#','#Form.departureDate#', '#Form.returnDate#', #Form.price#, '#Form.tripLeader#', '#Form.photo#', #Form.baseCost#, #Form.numberPeople#, '#Form.depositRequired#')`
	`</cfquery>`

UNIX users, using PointBase	`<!--- Insert the new trip record into the Compass Travel database. --->`
	`<!--- Use local variables to convert dates to JDBC format (yyyy-mm-dd) from input format (mm/dd/yyyy). --->`
	`<cfset JDBCdepartureDate = #Right(Form.departureDate,4)# & "-" & #Left(Form.departureDate,2)# & "-" & #Mid(Form.departureDate,4,2)#>`
	`<cfset JDBCreturnDate = #Right(Form.returnDate,4)# & "-" & #Left(Form.returnDate,2)# & "-" & #Mid(Form.returnDate,4,2)#>`
	`<cfquery name="AddTrip" datasource="CompassTravel">`
	`INSERT INTO Trips (tripName, eventType, tripDescription, tripLocation, departureDate, returnDate, price, tripLeader, photo, baseCost, numberPeople,depositRequired)`
	`VALUES ('#Form.tripName#', #Form.eventType#, '#Form.tripDescription#', '#Form.tripLocation#', Date'#JDBCdepartureDate#', Date'#JDBCreturnDate#', #Form.price#,'#Form.tripLeader#', '#Form.photo#', #Form.baseCost#, #Form.numberPeople#, '#Form.depositRequired#')`
	`</cfquery>`

3. Save the file.

To test the modified code:

1. Open the tripedit.cfm page in your browser.
2. In the tripedit.cfm page, enter in the fields the values in the following table, and then click Save.

Field	Value
Trip Name	NH White Mountains
Event Type	Mountain Climbing
Trip Description	Climb the 5 highest peaks in the New Hampshire White Mountains.
Trip Location	Northeastern New Hampshire
Departs	05/01/2005
Returns	05/10/2005
Number of People	15
Price	1200
Base Cost	600
Deposit Required	Yes
Trip Leader	Tom Finn
Photo File Name	whitemountains.jpg

   After the new trip is written to the database, the following message appears: You have added NH White Mountains to the trips database.
3. To verify that the trip was saved, open the tripsearchform.cfm page in the my_app directory in your browser.
4. In the Trip Search page, in the Trip Location drop-down list, select the Begins With option, and enter the value **Nor** in the text box.
5. Click Search.

   The TripResults page appears.
6. Click NH White Mountains to display the details of the trip you just added. Verify that all the fields were saved correctly.
7. Click the Delete button to delete this record so that you can reuse some of the steps of this exercise in Exercise 3: Adding data using the cfinsert tag.

Reviewing the code

The following table describes the SQL INSERT and `cfquery` code that ColdFusion uses to add data:

Code	Explanation
`<cfquery name="AddTrip" datasource="CompassTravel">`	Using the `datasource` attribute, the `cfquery` tag connects to the CompassTravel data source and returns a result set identified by the `name` attribute.
`INSERT INTO Trips (TripName, EventType, tripDescription, tripLocation, departureDate, returnDate, price, tripLeader,photo, baseCost, numberPeople, depositRequired) VALUES ('#Form.TripName#', #Form.EventType#, '#Form.tripDescription#', '#Form.tripLocation#', '#Form.departureDate#', '#Form.returnDate#', #Form.price#, '#Form.tripLeader#', '#Form.photo#', #Form.baseCost#, Form.numberPeople#, '#Form.depositRequired#)`	The SQL INSERT statement identifies the data to insert into the Trips table. The table column names are cited in a comma-separated list surrounded by parentheses `(TripName, EventType....)` after the table name Trips. The `VALUES` keyword indicates the list of values that are inserted into the columns in the same order as the columns are specified earlier in the statement. The values refer to form variables that are passed from the data entry form to the action page. The variables are surrounded by number signs; for example, `#Form.baseCost#`. Also, if the column data type is a String data type, the values are surrounded by single-quotation marks; for example: `'#Form.TripName#'`.

For more information about adding data to a database using SQL and the `cfquery` tag, see *ColdFusion MX Developer's Guide*. For more information about SQL, consult any SQL primer.

Exercise 3: Adding data using the cfinsert tag

For developers who prefer not to have to remember SQL syntax to add information to SQL databases, ColdFusion simplifies the coding for inserting SQL rows through the use of the `cfinsert` tag. As you might expect, the `cfinsert` tag has `datasource` and `tablename` attributes to specify where the data is inserted. The tag also has a `formfields` attribute to identify which fields to insert. The `formfields` attribute specifies a comma-separated list of form fields to insert. If this attribute is not specified, all the fields in the form are included in the operation. The following example uses the `cfinsert` tag with these attributes:

```
<cfinsert datasource="CompassTravel" tablename="Trips"
  formfields="tripName, eventType, tripDescription, tripLocation,
  departureDate, returnDate, price, tripLeader, photo, baseCost,
  numberPeople, depositRequired">
```

The `cfinsert` tag used in the preceding code snippet uses the following attributes:

Attribute	Description
datasource	The data source name associated with the database where the data is inserted.
tablename	The name of the SQL table within the database where the data is inserted.
formfields	A comma-separated list of form fields to insert.

To add data using cfinsert:

1. Open the tripeditaction.cfm file from the my_app directory in your editor.
2. Remove the entire AddTrip `cfquery` code block that you added in Exercise 2: Adding trips with SQL INSERT statements:

```
<cfquery name="AddTrip" datasource="compasstravel">
  INSERT INTO Trips (tripName, eventType, tripDescription,
    tripLocation,departureDate, returnDate, price, tripLeader,
    photo, baseCost, numberPeople, depositRequired)
  VALUES ('#Form.tripName#', #Form.eventType#,
    '#Form.tripDescription#',
    '#Form.tripLocation#','#Form.departureDate#',
    '#Form.returnDate#',
    #Form.price#, '#Form.tripLeader#', '#Form.photo#',
    #Form.baseCost#, #Form.numberPeople#, '#Form.depositRequired#')
</cfquery>
```

3. Add the following `cfinsert` tag to insert data into the trips table in the same location as the code that you just deleted:

```
<cfinsert datasource="CompassTravel" tablename="trips">
```

4. Save the file.

To test the modified code:

1. Open the tripedit.cfm page in your browser.
2. In the tripedit.cfm page, enter in the fields the values in the following table, and then click Save:

Field	Value
Trip Name	NH White Mountains
Event Type	Mountain Climbing
Trip Description	Climb the 5 highest peaks in the New Hampshire White Mountains.
Trip Location	Northeastern New Hampshire
Departs	05/01/2005
Returns	05/10/2005
Number of People	15
Price	1200
Base Cost	600
Deposit Required	Yes
Trip Leader	Tom Finn
Photo File Name	whitemountains.jpg

After the new trip is written to the database, the following message appears: You have added NH White Mountains to the trips database.

3. To verify that the trip was saved, open the tripsearchform.cfm page in the my_app directory in your browser.
4. In the Trip Search page, in the Trip Location drop-down list, select the Begins With option, and enter the value **Nor** in the text box.
5. Click Search.

   The TripResults page appears.
6. Click NH White Mountains to display the details of the trip you just added. Verify that all the fields were saved correctly.

For more information about adding data to a database using the `cfinsert` tag, see *ColdFusion MX Developer's Guide*.

Exercise 4: Updating a SQL row using the cfupdate tag

To update an existing SQL row, ColdFusion offers a simple approach through the use of the `cfupdate` tag. Like the `cfinsert` tag, the `cfupdate` tag has `datasource` and `tablename` attributes to specify where the data is to be inserted. The tag also has a `formfields` attribute to identify which fields are to insert. The `formfields` attribute specifies a comma-separated list of form fields to insert. If this attribute is not specified, all the fields in the form are included in the operation.

All the fields of the tripedit.cfm page have corresponding columns in the Trips table, so you can omit the `FormFields` attribute for both the `cfinsert` and `cfupdate` tags. If the `tripID` form field is passed from the Trip Edit page, the `cfupdate` tag is used; otherwise the `cfinsert` tag is executed. The following example uses the `cfupdate` and `cfinsert` tags without the `formfields` attribute:

```
<cfif not isdefined("form.tripID")>
  <cfinsert datasource="CompassTravel" tablename="Trips">
    <cflocation url="tripdetail.cfm">
  <cfelse>
  <cfupdate datasource="CompassTravel" tablename="Trips">
    <cflocation url="tripdetail.cfm?ID=#Form.tripID#">
</cfif>
```

Reviewing the code

The following tables describes the `cfinsert` and `cfupdate` code:

Code	Explanation

`<cfif not isdefined("form.tripID")>`
`  <cfinsert datasource="CompassTravel"`
`  tablename="Trips">`
`  <cflocation url="tripdetail.cfm">`
`<cfelse>`
`<cfupdate datasource="CompassTravel"`
`  tablename="Trips">`
`<cflocation`
`  url="tripdetail.cfm?ID=#Form.tripID#">`
`</cfif>`
	The `IsDefined` function determines whether the hidden field `tripID` was passed to the action page from the tripedit.cfm page. If there is a current trip, the `IsDefined` function returns True. When there is no current trip, the `cfif` statement is True. When the `cfif` statement is True, the `cfinsert` tag executes and the main page appears with the updated trip. If the `cfif` statement evaluates to False, the `cfinsert` statement executes and the first trip appears in the main page.

To update the database using a cfupdate tag:

1. Open the tripeditaction.cfm file in the my_app directory in your editor.
2. Add the following code before the `</body>` tag at the end of the file:

```
<!--- Update the database --->
  <cfif isOk EQ "Yes">
     <cfif isdefined("form.tripID")>
       <cfupdate datasource="CompassTravel" tablename="trips">
         <cflocation url="tripdetail.cfm?ID=#Form.tripID#">
     <cfelse>
       <cfinsert datasource="CompassTravel" tablename="TRIPS">
          <cflocation url="tripdetail.cfm">
     </cfif>
  <cfoutput>You have added #Form.TripName# to the trips database.
  </cfoutput>
  </cfif>
```

3. Save the file.

For more information about adding data to a database using the `cfupdate` tag, see *ColdFusion MX Developer's Guide*.

Exercise 5: Linking the Trip Edit page to the main page

As discussed in "Lesson 7: Validating Data to Enforce Business Rules" on page 103, the action page for the maintenance buttons on the main page is maintenanceaction.cfm. You previously added code for the Search and Delete buttons. In this exercise, you add code for the Add and Edit buttons.

To link the Add and Edit buttons on the Trip Detail page:

1. Open the maintenanceaction.cfm file in the my_app directory in your editor.
2. Locate the `</cfif>` tag at the end of the file.

3. Insert the highlighted code:

```
<cfelseif IsDefined("Form.btnEdit")>
<!--- Code to execute if the user clicked Edit. --->
  <cflocation url="tripedit.cfm?ID=#Form.RecordID#">
<cfelseif IsDefined("Form.btnAdd")>
<!--- Code to execute if the user clicked Add. --->
  <cflocation url="tripedit.cfm">
```

4. Save the file.

Exercise 6: Inserting values in the edit form

Notice that when the user clicks the Add button, the maintenanceaction.cfm page navigates to the tripedit.cfm page without passing any arguments. Conversely, when the user clicks the Edit button, the Trip Edit page passes the current record ID. The Trip Edit page must handle both cases. When a `RecordID` is passed on the URL, the tripedit.cfm page must query the database and fill the form with the data for the corresponding trip.

To insert values in the edit form:

1. Open the tripedit.cfm file in your editor.
2. Enter or copy and paste the following code after the `cfquery` block that appears at the top of the file:

```
<cfif IsDefined("URL.ID")>
  <cfquery name="TripQuery" datasource="CompassTravel" maxrows="1">
    SELECT tripName, eventType, tripDescription, tripLocation,
    departureDate, returnDate, price, tripLeader, photo, baseCost,
    numberPeople, depositRequired, tripID
    FROM trips
  <cfif IsDefined("URL.ID")>
    WHERE tripID = #url.ID#
  </cfif>
  </cfquery>
<!--- Set the local variables --->
  <cfset tripName = '#TripQuery.tripName#'>
  <cfset eventType = '#TripQuery.eventType#'>
  <cfset tripDescription = '#TripQuery.tripDescription#'>
  <cfset tripLocation = '#TripQuery.tripLocation#'>
  <cfset departureDate = DateFormat(#TripQuery.departureDate#,
    "mm/dd/yyyy")>
  <cfset returnDate = DateFormat(#TripQuery.returnDate#,"mm/dd/yyyy")>
  <cfset price = #TripQuery.price#>
  <cfset tripLeader = '#TripQuery.tripLeader#'>
  <cfset photo = '#TripQuery.photo#'>
  <cfset baseCost = #TripQuery.baseCost#>
  <cfset numberPeople = #TripQuery.numberPeople#>
  <cfif isDefined("TripQuery.depositRequired")>
    <cfset depositRequired = 'Yes'>
    <cfelse>
    <cfset depositRequired = 'No'>
  </cfif>
<cfelse>
  <cfset tripName = ''>
```

```
    <cfset tripDescription = ''>
    <cfset eventTypeIdentifier = #GetEvents.eventType#>
    <cfset tripLocation = ''>
    <cfset departureDate = ''>
    <cfset returnDate = ''>
    <cfset price = ''>
    <cfset tripLeader = ''>
    <cfset photo = ''>
    <cfset baseCost = ''>
    <cfset numberPeople = ''>
    <cfset depositRequired = 'Yes'>
  </cfif>
```

3. To make the appropriate values appear in the form, insert the code that is highlighted in the following table:

   Note: If the user clicks the Add button, the values from the currently displayed trip detail appear in the form; otherwise, all values, except those in the event type field, are blank.

4. Save the file.

Code	Explanation

`<TD><cfinput name= "tripName"`
`  maxlength = "50"`
`  size = "50"`
`  required = "Yes"`
`  value="tripName"`
`  message = "Trip name must not be blank">`
`</TD>`
	Set the value of the Trip Name to be blank if the ID is undefined; otherwise, set the value equal to the Trip Name that appeared on the Trip Detail page.

`<TD><cftextarea name="tripDescription"`
`  required="Yes"`
`  message="Trip description must not be blank.">`
`  value="#tripDescription#"`
`  </cftextarea>`
`</TD>`
	Set the value of the Trip Description to be blank if the ID is undefined; otherwise, set the value equal to the Trip Description that appeared on the Trip Detail page.

`<td><cfselect size="1" name="eventType" required="Yes"`
`  message="Type of event must be selected.">`
`  <cfoutput query="GetEvents">`
`  <option value=`
`"#GetEvents.eventTypeID#">#GetEvents.eventType#`
`  </option>`
`  </cfoutput>`
`</cfselect>`
`</td>`
	Set the value of the Event Type to be the list of events.

`<TD><cfinput size="50" name="tripLocation"`
`  required="Yes"`
`  value="#tripLocation#"`
`  message="Trip location must not be blank.">`
`</TD>`
	Set the value of the Trip Location to be blank if the ID is undefined; otherwise, set the value equal to the Trip Location that appeared on the Trip Detail page.

Code	Explanation
<TD><cfinput name="departureDate" size="10" required="Yes" validate="date" **value="#departureDate#"** message="Departure date must be a valid date."> </TD>	Set the value of the Departure Date to be blank if the ID is undefined; otherwise, set the value equal to the Departure Date that appeared on the Trip Detail page.
<TD><cfinput name="returnDate" size="10" required="Yes" validate="date" **value="#returnDate#"** message="Return date must be a valid date."> </TD>	Set the value of the Return Date to be blank if the ID is undefined; otherwise, set the value equal to the Return Date that appeared on the Trip Detail page.
<TD><cfinput name="numberPeople" size="6" required="Yes" validate="integer" **value="#numberPeople#"** message="The number of people field must be a number and cannot be blank."> </TD>	Set the value of the Number of People to be blank if the ID is undefined; otherwise, set the value equal to the Number of People that appeared on the Trip Detail page.
<TD><cfinput name="price" size="10" required="Yes" validate="integer" **value="#price#"** message="Price is required and must be numeric."> </TD>	Set the value of the Price to be blank if the ID is undefined; otherwise, set the value equal to the Price that appeared on the Trip Detail page.
<TD><cfinput name="baseCost" size="10" required="Yes" validate="integer" **value="#baseCost#"** message="Base cost is required and must be numeric."> </TD>	Set the value of the Base Cost to be blank if the ID is undefined; otherwise, set the value equal to the Base Cost that appeared on the Trip Detail page.
<TD><cfinput name="depositRequired" type="checkbox" **value="#depositRequired#" >** </TD>	Set the value of the Deposit Required to be "Yes" if the ID is undefined; otherwise, set the value equal to the Deposit Required that appeared on the Trip Detail page.
<TD><cfinput name="tripLeader" maxlength="50" size="50" required="Yes" **value="#tripLeader#"** message="A trip leader must be specified."></TD>	Set the value of the Trip Leader to be blank if the ID is undefined; otherwise, set the value equal to the Trip Leader that appeared on the Trip Detail page.
<TD><cfinput name="photo" maxlength="50" size="50" required="Yes" **value="#photo#"** message="Valid photo file name must be specified."> </TD>	Set the value of the Photo to be blank if the ID is undefined; otherwise, set the value equal to the Photo that appeared on the Trip Detail page.

Reviewing the code

The following table describes the code that ColdFusion uses to properly initialize the Trip Edit form:

Code	Explanation
<cfif IsDefined("URL.ID")> <cfquery name="TripQuery" datasource="CompassTravel" maxrows="1"> SELECT tripName, eventType, tripDescription, tripLocation, departureDate, returnDate, price, tripLeader, photo, baseCost, numberPeople, depositRequired, tripID FROM trips <cfif IsDefined("URL.ID")> WHERE tripID = #ID# </cfif> </cfquery> <!--- Set the local variables ---> <cfset tripName = '#TripQuery.tripName#'> <cfset eventType = #TripQuery.eventType#> <cfset tripDescription = TripQuery.tripDescription#'> <cfset tripLocation = '#TripQuery.tripLocation#'> <cfset departureDate = DateFormat(#TripQuery.departureDate#,"mm/dd/yyyy")> <cfset returnDate = DateFormat(#TripQuery.returnDate#,"mm/dd/yyyy")> ... <cfelse> <cfset tripName = ''> <cfset eventType = ''> ... </cfif>	The ColdFusion function `IsDefined` determines whether an ID argument was passed as part of the invoking URL. When an ID is passed, it is used in the WHERE clause of the SQL SELECT statement to retrieve the information about the current trip. The application then instantiates local variables from the results of the SQL query. The ColdFusion `DateFormat` function formats the date fields. If the user clicks the Add button to add a new trip, there is no ID passed as a URL argument. In this case, the local variables are instantiated to blank.

To test the modified code:

1. Open the tripdetail.cfm page in your browser.
2. Do the following tasks:
   a. Click the Edit button.
   b. Double the price of the current trip.
   c. Click Save.

Summary

In this lesson you used the SQL DELETE statement, and the `cfinsert` and `cfupdate` tags to delete, add, and update data to a table.

CHAPTER 13
Lesson 10: Restricting Access to ColdFusion Applications

This chapter provides information on using the Login Wizard to restrict access to a new or existing Macromedia ColdFusion application. You will learn about the Login Wizard, and then add user authentication to the Compass Travel application.

This lesson contains the following exercises:

Exercise 1: About the Login Wizard . . . . . .143.

Exercise 2: Requiring users to log in to the Compass Travel application . . . . . .147.

Exercise 1: About the Login Wizard

ColdFusion MX 7 provides Macromedia Dreamweaver MX Extensions to assist you in creating a ColdFusion MX application that requires login, or user, authentication. The user authentication interface that you create using the Login Wizard can perform simple authentication, authentication against a Microsoft Windows NT domain, or authentication against a Lightweight Directory Access Protocol (LDAP) server.

In Dreamweaver MX, you open the Login Wizard, which guides you through the user authentication interface creation process. The wizard then creates or modifies the files necessary to add user authentication to your ColdFusion MX application.

Installing the Login Wizard

The Login Wizard is installed with ColdFusion MX 7 when you install the Dreamweaver Extensions. If you did not install the Dreamweaver Extensions when you installed ColdFusion MX 7, you can install them separately.

To install the Dreamweaver Extensions:

1. Ensure that you have installed Dreamweaver 7.0.1 or later.
2. Make *coldfusion*/CFIDE/installers the current directory.
3. Double-click the CFMX7DreamWeaverExtensions.mxp file.
4. Follow the prompts in the installer.

Preparing to run the Login Wizard

Before using the Login Wizard, you should determine the following information:

- Which of the authentication types you want to use to restrict access to your application. The types include the following:
  - LDAP, which grants access to users who have an entry in a known LDAP server.
  - NT, which grants access to users who belong to the specified Windows domain.
  - Simple authentication, which grants access to users who enter the user name and password that you specify in the Login Wizard.
- Whether you want the login page to appear as either of the following:
  - Basic authentication, which uses the browser login pop-up window to prompt the user for login credentials.
  - ColdFusion Login page, which uses a web page to prompt the user for login credentials.

If the application for which you want to restrict access is deployed on a remote server, you should synchronize the files that constitute the application with those that reside on your local server. You should then use the Login Wizard to modify the files on your local server, and perform testing before redeploying the application on the remote server. You can move the application files to the remote server either by using the Login Wizard, which uses Dreamweaver, or by using Dreamweaver directly.

Creating an application that includes user authentication

If you are creating an application, you can use the Login Wizard to create the files necessary for user authentication before creating any other application files. The following table lists the files that the Login Wizard creates for each type of authentication and type of login page:

File	Description
Application.cfc	This file is invoked every time any file in or under this directory is called. You can modify this file as appropriate for your application; however, you should not alter the following line of code: `<cfinclude template="mm_wizard_application_include.cfm">`
index.cfm	You can replace this file with your own index.cfm file.
mm_wizard_application_include.cfm	This file contains the logic to determine if the user is a valid user. If not, the application automatically redirects the user to a login form. This file is an included call from Application.cfc.
mm_wizard_authenticate.cfc	This file contains the method required to perform user authentication. It creates a <cfloginuser>, which is the ColdFusion MX 7 version of an authenticated user.
mm_wizard_login.cfm	The ColdFusion Login page.
readme.txt	Describes the files that the Login Wizard creates.

Adding user authentication to an existing application

You can use the Login Wizard to add user authentication to an existing application. The Login Wizard modifies the existing Application.cfc. If your application uses an Application.cfm file instead of an Application.cfc file, ColdFusion MX ignores the Application.cfm file when the Login Wizard creates the Application.cfc file in the same directory as the existing Application.cfm file.

If your application already contains a file named index.cfm, the Login Wizard creates the file mm_wizard_index.cfm. You can use your existing index.cfm file without modifying it.

Starting the Login Wizard

After you install the Login Wizard, you can open it from Dreamweaver.

To start the Login Wizard:

1. Start Dreamweaver.
2. Ensure that the active site is the one for which you want to restrict access.

   Note: The currently active site must be a local site. The Remote Info Access and the Test Server Access must be RDS

3. In Dreamweaver, select Commands > CF Login Wizard.
4. Click Begin the Wizard.
5. Do one of the following:
   - If the site for which you want to restrict access is the currently active site in Dreamweaver, select the Secure Your Active Site option.
   - To restrict access to a different site, select the Secure a Specific Directory option, click Browse, and then select the application for which you want to restrict access.
6. Click Next.
7. Depending on the type of authentication you want to use, continue with the instructions in the appropriate section:
   - Creating a login using simple authentication
   - Creating a login using LDAP authentication
   - Creating a login using Windows NT authentication

Creating a login using simple authentication

When you create a Login using simple authentication, you specify one username and password that all users enter to gain access to your application.

To create a login using simple authentication:

1. Start the Login Wizard. (See "Starting the Login Wizard" on page 145.)
2. Select Simple as the type of authentication.
3. Select one of the following:

   Basic Authentication to use the browser dialog box to prompt the user for credentials

   ColdFusion Login page to prompt the user in a web page
4. Click Next.
5. Enter the user name and password that you want users to enter to have access to your application.
6. Click Next.

   Note: You can specify only one user name and password combination for each application.

   The Login Wizard creates the files necessary to restrict access to your application. The Wizard then displays a list of the files.
7. To deploy the modified application to your production server, select the check box.
8. Click Done.

Creating a login using LDAP authentication

If the users of your application have entries in a known LDAP server, you can create a login using LDAP authentication.

To create a login using LDAP authentication:

1. Start the Login Wizard. (See "Starting the Login Wizard" on page 145.)
2. Select LDAP as the type of authentication.
3. Select one of the following:

   Basic Authentication to use the browser dialog box to prompt the user for credentials

   ColdFusion Login page to prompt the user in a web page
4. Enter the required information in each text box.
5. Click Verify LDAP Server to verify connection to the LDAP server.
6. Click Next.
7. Select the check box to have the Login Wizard synchronize the files on the remote server.
8. Click Done.

Creating a login using Windows NT authentication

1. Start the Login Wizard. (See "Starting the Login Wizard" on page 145.)
2. Select NT as the type of authentication.
3. Select one of the following:

   Basic Authentication to use the browser dialog box to prompt the user for credentials

   ColdFusion Login page to prompt the user in a web page
4. Enter the name of the domain for whose members you want to grant access to your application.

   Note: The Login Wizard does not perform validation of the domain name and works only on Windows.
5. Click Next.
6. Select the check box to have the Login Wizard synchronize the files on the remote server.
7. Click Done.

Exercise 2: Requiring users to log in to the Compass Travel application

To ensure that only employees of Compass Travel can access the application you have built, you use the Login Wizard to provide user authentication. You specify only one username and password that all employees of Compass Travel will use to gain access to the application.

Note: To perform the steps in this exercise, you must be running Windows. In addition, the site that contains the Compass Travel application must be a site managed by Dreamweaver.

To add user authentication to the Compass Travel application:

1. Start Dreamweaver.
2. Verify that the Compass Travel site is the current site.
3. In Dreamweaver, select Commands > CF Login Wizard.
4. Click Begin the Wizard.
5. Select the Secure Your Active Site option.
6. Click Next.
7. Select Simple as the type of authentication.
8. Select Basic Authentication.
9. Enter **CompassStaff** as the user name.
10. Enter **Columbus92** as the password.

    The Login Wizard creates the files necessary to restrict access to your application. The Wizard then displays a list of the files.
11. Click Done.

To test the authentication:

1. Open the tripdetail.cfm page in the my_app directory in your browser.
2. When prompted, enter **CompassStaff** as the user name.
3. Enter **Columbus92** as the password.

Summary

In this lesson, you learned how to use the Login Wizard to provide user authentication for a site, and added user authentication to the Compass Travel application.

You have completed the Compass Travel application. As you create your own ColdFusion MX applications, see *ColdFusion MX Developer's Guide* and *CFML Reference*.

INDEX

A

action pages
  creating 107
  defined 28
actions based on buttons 127
adding
  database maintenance buttons 100
  navigation buttons 98
  rows to a table 131
AND operator 75
answers in solution files 47
application. *See* tutorial
authentication
  about 143
  LDAP 146
  simple 146
  Windows NT 147

B

browsing, database records 123
business rules 108
buttons
  database maintenance 100
  delete 138
  determining actions based on 127
  linking to actions 138
  navigation 98
  search 138

C

cfform tag, defined 103
cfif t ag 76
cfif tag 82, 109
cfinput tag 103
cfinsert tag 129
cflocation tag 123
CFML
  basics 15, 30
  building applications 13
  commenting code 30
  elements 18, 23
  expressions 23, 27
  functions 19
  operators 26
  SQL 37
  tags 18
  variables 21
cfoutput tag 16, 62
cfquery tag 61
cfselect tag 103
cfset tag 76, 82, 109
cfupdate tag 129, 137
check boxes, validating 111
client-side validation 115
ColdFusion components
  about 63
  copying queries to 65
  creating 64
  invoking methods in 65
ColdFusion forms 116
ColdFusion markup language. *See* CFML
ColdFusion MX 7
  about 11
  Administrator 12
  application server 11
  components 11
  described 11, 12
ColdFusion pages
  browsing 17
  creating 16
  described 15, 18
  saving 17
comment tag, defined 30

conditional processing 27, 28, 76, 82
contains operator 73
context root 17
creating
  action pages 107
  CFCs 64
cross-field edits 108
currency, formatting 93

D

data sources
  defined 36
  supported 13
data types, recognizing 44
data validation
  about 108
  client-side 115
  server-side 109
data, retrieving 61
databases
  browsing records 123
  CFML tags 37
  connecting to 53
  data types 44
  designing 43
  establishing table relationships 45
  fields 32
  fundamentals 31, 37
  primary key 45
  records 32
  relational 32
  tables 32
  understanding 31
DateFormat function 16, 85
dates
  displaying 66
  formatting 93
debugging information, remote clients 57
debugging options, enabling 56
debugging settings, described 55, 57
deleting records 129
development tools, described 13, 57
displaying
  dates 66
  query results 68, 88
  search results 79
  trip information 60
  trips by price 66
  variables 62
DollarFormat function, defined 85
Dreamweaver
  creating CFC queries 67, 78, 87
  creating CFCs 64
  creating queries 61
  Extensions 143
  invoking methods 66, 67, 68, 80
  Login Wizard 143
dynamic queries 75
dynamic SQL 75, 124
dynamically populating lists 119

E

editors, supported 15
enable searching 127
ends with operator 73
equals 73
expressions
  building 24
  character case 24
  defined 23
  denoting functions or variables 24
  literal values 25
  operators 25
  quotation marks 25

F

FileExists function 103
foreign key, defined 45
form page, defined 28
form processing, defined 28, 29
formatting
  currency 93
  dates 93
forms 72
  ColdFusion 116
  for searching 73
  HTML 73
  inserting values in 139
  trip edit 104
functions
  DateFormat 16, 85
  defined 19, 20
  DollarFormat 85
  FileExists 103
  IsDefined 85
  nesting 21
  Now 16
  operation on values 21
  syntax 20
  URLEncodedFormat 85

G

greater than operator 73

H

HTML forms 73

I

inserting, values in forms 139
invoking methods 65, 68
IsDefined function 85

J

J2EE configuration 17

L

LDAP authentication 146
LDAP server 143
less than operator 73
limiting, number of rows 125
linking, buttons and actions 127
lists, dynamically populating 119
local variables, creating 109
Login Wizard 143

M

Macintosh OS X
  data source 53
  saving CFM pages 17
Macromedia ColdFusion MX 7. *See* ColdFusion MX 7
Macromedia Dreamweaver MX
  configuring 51
  defined 50
methods, invoking 68
Microsoft Access
  about 82, 133
  configuration 53

N

navigation buttons 98
Now function, defined 16
number signs, using 62

O

operators
  about 25, 27
  for querying 73

P

photo file, testing existence of 120
PointBase database
  about 82, 133
  configuration 54
primary key, defined 45

Q

queries
  copying to CFCs 65
  displaying results 62
  dynamic 75
  dynamic SQL in 77
  search operators 73
query results
  displaying 62, 68, 88
  sorting 66

R

radio buttons, validating 111
records
  adding 131
  deleting 129
  updating 137
relational databases, defined 32
result set, limiting 125
retrieving data 61
reusable code, writing 63
rows
  adding 131
  limiting number of 125
  updating 137
rules, for data validation 108

S

search operators 73
searching
  databases 72
  enabling 127
SELECT statement 61
separating business logic from presentation 63
serve- side validation 109
simple authentication 146
single-field edits 108
solution files, how to use 47
sorting query results 66

SQL
AND operator 75
commands 34
data source 36
described 34, 37
dynamic 75, 124
INSERT statement 131
ORDER BY statement 36
security risks 91
SELECT statement 34, 61
using 61
WHERE clause 35
writing statements 37
starts with operator 73
structured code 63

T

tags
attributes 19
cfform 103
cfif 76, 82, 109
cfinput 103
cfinsert 129
cflocation 123
cfoutput 16, 62
cfquery 61
cfselect 103
cfset 76, 82, 109
cfupdate 129
defined 18, 19
syntax 18
testing
conditons 82
deleting records 131
existence of a file 120
validation code 114
variable existence 110
trip detail page, creating 86
trip details, browsing 123
trip information, displaying 60
trip list, displaying 62
trip search page 73
trip search results page 79
tutorial
adding and updating SQL data 129
application business rules 108
application requirements 42
creating main application page 85, 95, 97, 101
data requirements 43
database configuration 53
database design 43
enforcing business rules 122
file structure, described 49
implementing browsing and maintenance functions 123
preparing to use 41
requirements 48
validating trip photo 120
working directories 47

U

UNIX PointBase database 82, 133
updating records 137
URL structure, defined 123
URLEncodedFormat function, defined 85
user authentication 143, 144

V

validation 108
check boxes 111
client-side 115
radio buttons 111
server-side 109
variables
creating 21
defined 21, 23
displaying 62
displaying output 23
local 109
naming 21
scope 22
testing existence of 110

W

WHERE clauses 75
WHERE subclauses 82
Windows NT authentication 147
working directories 47