macromedia®
COLDFUSION®
MX

Getting Started Building ColdFusion MX Applications

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.

Apple Disclaimer
APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

Acknowledgments
Project Management: Stephen M. Gilson

Writer: Cheryl Smith

Editor: Linda Adler

First Edition: May 2002

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

CONTENTS

ABOUT THIS BOOK

Getting Started Building ColdFusion MX Applications is intended for anyone who wants to learn how to use ColdFusion MX to create web applications.

This book consists of two parts:

- Part I introduces related web technologies and explains how ColdFusion works. This part also introduces the ColdFusion Markup Language (CFML), basic database concepts, and information about how to configure your development environment for the tutorial in Part II of this book.
- Part II provides a complete tutorial of everything you need to know to build a sample ColdFusion application. It consists of six lessons. Ideally, you should work through each of the lessons in the order they are presented.

Contents

Developer resources

Macromedia, Inc. is committed to setting the standard for customer support in developer education, documentation, technical support, and professional services. The Macromedia website is designed to give you quick access to the entire range of online resources. The following table shows the locations of these resources:

Resource	Description	URL
Macromedia website	General information about Macromedia products and services	http://www.macromedia.com
Information on ColdFusion	Detailed product information on ColdFusion and related topics	http://www.macromedia.com/coldfusion
Macromedia ColdFusion Support Center	Professional support programs that Macromedia offers	http://www.macromedia.com/support/coldfusion
ColdFusion Online Forums	Access to experienced ColdFusion developers through participation in the Online Forums, where you can post messages and read replies on many subjects relating to ColdFusion	http://webforums.macromedia.com/coldfusion/
Installation Support	Support for installation-related issues for all Macromedia products	http://www.macromedia.com/support/email/isupport
Training	Information about classes, on-site training, and online courses offered by Macromedia	http://www.macromedia.com/support/training
Developer Resources	All the resources that you need to stay on the cutting edge of ColdFusion development, including online discussion groups, Knowledge Base, technical papers, and more	http://www.macromedia.com/desdev/developer/
Reference Desk	Development tips, articles, documentation, and white papers	http://www.macromedia.com/v1/developer/TechnologyReference/index.cfm
Macromedia Alliance	Connection with the growing network of solution providers, application developers, resellers, and hosting services creating solutions with ColdFusion	http://www.macromedia.com/partners/

About Macromedia ColdFusion MX documentation

The ColdFusion documentation is designed to provide support for the complete spectrum of participants. The print and online versions are organized to let you quickly locate the information that you need. The ColdFusion online documentation is provided in HTML and Adobe Acrobat formats.

Printed and online documentation set

The ColdFusion documentation set consists of the following titles:

Book	Description
Installing ColdFusion MX	Describes system installation and basic configuration for Windows NT, Windows 2000, Solaris, Linux, and HP-UX.
Administering ColdFusion MX	Describes how to use the ColdFusion Administrator to manage the ColdFusion environment, including connecting to your data sources and configuring security for your applications,
Developing ColdFusion MX Applications with CFML	Describes how to develop your dynamic web applications, including retrieving and updating your data, using structures, and forms.
Getting Started Building ColdFusion MX Applications	Contains an overview of ColdFusion features and application development procedures. Includes a tutorial that guides you through the process of developing an example ColdFusion application.
Using Server-Side ActionScript in ColdFusion MX	Describes how Macromedia Flash movies executing on a client browser can call ActionScript code running on the ColdFusion server. Includes examples of server-side ActionScript and a syntax guide for developing ActionScript pages on the server.
Migrating ColdFusion 5 Applications	Describes how to migrate a ColdFusion 5 application to ColdFusion MX. This book describes the code compatibility analyzer that evaluates your ColdFusion 5 code to determine any incompatibilities within it.
CFML Reference	Provides descriptions, syntax, usage, and code examples for all ColdFusion tags, functions, and variables.
CFML Quick Reference	A brief guide that shows the syntax of ColdFusion tags, functions, and variables.
Working with Verity Tools	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.
Using ClusterCATS	Describes how to use Macromedia ClusterCATS, the clustering technology that provides load-balancing and failover services to assure high availability for your web servers.

Viewing online documentation

All ColdFusion documentation is available online in HTML and Adobe Acrobat Portable Document Format (PDF) files. To view the HTML documentation, open the following URL on the web server running ColdFusion: http://*web_root*/cfdocs/dochome.htm.

ColdFusion documentation in Acrobat format is available on the ColdFusion product CD-ROM.

Getting answers

One of the best ways to solve particular programming problems is to tap into the vast expertise of the ColdFusion developer communities on the ColdFusion Forums. Other developers on the forum can help you figure out how to do just about anything with ColdFusion. The search facility can also help you search messages from the previous 12 months, allowing you to learn how others have solved a problem that you might be facing. The Forums is a great resource for learning ColdFusion, but it is also a great place to see the ColdFusion developer community in action.

Contacting Macromedia

Corporate headquarters	Macromedia, Inc. 600 Townsend Street San Francisco, CA 94103 Tel: 415.252.2000 Fax: 415.626.0554 Web: http:// www.macromedia.com
Technical support	Macromedia offers a range of telephone and web-based support options. Go to http://www.macromedia.com/support/coldfusionfor a complete description of technical support services. You can make postings to the ColdFusion Support Forum (http://webforums.macromedia.com/coldfusion) at any time.
Sales	Toll Free: 888.939.2545 Tel: 617.219.2100 Fax: 617.219.2101 E-mail: sales@macromedia.com Web: http://www.macromedia.com/store

PART I

Welcome to ColdFusion

Part I provides an introduction to ColdFusion. It defines ColdFusion and provides an overview of the ColdFusion Markup Language (CFML) . It also provides generic database concepts, and information about how to prepare your development environment for using the tutorial in Part II of this book.

CHAPTER 1
Introducing ColdFusion MX

This chapter introduces the core technologies that are the foundation for Macromedia ColdFusion MX. It provides a basic understanding of the Internet, web servers, URLs, HTML, and JavaScript. In addition, it introduces the basic concepts about ColdFusion MX, how it works, and the various components that comprise it.

Contents

The Internet and related technologies

ColdFusion MX is tightly integrated with the Internet and the World Wide Web. For a better understanding of ColdFusion MX, the following sections provide a basic description about the Internet and its related technologies.

The Internet

One way to think of the Internet is to picture it as a wide-area network that spans multiple geographic locations. Each location in this enormous network is comprised of a group of computers that are relatively close in proximity to each other and are connected by hardware and cabling.

Users communicate from one location to another using a communication protocol known as **IP** (Internet Protocol). This protocol, running on each computer connected to the Internet, ensures that communication breakdowns do not occur and that the networked computers can communicate and exchange data properly with each other.

Every computer connected to the Internet has a unique IP address. If duplicate IP addresses existed, information using a given address could end up in the wrong place. It would be like using the same street address for two residences. As a sender of information, you would never know if the intended receiver did receive your information.

Internet applications

Software applications that run on the Internet are known as **Internet applications**. The following table lists some of the most popular Internet applications:

Application	Description
WWW	World Wide Web. The web is a hypertext information system. It lets you read and navigate to text and visual information in a nonlinear way that is based on what you want to read next. This freely available information is linked together in various ways on the Internet and is available for you to browse whenever you want. A website is a location on the World Wide Web. When you view a page, your browser connects to that website to get the information.
E-mail	Electronic mail. E-mail programs, such as Microsoft Outlook, let you send and receive mail electronically over the Internet.
FTP	File Transfer Protocol. This protocol lets you transfer information between hosts using an FTP site.
Telnet	Telnet lets you log on to a computer from a remote location.
NFS	Network File System. NFS lets you share files between hosts.

Internet applications communicate across the Internet by using IP. IP transmits application data in small packets to a destination IP address. The receiving host processes the information that it receives.

Intranet applications

An **intranet** is a private LAN (Local Area Network) or WAN (Wide Area Network) that lets you use and interact with your Internet-based applications in a secure environment. These private networks exist in large corporations, small companies, and even home offices. Private networks let companies and organizations determine who can share their information and who can access it.

An **intranet application** is an application that works on a private intranet (network). It differs from an Internet application only in who can access it and the location of the client computer accessing it. An intranet application can also operate over a public Internet. When an intranet application runs on the Internet, the application is called an Internet application. These terms, *Internet applications* and *intranet applications,* are used interchangeably throughout this book.

Web servers

A **web server** is a software program that serves web pages to requesting clients. The web server software runs on any computer. Often people refer to the host running the web-server software as the web server, and think of it as the hardware. However, technically, the web server is just the software program and not the hardware.

How a web server and connecting hosts communicate

When a user at a specific IP address requests a file, the web server retrieves that file and returns it to the requesting IP address. The contents of a file are not important to the web server. It is the web browser that makes the request and interprets and displays the data in the file that was returned from the web server.

When you make a request from a web server, an IP connection is made across the Internet between the client making the request and the host running the web-server software. As soon as a request is satisfied by the web server, the Internet connection between the client and the host breaks. A page containing images or links to other pages all require separate connections. Often, it takes many requests to retrieve all the information on one web page.

Web pages

The information on the World Wide Web is presented in web pages. You can create web pages using a series of client-side technologies. A web page can include a variety of information: text, lists, forms for capturing data, tables for presenting data, scripts that perform a function, multimedia content that animate pages, and so on.

No matter the content of the web page, the web browser must process and display the page.

Web browsers

A **web browser** is a software program residing on a computer that you use to view pages on and navigate the World Wide Web. When you use a browser to request a page on a website, that browser is making a web connection to a web server.

As mentioned previously, the web browser processes the web pages that it receives from a web server and displays the pages to the user. Depending on the browser that you use and the features it includes, you might be able to play multimedia files, view and interact with Java applets, read your e-mail, or use other advanced features.

Some of the most popular web browsers today are Microsoft Internet Explorer, Netscape Navigator, and Mozilla. Unfortunately, most browsers today parse web pages differently. Web designers must pay special attention to the way a browser behaves, or users might not see the pages as the designers intended. Therefore, web designers test their pages on multiple browsers before publishing them on their website.

HTML

Web page authors create plain text files using the Hypertext Markup Language. This language, known as HTML, consists of a series of simple-to-learn tags. You can use these tags to mark up a page of text. Tags can indicate page elements, structure, formatting, hypertext links to other locations, and so on. Web browsers read the HTML tags and format the text and styles that appear on the computer screen.

HTML tags usually have a starting and ending tag, surrounding the text that they affect. The starting tag turns on a feature (heading, bold, and so on), and the ending tag turns it off. All ending tags have a forward slash (/) preceding the tag name.

Most HTML tags look like this:

```
<TheTagName>text</TheTagName>
```

The tag name is always enclosed in angle brackets (<>) and is case-insensitive, which means that you can specify the tag name in uppercase, lowercase, or mixed case.

Most web browsers let you view the source of an HTML page. This option is usually located in a menu or a button.

JavaScript

Web developers write JavaScript to create small programs that run in the browser. JavaScript is one of the most popular client-side scripting languages today. It is supported by almost all browsers on the market. Web developers use JavaScript to do these actions:

- Validate user actions.
- Create scrolling messages in a browser's status bar.
- Animate text or images.

JavaScript can be inserted in the HTML file. HTML uses tags to mark the start and end of the code. The <script> tag tells the browser that the following chunk of text, bounded by the closing </script> tag, is not HTML, but rather script code to be processed.

Although using JavaScript seems much like inserting HTML content, JavaScript is more difficult to learn than HTML. For more information about JavaScript, see any JavaScript primer.

URLs

Every piece of information on the World Wide Web has a unique address. This address is called a Uniform Resource Locator, or URL. A **URL** is a pointer to some bit of data on the web. This information might include a web document, a file on a FTP site, a posting on Usenet, or even an e-mail address. URLs contain information about the following:

- How to get the information (what protocol to use: FTP, HTTP, and so on)
- The Internet host name to contact (for example, www.macromedia.com; http://localhost/mysite; or ftp.mysite.com)
- The directory or other location to locate the requested information

In addition, you use special URLs to send e-mail and for using the Telnet program.

Understanding web application servers

As explained previously, web browsers make requests, and web servers fulfill those requests by returning the requested information to the browser. This information is usually HTML files, as well as other types.

When you think about it, web servers capabilities are limited because all they do is wait for requests to arrive and attempt to fulfill those requests as soon as possible. Web servers do not let you do the following tasks:

- Interact with a database.
- Serve up customized information based on user preferences or requests.
- Validate user actions.

Web servers, basically, locate information and return that information to a web browser.

To extend the capabilities of a web server, you need a web application server. A **web application server** is a software program that lets the web server do more tasks, like those listed in the previous paragraph.

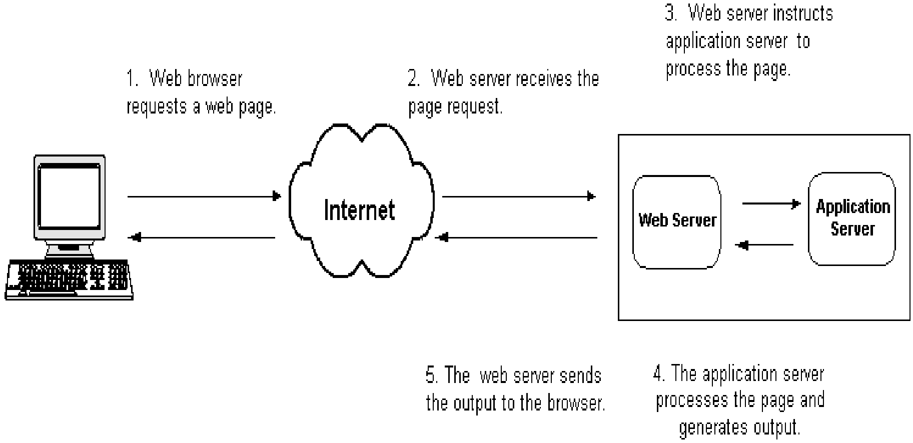
How a web server and web application server work together

The following steps explain how a web server processes a page that also needs processing by a web application server:

1 The user requests a page by typing a URL in a browser, and the web server receives the request.

2 The web server looks at the MIME type (or file extension) to determine whether a web application server must process the page. Then one of the following actions occur:

   - If the MIME type indicates that the file is a simple web page (typically an HTM extension), then the web server fulfills the request and sends the file to the browser.
   - If the MIME type indicates that the requested file is a page that a web application server must process (CFM or CFC extension for ColdFusion requests), then the web server passes it to the web application server. The web application server processes the page and sends the results to the web server, which returns those results to the browser. The following figure shows this process:


3. Web server instructs application server to process the page.
1. Web browser requests a web page.
2. Web server receives the page request.
Internet
Web Server
Application Server
5. The web server sends the output to the browser.
4. The application server processes the page and generates output.

Web application servers process code in a page that a browser and web server cannot interpret. The web server recognizes these requested pages by the file extension and forwards it to the web application server for action. The web application server interprets the programming instructions in the page and generates output that a web browser can interpret. Then the web server returns the output to the browser.

By using a web application server, web developers can build highly interactive and data-rich sites, such as the following:

- Create shopping carts and e-commerce sites.
- Query other database applications for data.
- Dynamically populate form elements.
- Respond with e-mail immediately after a user submits a form.

What is 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 Administrator

The following sections describe these core components in more detail.

The ColdFusion application server

The ColdFusion application server is a software program that 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 file extension. A ColdFusion page looks like an HTML page but contains special tags that instruct the ColdFusion server to perform specific operations.

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:
  - Tags that begin with `cf`.
  - Variables and functions that are always surrounded by pound signs (#).

2 If the Coldfusion server finds any HTML or plain text in the page, the ColdFusion server returns it to the web server untouched.

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.

The ColdFusion Markup Language

ColdFusion Markup Language (CFML) is a 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*; for example:

```
<cfstarttagname> </cfendtagname>
```

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 JSP tag libraries).

Interacting with data sources

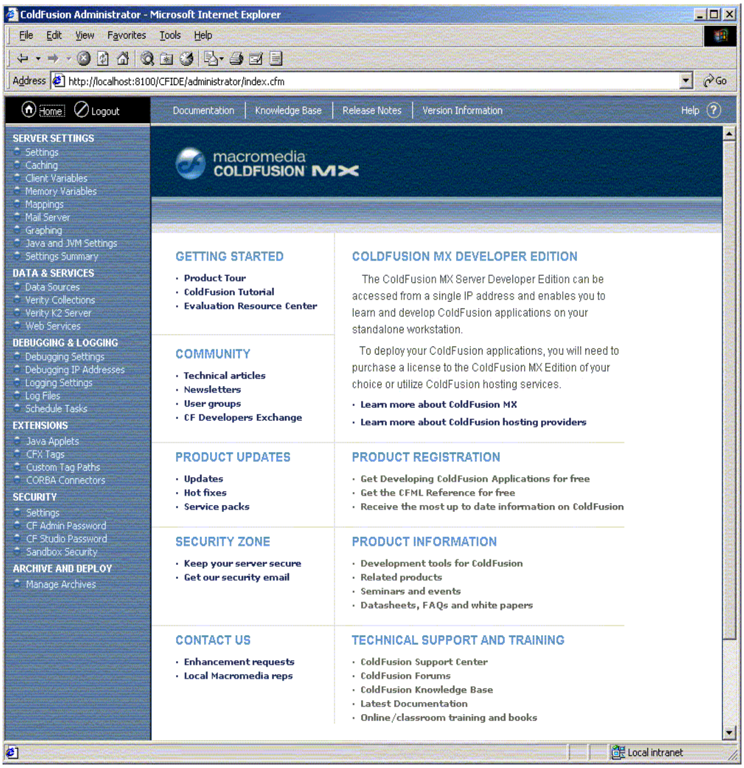
ColdFusion applications can interact with any database that supports a JDBC technology-based driver. A JDBC technology-based driver uses an Application Programming Language (API) to execute SQL statements to databases on most platforms. However, ColdFusion is not limited to JDBC data sources. You can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

Development tools

While you can code your ColdFusion application with NotePad or any HTML editor, 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 4, "Configuring Your Development Environment" on page 35.

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.



The ColdFusion MX Server Developer Edition can be accessed from a single IP address and enables you to learn and develop ColdFusion applications on your standalone workstation.

To deploy your ColdFusion applications, you will need to purchase a license to the ColdFusion MX Edition of your choice or utilize ColdFusion hosting services.

Learn more about ColdFusion MX
Learn more about ColdFusion hosting providers
PRODUCT REGISTRATION
Get Developing ColdFusion Applications for free
Get the CFML Reference for free
Receive the most up to date information on ColdFusion
PRODUCT INFORMATION
Development tools for ColdFusion
Related products
Seminars and events
Datasheets, FAQs and white papers
TECHNICAL SUPPORT AND TRAINING
ColdFusion Support Center
ColdFusion Forums
ColdFusion Knowledge Base
Latest Documentation
Online/classroom training and books
Local intranet

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

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

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

Using ColdFusion MX with Macromedia Flash MX

Macromedia Flash MX is designed to overcome the many limitations of HTML and solve the problem of providing efficient, interactive, user interfaces for Internet applications. ColdFusion MX is designed to provide a fast efficient environment for developing and providing data-driven Internet applications.

By using ColdFusion MX and Flash MX together, you can easily create complete visually interactive applications for the Internet. ColdFusion MX provides a native Flash connectivity that ensures visual applications created with Flash MX can easily and securely interact with ColdFusion. Flash MX clients communicate efficiently with ColdFusion by using an Action Message Format protocol over HTTP. This protocol provides fast, lightweight, binary transfer of data between the Flash client and ColdFusion.

By using the following features of ColdFusion MX and Flash MX, you can create efficient data-driven Internet applications with visually interactive user interfaces:

- **Flash MX ActionScript commands** Lets you connect to ColdFusion components (CFC) and ColdFusion pages.
- **Flash MX development application debugger** Lets you trace your application logic as it executes between Flash and ColdFusion.
- **ColdFusion MX Server-Side ActionScript** Lets Flash programmers familiar with ActionScript create ColdFusion services, such as SQL queries, for use by Flash clients.

For more information about using Server-Side ActionScript in ColdFusion MX, see *Using Server-Side ActionScript in ColdFusion MX*. For more information about developing Flash application in ColdFusion, see *Developing ColdFusion MX Applications with CFML*. For more information about using Flash MX, go to Macromedia on the web (www.macromedia.com).

CHAPTER 2
CFML Basics

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

Contents

Working with ColdFusion pages

As discussed in Chapter 1, 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 (discussed in Chapter 4)
- Macromedia HomeSite+ (discussed in Chapter 4)
- 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.

Note: This book shows 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 book.

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 server is installed. If you are creating your pages on a local server (on which ColdFusion is running), then you can save the pages locally; if you are using a remote server, then you must save your pages on that server.

To publish ColdFusion pages on the Internet, you must save the pages under the *web-root* directory.

To save the code you just typed (to create a ColdFusion page):

1 Create a new directory called test under the *web_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.

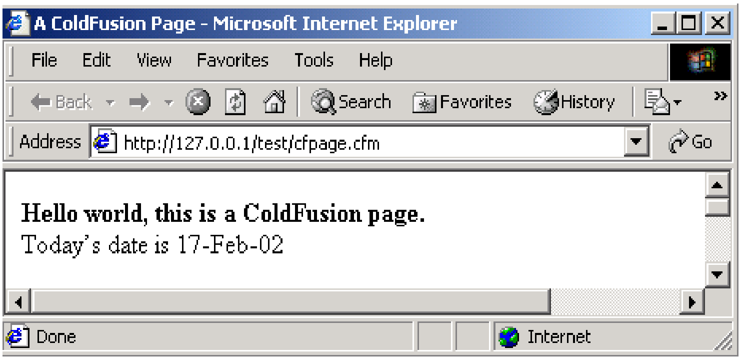
To view the ColdFusion page:

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

http://127.0.0.1/test/cfpage.cfm

The address 127.0.0.1 refers to the localhost and is only valid when you view pages locally. The URL for a remote site would include the IP address of the server where ColdFusion is installed; for example: http://<serveripaddress>/test/cfpage.cfm.

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


A ColdFusion Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://127.0.0.1/test/cfpage.cfm
Go
Done
Internet

Hello world, this is a ColdFusion page.
Today's date is 17-Feb-02

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 page 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, 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.

Understanding 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 you want to accomplish with ColdFusion will involve using tags and functions.

You will 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, 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 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`.
- A start and end tag.
- Use of attributes (like html tags), and most attributes have values.

Some ColdFusion tags, such as `cfset`, omit the closing 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 the ColdFusion server about the details of an operation. For example, to update a database table, the server needs to know specifics 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

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 containing 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, thousand separator, and dollar sign. The pounds signs (#) around the function instruct ColdFusion to evaluate the content between the pound signs and display the value.

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.

Nesting functions

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

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

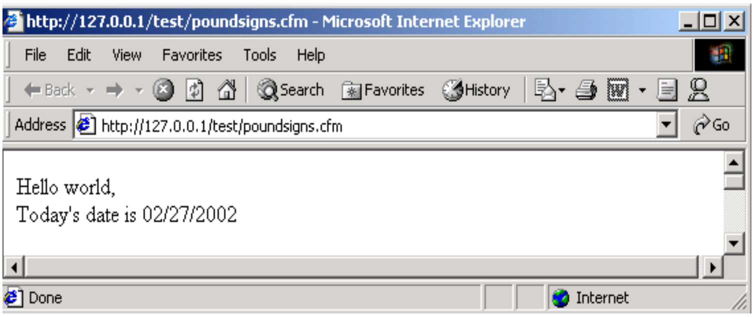
In this example, the Now() function generates the date, and then the DateFormat function formats the date.

Functions and pound signs

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

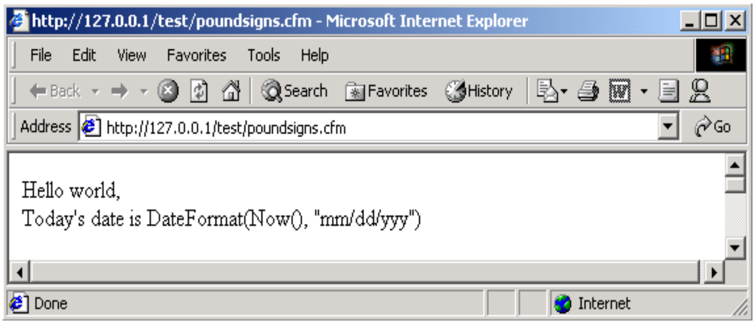
```
<cfoutput>
Hello world, <br>
Today's date is #DateFormat(Now(), "mm/dd/yyyy")#
</cfoutput>
```

The following figure shows the output of this example:


http://127.0.0.1/test/poundsigns.cfm - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
History
Address
http://127.0.0.1/test/poundsigns.cfm
Go
Done
Internet

Hello world,
Today's date is 02/27/2002

If you did not include the pound signs around the `DateFormat(Now(), "mm/ddyyy")` function, the output for the previous example would display as follows:


http://127.0.0.1/test/poundsigns.cfm - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
History
Address
http://127.0.0.1/test/poundsigns.cfm
Go
Done
Internet

```
Hello world,
Today's date is DateFormat(Now(), "mm/dd/yyy")
```

For more information about how to use pound signs with functions, see *Developing ColdFusion MX Applications with CFML.*

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 a variable 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. The syntax for this tag is:

```
<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 (') 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 shown have been about local variables. **Local variables** are variables that you can use only on the current ColdFusion page. As shown in the previous example, a Variables prefix was used to reference an existing variable on the page. Using a prefix when referencing a variable is important because ColdFusion supports many types of variables. The syntax for referencing a local variable is as follows:

```
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

ColdFusion supports many types of variables. Each type 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 cfset or cfparam. Most often you define the variable on the current page or on a page that you include using cfinclude.
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 are named 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 book. 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:

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

To return the value of a variable, you must always surround the variable name with pound 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 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)

Many examples of expressions were shown in this chapter; 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 program.

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 book.)

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

- Character case consistency
- When to use the pound (#) sign
- 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 pound signs to denote functions or variables

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

The following list identifies some common ways to use pound signs:

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

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

  If you omit the pound signs, the text, not the value, appears on the page.

- In the following example, you do not include the pound signs because you are using `cfset` to assign one variable's value to another value:

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

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

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

  The result is the following output:

  #1. Your name.

For more information and examples on using pound signs in expressions, see *Developing ColdFusion MX Applications with CFML*.

Specifying quotation marks around values

When assigning literal values to variables, you must surround the literal value with double quotation marks or single 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; for example:

```
<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 double the 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 using 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 these operators as the start or end of a tag.

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

Nonsupported 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.

Nonsupported logical operator	Equivalent ColdFusion decision operator	Description
>	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 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 \ (trailing slash) 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.

Understanding conditional processing

To this point, all the coding examples shown 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 if 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`.

Conditional processing tags

ColdFusion provides several tags that let you control how a page is processed. 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 book, 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 *Developing ColdFusion MX Applications with CFML*.

Using cfif 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 for a `cfif` statement is as follows:

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

In the previous 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, then the code between the `cfelse` tag and the `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 cfelseif to evaluate multiple expressions

To evaluate multiple expressions in a `cfif` statement, you can use `cfelseif` and `cfelse` 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. In this example, you created 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:

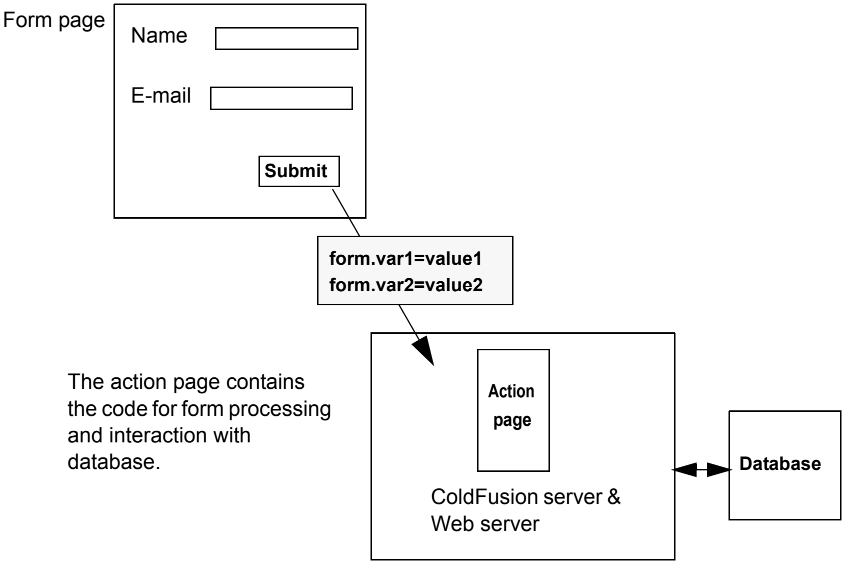
1 Creating the layout for the form itself.

2 Writing the code to process the submitted information.

Form processing

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:


Form page
Name
E-mail
Submit
form.var1=value1
form.var2=value2
The action page contains the code for form processing and interaction with database.
Action page
ColdFusion server & Web server
Database

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 processes 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 book, you will 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 the 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 --- >
```

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.

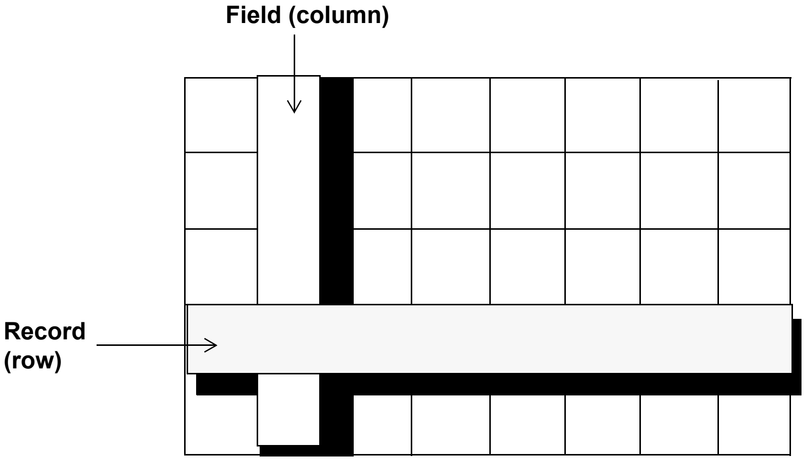
Contents

Understanding 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 get you started with ColdFusion.

What is a relational database?

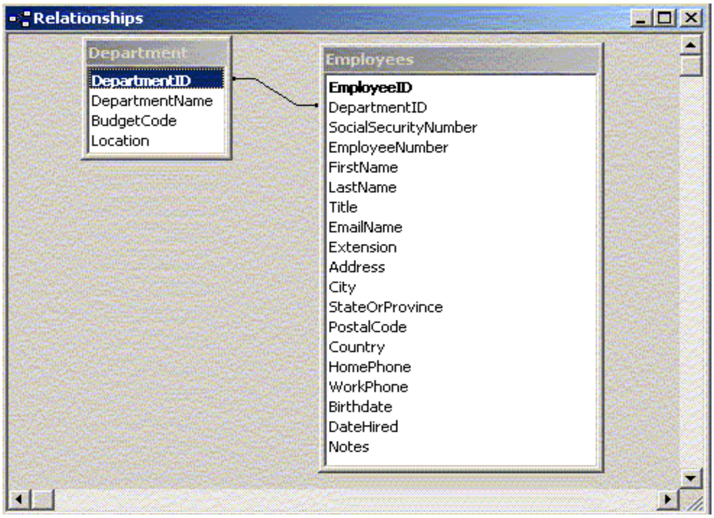
A **relational database** is a structured collection of information that is related to a particular subject or purpose, such as an inventory database or a human resources database. You use databases to manage information. Information, such as product name, cost, and on-hand inventory, is stored in a 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 address, city, state, zip, and salary are all stored in their own columns. 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.

Understanding relational tables

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



Because you have multiple departments for employees, but you would not store the information about the departments in every employee row for several 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.

Using SQL to interact with a database

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 these SQL commands:

Command	Description
SELECT	Use to retrieve (query) information in a database.
INSERT	Use to add records to a database.
UPDATE	Use to update information in a database.
DELETE	Use to delete information in a database.

In Part II of this book, you will be introduced to the syntax of these commands when you use them to build a ColdFusion application that interacts with a database. For additional information about SQL, consult any SQL primer.

Using SQL with ColdFusion

ColdFusion communicates with your data source through a database interface called 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 administrator. Chapter 4, "Configuring Your Development Environment" on page 35 discusses how to configure the sample data source file that is supplied for use with Part II of this book. For more information about configuring a data source in ColdFusion, see *Installing ColdFusion MX* or *Developing ColdFusion MX Applications with CFML*.

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 need to include SQL statements in your CFML statements; for example:

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

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

CFML tags that interact with a database

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

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

In Part II of this book, you will be introduced to these tags when you use them to interact with the sample database. For more information about interacting with a database, see *Developing ColdFusion MX Applications with CFML* or *CFML Reference*.

CHAPTER 4
Configuring Your Development Environment

This chapter describes how to set up your development environment for the tutorial in Part II of this book. It specifies the tutorial file structure, and how to configure the database connection and debugging options in the ColdFusion Administrator. Additionally, it provides a brief overview of using Macromedia Dreamweaver MX or Macromedia HomeSite+ for ColdFusion development.

Contents

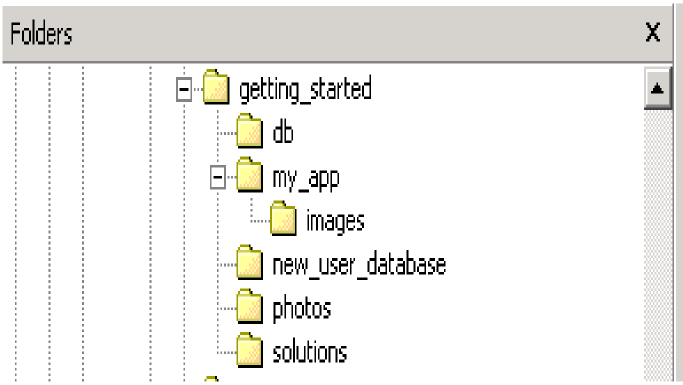
Verifying the tutorial file structure

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

The files required to complete the Compass Travel tutorial (in Part II of this book) 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:

- For local third-party web server configurations, the files are installed in: webroot\cfdocs\getting_started.
- For stand-alone ColdFusion web server configurations, the files are installed in: cfusionmx\webroot\cfdocs\getting_started.

The following figure shows the getting_started directory structure:



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 Part II of this book.

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 the sample PointBase database.

Save all the files that you create for the tutorial application (in Part II of this book) 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.

Configuring database connection and debugging options

Prior to ColdFusion development, use the ColdFusion MX Administrator to define the connection to the sample database file and any optional debugging options.

To access the ColdFusion Administrator, do either of the following:

- Select **Start > Programs > Macromedia ColdFusion MX > ColdFusion MX Administrator**.
- Open a browser and go to one of the following URLs:
  - External web server users: http://localhost/CFIDE/administrator
  - Stand-alone web server users: http://localhost:8100/CFIDE/administrator

Note: If you are acessing the ColdFusion Administrator from a remote client, you must replace localhost with the 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 Administrator. Prior to using Part II of this book to build 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 users.

To define the connection to the sample Microsoft Access database:

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

2 In the Add New Data Source dialog box, 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 list box	Select Microsoft Access [Macromedia] .

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

The Macromedia Microsoft Access Data Source dialog box appears.

4 Specify the following:

Field	Action
Database File text box	Specify the location of the CompassTravel.mdb file. Click Browse to locate and select the CompassTravel.mdb file. By default, ColdFusion MX installs the CompassTravel.mdb file in one of the following locations: • For third-party web server configurations: web_root\cfdocs\getting_started\db For standalone ColdFusion web server configurations: cfusionmx\web_root\getting_started\db
Description text box	Enter the following: Database file for Compass Travel tutorial

5 Click Show Advanced Settings and ensure that the settings for CLOB and BLOB are 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:

- 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, then repeat the steps in this procedure to reconfigure this data source.
- Verify that the path specified for the Compass Travel database file is correct.

To define the sample PointBase database file:

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

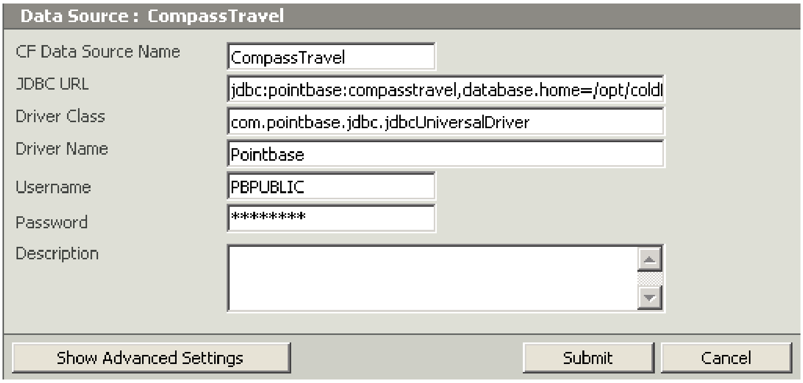
The Add New Data Source dialog box appears.

2 Specify the following:

Field	Action
Data Source name text box	Specify the name CompassTravel. Note: Ensure that the name of the datasource 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 PointBase data source dialog box appears:



Data Source : CompassTravel

CF Data Source Name	CompassTravel
JDBC URL	jdbc:pointbase:compasstravel,database.home=/opt/coldl
Driver Class	com.pointbase.jdbc.jdbcUniversalDriver
Driver Name	Pointbase
Username	PBPUBLIC
Password	********
Description	

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/coldfusionmx/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 ensure that the settings for CLOB and BLOB are 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.

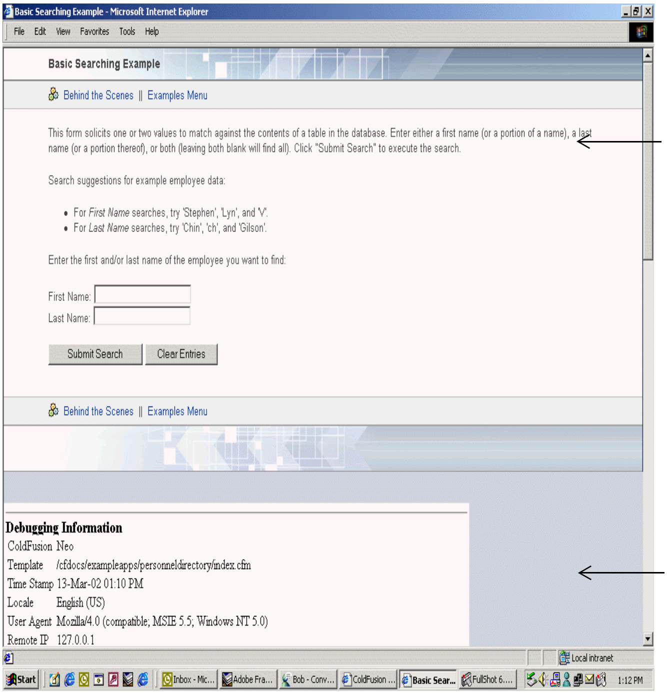
If the connection to the compass travel data source fails, do the following:

- 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, then repeat the steps in this procedure to reconfigure this data source.
- Verify that the JDBC URL specified for the Compass Travel pontbase files is correct.

Enabling debugging options

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



Basic Searching Example

This form solicits one or two values to match against the contents of a table in the database. Enter either a first name (or a portion of a name), a last name (or a portion thereof), or both (leaving both blank will find all). Click "Submit Search" to execute the search.

Search suggestions for example employee data:

- For *First Name* searches, try 'Stephen', 'Lyn', and 'V'.
- For *Last Name* searches, try 'Chin', 'ch', and 'Gilson'.

Enter the first and/or last name of the employee you want to find:

First Name:

Last Name:

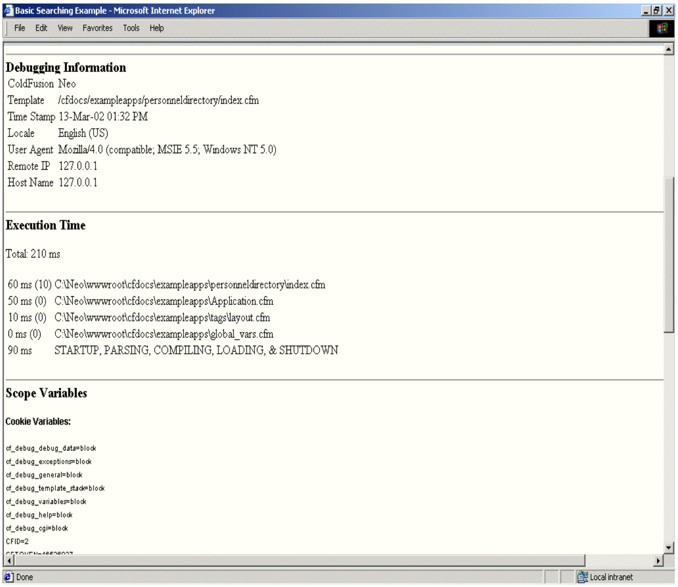
Debugging Information

ColdFusion Neo
Template /cfdocs/exampleapps/personneldirectory/index.cfm
Time Stamp 13-Mar-02 01:10 PM
Locale English (US)
User Agent Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Remote IP 127.0.0.1

The application form

The appended debugging information

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 Administrator. In the following example, the debugging output includes general information about the ColdFusion server, the execution time of the application, and variable information.


Basic Searching Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
Local intranet

```
Debugging Information
 ColdFusion  Neo
 Template    /cfdocs/exampleapps/personneldirectory/index.cfm
 Time Stamp  13-Mar-02 01:32 PM
 Locale      English (US)
 User Agent  Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
 Remote IP   127.0.0.1
 Host Name   127.0.0.1

Execution Time

Total: 210 ms

 60 ms (10) C:\Neo\wwwroot\cfdocs\exampleapps\personneldirectory\index.cfm
 50 ms (0)  C:\Neo\wwwroot\cfdocs\exampleapps\Application.cfm
 10 ms (0)  C:\Neo\wwwroot\cfdocs\exampleapps\tags\layout.cfm
 0 ms (0)   C:\Neo\wwwroot\cfdocs\exampleapps\global_vars.cfm
 90 ms      STARTUP, PARSING, COMPILING, LOADING, & SHUTDOWN

Scope Variables

Cookie Variables:

cf_debug_debug_data=block
cf_debug_exceptions=block
cf_debug_general=block
cf_debug_template_stack=block
cf_debug_variables=block
cf_debug_help=block
cf_debug_cgi=block
CFID=2
```

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

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

To enable debugging options:

1 In the ColdFusion Administrator, select **Debugging and Logging > Debugging.**

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

2 Select the Enable Debugging check box.

  When you select this option, the debugging service is enabled for all options already selected on the page.

3 On 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 the tutorial in Part II of this book, 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 Part II of 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), this procedure is not necessary.

To recieve debugging information when using a remote client:

1 In the ColdFusion Administrator, select **Debugging and Logging > Debugging IP Addresses.**

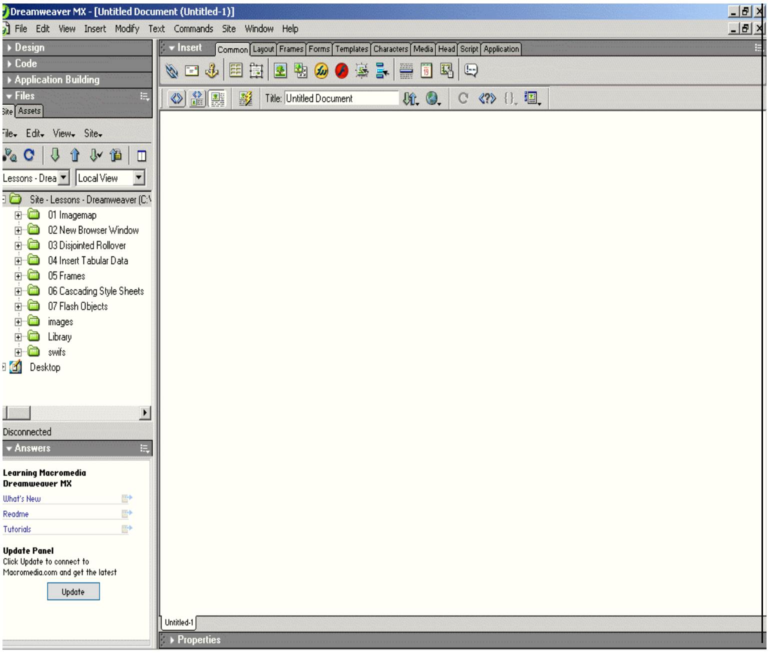
The Debugging IP Address page appears.

2 In the IP Address text box, enter the IP address of your remote client.

3 Click Add.

Macromedia development environment tools

Macromedia Dreamweaver MX is the preferred development environment for building ColdFusion MX applications. It combines the best code editing features of ColdFusion Studio with the visual design features of Dreamweaver.



Update Panel
Click Update to connect to Macromedia.com and get the latest

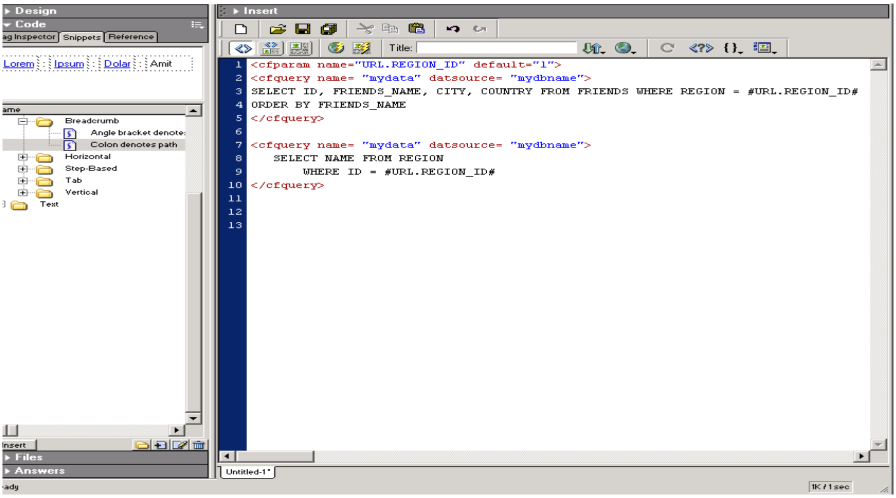
Dreamweaver MX window

Dreamweaver MX supports the latest ColdFusion MX features and tags. It also includes Macromedia HomeSite+, which combines all the features of ColdFusion Studio and HomeSite 5, along with support for the latest ColdFusion MX tags.

With Dreamweaver MX or HomeSite+, you can author and test your application code from a local or remote client. Both of these tools let you 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 and HomeSite+ for ColdFusion development.

The Dreamweaver MX environment

As a ColdFusion developer, you can build ColdFusion MX applications by writing the code manually or generating the code by using one of the code-generating tools provided with Dreamweaver MX.



```
<cfparam name="URL.REGION_ID" default="1">
<cfquery name= "mydata" datsource= "mydbname">
SELECT ID, FRIENDS_NAME, CITY, COUNTRY FROM FRIENDS WHERE REGION = #URL.REGION_ID#
ORDER BY FRIENDS_NAME
</cfquery>

<cfquery name= "mydata" datsource= "mydbname">
   SELECT NAME FROM REGION
        WHERE ID = #URL.REGION_ID#
</cfquery>
```

Features for ColdFusion developers

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

- Rich tag editors for quickly setting attributes and values for every CFML tag.
- Code hints for writing CFML tag attributes.
- Code validator for validating code readiness against other ColdFusion versions.
- Tag chooser with integrated reference material for inserting ColdFusion tags.
- Snippets panel for reusing code.
- Integrated debugging display for quickly pinpointing problem areas in the code.
- Remote ColdFusion server connection for browsing remote data sources and files.

If you plan to use Dreamweaver MX or HomeSite+ to build the sample ColdFusion application in Part II of this book, see the following sections for information about configuring these tools for ColdFusion development.

Configuring Dreamweaver MX for ColdFusion development

Before you use Dreamweaver MX to create the sample application in Part II of this book, you must configure Dreamweaver to recognize the tutorial files and data sources.

To configure Dreamweaver MX to create the sample application:

1 Create a site that contains the tutorial files.

For information about how to create a site in Dreamweaver MX, see the Dreamweaver MX online Help or *Using Dreamweaver MX*.

2 Specify ColdFusion as the application server document type.

For information about how to specify ColdFusion as the server document type, see the Dreamweaver MX online Help or *Using Dreamweaver MX*.

3 Specify ColdFusion MX as the site application server.

For information about how to specify ColdFusion as your application server, see the Dreamweaver MX online Help or *Using Dreamweaver MX*.

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).

Configuring HomeSite+ for ColdFusion development

Before you use Macromedia HomeSite+ to create the sample application in Part II of this book, you must configure HomeSite+ to recognize the tutorial files and data sources.

To use HomeSite+ to create the sample application:

1 Establish a secure connection to the ColdFusion Server environment where the tutorial files are installed. Establishing a connection to the ColdFusion Server is a prerequisite for accessing the ColdFusion data sources.

For information, see the online Help or product documentation for HomeSite+.

2 Enable internal browsing to view and process the ColdFusion application files.

For information, see the online Help or product documentation for HomeSite+.

3 Determine how you want to work with the tutorial files. You can view and access the tutorial files using the Resource Area in the HomeSite + window. You can also set up a project to manage the files more efficiently.

For more information, see the online Help or the product documentation for HomeSite+.

Note: The tutorial provided in this book does not describe how to use HomeSite +. The purpose of this tutorial is to teach you how to build ColdFusion applications using ColdFusion Markup Language (CFML). For more information about HomeSite +, see the product documentation or online Help.

PART II

Building a ColdFusion Application

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

LESSON 1
Preparing to Build the Sample Application

In this tutorial, you will build a simple ColdFusion web application 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 made available to the public. You will build the sample tutorial application to assist 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 in determining the application design. This lesson provides an overview of these application design steps, while the remainder of this book guides you through the lessons on constructing the sample application.

Application development steps

Most software applications perform three major functions:

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

The process steps to develop these major functions varies from project to project. In this tutorial you will review or participate in the following application development steps to build the Compass Travel Trips 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.
3	Design the database for your application by exploring the database tables that will store the trip information.
4	Develop the ColdFusion application pages.

An overview of each of these application development steps is explored in greater detail in the following sections.

Determining the application functional requirements

Before you can build the sample application, you must understand the functional requirements underpinning its design. The design of the sample application centers around the daily tasks performed by Compass Travel's trip coordinators. These tasks are listed in the following table:

Trip coordinator task	Description
Produce current trip listing	To help Compass Travel agents take trip reservations over the phone 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 following functions:

Functional requirement	
1	The ability to generate trip listings
2	A trip query facility based on user supplied criteria
3	Trip browsing functionality
4	The ability to add a new trip
5	The ability to delete an existing trip
6	The ability to edit an existing trip
7	A mechanism to 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. Central to every requirement is the notion 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.

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:

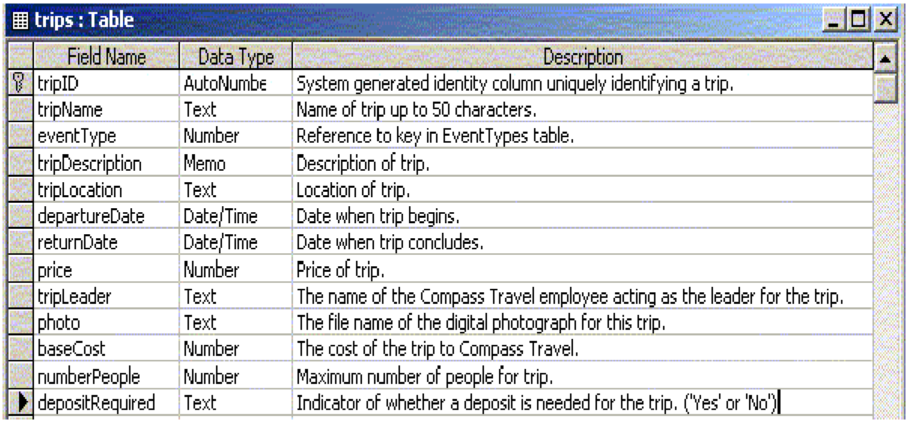
- Trip name
- Type of event (surfing, mountain climbing, kayaking, etc.)
- Trip description
- Trip location
- An indicator of whether a deposit is required
- Departure date
- Return date
- Total number of people who can attend the trip
- Price
- Base cost
- Assigned trip leader
- Trip photograph

By collecting the preceding information about each trip, the Compass Travel website can market its trips online to the general public. Customers 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 file name 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. Cost is not shown on the public website.

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 this:



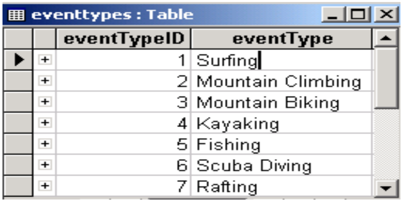
Field Name	Data Type	Description
tripID	AutoNumbe	System generated identity column uniquely identifying a trip.
tripName	Text	Name of trip up to 50 characters.
eventType	Number	Reference to key in EventTypes table.
tripDescription	Memo	Description of trip.
tripLocation	Text	Location of trip.
departureDate	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 classifies 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:

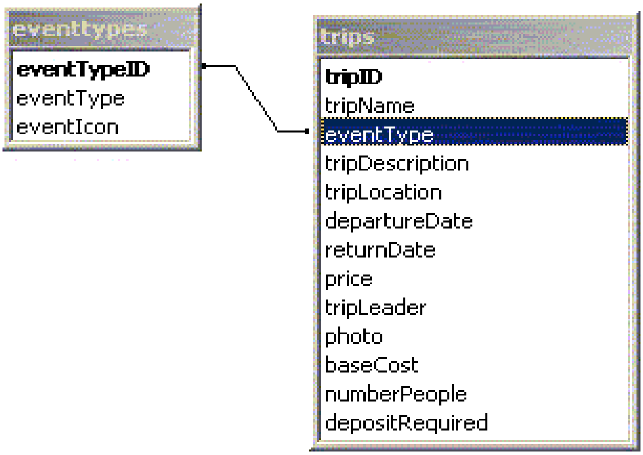


eventTypeID	eventType
1	Surfing
2	Mountain Climbing
3	Mountain Biking
4	Kayaking
5	Fishing
6	Scuba Diving
7	Rafting

Establishing a relationship between the two tables

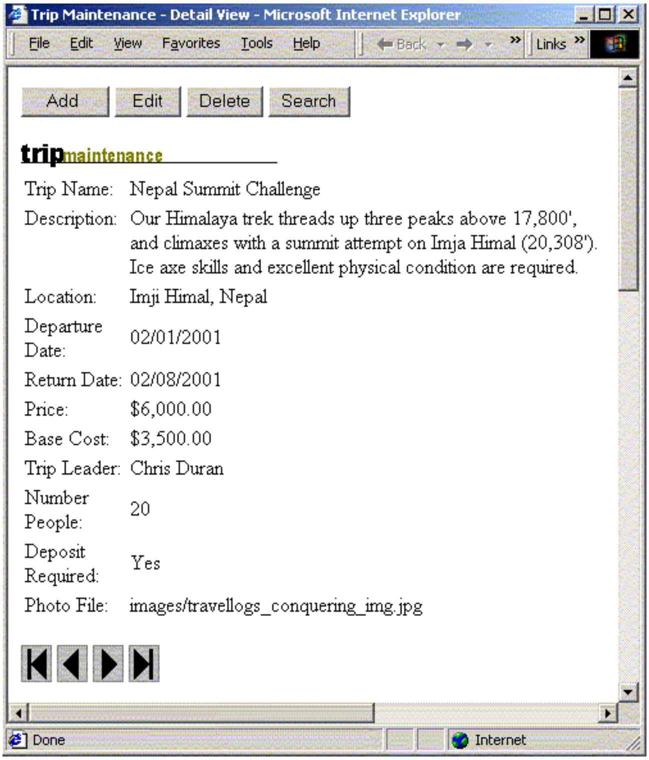
When the user selects an event type from the list obtained from reading the eventtypes table, the correct event type must be saved to the trips table with all the other trip related data. The application could store the eventType (for example, mountain climbing) itself into the eventType column in the 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:



Developing the sample application

Given the application functional requirements and the database provided, you are ready to use ColdFusion to develop the Trips Maintenance application. The remaining lessons in the tutorial will 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:



Trip Name:	Nepal Summit Challenge
Description:	Our Himalaya trek threads up three peaks above 17,800', and climaxes with a summit attempt on Imja Himal (20,308'). Ice axe skills and excellent physical condition are required.
Location:	Imji Himal, Nepal
Departure Date:	02/01/2001
Return Date:	02/08/2001
Price:	$6,000.00
Base Cost:	$3,500.00
Trip Leader:	Chris Duran
Number People:	20
Deposit Required:	Yes
Photo File:	images/travellogs_conquering_img.jpg

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:

- Develop a trip search facility (Lesson 2).
- Build the main application page (Lesson 3).
- Implement browsing and trip maintenance functions (Lesson 4).
- Write code to enforce Compass Travel business rules (Lesson 5).
- Develop the trip add and update pages (Lesson 6).

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 exercise(s).

Depending on your programming experience, you can read the entire lesson then proceed to the hands-on exercises, or you can skip some information in the lesson and proceed directly to the exercises.

Working directories

The following table describes the working directories for this tutorial:

Directory	Description
my_app	You will save all your source code in this directory.
solutions	You can find solutions to all the exercises in this directory.
db	You will use this directory as the working directory for the Compass Travel database. Note: As discussed earlier in Chapter 4, you should verify that the Compass Travel database file residing in the db directory is the original file supplied. This file should have the same date as the file in the new_user_db directory. If date on the file in the db directory is different, replace the file in db directory with a copy of the Compass Travel database file that is located in the new_user_db directory.
photos	You will use this directory to access existing trip photographs.
images	You will use this directory to access application image files.

Locating the working directories

You can locate the working directories for this tutorial under your web root directory. For example, the directory path on your computer might be:

(on Windows NT) <mywebserverdocroot>\cfdocs\getting_started
(on UNIX) <mywebserverdocroot>/cfdocs/getting_started

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

Configuration	URL
For local third-party web server configuration	http://localhost/cfdocs/getting_started/my_app/‹pagename›.cfm
For standalone ColdFusion web server configuration	http://localhost:8100/cfdocs/getting_started/my_app/‹pagename›.cfm

For more information about the tutorial file structure and the location of the getting_started subdirectories, see "Verifying the tutorial file structure" on page 36.

Requirements

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

- **ColdFusion Server** For information on how to install ColdFusion Server, see *Installing ColdFusion MX*.
- **Database Management System** A database management system can be installed on the same computer as the ColdFusion Server or on a separate computer. For the purpose of this tutorial, a Microsoft Access database file for MS Windows users and a PointBase database file for UNIX users has been provided. For information about how to configure the Compass Travel datasource, see "Configuring Your Development Environment" on page 35. For additional information about configuring a data source, see *Installing ColdFusion MX*.
- **Web browser** You can use Internet Explorer (4.0 or later) or Netscape Navigator (6.0 or later).
- **Text editor or IDE** (*Interactive Development Environment*) Macromedia recommends that you use Dreamweaver MX. However, you can use HomeSite+, ColdFusion Studio, any text editor, or IDE. In the exercises in this tutorial, the term **editor** means Dreamweaver MX, HomeSite+, ColdFusion Studio, or any text editor or IDE of your choice.

Note: The default file extension used for ColdFusion application pages is `.cfm`.

LESSON 2
Writing Your First ColdFusion Application

In this lesson, you begin the construction of a ColdFusion web application for the fictitious company, Compass Travel. The exercises in this lesson guide you through the steps of creating queries and forms to search for and display trip offering information from the Compass Travel relational database.

This lesson explains how to do the following tasks:

- Construct a query to retrieve information from a database.
- Develop a search form to accept user criteria.
- Use dynamic SQL to build a flexible search query.
- Develop a results form to display the result of the search.

ColdFusion tags and functions introduced in this lesson

The following table identifies the ColdFusion tags and functions that you use in this lesson to build your first ColdFusion application:

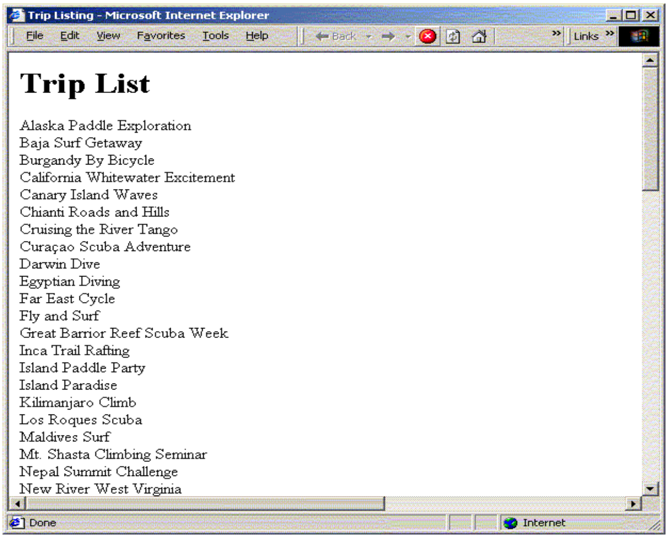
Element	Type	Description
`PreserveSingleQuotes`	Function	Enables you to use single quotation marks in variables used in SQL statements.
`cfoutput`	Tag	Instructs the server to show the results of variables, functions, or text that is specified between the `cfoutput` start and end tags.
`cfquery`	Tag	Submits SQL statements to a JDBC data source.
`cfif`	Tag	Creates conditional statements.
`cfset`	Tag	Defines a ColdFusion variable. If the variable already exists, `cfset` resets it to the specified value.

Creating your first ColdFusion application

As you recall from Lesson 3, 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 lesson.

The following list identifies the components that you will create in this lesson:

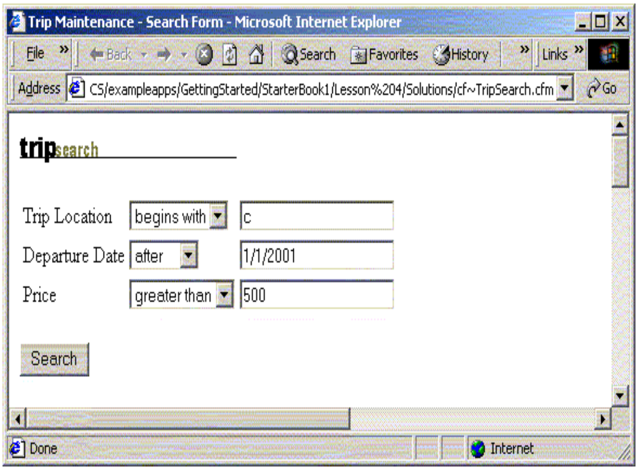
- **Dynamic Trip List page** The purpose of the Trip List page is to present an up-to-date lists of trips on the Compass Travel website.



Trip List

Alaska Paddle Exploration
Baja Surf Getaway
Burgandy By Bicycle
California Whitewater Excitement
Canary Island Waves
Chianti Roads and Hills
Cruising the River Tango
Curaçao Scuba Adventure
Darwin Dive
Egyptian Diving
Far East Cycle
Fly and Surf
Great Barrior Reef Scuba Week
Inca Trail Rafting
Island Paddle Party
Island Paradise
Kilimanjaro Climb
Los Roques Scuba
Maldives Surf
Mt. Shasta Climbing Seminar
Nepal Summit Challenge
New River West Virginia

- **Trip Search form** 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.


Trip Maintenance - Search Form - Microsoft Internet Explorer
File
Back
Search
Favorites
History
Links
Address
Go
Done
Internet

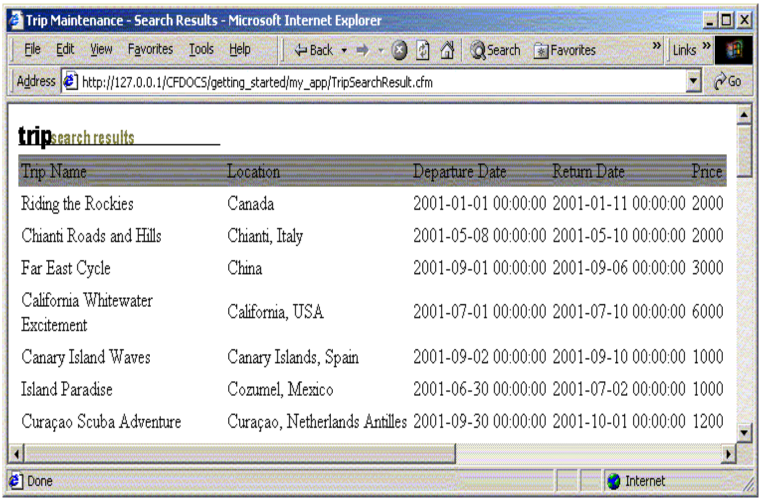
Address: C5/exampleapps/GettingStarted/StarterBook1/Lesson%204/Solutions/cf~TripSearch.cfm

tripsearch

Trip Location	begins with	c
Departure Date	after	1/1/2001
Price	greater than	500

Search

- **Trip Search Results page** The purpose of the Trip Search Results page is to display the results of a trip search.


Trip Maintenance - Search Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Links
Address
http://127.0.0.1/CFDOCS/getting_started/my_app/TripSearchResult.cfm
Go
Done
Internet

tripsearch results

Trip Name	Location	Departure Date	Return Date	Price
Riding the Rockies	Canada	2001-01-01 00:00:00	2001-01-11 00:00:00	2000
Chianti Roads and Hills	Chianti, Italy	2001-05-08 00:00:00	2001-05-10 00:00:00	2000
Far East Cycle	China	2001-09-01 00:00:00	2001-09-06 00:00:00	3000
California Whitewater Excitement	California, USA	2001-07-01 00:00:00	2001-07-10 00:00:00	6000
Canary Island Waves	Canary Islands, Spain	2001-09-02 00:00:00	2001-09-10 00:00:00	1000
Island Paradise	Cozumel, Mexico	2001-06-30 00:00:00	2001-07-02 00:00:00	1000
Curaçao Scuba Adventure	Curaçao, Netherlands Antilles	2001-09-30 00:00:00	2001-10-01 00:00:00	1200

The primary users of these components are the Compass Travel coordinators and agents, not the general public.

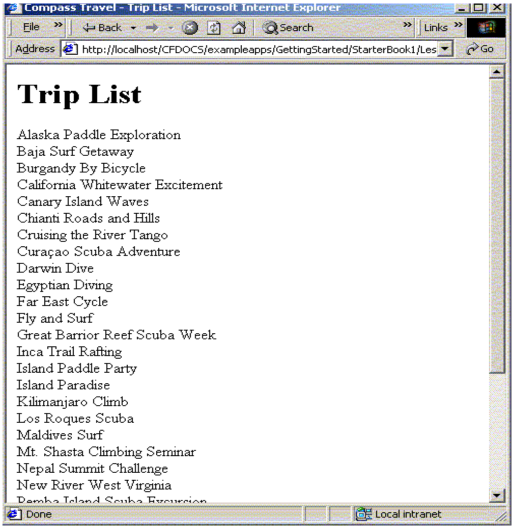
Application development steps

You will review or participate in the following application construction steps:

Steps	Description
1	Create a dynamic web page that displays a list of trips.
2	Design the constraints for the search interface.
3	Develop ColdFusion pages to capture user search criteria and display the results.

Using a web page to list trips

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 accessible by all offices, the trip coordinator added the following HTML web page to the site:


Compass Travel - Trip List - Microsoft Internet Explorer
File
Back
Search
Links
Address
http://localhost/CFDOCS/exampleapps/GettingStarted/StarterBook1/Les
Go
Done
Local intranet

Trip List

Alaska Paddle Exploration
Baja Surf Getaway
Burgandy By Bicycle
California Whitewater Excitement
Canary Island Waves
Chianti Roads and Hills
Cruising the River Tango
Curaçao Scuba Adventure
Darwin Dive
Egyptian Diving
Far East Cycle
Fly and Surf
Great Barrior Reef Scuba Week
Inca Trail Rafting
Island Paddle Party
Island Paradise
Kilimanjaro Climb
Los Roques Scuba
Maldives Surf
Mt. Shasta Climbing Seminar
Nepal Summit Challenge
New River West Virginia
Pemba Island Scuba Excursion

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.

Converting to a dynamic web page

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, since Compass Travel has built a database that contains a list of trips, you can build a more accurate and timely solution for the trip coordinator. To accomplish this, you must understand how to issue a SQL SELECT statement to retrieve the data from the Trips table in the Compass Travel database.

Understanding basic SQL SELECT statements

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 to hold information about people with the following rows:

LastName	FirstName	Address	City
Jones	Tom	12 State St	Boston
Adams	Anita	521 Beacon St	Boston
Green	Peter	1 Broadway	New York

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

```
SELECT LastName, FirstName FROM Persons
```

The results of this SQL statement contains the following data:

LastName	FirstName
Jones	Tom
Adams	Anita
Green	Peter

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 resulting 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 "%" sign to define wildcards (missing letters in the pattern) before and after the pattern.

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

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

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

LastName	FirstName
Jones	Tom
Adams	Anita

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 listed are true. The OR operator displays a row if ANY of the conditions listed are true. An example of a WHERE clause with multiple subclauses follows:

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

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

LastName
Adams

Note: The preceding SQL SELECT examples use single quotation marks around the value. SQL uses single quotation marks around text values. Most database systems will 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 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 book. For more information, consult a SQL reference.

Using SQL with cfquery to dynamically retrieve information

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 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 presented earlier in this lesson. In this example, you use `cfquery` to return all the trip names found in the tripName column within 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 as follows:

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

Displaying the query result using cfoutput

In Chapter 2, 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 pound 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 instance, when you want to display the trip names from the SQL query, you would simply use `#tripName#` within the output code block.

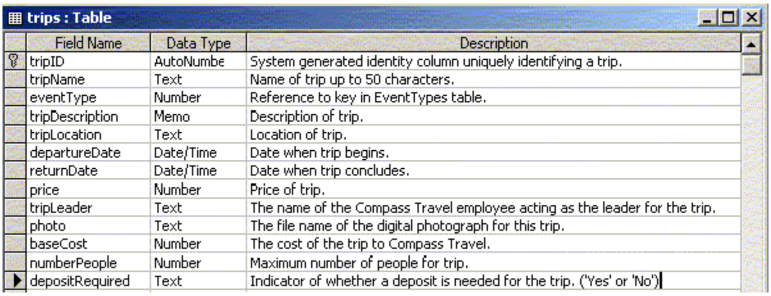
```
<cfoutput query="TripResult">
   #tripname#
</cfoutput>
```

For additional information about using SQL with `cfquery` and `cfoutput`, see *Developing ColdFusion MX Applications with CFML.*

Creating a dynamic web page

In the following exercises you will build a dynamic Trip Listing web page that is always current. The first exercise guides you through constructing a query to retrieve information from the database. In the second exercise, you will enhance the query to sort the query results and to display other pertinent trip information.

For your convenience, the following figure shows the Compass Travel Trips table. You can refer to this table to verify the names of the columns you use in the queries in the exercises.



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')

Exercise: building a query using SQL, cfquery, and cfoutput

Follow these steps to build a query that lists the current trips from the Compass Travel database.

To build the query:

1 Open an editor and create a new ColdFusion page (.cfm).

2 At the top of the file, enter the following code to dynamically retrieve the names of the current trips listed in the Compass Travel database:

```
<cfquery name="TripResult" datasource="compasstravel">
     SELECT tripName FROM trips
</cfquery>
<html>
   <head>
     <title>Trip Listing</TITLE>
   </head>
   <body>
     <h1>Trip List</h1>
     <cfoutput query="TripResult">#tripName#<BR></cfoutput>
   </body>
</html>
```

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

4 View the triplisting.cfm page in a browser. The page lists all the trip names retrieved from Compass Travel database.

Reviewing the code

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

Code	Explanation
`<cfquery name="TripResult" datasource="CompassTravel">`	ColdFusion query named "TripResult". Submits any SQL statement between the cfquery start and end tags to the data source specified in the datasource attribute.
`SELECT tripName FROM trips`	SQL SELECT statement to retrieve all tripName(s) from the trips table.
`<cfoutput query="TripResult">#tripName#<BR></cfoutput>`	Output code block. Displays the value of the column tripName for each row in the result set from the "TripResult" query.

Exercise: enhancing the query

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 alphabetic 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.

To enhance the trip listing query to meet these new requirements, you will modify the query you created in the previous exercise.

Follow these steps to enhance the query to meet the new requirements. Display the triplisting.cfm page in the browser after each step to ensure the corresponding requirement was met.

To enhance the query results:

1 To sort the trip names in alphabetical order in the triplisting.cfm page, modify the SQL SELECT statement within the `cfquery` tags as follows:

```
SELECT tripName FROM trips ORDER BY tripName
```

2 To display the departure, return date, and price for each trip, modify the same SQL statement.

a Modify the SQL SELECT statement, as follows:

```
SELECT tripName, departureDate, returnDate, price
FROM trips
ORDER BY tripName
```

b Change the output block (the code immediately preceding the `</cfoutput>` tag) from just `#tripName#` to include all three selected fields, as follows:

```
#tripName# departs: #departureDate# returns: #returnDate# price:
#price#<BR>
```

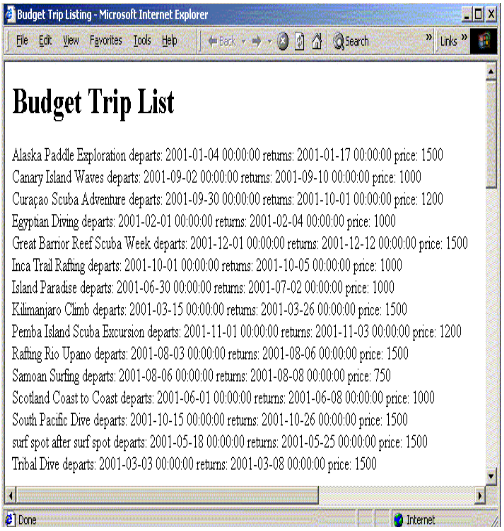
3 Create the Budget Trip List report by doing the following:

   a Modify the SQL SELECT statement, as follows:

```
SELECT tripName, departureDate, returnDate, price
FROM trips
WHERE price <= 1500
ORDER BY tripName
```

   b Change the heading tag from `<h1>Trip List</h1>` to `<h1>Budget Trip List</h1>`.

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


Budget Trip Listing - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Links
Done
Internet

Budget Trip List

```
Alaska Paddle Exploration departs: 2001-01-04 00:00:00 returns: 2001-01-17 00:00:00 price: 1500
Canary Island Waves departs: 2001-09-02 00:00:00 returns: 2001-09-10 00:00:00 price: 1000
Curaçao Scuba Adventure departs: 2001-09-30 00:00:00 returns: 2001-10-01 00:00:00 price: 1200
Egyptian Diving departs: 2001-02-01 00:00:00 returns: 2001-02-04 00:00:00 price: 1000
Great Barrior Reef Scuba Week departs: 2001-12-01 00:00:00 returns: 2001-12-12 00:00:00 price: 1500
Inca Trail Rafting departs: 2001-10-01 00:00:00 returns: 2001-10-05 00:00:00 price: 1000
Island Paradise departs: 2001-06-30 00:00:00 returns: 2001-07-02 00:00:00 price: 1000
Kilimanjaro Climb departs: 2001-03-15 00:00:00 returns: 2001-03-26 00:00:00 price: 1500
Pemba Island Scuba Excursion departs: 2001-11-01 00:00:00 returns: 2001-11-03 00:00:00 price: 1200
Rafting Rio Upano departs: 2001-08-03 00:00:00 returns: 2001-08-06 00:00:00 price: 1500
Samoan Surfing departs: 2001-08-06 00:00:00 returns: 2001-08-08 00:00:00 price: 750
Scotland Coast to Coast departs: 2001-06-01 00:00:00 returns: 2001-06-08 00:00:00 price: 1000
South Pacific Dive departs: 2001-10-15 00:00:00 returns: 2001-10-26 00:00:00 price: 1500
surf spot after surf spot departs: 2001-05-18 00:00:00 returns: 2001-05-25 00:00:00 price: 1500
Tribal Dive departs: 2001-03-03 00:00:00 returns: 2001-03-08 00:00:00 price: 1500
```

Note that the dates and prices in the preceding listing are not formatted. In Lesson 3 you will enhance the look of this page.

Developing a search capability

The dynamic listings developed in the previous exercise 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 results of the user's criteria selection are then posted to a search results page. The logic contained within the search results page builds the SQL SELECT statement contained in a `cfquery` tag using ColdFusion string manipulation. Finally, the action page displays the result using the `cfoutput` tag. This approach of building and executing SQL statements on the fly is called **dynamic SQL**.

Dynamic SQL

Dynamic SQL is a term used to refer to SQL code 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.

Designing the search criteria page

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

- List the trips located in Hawaii.
- Identify the trips with a price greater than $3,000.
- Show the trips departing after 11/11/2002 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. Two of the most important considerations are as follows:

- 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 lesson, the Compass Travel trip coordinator will search the trips based on tripLocation, departureDate, and price. These queryable columns, therefore, will be the only ones contained in the WHERE clause of the generated SQL Statement. Further, the coordinator will have no control over which columns are returned in the record set. The query will always return 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 decided on the queryable columns (tripLocation, departureDate, and price), you can build a simple form that allows the user to 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'
```

But, what if 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. When designing the Search Criteria page, you must decide which operators to support for each of the queryable columns. The operators that you use depends on the data type of the SQL column.

For example, 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

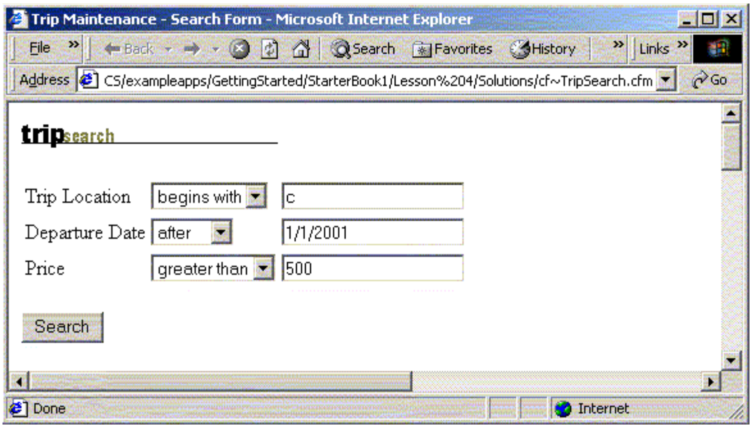
Unlike trip location, it is not semantically correct to consider whether a price begins with "B". Typical SQL string operators are *equals*, *starts with*, *contains*, and *ends with*.

While many more operators are permissible, for simplification, you can use the following the operators for the Compass Travel queryable columns:

Queryable column	Query operators
tripLocation	is, begins with
departureDate	is, before, after
price	is, greater than, less than

Using SQL operators to create a search criteria page

A simple design for a search criteria page presents an operator list and data entry field for each of the queryable columns. Following this pattern, a page to collect the Compass Travel Trip search criteria looks like this:


Trip Maintenance - Search Form - Microsoft Internet Explorer
File
Back
Search
Favorites
History
Links
Address
C:/exampleapps/GettingStarted/StarterBook1/Lesson%204/Solutions/cf~TripSearch.cfm
Go
tripsearch
Done
Internet

Trip Location begins with c

Departure Date after 1/1/2001

Price greater than 500

Search

Since all the code used to produce the search criteria page is HTML, you are not requested to build this page. You will, however, use this page (tripsearch.cfm) later in this lesson to test the search action page. The source code for the Trip Search form (tripsearch.cfm) is as follows:

```
<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="EQUAL">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>
```

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">`	Input text control to capture value to test. There is one text control for each queryable column.

Building the Search Results page

Based on the queryable columns identified earlier, 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 then 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 can be generated depending on the criteria set on the search page:

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

In the previous 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 search action page uses a SQL SELECT statement to display an HTML table with the results of the user query using the `cfoutput` block.

Building the WHERE Clause with the cfif and cfset

The WHERE clause in a SQL SELECT 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 new 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 = "Dylan">
<cfset LastName = "Smith">
<cfset FullName = FirstName & " " & LastName>
<cfoutput>My name is #FullName#.</cfoutput>
```

results in the following text:

```
My name is Dylan Smith.
```

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 using 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.

Constructing the initial Trip Search Results page

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.

As mentioned previously, the SQL SELECT statement uses quotation marks to surround string variable values. Unfortunately, embedded quotation marks can cause problems when posting data to a web server. Normally, ColdFusion adds an escape character to a string that contains a quotation mark so that an error is not generated from the web server. The `PreserveSingleQuotes` function prevents ColdFusion from automatically escaping single quotation marks contained in the variable string passed to the function.

```
<!--- Create Where clause for query from data entered thru 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>
<!--- Query returning search results --->
      <cfquery name="TripResult" datasource="compasstravel">
         SELECT tripName,  tripLocation, departureDate, returnDate, price, tripID
         FROM trips
         WHERE #PreserveSingleQuotes(WhereClause)#
      </cfquery>
<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
```

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.
`<cfif Form.tripLocationValue GT "">`	`The cfif` tag tests to see if user entered anything in the Value input box for tripLocation criterion.

`SELECT tripName,  tripLocation,`
`departureDate, returnDate, price, tripID`
`FROM trips`
`WHERE #PreserveSingleQuotes(WhereClause)#`
	SQL query to execute.
	`PreserveSingleQuotes` ColdFusion function ensures that quotation marks will passed to the database server as intended.

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

Completing the Trip Search Results page

In the following exercises you will test and modify tripsearchresults.cfm. In the first exercise, you will test the Trip Search Results page by entering criteria on the Trip Search form and inspecting the results. In the second exercise, you will finish the code to construct the complete WHERE clause for all three queryable columns from the Trip Search form.

Exercise: testing the Trip Search Results page

Follow these steps to test the Trip Search Results page:

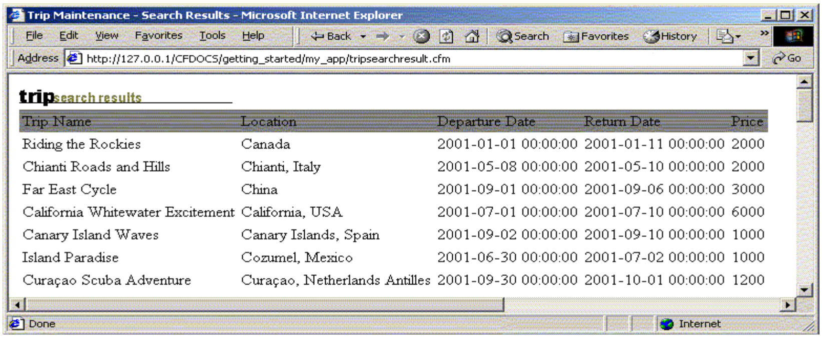
1 Copy the tripsearch.cfm and tripsearchresult.cfm files from the solutions directory to the my_app directory.

2 View the tripsearch.cfm from the my_app directory in your browser and do the following:

  a In the Trip Location drop-down list box 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:



Trip Maintenance - Search Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Go
Done
Internet

http://127.0.0.1/CFDOCS/getting_started/my_app/tripsearchresult.cfm

tripsearch results

Trip Name	Location	Departure Date	Return Date	Price
Riding the Rockies	Canada	2001-01-01 00:00:00	2001-01-11 00:00:00	2000
Chianti Roads and Hills	Chianti, Italy	2001-05-08 00:00:00	2001-05-10 00:00:00	2000
Far East Cycle	China	2001-09-01 00:00:00	2001-09-06 00:00:00	3000
California Whitewater Excitement	California, USA	2001-07-01 00:00:00	2001-07-10 00:00:00	6000
Canary Island Waves	Canary Islands, Spain	2001-09-02 00:00:00	2001-09-10 00:00:00	1000
Island Paradise	Cozumel, Mexico	2001-06-30 00:00:00	2001-07-02 00:00:00	1000
Curaçao Scuba Adventure	Curaçao, Netherlands Antilles	2001-09-30 00:00:00	2001-10-01 00:00:00	1200

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.

3 In the Trip Location drop-down list box of the Trip Search page, select the Is option, enter the value China, then click Search.

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

4 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.

Obviously, either of these conditions would produce a results page with no rows.

c Click the Search button.

The Search Result page should be identical to Step 3 because the code to build the WHERE clause in the Trip Results page does not include departure date and price.

Exercise: 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 In an editor, open the tripsearchresult.cfm page in the my_app directory, then locate the following comment line:

```
<!--- Query returning search results --->
```

2 To build the `departureDate` WHERE subclause, enter the code in the following table immediately following the comment line.

For	Enter this code
Windows users, using the MS Access database file	<!--- Build subclause for departureDate ---> <cfif Form.departureValue GT ""> <cfif Form.departureOperator EQ "EQUALS"> <cfset WhereClause = WhereClause & " and departureDate = " & Form.departureValue> <cfelseif Form.departureOperator EQ "AFTER"> <cfset WhereClause = WhereClause & " and departureDate > " & Form.departureValue> <cfelseif Form.departureOperator EQ "BEFORE"> <cfset WhereClause = WhereClause & " and departureDate < " & Form.departureValue> </cfif> </cfif>
UNIX users, using the 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>

3 To build the `price` WHERE subclause, enter the following code after the code you entered in step 2.

```
<!--- 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>
```

4 Verify that the `price` and `departureDate` are now considered in the query, as in step 4 in the previous exercise:

   a Open the tripsearch.cfm page in the my_app directory in your browser.

   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.

   c Click the Search button.

   Now, 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 then the one presented here. This more extensible approach is beyond the scope of this tutorial, however.

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 built 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 main navigation page for the Trip Maintenance application. This page will display detail information about the currently selected trip. Additionally, it will provide buttons to do trip maintenance and browsing through the trips table. You will also link the trip search facility that you built in this lesson to the main page that you will build in the next.

LESSON 3
Creating a Main Application Page

In this lesson you will enhance the Compass Travel Trip Maintenance application. The exercises in this lesson guide you through the following steps:

- Transforming the search facility that you built in the previous lesson into a drill-down facility for trip queries
- Creating a Trip Detail page to display complete details of a trip
- Implementing a link from Trip Detail page to the search facility
- Converting the Trip Detail page into the main application page by including buttons for navigating to other pages and browsing trip records in the database

This lesson explains how to do the following tasks:

- Use `URLEncodedFormat` to safely link the Search Results page to the Trip Detail page.
- Use the `DateFormat` and `DollarFormat` functions to properly format date and dollar variables.
- Use the `MOD` function and a query variable to properly alternate the back color of rows in the Trip Search Results table.

ColdFusion tags and functions introduced in this lesson

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

Element	Type	Brief description
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.
DollarFormat	Function	Returns a number as a string formatted with two decimal places, thousands separator, and dollar sign. If the specified number is negative, parentheses are used.
DateFormat	Function	Returns a formatted date/time value. If no mask is specified, `DateFormat` returns the date value in the dd-mmm-yy format. `DateFormat` supports dates that have the U.S. date format.
URLEncodedFormat	Function	Converts a text string into a string that can safely be used in a URL.

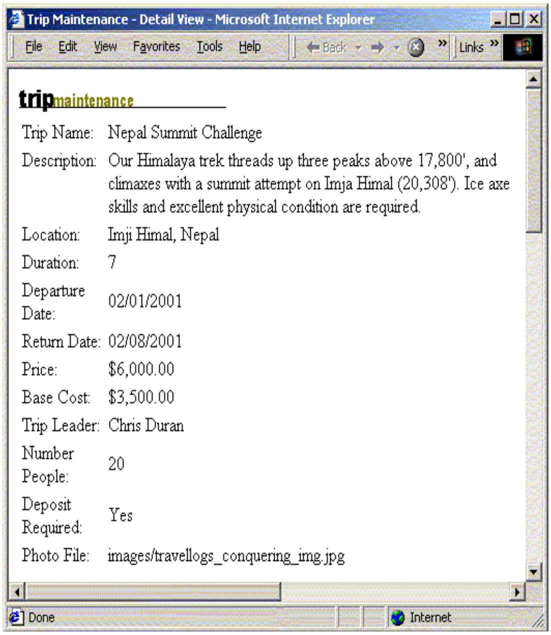
Enhancing the Trip Maintenance application

In this lesson you will enhance the Trip Maintenance application that you created in Lesson 2. You will modify the application to include a main application page that lets Compass Travel employees do these tasks:

- View additional details about a trip.
- Scan records in the Trips database table.
- Modify or search for records in the Trips database table.

The following list identifies the application components that you will create in this lesson:

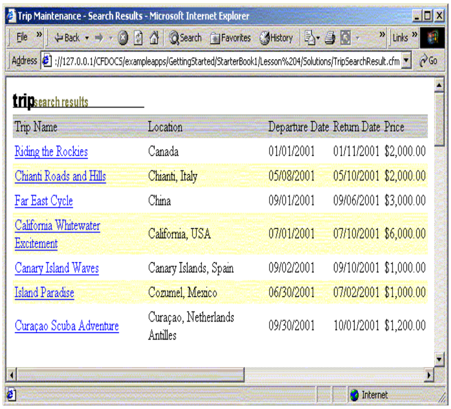
- **Trip Detail Page**. The Trip Detail page displays additional information about a trip that was initially selected from the Trip Search Results page.


Trip Maintenance - Detail View - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Links
Done
Internet

tripmaintenance

Trip Name:	Nepal Summit Challenge
Description:	Our Himalaya trek threads up three peaks above 17,800', and climaxes with a summit attempt on Imja Himal (20,308'). Ice axe skills and excellent physical condition are required.
Location:	Imji Himal, Nepal
Duration:	7
Departure Date:	02/01/2001
Return Date:	02/08/2001
Price:	$6,000.00
Base Cost:	$3,500.00
Trip Leader:	Chris Duran
Number People:	20
Deposit Required:	Yes
Photo File:	images/travellogs_conquering_img.jpg

- **Enhanced Trip Search Results page.** The original purpose of the Trip Search Results page in Lesson 2 was to display the results of a trip search. In this lesson, you will enhance this page to provide a useful drill-down mechanism for accessing additional information about trips that meet the search criteria.



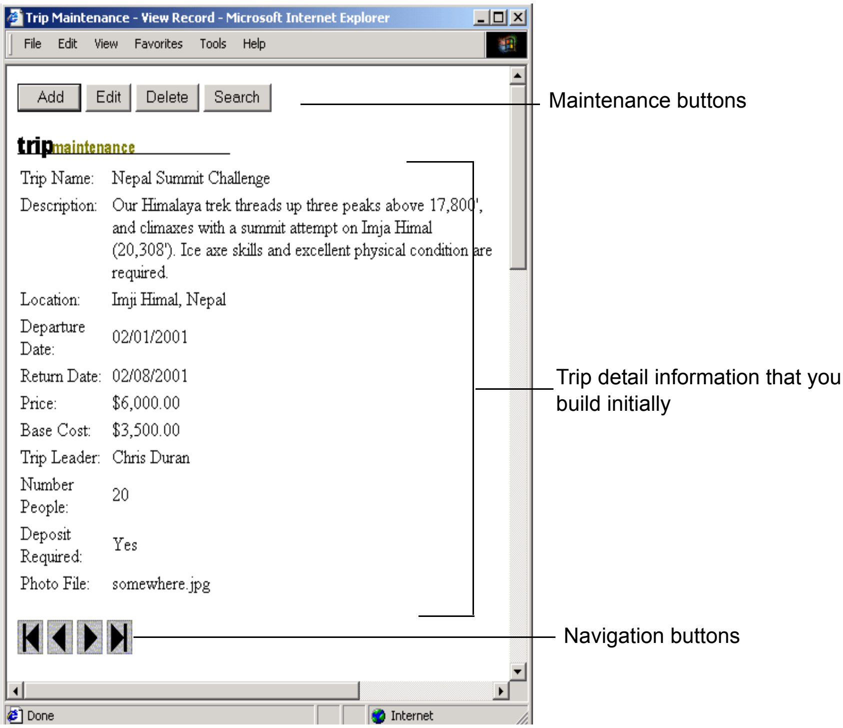
Trip Maintenance - Search Results - Microsoft Internet Explorer
File
Back
Search
Favorites
History
Links
Address
Go
Internet

://127.0.0.1/CFDOCS/exampleapps/GettingStarted/StarterBook1/Lesson%204/Solutions/TripSearchResult.cfm

tripsearch results

Trip Name	Location	Departure Date	Return Date	Price
Riding the Rockies	Canada	01/01/2001	01/11/2001	$2,000.00
Chianti Roads and Hills	Chianti, Italy	05/08/2001	05/10/2001	$2,000.00
Far East Cycle	China	09/01/2001	09/06/2001	$3,000.00
California Whitewater Excitement	California, USA	07/01/2001	07/10/2001	$6,000.00
Canary Island Waves	Canary Islands, Spain	09/02/2001	09/10/2001	$1,000.00
Island Paradise	Cozumel, Mexico	06/30/2001	07/02/2001	$1,000.00
Curaçao Scuba Adventure	Curaçao, Netherlands Antilles	09/30/2001	10/01/2001	$1,200.00

- **Main Application page** In the beginning of this lesson, you will develop the Trip Detail page (see the following figure). Later in this lesson, you will convert the Trip Detail page into the main Trip Maintenance application page. The main application page will include additional buttons for navigating to other ColdFusion pages and browsing the trip database records.



tripmaintenance

Trip Name:	Nepal Summit Challenge
Description:	Our Himalaya trek threads up three peaks above 17,800', and climaxes with a summit attempt on Imja Himal (20,308'). Ice axe skills and excellent physical condition are required.
Location:	Imji Himal, Nepal
Departure Date:	02/01/2001
Return Date:	02/08/2001
Price:	$6,000.00
Base Cost:	$3,500.00
Trip Leader:	Chris Duran
Number People:	20
Deposit Required:	Yes
Photo File:	somewhere.jpg

Maintenance buttons

Trip detail information that you build initially

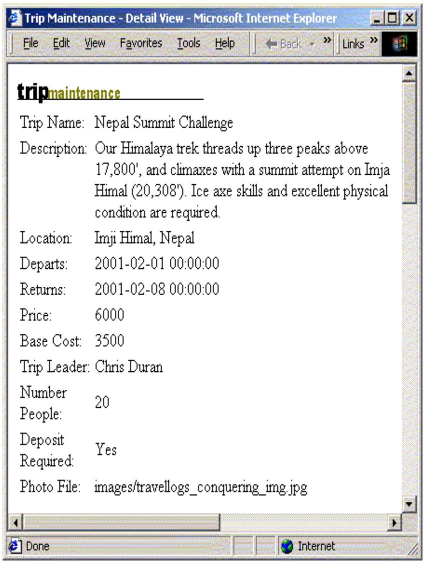
Navigation buttons

The primary users of these components will be the Compass Travel coordinators and agents, not the general public.

Showing additional trip details

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 displayed, the user should be able to click on any row to display the detailed trip data.

In the following 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 will build:


Trip Maintenance - Detail View - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Links
tripmaintenance
Done
Internet

Trip Name: Nepal Summit Challenge

Description: Our Himalaya trek threads up three peaks above 17,800', and climaxes with a summit attempt on Imja Himal (20,308'). Ice axe skills and excellent physical condition are required.

Location: Imji Himal, Nepal

Departs: 2001-02-01 00:00:00

Returns: 2001-02-08 00:00:00

Price: 6000

Base Cost: 3500

Trip Leader: Chris Duran

Number People: 20

Deposit Required: Yes

Photo File: images/travellogs_conquering_img.jpg

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

- You can call the TripDetail 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 Results page. This will offer the user drill-down capability when searching for trips. You will link the Trip 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 the next lesson.

Exercise: building a Trip Detail page

Follow these steps to build a Trip Detail page.

To build a Trip Detail page:

1 Open your editor and create a new ColdFusion page. Remove any lines if your editor adds.

2 To create a query to select a single trip from the Trips table, enter the following code:

```
<cfquery name="TripQuery" dataSource="compasstravel" maxRows=1>
SELECT tripID, tripName, tripDescription, tripLocation, departureDate,
returnDate, price, tripLeader, photo, baseCost, numberPeople, depositRequired
FROM TRIPS
<cfif IsDefined("URL.ID")>
WHERE tripID = #URL.ID#
</cfif>
</cfquery>
```

3 To output the results from the query, append the following `cfoutput` code after the code you added in step 2.

```
<cfoutput query="TripQuery">
   <img src="images/tripmaintenance.gif">
</cfoutput>
```

4 To display the data fields on the Trip Detail page, place the following code above the `</cfoutput>` tag you added in step 3. Alternatively, you can copy this HTML code from the tripdetail.txt file in the solutions directory.

```
<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>
```

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

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

6 Insert the ending body and html tags at the end of the page:

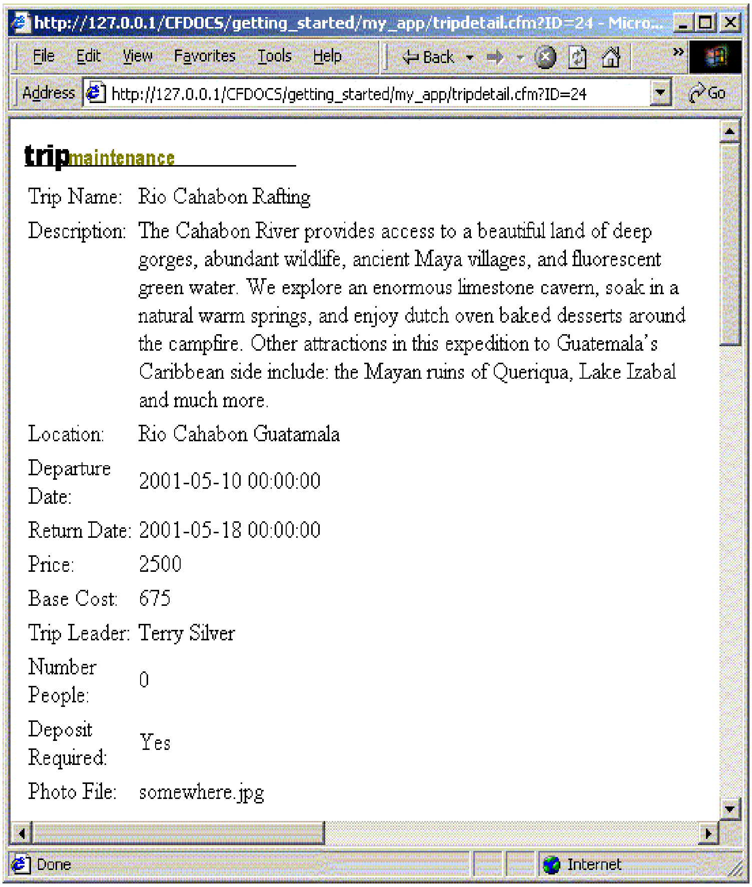
```
</body>
</html>
```

7 Save the file as `tripdetail.cfm` in the my_app directory.

8 The Rio Cahabon Rafting trip has an tripID of 24. To view the trip detail for the trip in your browser enter one of the following URLs::

Web server configuration	URL
For stand-alone ColdFusion web server configuration	http://localhost:8500/cfdocs/getting_started/my_app/tripdetail.cfm
For local third-party web server configuration	http://localhost/cfdocs/getting_started/my_app/tripdetail.cfm

The following page shows the expected result:


http://127.0.0.1/CFDOCS/getting_started/my_app/tripdetail.cfm?ID=24 - Micro...
File Edit View Favorites Tools Help
Back
Address
Go
Done
Internet

http://127.0.0.1/CFDOCS/getting_started/my_app/tripdetail.cfm?ID=24

tripmaintenance

Trip Name:	Rio Cahabon Rafting
Description:	The Cahabon River provides access to a beautiful land of deep gorges, abundant wildlife, ancient Maya villages, and fluorescent green water. We explore an enormous limestone cavern, soak in a natural warm springs, and enjoy dutch oven baked desserts around the campfire. Other attractions in this expedition to Guatemala's Caribbean side include: the Mayan ruins of Queriqua, Lake Izabal and much more.
Location:	Rio Cahabon Guatamala
Departure Date:	2001-05-10 00:00:00
Return Date:	2001-05-18 00:00:00
Price:	2500
Base Cost:	675
Trip Leader:	Terry Silver
Number People:	0
Deposit Required:	Yes
Photo File:	somewhere.jpg

Reviewing the code

The following table describes the ColdFusion code used to build the Trip Detail page:

Code	Explanation
`<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, we want to only show a single row at a time, therefore, `maxRows` is set to 1.

`<cfif IsDefined("URL.ID")>`
`   WHERE tripID = #URL.ID#`
`</cfif>`
	The URL.ID specifies a parameter that can be contained within 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 function IsDefined to determine if a parameter is passed within the URL. It can also be used to determine if the user has entered data in form fields prior to the form post action.

As you can see, 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 2. However, before you link the search facility you built in Lesson 2, you need to understand a potential security risk using dynamic SQL. The following section describes this risk and how to code around it.

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 may 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 supporting 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, 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>
```

Now if non-numeric data is passed within the URL ID field, the `Val` statement returns 0, and the trip with ID 0 displays (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. You must use the `Val` function in your applications to eliminate the risk.

Linking the Search Results page to the Trip Detail page

In the next 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 the trip name entries in the results page to links, which will display the trip's detailed information in the detail page.

Exercise: linking the Search Results page with the Trip 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 pages:

1 Open the tripsearchresult.cfm page from the my_app directory and replace the `#tripName#` in the `cfoutput` block with the following code:

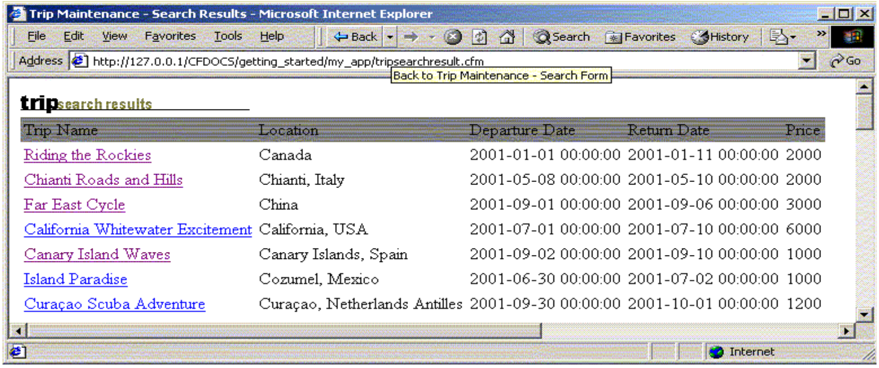
```
<a href="tripdetail.cfm?ID=#URLEncodedFormat(tripID)#">
#tripName# </a>
```

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.

2 Save the file and view the tripsearch.cfm page from the my_app directory in your browser.

3 In the Trip Location drop-down list box, select Begins With and type the value C in the trip location text box then click Search.

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


Trip Maintenance - Search Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
History
Address
http://127.0.0.1/CFDOCS/getting_started/my_app/tripsearchresult.cfm
Go
Back to Trip Maintenance - Search Form
Internet

tripsearch results

Trip Name	Location	Departure Date	Return Date	Price
Riding the Rockies	Canada	2001-01-01 00:00:00	2001-01-11 00:00:00	2000
Chianti Roads and Hills	Chianti, Italy	2001-05-08 00:00:00	2001-05-10 00:00:00	2000
Far East Cycle	China	2001-09-01 00:00:00	2001-09-06 00:00:00	3000
California Whitewater Excitement	California, USA	2001-07-01 00:00:00	2001-07-10 00:00:00	6000
Canary Island Waves	Canary Islands, Spain	2001-09-02 00:00:00	2001-09-10 00:00:00	1000
Island Paradise	Cozumel, Mexico	2001-06-30 00:00:00	2001-07-02 00:00:00	1000
Curaçao Scuba Adventure	Curaçao, Netherlands Antilles	2001-09-30 00:00:00	2001-10-01 00:00:00	1200

4 To view the Trip Detail page for a trip, click on 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 the next exercise.

Enhancing the look of the search results and detail pages

The Trip Maintenance search now provides a useful drill-down mechanism for locating trips. While this application is functionally sound, the appearance of the dates and dollar amounts could be improved.

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

Another part of the application that could be improved is the Trip Search Results table. In a large list, it is sometimes difficult to correctly read the contents of a row in the middle of the table because it is sandwiched between two other rows. To avoid mistakes in reading the table, it would be helpful to alternate the background color of each row in the table.

The HTML table row `<tr>` tag has a `bgcolor` attribute to change the background color for a given row. To highlight every other row, the background color could be changed alternatively to yellow or white. To change the background color for each row, you must determine whether the row is an odd or even row. Therefore, you must obtain the sequence number of the current row and test if it is evenly divisible by 2.

As described in Chapter 2, ColdFusion offers a modulus function (`MOD`) that returns the remainder (modulus) after a number is divided by a divisor; for example,`10 MOD 3` is 1.

If the `MOD` function returns a value greater than 0, a `cfif` test of its result evaluates to `True`. If the `MOD` function returns 0, the `cfif` check fails.

The following example uses the MOD function to alternate the background color of table rows:

```
<cfoutput query="tripResult">
<cfif CurrentRow Mod 2>
   <cfset BackColor="White">
<cfelse>
   <cfset BackColor="Yellow">
</cfif>
<tr bgcolor= "#BackColor#">
```

Notice that the MOD function uses a variable called CurrentRow. Notice also that CurrentRow is not defined anywhere in tripsearchresult.cfm. The CurrentRow variable is one of a few variables that ColdFusion automatically exposes that provide information about the query. These variables are often called **query variables**.

Query variables

In addition to using the cfquery tag to return data from a ColdFusion data source, you also can use it to return information about a query. To do this, the cfquery tag creates a query object, providing information in query variables as described in the following table:

Variable Name	Description
query_name.recordCount	The number of records returned by the query.
query_name.currentRow	The current row of the query being processed by cfoutput.
query_name.columnList	A comma-delimited list of the query columns.

When a query variable is referenced within a cfoutput block, the qualifying query_name is assumed to be the query identified in the QUERY attribute of the cfoutput tag and does not need the qualifier query_name. That is why the CurrentRow variable is unqualified in the previous modulus code example.

For more information about using the modulus function or query variables in ColdFusion applications, see *Developing ColdFusion MX Applications with CFML*.

Exercise: formatting the display

In this exercise, you format the currency and date fields in the Trip Search Results page and the Trip Detail page. Additionally, you modify the Trip Search Results page to alternate the background color of the result table rows.

To format the table:

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

Existing code	Change to
#departureDate#	#dateformat(departureDate, "mm/dd/yyyy")#
#returnDate#	#dateformat(returnDate, "mm/dd/yyyy")#

Existing code	Change to
#price#	#dollarformat(price)#
#baseCost#	#dollarformat(baseCost)#

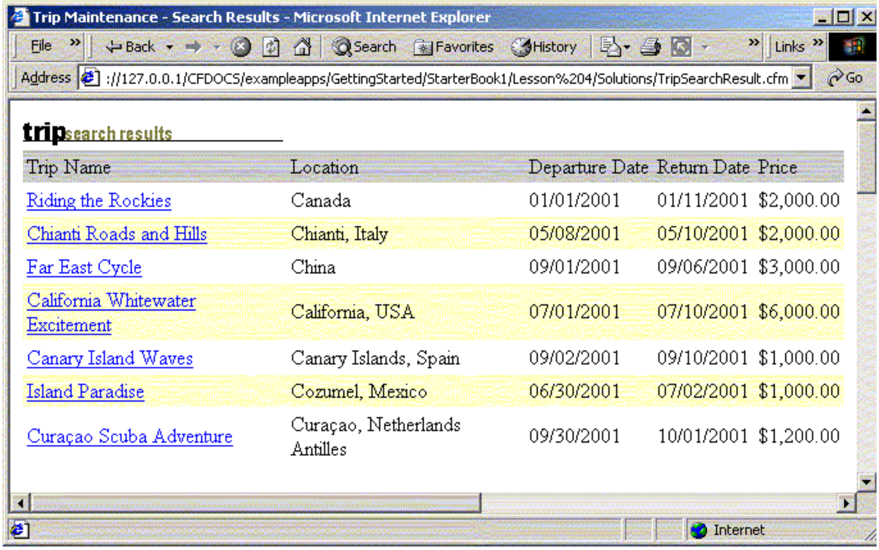
2 To format the currency and date fields on the Trips Search Results page, open the tripsearchresult.cfm in your editor and make the same changes for `departureDate`, `returnDate`, and `price` as in step 1.

3 To alternate the background color of the rows of the search results table, delete the table row tag `<tr>` that immediately follows the `<cfoutput query="TripResult">`, and replace it with the following code:

```
<cfif CurrentRow Mod 2>
   <cfset BackColor="White">
<cfelse>
   <cfset BackColor="Yellow">
</cfif>
<TR bgcolor= #BackColor#>
```

4 Save the files then open your browser and navigate to the tripsearch.cfm page in the my_app directory. Again, enter Begins With and C in the location search criteria, and click `Search`.

The Trip Search Results page appears:


Trip Maintenance - Search Results - Microsoft Internet Explorer
File
Back
Search
Favorites
History
Links
Address
Go
Internet

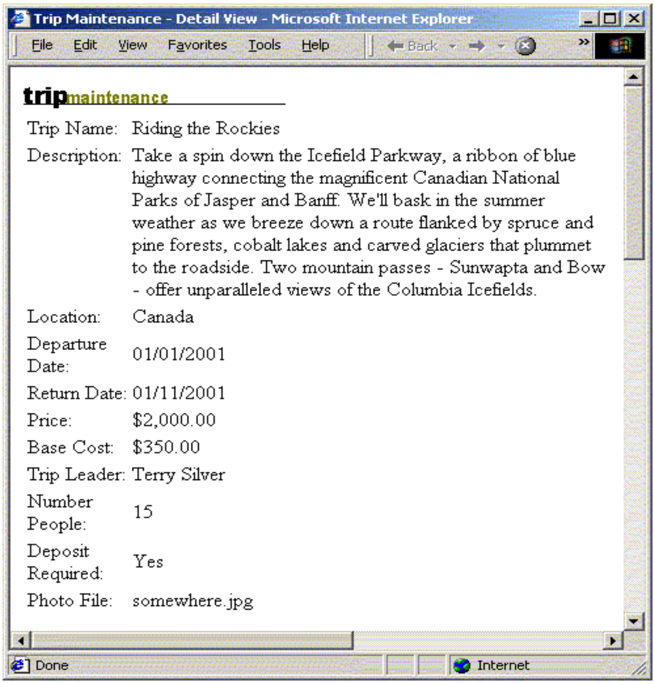
://127.0.0.1/CFDOCS/exampleapps/GettingStarted/StarterBook1/Lesson%204/Solutions/TripSearchResult.cfm

tripsearch results

Trip Name	Location	Departure Date	Return Date	Price
Riding the Rockies	Canada	01/01/2001	01/11/2001	$2,000.00
Chianti Roads and Hills	Chianti, Italy	05/08/2001	05/10/2001	$2,000.00
Far East Cycle	China	09/01/2001	09/06/2001	$3,000.00
California Whitewater Excitement	California, USA	07/01/2001	07/10/2001	$6,000.00
Canary Island Waves	Canary Islands, Spain	09/02/2001	09/10/2001	$1,000.00
Island Paradise	Cozumel, Mexico	06/30/2001	07/02/2001	$1,000.00
Curaçao Scuba Adventure	Curaçao, Netherlands Antilles	09/30/2001	10/01/2001	$1,200.00

5 In the Trip Search Result page, click the link for Riding the Rockies.

The properly formatted Trip Detail page appears:



Trip Name: Riding the Rockies

Description: Take a spin down the Icefield Parkway, a ribbon of blue highway connecting the magnificent Canadian National Parks of Jasper and Banff. We'll bask in the summer weather as we breeze down a route flanked by spruce and pine forests, cobalt lakes and carved glaciers that plummet to the roadside. Two mountain passes - Sunwapta and Bow - offer unparalleled views of the Columbia Icefields.

Location: Canada

Departure Date: 01/01/2001

Return Date: 01/11/2001

Price: $2,000.00

Base Cost: $350.00

Trip Leader: Terry Silver

Number People: 15

Deposit Required: Yes

Photo File: somewhere.jpg

Creating the main application page from the Trip Detail page

To this point in the tutorial, 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, however, that you have not addressed: the ability to browse through the Trips table, and the ability to add, delete, and edit trip information.

The trip coordinator must be viewing the details of a specific trip to edit trip information or delete a trip. The Trip Detail page provides this trip-specific detail information. Therefore, you will use the Trip Detail page as the main Trip Maintenance application page.

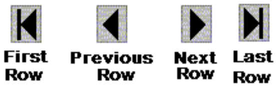
You will modify the Trip Detail page so that it can act as a main switchboard to accomplish this additional functionality. The Trip Detail page shows information about a single trip. You will convert the Trip Detail page into the main application page by adding the following functionality:

- Navigation buttons to browse the database
- Database maintenance buttons to edit, delete, or add new trips
- An additional button to launch the search facility

Adding navigation buttons to browse database

The drill-down search function developed in the last exercise is very useful when the user knows some search criteria to enter. Unfortunately, however, 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 user does not know the criteria to search for in advance.

For example, an anomaly may exist on any trip record, the trip coordinator would have no idea what search criteria to specify on the search form to find these problem records. To solve this problem, the browse function gives the coordinator the ability to navigate sequentially through the trips table using the single record trips display. The following figure shows the navigation buttons. The label under each button describes which row to display relative to the currently displayed row:



Exercise: adding navigation buttons to the Trip Detail page

In this exercise, you use the HTML `form` and `input` tags to add the navigational buttons to the Trips Detail page.

To add navigation:

1 Open the tripdetail.cfm in the my_app directory in your editor.

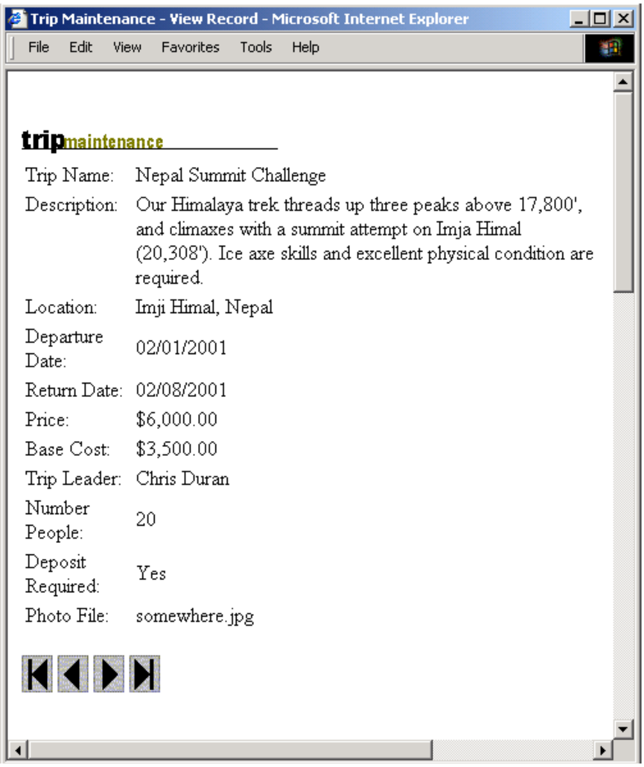
2 To implement the trip navigation buttons, insert the following code between the `</table>` and`</cfoutput>` tags in the tripdetail.cfm file:

```
<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: Notice that the current trip record ID (`tripID`) is hidden within the form code. This is desirable because the action page must have the current record ID in order to build the query that navigates to the appropriate record in the trips database table.

3 Save the file and view the updated tripdetail.cfm page in a browser.

The Trip Search Results page appears:



Trip Maintenance - View Record - Microsoft Internet Explorer
File Edit View Favorites Tools Help
tripmaintenance

Trip Name:	Nepal Summit Challenge
Description:	Our Himalaya trek threads up three peaks above 17,800', and climaxes with a summit attempt on Imja Himal (20,308'). Ice axe skills and excellent physical condition are required.
Location:	Imji Himal, Nepal
Departure Date:	02/01/2001
Return Date:	02/08/2001
Price:	$6,000.00
Base Cost:	$3,500.00
Trip Leader:	Chris Duran
Number People:	20
Deposit Required:	Yes
Photo File:	somewhere.jpg

4 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 the next lesson.

Reviewing the code

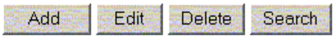
The following table describes the navigation code in the Trip Detail page:

Code	Explanation
`<form action="navigationaction.cfm" method="post">`	Form tag identifying navigationaction.cfm 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 are image type HTML input tags.

Adding database maintenance buttons

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



As described earlier, 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 will use an HTML `input` tag to hide the current `recordID` and post it to the maintenanceaction.cfm page.

Exercise: add Maintenance Buttons to Trip Detail Page

Follow these steps to add the database maintenance buttons to the Trip Detail page.

To add maintenance buttons:

1 In your editor, open the `tripdetail.cfm` from my_app subdirectory.

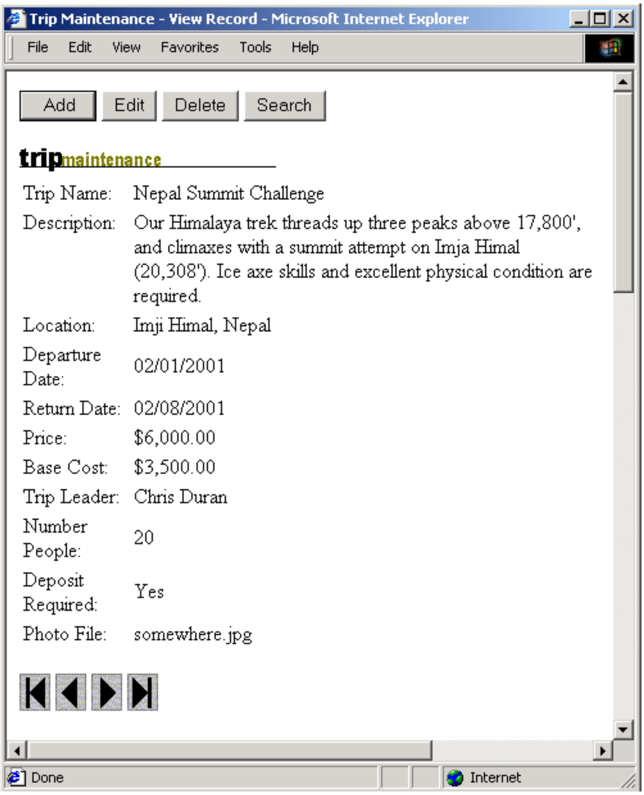
2 Enter the following code immediately after the `<cfoutput query="TripQuery">` tag in the `tripdetail.cfm` file:

```
<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 and view the updated `tripdetail.cfm` page in a browser (http://localhost/CFDOCS/getting_started/my_app/tripdetail.cfm).

The page appears as follows:



Trip Name:	Nepal Summit Challenge
Description:	Our Himalaya trek threads up three peaks above 17,800', and climaxes with a summit attempt on Imja Himal (20,308'). Ice axe skills and excellent physical condition are required.
Location:	Imji Himal, Nepal
Departure Date:	02/01/2001
Return Date:	02/08/2001
Price:	$6,000.00
Base Cost:	$3,500.00
Trip Leader:	Chris Duran
Number People:	20
Deposit Required:	Yes
Photo File:	somewhere.jpg

4 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 the next lesson.

Summary

In this lesson, you transformed the search facility you built in Lesson 2 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 `DollarFormat` and `DateFormat` functions. You linked the Trip Search Results page with the Trip Detail page. You used the `URLEncodedFormat` CFML function to ensure that data was correctly from one page to the other. Finally, by adding trip maintenance and navigation buttons, you converted the Trip Detail page into the main Trip Maintenance application page.

In the next lesson

In the next lesson, you will build the action pages required to implement the navigation and maintenance buttons on the main Trip Maintenance application page.

LESSON 4
Validating Data to Enforce Business Rules

In this lesson, you will enhance the Compass Travel Trip Maintenance application. The exercises in this lesson will guide you through the steps of enhancing the application to provide a page for the trip coordinator to add new trip offerings and update existing trips. Further, you will add logic to validate that data entered against Compass Travel business rules.

This lesson explains how to do the following tasks:

- Create a data entry form to capture user input information
- Develop an action page to process posted form variables
- Validate captured data in three ways

ColdFusion tags and functions introduced in this lesson

The following table identifies 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 `cfform` to place radio buttons, checkboxes, or text boxes. Provides input validation for the specified control type.
`cfselect`	Tag	Used inside `cfform`, `cfselect` lets you construct a drop-down list box form control. You can populate the drop-down list box from a query, or use the option tag. You further 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; otherwise it returns False.

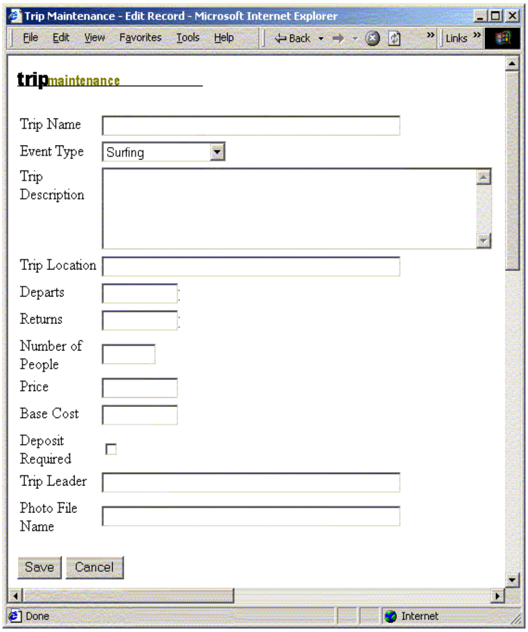
Enhancing the Trip Maintenance application

In this lesson and the next, you will create the code to implement the remaining maintenance buttons on the main Trip Maintenance application page. The remaining buttons are Add and Edit.

You will develop the data entry form to capture new trip information and validate the data entered against Compass Travel business rules. You will then modify the data entry form to edit existing trips. In the next lesson, you will continue to build on this application by adding logic to insert and update the data to the Compass Travel database.

The following list identifies the trip edit components that you will create in this lesson.

- **Trip Edit page** You will create the Trip Edit page (see the following figure) bto add new trips and edit existing trips. The page will be launched from the Add and Edit buttons on the main Trip Maintenance application page (tripdetail.cfm). 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 3.



Trip Name

Event Type Surfing

Trip Description

Trip Location

Departs

Returns

Number of People

Price

Base Cost

Deposit Required □

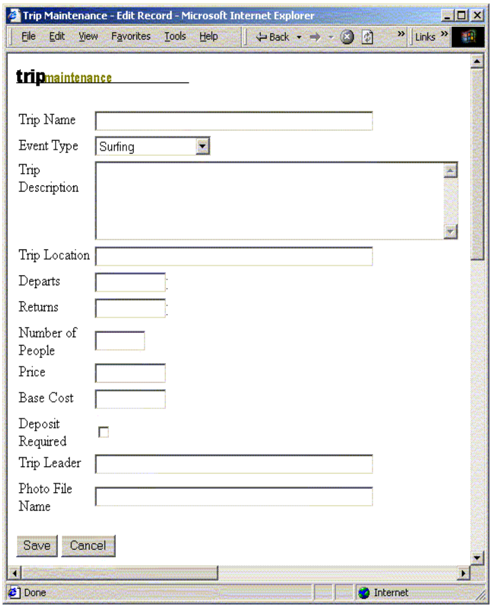
Trip Leader

Photo File Name

- **Trip Edit Action page** You will develop an action page that will insert or update trip data passed from the Trip Edit page into the trips table of the Compass Travel database. In this lesson you will only add the logic to validate the data entered on the Trip Edit page.

Using an HTML form to collect data

Based on the data requirements determined in Lesson 1, the following figure shows the Trip Edit data collection page:



Event Type Surfing

Exercise: view the source and test the Trip Edit page

To view the source and test the Trip Edit data collection form:

1 Open an editor, then locate and open the file tripedit1.cfm in the solutions directory `\cfdocs\getting_started\solutions` under your web root directory.

2 Review the HTML source code used to create the Trip Edit page. If you are not fluent in HTML, 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.

Tag	Description
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 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.

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

4 View the tripedit.cfm in a browser and test the form by entering a trip name in the Trip Name field then clicking Save. An error occurs.

5 View the form source (tripedit.cfm) in an editor to try to determine the cause of the error. Notice that the `<form>` tag on line 6 of the source code has an `action` attribute. This attribute indicates the page to receive the form values posted by the tripedit.cfm page. Since the page, tripeditaction.cfm, does not exist yet, the ColdFusion Server sends an error.

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

Developing code to validate data and enforce business rules

As described in Lesson 1, it is important to define the right data type for each column on the tables in the database. A fundamental concern, therefore, 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 assure 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 requiring single 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.	single-field
	All trip dates must be valid future dates. Departure date must precede return date.	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 within the images directory of the Compass Travel website.	single-field

Ways to validate data

ColdFusion provides special tags to simply 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. Other validation edits, you can enforce on the server after the data entry form is submitted. You will explore these options in the following sections.

Validating data using a server-side action page

The first approach you will take to enforce Compass Travel business rules is to develop an 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 policy.

Using a cfif tag to enforce business rules

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

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

In the previous 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 "Trip name cannot be blank" to the browser.

Note: The keyword `EQ` is an operator used to test for equality. For more information about the `cfif` tag and its operators, see *Developing ColdFusion MX Applications with CFML*.

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. Therefore, an error occurs if the action page tries to use a variable that was not been passed.

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

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

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. No indicates a deposit is not required; Yes indicates a deposit is required.

Evaluating whether business rules fail

The purpose of the tripeditaction.cfm action page is to update the Compass Travel database, so it is important to make certain that all the business rules are passed successfully before the database insert is executed. Failure of any one of the rules negates the insert.

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 to see if 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 the previous example, cfset 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.

For more information about using the cfset and cfif tags and the IsDefined function, see *Developing ColdFusion MX Applications with CFML* or the *CFML Reference*.

Exercise: create an action page with server-side validation

In this exercise you build an action page (tripeditaction.cfm)to validate the data passed to the ColdFusion Server from the tripedit.cfm data entry page. You use the cfif and cfset tags to build edits that ensure the data passed is valid per the Compass Travel business rules. Additionally, you will use the ColdFusion IsDefined function to check to see if data was entered in the data entry form (tripedit.cfm).

To build trip edit action page and validate data passed:

1 Open an editor and create a new page called tripeditaction.cfm in the my_app directory. The new page appears as follows:

```
<html>
<head>
   <title>Untitled</title>
</head>

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

2 To ensure that Compass Travel business rule 7 is met, insert the following code above the <html> tag on the tripeditaction.cfm page. For your convenience, business rule 7 is repeated.

Business rule 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><br>
    </cfif>
</cfif>
```

Note: The code for business rule 7 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 Developing ColdFusion MX Applications with CFML.

3 Save the page.

4 Use the following steps to test the code to see if it meets the objective of business rule 7:

a View the tripedit.cfm page in the browser.

b In the form, enter the number 500 in both the Price and Base cost fields.

c Click the Save button.

The trip price error message displays: "Price must be marked up at least 20% above cost."

d Click the browser Back button to return to the tripedit.cfm page.

e To avoid the error, enter the number 800 in the Price field and click Save.

5 Complete all the business rules using server-side validation. You must insert the code for each business rule above the `<html>` tag. As an example, see the tripeditaction1.cfm page in the solutions directory.

Tip: You can either modify your new tripeditaction.cfm page to include the code necessary to meet all 10 business rules or you can copy the tripeditaction1.cfm page from the solutions directory to tripeditaction.cfm in the my_app directory.

6 Test various combinations to make sure 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 in a non-numeric 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 file name 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.

Drawbacks of validating data on the server-side

Validating data on the server-side has two drawbacks. First, since the action page is used for validation, the form page is not in the browser context when the error is trapped. The user will, therefore, not get immediate feedback from the page where the data was entered. Second, because data capture occurs on the client and validation 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, then only valid data is posted to the server and traffic is reduced.

Validating data on the client using ColdFusion form tags

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 that provide advantages of client-side data validation.These data input tags include `cfinput text`, `cfinput radio`, `cfinput checkbox`, `cfselect`, and others.

Among the improvements over standard HTML tags, ColdFusion form tags offer 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 displayed if validation fails.
range	The range of permissible values for this tag.
required	An indicator of whether data is required for the corresponding tag.

Examples of using the improved ColdFusion form tags

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 ColdFusion form tag)

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

```
<input size=3 name=duration>
Code on the server (tripeditaction.cfm page):
<!--- Duration 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 ColdFusion form tag

The following code is on the client:

```
<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`. 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.

Exercise: modify Trip Edit page to exploit ColdFusion form tags

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

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

1 Open the tripedit.cfm in the my_app directory in your editor.

2 Locate and change the `<form>` and `</form>` tags to `<cfform>` and `</cfform>`, respectively.

3 Change the <input> tags to `<cfinput>` tags and `<select>` tags to `<cfselect>` tags. Note that 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 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 requires 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 be displayed if validation fails. The message reflects the text that describes the business rule.

For example, the Trip Name field requires the following code:

```
<cfinput maxlength = "50" size = "50" required = "Yes" name= "tripName"
      message = "Trip name must not be blank">
```

Tip: For additional help, review the completed code in the tripedit2.cfm within the solutions directory. For more details about using ColdFusion form tags and their attributes, see *Developing ColdFusion MX Applications with CFML*.

5 In your editor, open the tripeditaction.cfm in the my_app directory and delete the code for the following single-field validation rule:

- Trip name is required.
- Trip leader is required.
- Photo file name 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.

Tip: You can either remove the single-field validations yourself or use tripeditaction2.cfm file in the solutions directory. The file tripeditaction2.cfm in the solutions directory is a copy of tripeditaction with the single-field edits deleted. Copy tripeditaction2.cfm in the solutions directory to tripeditaction.cfm in the my_app directory.

6 When you finish deleting the single-field validation rules, save the file.

The modified tripeditaction.cfm page appears as follows:

```
<!--- Action Page to edit and save Trip information for Compass Travel. --->
<!--- Single field edits have been removed in favor of client-side edits. --->
<!--- Make the passportRequired variable be No if it is not set
    (check box is empty) --->
<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>
```

7 View the tripedit.cfm page in a browser and 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 like **12/32/2002**.
- Enter a non-numeric value in number of people such as **one**.

Using cfselect tag to present valid event types

Currently the event types in tripedit.cfm are hard coded:

```
<!--- Field: eventType --->
<tr>
   <td valign="top">Type of Event
   </td>
   <td>
<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>
   </td>
</tr>
```

As described in Lesson 1, the tutorial application design includes a database table that holds event types. The event type in the Trips table is an identifier used as a foreign key to the eventtypes table (which holds the actual event names). In the previous code, each `option` tag contains a `value` attribute and option text, such as `Surfing`. These values come from the eventtypes table so that they are not hard-coded. The eventtypes table column `eventTypeID` is used for the value attribute and the `eventType` for the literal value that is displayed 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 option tags, you can replace the HTML `select` tag with `cfselect`.

The 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:

```
<!--- Field: eventType --->
  <tr>
    <td valign="top">Type of Event
    </td>
    <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>
  </tr>
```

Exercise: use eventtypes table to load event types

Do the following steps to modify the Trip Edit page to display a list of event types from the eventtypes table and add validation.

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 in the my_app directory in your editor.

3 Add the following code above the <html> tag:

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

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

Using other client-side script to reduce edits on the server

If you were interested in moving as much of the business rule logic to the client as possible, you might use other client-side scripting languages, such as JavaScript. By exploiting ColdFusion form tags, you moved most of the responsibility for the business rule checking from the server to the client. This section explains how to migrate cross-field business rules to the client using JavaScript.

Web browsers can execute scripts based on events triggered on the current page. One of the most popular scripting languages is JavaScript. ColdFusion Form tags include an `onValidate` attribute that lets you specify your own JavaScript function for custom validation.

The JavaScript form object, input object, and input object value are passed to the specified JavaScript function. The function returns True if validation succeeds and False otherwise. The `onValidate` and `validate` attributes are mutually exclusive.

Recall the Compass Travel New Trip business rule 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.

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.

One reason this rule is a good candidate for a JavaScript function is that the test for a future date cannot be done using the ColdFusion form tags attributes such as `validate` and `range`. The following JavaScript function (`isitFutureDate`) tests whether a date is a valid future date.

```
function isitFutureDate(oForm, oTag, dateString) {
/*
   function isitFutureDate
   parameters: oForm, oTag, dateString
   returns: boolean

   oForm is the CFForm object.  All onvalidate calls pass this argument.
          This function ignores it.
   oTag is the CFForm current tag object.  All onvalidate calls pass this
          argument.  This function ignores it.
   dateString is the value of the current tag object. It should be a date passed
          as a string in the following
   format: MM/DD/YYYY. This means that months and days require leading zeros!!

   Returns true if the date passed is greater than today's date
   Returns false if the date passed is NOT greater than todays
   date.
*/

   // Check to make sure the date is zero filled with 4 digit year and
   //therefore 10 characters long.
   if (dateString.length != 10)
      return false;
var now = new Date();
   var today = new Date(now.getYear(),now.getMonth(),now.getDate());
var testdate = new Date(dateString.substring(6,10),
          dateString.substring(0,2)-1,
          dateString.substring(3,5));
    if (testdate > now)
      return true;
    else
      return false;
}
```

Another reason that rule 6 requires JavaScript scripting is that it tests the values of more than one field in a single edit. You must ensure that the return date field is greater than departure date field. To do this, you add a JavaScript function to validate the trip date range entered, and specify the function on the `onValidate` attribute of the `returnDate` `cfinput` tag.

```
function validateTripDateRange(oForm, oTag, dateString)
{
  /*
  parameters: oForm, oTag, dateString
  returns: boolean

  oForm is the CFForm object.  All onvalidate calls pass this argument.
    This function ignores it.
  oTag is the CFForm current tag object.  All onvalidate calls pass this argument.
    This function ignores it.
  dateString is the value of the current tag object. It should be a date
    passed as a string in the following
  format: MM/DD/YYYY. This means that months and days require leading zeros!!

  Returns true if the date passed is a future date greater than the departure date
  Returns false if the date passed is NOT a future date greater than departure
    date.
*/

//Edit to make sure that Return date is Later than departure date.
  var returnDateString;

  //First check to see if the departure Date is a valid future date
  if (isitFutureDate(oForm, oTag, dateString) == false)
    return false;

  var departureDate = new Date(dateString.substring(6,10),
        dateString.substring(0,2)-1,
        dateString.substring(3,5));

  returnDateString = document.forms(0).item("returnDate").value;

  var returnDate = new Date(returnDateString.substring(6,10),
        returnDateString.substring(0,2)-1,
        returnDateString.substring(3,5));

  if (departureDate < returnDate)
    return true;
  else
    return false;
}
```

The important point about the preceding JavaScript is that you can use two functions, `isitFutureDate` and `validateTripDateRange`, to verify whether a date is in the future and the trip date range is valid, respectively.

Exercise: add JavaScript-based validation code

In this exercise you will modify the Trip Insert page to validate the departure and return dates using the JavaScript functions provided.

To validate the departure and return dates using JavaScript functions:

1 Open tripedit.cfm in your editor and do one of the following:

Copy example code provided Copy the tripsedit3.cfm file from the solutions directory and rename it to tripedit.cfm in the my_app subdirectory

or

Add JavaScript-based validation code to tripedit.cfm Follow these steps to modify the tripedit.cfm page:

a Copy and insert the text from the scriptvalidation.txt in the solutions directory right before the HTML `body` tag in tripedit.cfm.

b Modify the `departureDate` and `returnDate` input tags to include the `onValidate` attributes as follows:

```
<cfinput name="departureDate" size="10" validate="date"
   onvalidate="isitFutureDate" message="Departure date must be a valid
   future date (mm/dd/yyyy).">
<cfinput size="10" name="returnDate" validate="date"
   onvalidate="validateTripDateRange" message="Return date must be a valid
   date greater than departure date (mm/dd/yyyy).">
```

c Save the tripedit.cfm in the my_app directory.

2 Test this page by opening the tripedit.cfm page in your browser.

3 Test the date validation by checking that each of the following tasks fail:

a Enter a date in the past for the departure date field; for example, **01/01/2000**.

b Enter a departure date greater than the return date; for example, enter **02/01/2004** for the departure date, and **01/01/2004** for the return date.

4 Test the date validation and ensure that the proper data succeeds; for example, enter **01/01/2004** for the departure date, and **01/14/2004** for the return date.

5 Test for an invalid date by entering **12/32/2002** for the return date.

You would expect the application to reject this date. It does not. This is because the `validate` attribute of a `cfinput` tag (returnDate in this example) is ignored when there is a JavaScript routine specified in the `onValidate` attribute. To correct this, you must write a test to validate the date using JavaScript (not addressed in this tutorial).

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. The last trip photo business rule has two parts:

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

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

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

To check to see if a file exists, ColdFusion provides a `FileExists` function. The syntax of this function is:

```
FileExists(absolute_path)
```

This function returns `True` if the file specified in the argument does exist; otherwise, it returns `False`. Assume that a data entry page has an input tag named "testFileName". If a user types in a valid file name, the action page snippet from the following example would display the message “The test file exists”:

```
<cfset fileLocation = "c:\inetpub\wwwroot\images\">
<cfif IsDefined("form.testFileName")>
    <cfif form.testFileName is not "">
  <!---Concatenate the File Location with the FileName passed in--->
    <cfset fileLocation = fileLocation  & form.testFileName>
      <cfif FileExists(fileLocation)>
        <cfoutput> The test file exists. </cfoutput>
      </cfif>
    </cfif>
</cfif>
```

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, then the photos are stored in the C:\inetpub\wwwroot\cfdocs\getting_ started\photos directory.

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

Reviewing the code

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

Code	Explanation
`<cfif IsDefined("form.testFileName")>`	The `cfif` tag checks to see if the form variable `testFileName` has been entered.
`<cfset fileLocation = fileLocation & form.testFileName>`	The ColdFusion `&` operator in the `cfset` tag combines the original value for `fileLocation` from the first source line ("c:\inetpub\wwwroot\images\") with the value of the `testFileName` form variable.
`<cfif FileExists(fileLocation)>`	FileExists checks to see if the file indicated by the fileLocation variable exists at the specified disk location.

Exercise: use FileExists function to verify the existence of photo file name

In this exercise, you will use the Cold Fusion `FileExists` function to ensure that the photo file name entered in the Trip Edit page exists in the location specified.

To verify that the photo file name exists:

1 Open the tripeditaction.cfm in the my_app directory in your editor.

2 In the tripeditaction.cfm page, do the following:

a Add logic to check that the user entered a valid photo file name by copying the code from the photofilecheck.txt file in the solutions directory pasting it immediately following the first <cfset isOk = "Yes"> statement.

b Verify that code you copied in **step a** 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:\cfusionmx\wwwroot\CFDOCS\getting_started\Photos\">
```

or

```
<cfset PhotoLocation =
"C:\Inetpub\wwwroot\CFDOCS\getting_started\Photos\">
```

- For Linux or Solaris systems:

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

or

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

3 Save the page and test it by opening the tripedit.cfm page in your browser.

Testing recommendations:

a In the Trip Edit page entering valid information in all the required fields but the Photo File field.

b In the Photo File 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 file name in the Trip Edit page with somewhere.jpg and click Save.

The following message appears: Trip added.

Summary

As described in this lesson, ColdFusion offers several 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. Additionally, you can use JavaScript with ColdFusion to execute client-side functions.

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.

LESSON 5
Implementing the Browsing and Maintenance Database Functions

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

This lesson explains how to do the following tasks:

- Use SQL SELECT to move to the first, last, next and previous rows in the trips table
- Add code to the main application page to browse the trips table
- Use SQL DELETE to delete rows in the trips table
- Use `cflocation` to link the search facility built Lesson 1 to the main Trip Maintenance application page.

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:

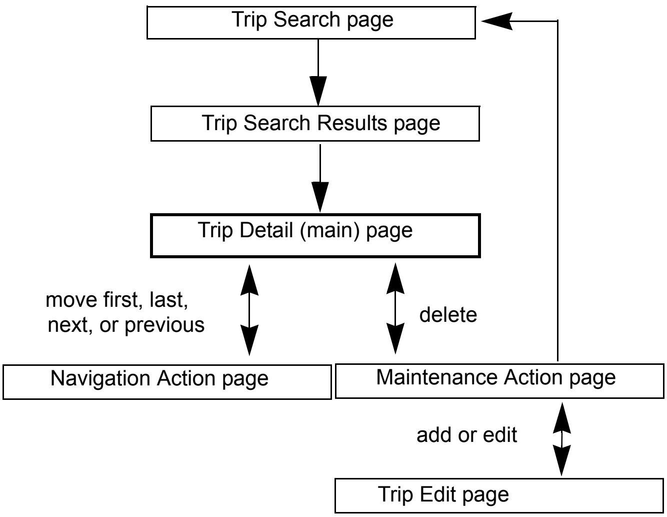
Element	Type	Brief description
cflocation	Tag	Opens a ColdFusion page or HTML file.
URL	Structure	Structure to hold the variables within a URL.

Enhancing the Trip Maintenance application

In this lesson, you will make enhancements to the sample Trip Maintenance application that you created in previous lessons. In Lesson 4, you added buttons to the Trip Detail page to browse, add, edit, delete or search for records in the database. In this lesson you will build the action pages that implement the actions for these buttons.

Page flow

You will combine the Trip Detail page and the search pages you built in the previous lessons with a maintenance action page and a navigation action page you will build in this lesson. In the final two lessons you will build a Trip Edit page to complete the trip maintenance facility. The following figure shows the flow of the finished Trip Maintenance application pages:



Notice that the Trip Detail page is at the center of the Trip Maintenance application. Depending on the user action, the Trip Detail page navigates the records in the database or connects to the appropriate page to add, edit, delete, or search for records in the database.

In order for the application to process the user actions from the Trip Detail page, you must build the two action pages for the navigation and maintenance functions.

Navigation action page

This navigation action page determines which triprecord displays on the Trip Detail page after the user presses 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 display on the Trip Detail page. In this dynamic SQL statement the proper `tripID` is passed as a parameter to the URL then redirects it to the Trip Detail page.

Maintenance action page

The maintenance action page processes a user's maintenance request from the Trip Detail page. The request can be any of the following actions:

- Delete the currently displayed trip.
- Launch the search facility.
- Add a new trip (implemented in the previous two lessons).
- Update the currently displayed trip (implemented in the previous two lessons).

Application development steps

You will review or participate in the following application construction steps:

Steps	Description
1	Build the navigation action page to navigate and display the proper trip record.
2	Build the maintenance action page to process the user's selection on the Trip Detail page.

Using dynamic SQL to browse (navigate) the Trips table

The `tripID` uniquely identifies a trip in the Trips table. In Lesson 3, you displayed the Trip Detail page for a trip by passing the ID as a parameter of the URL launching the detail page. Therefore, you would navigate to 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 with a proper URL identifying the correct `tripID` based on the navigation button clicked. Unfortunately, because trips are added and later deleted, trips might not be ordered sequentially by ID. There can be missing IDs where trips were deleted. Therefore, if the current trip ID is 1 and the user clicks the next navigation button, it will not navigate to 2.

In order to ensure that the proper `tripID` is retrieved, you must create a query to 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 2, you used ColdFusion string manipulation to construct the proper SQL SELECT WHERE clause. In this lesson, you will use a similar approach to build the WHERE clause for navigation. Additionally, it is necessary to use the proper ORDER BY clause to select the correct trip row.

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

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.

Limiting the number of result rows

Each of the SQL statements in the previous table return 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, since only the initial row in the result set is needed for the Trip Detail page display.

ColdFusion provides the maxRows attribute on 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.

The navigation action page

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 then 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. The following code (from the navigationaction.cfm) processes the navigation button requests on the Trip Detail page:

```
<!---  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")>
     ORDER BY tripID DESC

   </cfif>

</cfquery>
   <cfif TripQuery.RecordCount is 1>

   <cflocation url="tripdetail.cfm?ID=#TripQuery.tripID#">

<cfelese>
   <cflocation url="tripdetail.cfm?ID=#Form.RecordID#">

</cfif>
```

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" will be 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>`
 	
`</cfquery>`
	The `cfif` tag checks if the user pressed a navigation button on the browse page. The X property is checked since the buttons on the detail page use image type HTML input tags. The X property is a mouse offset that gets sent when you click a graphic button.
	
	The WHERE and ORDER BY clauses will vary depending on the navigation button clicked by the user.

`<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 if the query returned a row to display. If it did, use that `tripID` to form a URL to navigate to using the `cflocation` tag. If the query returned no rows, navigate back to the detail page with current record id passed in the hidden form variable `RecordID`.

Exercise: implement trip record browsing (navigation)

Follow these steps to implement the trip record browsing functionality (navigation buttons) on the Trip Detail page. In this exercise, you will use the supplied navigation action page to implement and test the navigation buttons on the Trip Detail page.

To implement the trip record browsing functionality:

1 Copy the navigationaction.cfm file from the solutions subdirectory in the getting_started directory to the my_app directory.

2 View the tripdetail.cfm page from the my_app directory in a browser and test the navigation buttons as follows:

a Click Next Row.

The Trip Detail page shows information about the second trip.

b Click Previous Row.

The Trip Detail page shows information about the first trip.

c Click Last Row.

The Trip Detail page shows information about the last trip.

d Click First Row.

The Trip Detail page shows information about the first trip.

Building the maintenance action page

The maintenance action page (maintenanceaction.cfm) handles the user's maintenance requests. The delete request is handled directly within the maintenance action page. The search request is accomplished by linking the Trip Search page. The maintenance action page navigates to another page for data capture for add and edit requests.

You will build the tripedit.cfm page to capture the information for add and edit requests in the next lesson. The following table identifies the button clicked on the Trip Detail page with the action taken in the maintenance action page:

Button	Action taken in maintenaceaction.cfm
Search	Navigate to tripsearch.cfm built in Lesson 4.
Delete	Execute SQL DELETE statement for current `tripID`.
Edit	Navigate to tripedit.cfm with ID parameter in URL set to current `tripID`.
Add	Navigate to tripedit.cfm with ID parameter in URL set to blank.

Maintenance action page code

ColdFusion creates a variable only for the button that the user clicked. Therefore, the `IsDefined` function is used to test which action to take. The following code is an excerpt from the maintenanceaction.cfm page that tests which action to take:

```
<cfif IsDefined("Form.btnSearch")>
...
<cfelseif IsDefined("Form.btnDelete")>
...
```

```
<cfelseif IsDefined("Form.btnEdit")>
...
<cfelseif IsDefined("Form.btnAdd")>
...
</cfif>
```

The first two buttons are the easiest to handle because they do not require building any new pages. Therefore, you will implement the functionality for the `Search` and `Delete` buttons first.

Linking the Trip Detail page to the Trip Search page

The ColdFusion `cflocation` tag navigates from the current page to a target HTML or CFML page. The URL attribute contains the name of the target page and any arguments. The code to navigate to the search page (tripsearch.cfm) follows:

```
<cflocation url="tripsearch.cfm">
```

Deleting the current trip record shown on the Trip Detail 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.

SQL DELETE Statement

The SQL DELETE statement removes existing rows in a relational table. The format of the DELETE statement is as follows:

```
DELETE FROM table_name WHERE column_name = some_value
```

Consider a database table named Clients that holds information about people with 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 processed the preceding statement, the table would contain 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. The RecordID field holds the tripID. Therefore, using the hidden `RecordID` input tag from the Trip Detail page, the following SQL statement deletes a row from the Trips table reads:

```
DELETE FROM trips WHERE tripID = #Form.RecordID#
```

Exercise: handle search and delete in maintenance action page

In this exercise, you will link the search function developed in Lesson 4 to the main page. Further, you will provide code to support the trip delete function. Then you will test this functionality by deleting a trip, then searching for it to ensure it was deleted successfully.

Follow these steps to create the maintenance action page that implements the search and delete functionality.

To create the maintenance action page:

1 In your editor, create a CFM page.

2 Delete any default lines of code if your editor automatically adds them.

3 To handle the Search and Delete buttons from the Trip Detail page, enter the following code:

```
<!---   SEARCH BUTTON--->
<cfif IsDefined("Form.btnSearch")>
     <cflocation url="tripsearch.cfm">
<!---   DELETE BUTTON --->
<cfelseif IsDefined("Form.btnDelete")>
   <cfquery name="DeleteRecord" dataSource="CompassTravel">
     DELETE FROM trips WHERE tripID = #Form.RecordID#
   </cfquery>
   <cflocation url="tripdetail.cfm">
</cfif>
```

4 Save the page as maintenanceaction.cfm in the my_app directory.

5 View the tripdetail.cfm page in a browser.

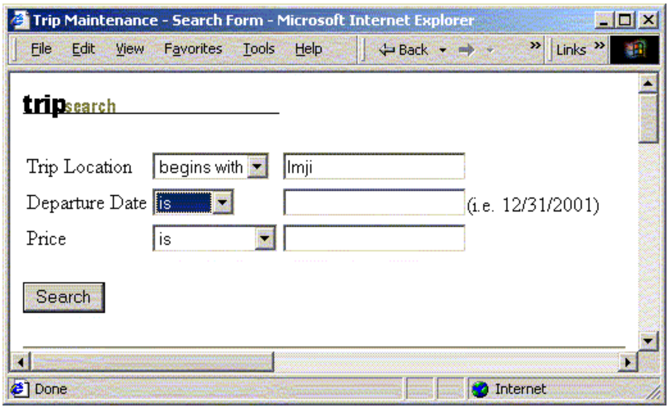
The current trip is Nepal. Notice that the destination for Nepal Summit Challenge trip is Imji Himal, Nepal.

6 Click Search.

The Trip Search page appears.

7 In the Trip Search page, select **begins with** in the selection box for Trip Location. Then enter Imji in the trip location value field.

The following figure shows the search form looks:



Trip Maintenance - Search Form - Microsoft Internet Explorer

tripsearch

Trip Location: begins with | Imji

Departure Date: is | (i.e. 12/31/2001)

Price: is

Search

8 Click Search. Verify that only a single trip is found whose location **begins with** Imji.

9 To return to the Trip Detail page for this trip, click the hyperlink for Imji.

10 In the Trip Detail page, click Delete to remove this record from the Trip database file.

11 Verify that the trip record was removed from the Trips database by repeating the search in step 7.

Summary

In this lesson, you converted the Trip Detail page from the search result display into a trip browser by using `cfquery` with SQL SELECT. The browser enables users to navigate sequentially through the Trips table. 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. Additionally, you used `cfquery` and SQL DELETE to delete the current trip.

In the next lesson

You have implemented two of the four maintenance buttons on the Trip Detail page. In the next lesson, you will implement the Add and Edit buttons. Because these functions require user input, you will build a new page, tripedit.cfm, to capture this information. The user input must be validated against the Compass Travel business rules before the data is saved to the database. Therefore, in the next lesson, you also will implement the business rule validation. In the final lesson, you will add the database update logic.

LESSON 6
Adding and Updating SQL Data

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

This lesson explains how to do the following tasks:

- Modify the Trip Edit page to link the add and update functions to the main application page.
- Write code to insert new trips using SQL.
- Write code to insert new trips without using SQL.
- Write code to update several records in the Trips table.

ColdFusion tags and functions introduced in this lesson

The following table identifies the ColdFusion tags and functions 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 a records in a JDBC data source.

Completing the Trip Maintenance application

In Lesson 5, you created the tripeditaction.cfm page to contain server side edits for the trip edit data entry form. In this final lesson, you will complete the tripeditaction.cfm page. To complete the action page, you will write code to do these tasks:

- Add trips to the Compass Travel database using the cfquery tag and the SQL INSERT statement.
- Add trips to the Compass Travel database using the `cfinsert` tag.
- Update the current trip using the `cfupdate` tag.
- Link the current trip to be updated or added by the tripeditaction.cfm with the tripedit.cfm page.

In addition to completing the Trip Maintenance application, you will develop a ColdFusion page to update all the prices in the database using the cfquery tag and the SQL UPDATE statement.

Writing code to save new trips to the database

In Lesson 5, 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.

Adding data using SQL INSERT with cfquery approach

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 format of the INSERT statement is as follows:

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

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

After the following SQL statement executes:

```
INSERT INTO Clients
VALUES ('Smith', 'Kaleigh', '14 Greenway', 'Windham')
```

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 SQL, you must surround any text or date values with single quotation marks but numeric values are not.

Alternatively, you can specify the columns for which you want to insert data. This approach lets you insert data to some columns while omitting others. The syntax for this approach is as follows:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

For example, the syntax to add Kaleigh Smith of Windham, with the address unknown, you use the named column approach:

```
INSERT INTO Clients (LastName, FirstName, City)
VALUES ('Smith', 'Kaleigh', 'Windham')
```

You used the `cfquery` tag to execute SQL from ColdFusion. The `cfquery` tag passes SQL statements to your data source. As described in Chapter 4, a data source stores information about how to connect to an indicated data provider, such as a relational database management system. The data source you established in Chapter 4 stored information on how to access the Compass Travel database. The data source name was called “CompassTravel”.

Exercise: insert trip data using SQL INSERT and cfquery

In this exercise you will add code to pass the data entered on the Trip Maintenance collection form and insert into the Compass Travel database. To do this, you will be modifying the trip insert action page to use the SQL INSERT statement and the ColdFusion `cfquery` tag.

To add data using SQL INSERT and cfquery:

1 Open tripeditaction.cfm in the my_app directory in your editor.

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 to insert the data from the Form variables into the Trips table.

Tip: To save time, you can copy this code from the tripsinsertquery.txt file (for Windows users) or from tripinsertqueryunix.txt (for UNIX users) in the solutions directory.

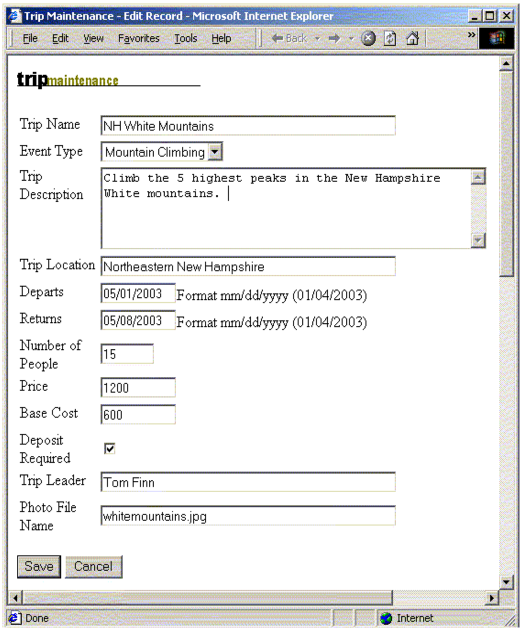
For	Code

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 page and test it by opening the `tripedit.cfm` in your browser.

4 In the tripedit.cfm page, fill in the fields with the values in the following figure, then click Save:

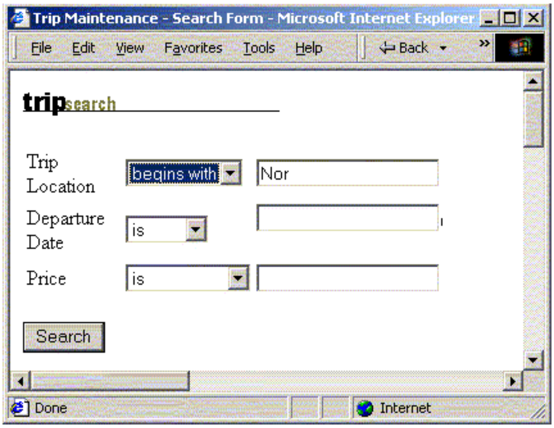

Trip Maintenance - Edit Record - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
tripmaintenance
Done
Internet

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	05/01/2003 Format mm/dd/yyyy (01/04/2003)
Returns	05/08/2003 Format mm/dd/yyyy (01/04/2003)
Number of People	15
Price	1200
Base Cost	600
Deposit Required	☑
Trip Leader	Tom Finn
Photo File Name	whitemountains.jpg

After the new trip is written to the database, the following message appears: Trip is added.

5 To verify that the save worked, open tripsearch.cfm in the my_app directory in your browser.

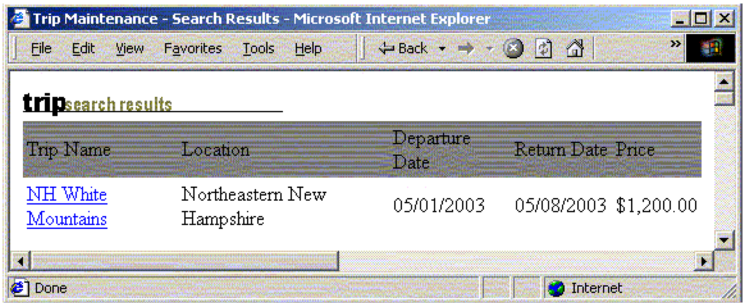
6 In the Trip Search page, enter Begins With **Nor** in the Trip Location criterion value in the Search page as in the following figure:


Trip Maintenance - Search Form - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
tripsearch
Search
Done
Internet

Trip Location	begins with	Nor
Departure Date	is	
Price	is	

7 Click Search.

The TripResults page appears:



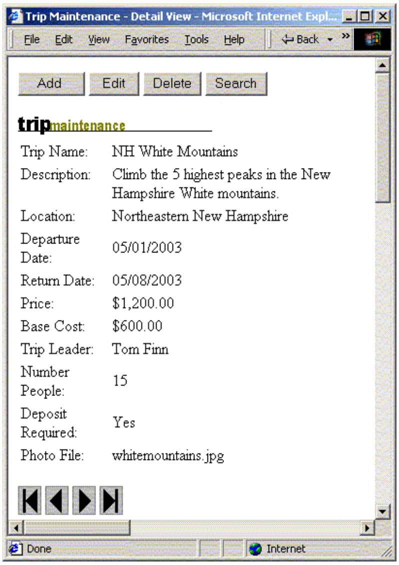
Trip Maintenance - Search Results - Microsoft Internet Explorer

tripsearch results

Trip Name	Location	Departure Date	Return Date	Price
NH White Mountains	Northeastern New Hampshire	05/01/2003	05/08/2003	$1,200.00

8 Click the link to the NH White Mountains to display the details of the trip you just added. Verify that all the fields were saved correctly.

The following page appears:



Trip Maintenance - Detail View - Microsoft Internet Expl...

tripmaintenance

Trip Name:	NH White Mountains
Description:	Climb the 5 highest peaks in the New Hampshire White mountains.
Location:	Northeastern New Hampshire
Departure Date:	05/01/2003
Return Date:	05/08/2003
Price:	$1,200.00
Base Cost:	$600.00
Trip Leader:	Tom Finn
Number People:	15
Deposit Required:	Yes
Photo File:	whitemountains.jpg

9 Click the Delete button to delete this record so that you can reuse the steps 4-8 of this exercise in the next exercise.

Reviewing the code

The following table describes the SQL INSERT and cfquery code used to add data:

Code	Explanation

`<cfquery name="AddTrip"`
`datasource="CompassTravel">`
	Using the `datasource` attribute, `cfquery` connects to the data source CompassTravel 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 that the data are to be inserted into the Trips table. The table column names are cited in a comma separated list surrounded by parenthesis (`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 passed from the data entry form to the action page. The variables are surrounded by pound signs; for example, `#Form.baseCost#`. Additionally, note that if the column data type is a string data type, then the values are surrounded by single quotation marks; for example: `'#Form.TripName#'`.

For more information about adding data to a database using SQL and `cfquery`, see *Developing ColdFusion MX Applications with CFML*. For more information about SQL, consult any SQL primer.

Adding data using the simpler, cfinsert approach

For those who would 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. `Formfields` is a comma-separated list of form fields to insert. If this attribute is not specified, all fields in the form are included in the operation. The following example uses the `cfinsert` 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 previous 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 are inserted.
formfields	A comma-separated list of form fields to insert.

Exercise: insert trip data using cfinsert

In this exercise, you change the approach the action page uses to insert the data into the database. You will replace the SQL INSERT statement with the `cfinsert` tag.

To add data using cfinsert:

1 Open tripeditaction.cfm from the my_app directory in your editor and do the following:

   a Remove the entire AddTrip cfquery that you added in the last exercise (from the beginning `<cfquery name ="AddTrip" datasource="CompassTravel">` tag to the `</cfquery>` end tag).

   b 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">
   ```

2 Save the page and test it by opening the tripedit.cfm page in your browser.

3 Follow steps 4 through 9 in the previous exercise to verify this approach to inserting new trips.

For more information about adding data to a database using the `cfinsert` tag, see *Developing ColdFusion MX Applications with CFML*.

Updating a SQL row using cfupdate

To update an existing SQL row, ColdFusion offers a simple approach for updating SQL rows through the use of the `cfupdate` tag. Like `cfinsert`, 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 be inserted. `Formfields` is a comma-separated list of form fields to insert. If this attribute is not specified, all 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 TripEdit page the cfupdate tag is used otherwise the `cfinsert` tag is executed. The following example uses the `cfupdate` and `cfinsert` 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 ColdFusion function `IsDefined` determines whether the hidden field tripID was passed to the action page from tripedit.cfm. 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 displays with the updated trip. If the cfif statement evaluates to False, the cfinsert statement executes and the first trip displays in the main page.

Exercise: update trip data using cfupdate

In this exercise, you will add the code to update the trip data into the database. You will add the `cfupdate` tag to the tripeditaction.cfm page.

To update the database using cfupdate:

1 In an editor, open tripeditaction3.cfm from the solutions directory.

2 Review the code to update the database (the last 12 lines of code).

3 Verify that the correct photolocation path is specified. This 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:\cfusionmx\wwwroot\CFDOCS\getting_started\Photos\">
  ```
  or
  ```
  <cfset PhotoLocation =
    "C:\Inetpub\wwwroot\CFDOCS\getting_started\Photos\">
  ```
- For Linux or Solaris systems:
  ```
  <cfset PhotoLocation = "/opt/coldfusionmx/wwwroot/cfdocs/
    getting_started/photos/">
  ```
  or
  ```
  <cfset PhotoLocation = "/<webserverdocroot>/cfdocs/
    getting_started/photos/">
  ```

4 Save the file as tripeditaction.cfm in the my_app directory.

For more information about adding data to a database using the `cfupdate` tag, see *Developing ColdFusion MX Applications with CFML*.

Now that you have built the data entry form adding new and updating existing trips, you will add the logic to link it to the main trip page to test the update logic.

Linking the Trip Edit page to the main page

As discussed in Lesson 4, 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 the next exercise you will insert the code to call tripedit.cfm from maintenance.cfm as follows:

```
<!---   EDIT BUTTON --->
<cfelseif IsDefined("Form.btnEdit")>
   <cflocation url="tripedit.cfm?ID=#Form.RecordID#">
<!---   ADD BUTTON --->
<cfelseif IsDefined("Form.btnAdd")>
   <cflocation url="tripedit.cfm">
</cfif>
```

Notice that when the user clicks the Add button, the maintenanceaction.cfm navigates to tripedit.cfm passing no 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, tripedit.cfm must query the database and fill the form with the data for the corresponding trip. The following code properly initializes the trip edit form:

```
<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")>
   <cfset price = #TripQuery.price#>
   <cfset tripLeader = '#TripQuery.tripLeader#'>
   <cfset photo = '#TripQuery.photo#'>
   <cfset baseCost = #TripQuery.baseCost#>
   <cfset numberPeople = #TripQuery.numberPeople#>
   <cfset depositRequired = '#TripQuery.depositRequired#'>
   <cfelse>
   <cfset tripName = ''>
   <cfset eventType = ''>
   <cfset tripDescription = ''>
   <cfset eventTypeIdentifier = #TripQuery.eventType#>
   <cfset tripLocation = ''>
   <cfset departureDate =  ''>
   <cfset returnDate = ''>
   <cfset price = ''>
   <cfset tripLeader = ''>
   <cfset photo = ''>
   <cfset baseCost = ''>
   <cfset numberPeople = ''>
   <cfset depositRequired = ''>
</cfif>
```

Reviewing the code

The following table describes the code used 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")>`
	The ColdFusion function `IsDefined` determines whether an ID argument was passed as part of the invoking URL.
	The ID argument is passed when TripEdit is invoked when the Edit button is clicked on the main page.
	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 program then instantiates local variables from the results of the SQL query.
	ColdFusion `DateFormat` function formats the date fields.

`...`
`<cfelse>`
`<cfset tripName = ''>`
`<cfset eventType = ''>`
`...`
`</cfif>`
	If TripEdit is called to add a new trip, then there is no ID passed as a URL argument. In this case, the local variables are instantiated to blank.

Exercise: linking the Add and Edit buttons

In this exercise you will link the Add and Edit buttons on the Trip Detail page with the Trip Edit page.

To link the add and update buttons on the Trip Detail page:

1 Open maintenanceaction.cfm in the my_app directory in your editor.

2 Locate the `</cfif>` tag at the end of the file.

3. Insert the following code just before the last line:

```
<!---  EDIT BUTTON --->
  <cfelseif IsDefined("Form.btnEdit")>
    <cflocation url="tripedit.cfm?ID=#Form.RecordID#">
<!---  ADD BUTTON --->
  <cfelseif IsDefined("Form.btnAdd")>
    <cflocation url="tripedit.cfm">
  </cfif>
```

4. Save maintenanceaction.cfm.
5. Open tripedit4.cfm in the solutions directory in your editor.
6. Open the file initvariables.txt from the solutions directory.
7. Copy the contents of the initvariables.txt and paste it before the `<HTML>` tag in the tripedit4.cfm page.
8. Save the file as tripedit.cfm in the my_app directory.
9. Test update logic by opening the tripdetail.cfm page in your browser and doing the following tasks:
   a. Click the Edit button.
   b. Double the price of the current trip.
   c. Click Save.

The ColdFusion `cfupdate` works well for updating a single record. To update several records in a single query, you must use the SQL UPDATE statement in conjuction with `cfquery`.

SQL Update

The SQL UPDATE statement updates or changes rows in a relational table. The syntax of the UPDATE statement is as follows:

```
UPDATE table_name SET column_name = new_value
WHERE column_name = some_value
```

Consider a database table named Clients that contains information about people in the following rows:

PersonID	LastName	FirstName	Age
1	Green	Tom	12
2	Wall	Peter	42
3	Madigan	Jess	20

After the following SQL statement executes:

```
UPDATE Clients SET LastName = 'Pitt'
WHERE ID = 3
```

the table contains the following rows:

PersonID	LastName	FirstName	Age
1	Green	Tom	12
2	Wall	Peter	42
3	Pitt	Jess	20

Update several rows

The UPDATE statement updates all rows that meet the criteria found in the WHERE clause. If there is no WHERE clause, every row of the table is updated. After the following SQL statement executes:

```
UPDATE Clients SET Age = Age + 1
WHERE ID = 3
```

the table contains the following rows:

PersonID	LastName	FirstName	Age
1	Tom	Green	12
2	Peter	Green	42
3	Pitt	Jess	21

Updating multiple records

The `cfupdate` statement works well when you want to update the current record within a `cfquery`. Alternatively, you can update several rows within a table by issuing a single query using `cfquery` and the SQL UPDATE statement. For example, if the base cost of all trips increased by 5%, you could issue the following query:

```
<!-- Routine to increase trip base Cost by 5% -->
<cfquery name="TripQuery" dataSource="CompassTravel">
   UPDATE Trips SET baseCost = baseCost * 1.05
</cfquery>
```

Exercise: using SQL UPDATE with cfquery

In this exercise, you will develop a page to increase the price of every trip by 10%. This page runs only once and is not part of the Trips Maintenance application. This exercise shows how to update many database rows using a single SQL UPDATE statement and the ColdFusion `cfquery` tag.

To update multiple database rows using SQL UPDATE with cfquery:

1 In an editor, open a new page and save it as priceincrease.cfm in the my_app directory.

2 Remove any lines of code that your editor added.

3 Add the following code:

```
<!---Routine to increase trip price by 10% --->
<cfquery name="TripQuery" dataSource="CompassTravel">
    UPDATE trips SET price = price * 1.1
</cfquery>
<cfoutput> New prices are now in effect.</cfoutput>
```

4 Save the page then test the page by doing the following tasks:

   a Use the Trip Maintenance application to take note of the price of any trip by viewing tripdetail.cfm in a browser.

   b Test by opening the priceincrease.cfmpage in your browser. This page automatically updates the prices in the trips table.

   c Use the Trip Maintenance application to verify that the query successfully increased the price of the trip by 10%. To do this, navigate to the tripdetail.cfm and locate the trip you noted in step 4a. The price of this trip is now 10% higher.

Summary

In this lesson you used the `cfinsert` and `cfupdate` tags to add and update data to a SQL table. You also have used the SQL UPDATE statement in conjuction with the `cfquery` tag to effect a trip price increase for all rows in the Trips table.

You have completed the Getting Started tutorial. You should understand how you can combine CFML and SQL to develop powerful applications. When compared with traditional development methods, ColdFusion helps speed the development in hand crafting a database solution like the one in this tutorial. Remarkably, however, depending on the editor or IDE that you use to develop applications, much of the work in this tutorial can be autogenerated using built-in wizards, which simplifies the development process even more.

INDEX