Use PHP 4, 5, or 6 and MySQL 5 to build dynamic Web database applications PDQ!

PHP & MySQL®

FOR DUMMIES®

3rd Edition

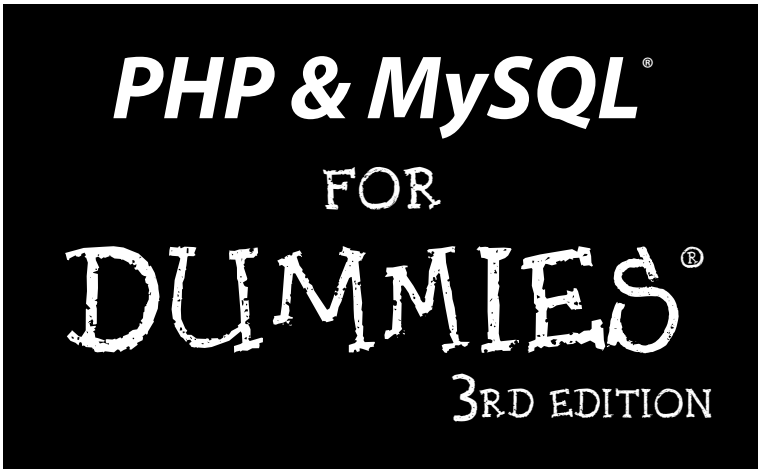
Companion Web site includes all code samples

A Reference for the Rest of Us!®

FREE eTips at dummies.com®



Janet Valade
Author of PHP & MySQL Everyday Apps For Dummies


PHP & MySQL®
FOR
DUMMIES®
3RD EDITION

PHP & MySQL® FOR DUMMIES® 3RD EDITION

by Janet Valade



Wiley Publishing, Inc.

PHP & MySQL® For Dummies®, 3rd Edition
Published by
Wiley Publishing, Inc.
111 River Street
Hoboken, NJ 07030-5774
www.wiley.com

Copyright © 2007 by Wiley Publishing, Inc., Indianapolis, Indiana

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, or online at http://www.wiley.com/go/permissions.

Reference for the Rest of Us!, The Dummies Way, Dummies Daily, The Fun and Easy Way, Dummies.com, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries, and may not be used without written permission. MySQL is a registered trademark of MySQL. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, please contact our Customer Care Department within the U.S. at 800-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002.

For technical support, please visit www.wiley.com/techsupport.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Control Number: 2006934828

ISBN-13: 978-0-470-09600-0

ISBN-10: 0-470-09600-4

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

3O/TQ/RQ/QW/IN



About the Author

Janet Valade is the author of *PHP 5 For Dummies, PHP & MySQL Everyday Apps For Dummies,* and *PHP & MySQL: Your visual blueprint for creating dynamic, database-driven Web sites,* as well as the author of first and second editions of this book. In addition, Janet is the author of *Spring into Linux* and a coauthor of *Mastering Visually Dreamweaver 8 and Flash 8.*

Janet has twenty years of experience in the computing field. Most recently, she worked as a Web designer and programmer in an engineering firm for four years. Before that, Janet worked for thirteen years in a university environment, where she was a systems analyst. During her tenure, she supervised the installation and operation of computing resources, designed and developed a data archive, supported faculty and students in their computer usage, wrote numerous technical papers, and developed and presented seminars on a variety of technology topics.

To keep in touch, see `janet.valade.com`.

Author's Acknowledgments

First, I want to express my appreciation to the entire open source community. Without those who give their time and talent, there would be no cool PHP and MySQL for me to write about. Furthermore, I never would have learned this software without the lists, where people generously spend their time answering foolish questions from beginners.

I want to thank my mother for passing on a writing gene, along with many other things. And my children always for everything. My thanks to my friends Art, Dick, and Marge for responding to my last-minute call for help. I particularly want to thank Sammy, Dude, Spike, Lucky, Upanishad, Sadie, and E. B. for their important contributions.

And, of course, I want to thank the professionals who make it all possible. Without my agent and the people at Wiley, this book would not exist. Because they all do their jobs so well, I can contribute my part to this joint project.

Publisher's Acknowledgments

We're proud of this book; please send us your comments through our online registration form located at www.dummies.com/register/.

Some of the people who helped bring this book to market include the following:

Acquisitions, Editorial, and Media Development

Project Editor: Susan Pink
(Previous Edition: Pat O'Brien)

Acquisitions Editor:

Copy Editor: Susan Pink
(Previous Edition: Teresa Artman)

Technical Editor: John Gosney

Editorial Manager: Jodi Jensen

Media Development Specialists: Angela Denny, Kate Jenkins, Steven Kudirka, Kit Malone, Travis Silvers

Media Development Coordinator: Laura Atkinson

Media Project Supervisor: Laura Moss

Media Development Manager: Laura VanWinkle

Media Development Associate Producer: Richard Graves

Editorial Assistant: Amanda Foxworth

Sr. Editorial Assistant: Cherie Case

Cartoons: Rich Tennant (www.the5thwave.com)

Composition Services

Project Coordinator: Erin Smith

Layout and Graphics: Lavonne Cook, Clint Lanhen, Barry Offringa, Lynsey Osborn, Heather Ryan

Proofreaders: Jessica Kramer, Techbooks

Indexer: Techbooks

Special Help
Heather Ryan

Publishing and Editorial for Technology Dummies

Richard Swadley, Vice President and Executive Group Publisher

Andy Cummings, Vice President and Publisher

Mary Bednarek, Executive Acquisitions Director

Mary C. Corder, Editorial Director

Publishing for Consumer Dummies

Diane Graves Steele, Vice President and Publisher

Joyce Pepple, Acquisitions Director

Composition Services

Gerry Fahey, Vice President of Production Services

Debbie Stailey, Director of Composition Services

Contents at a Glance

Introduction 1

Part I: Developing a Web Database Application Using PHP and MySQL 7

Chapter 1: Introduction to PHP and MySQL 9
Chapter 2: Setting Up Your Work Environment 21
Chapter 3: Developing a Web Database Application 37

Part II: MySQL Database 63

Chapter 4: Building the Database 65
Chapter 5: Protecting Your Data 93

Part III: PHP 111

Chapter 6: General PHP 113
Chapter 7: PHP Building Blocks for Programs 143
Chapter 8: Data In, Data Out 187
Chapter 9: Moving Information from One Web Page to the Next 255

Part IV: Applications 275

Chapter 10: Putting It All Together 277
Chapter 11: Building an Online Catalog 289
Chapter 12: Building a Members Only Web Site 327

Part V: The Part of Tens 357

Chapter 13: Ten Things You Might Want to Do Using PHP Functions 359
Chapter 14: Ten PHP Gotchas 367

Part VI: Appendixes 373

Appendix A: Installing MySQL 375
Appendix B: Installing PHP 391
Appendix C: Installing and Configuring Apache 407

Index 419

Table of Contents

Introduction 1

About This Book 1
Conventions Used in This Book 2
What You're Not To Read 3
Foolish Assumptions 3
How This Book Is Organized 4
Part I: Developing a Web Database Application Using PHP and MySQL 4
Part II: MySQL Database 4
Part III: PHP 4
Part IV: Applications 4
Part V: The Part of Tens 5
Part VI: Appendixes 5
Icons Used in This Book 5
Where to Go from Here 5

Part I: Developing a Web Database Application Using PHP and MySQL 7

Chapter 1: Introduction to PHP and MySQL 9

What Is a Web Database Application? 10
The database 11
The application: Moving data into and out of the database 11
MySQL, My Database 12
Advantages of MySQL 13
How MySQL works 14
Communicating with the MySQL server 15
PHP, a Data Mover 15
Advantages of PHP 16
How PHP works 16
MySQL and PHP, the Perfect Pair 18
Advantages of the relationship 18
How MySQL and PHP work together 18
Keeping Up with PHP and MySQL Changes 19

Chapter 2: Setting Up Your Work Environment 21

The Required Tools 21
Finding a Place to Work 22
A company Web site 22
A Web hosting company 24
Setting up and running a Web site on your local computer 26

Testing, Testing, 1, 2, 3 32
Understanding PHP/MySQL functions 32
Testing PHP 33
Testing MySQL 35

Chapter 3: Developing a Web Database Application 37

Planning Your Web Database Application 37
Identifying what you want from the application 38
Taking the user into consideration 40
Making the site easy to use 41
Leaving room for expansion 41
Writing it down 42
Presenting the Two Running Examples in This Book 42
Stuff for Sale 42
Members Only 43
Designing the Database 44
Choosing the data 44
Organizing the data 46
Designing the Sample Databases 50
Pet Catalog design process 51
Members Only design process 53
Types of Data 56
Character data 56
Numerical data 56
Date and time data 57
Enumeration data 57
MySQL data type names 57
Writing it down 59
Taking a Look at the Sample Database Designs 59
Stuff for Sale database tables 59
Members Only database tables 60
Developing the Application 61
Building the database 62
Writing the programs 62

Part II: MySQL Database 63

Chapter 4: Building the Database 65

Communicating with MySQL 65
Building SQL queries 66
Sending SQL queries 67
Building a Database 73
Creating a new database 73
Deleting a database 74
Adding tables to a database 74
Changing the database structure 76

Moving Data Into and Out of the Database 77
Adding information 78
Retrieving information 82
Combining information from tables 87
Updating information 92
Removing information 92

Chapter 5: Protecting Your Data 93

Controlling Access to Your Data 93
Understanding account names and hostnames 94
Finding out about passwords 96
Taking a look at account permissions 97
Setting Up MySQL Accounts 98
Identifying what accounts currently exist 100
Adding accounts 100
Adding and changing passwords 101
Changing permissions 102
Removing accounts and permissions 103
Backing Up Your Data 104
Restoring Your Data 107
Upgrading MySQL 110

Part III: PHP 111

Chapter 6: General PHP 113

Adding a PHP Section to an HTML Page 113
Writing PHP Statements 116
Using PHP Variables 119
Naming a variable 119
Creating and assigning values to variables 119
Dealing with notices 121
Using PHP Constants 122
Working with Numbers 123
Working with Character Strings 125
Single-quoted strings versus double-quoted strings 126
Joining strings 127
Working with Dates and Times 128
Setting local time 128
Formatting a date 129
Storing a timestamp in a variable 130
Using dates with MySQL 131
Comparing Values 132
Making simple comparisons 133
Matching character strings to patterns 135
Joining Comparisons with and/or/xor 139
Adding Comments to Your Program 141

Chapter 7: PHP Building Blocks for Programs 143

Useful Simple Statements....144
Using echo statements....145
Using assignment statements....148
Using increment statements....149
Using exit....150
Using function calls....150
Using PHP Arrays....151
Creating arrays....151
Viewing arrays....152
Removing values from arrays....154
Sorting arrays....154
Getting values from arrays....156
Walking through an array....158
Multidimensional arrays....160
Useful Conditional Statements....163
Using if statements....164
Using switch statements....167
Using Loops....168
Using for loops....169
Using while loops....170
Using do..while loops....172
Infinite loops....174
Breaking out of a loop....176
Using Functions....178
Using variables in functions....180
Passing values between a function and the main program....181
Using built-in functions....185

Chapter 8: Data In, Data Out 187

PHP and MySQL Functions....187
Making a Connection....189
Connecting to the MySQL server....190
Selecting the right database....191
Sending SQL queries....194
Getting Information from a Database....195
Sending a SELECT query....196
Getting and using the data....196
Using functions to get data....202
Getting Information from the User....206
Using HTML forms....207
Making forms dynamic....211
Using the information from the form....224
Checking the information....226
Giving users a choice with multiple submit buttons....236
Putting Information into a Database....238
Preparing the data....238
Adding new information....242
Updating existing information....247

Getting Information in Files 250
Using a form to upload the file 250
Processing the uploaded file....251
Putting it all together 252

Chapter 9: Moving Information from One Web Page to the Next . . .255

Moving Your User from One Page to Another 256
Moving Information from Page to Page 259
Adding information to the URL....260
Storing information via cookies....264
Passing information with HTML forms 267
Using PHP Sessions 267
Opening sessions....268
Using PHP session variables 269
Sessions without cookies 271
Making sessions private 273
Closing PHP sessions 274

Part IV: Applications 275

Chapter 10: Putting It All Together 277

Organizing the Application 277
Organizing at the application level 278
Organizing at the program level 279
Keeping It Private....285
Ensure the security of the computer 285
Don't let the Web server display filenames 286
Hide things 286
Don't trust information from users 287
Use a secure Web server 287
Completing Your Documentation....288

Chapter 11: Building an Online Catalog 289

Designing the Application....289
Showing pets to the customers 290
Adding pets to the catalog 291
Building the Database....291
Building the Pet table....292
Building the PetType table....295
Building the Color table....296
Adding data to the database....297
Designing the Look and Feel....299
Showing pets to the customers 299
Adding pets to the catalog 303
Writing the Programs....306
Showing pets to the customers 306
Adding pets to the catalog 312

Chapter 12: Building a Members Only Web Site 327

Designing the Application 328
Building the Database 328
Building the Member table 329
Building the Login table 332
Adding data to the database 333
Designing the Look and Feel 333
Storefront page 334
Login page 334
New Member Welcome page 336
Members Only section 336
Writing the Programs 337
Writing PetShopFront 338
Writing Login 340
Writing New_member 352
Writing the Members Only section 354
Planning for Growth 355

Part V: The Part of Tens 357

Chapter 13: Ten Things You Might Want to Do Using PHP Functions 359

Communicate with MySQL 359
Send E-Mail 360
Use PHP Sessions 362
Stop Your Program 362
Handle Arrays 362
Check for Variables 363
Format Values 363
Compare Strings to Patterns 365
Find Out about Strings 365
Change the Case of Strings 366

Chapter 14: Ten PHP Gotchas 367

Missing Semicolons 367
Not Enough Equal Signs 368
Misspelled Variable Names 368
Missing Dollar Signs 368
Troubling Quotes 369
Invisible Output 369
Numbered Arrays 370
Including PHP Statements 371
Missing Mates 371
Confusing Parentheses and Brackets 372

Part VI: Appendixes 373

Appendix A: Installing MySQL 375

On Windows 375
Downloading and installing MySQL 375
Running the MySQL configuration wizard 378
Starting and stopping the MySQL server 380
On Linux and Unix 381
Using RPM (Linux only) 382
From source files 383
On Mac 386
Verifying a Downloaded File 388
Configuring MySQL 389

Appendix B: Installing PHP 391

Installing PHP on Unix, Linux, or Mac with Apache 391
On Unix and Linux 391
On Mac OS X 394
Installation options 398
Configuring Apache for PHP 399
Installing PHP on Windows 400
Configuring your Web server for PHP 402
Configuring PHP 404

Appendix C: Installing and Configuring Apache 407

Selecting a Version of Apache 407
Installing Apache on Linux and Unix 408
Before installing 408
Installing 408
Starting and stopping Apache 410
Getting information from Apache 412
Installing Apache on Windows 412
Installing 412
Starting and stopping Apache 414
Getting information from Apache 415
Installing Apache on Mac 416
Configuring Apache 417
Changing settings 417
Changing the location of your Web space 418
Changing the port number 418

Index 419

Introduction

Welcome to the exciting world of Web database applications. This book provides the basic techniques to build any Web database application, but I certainly recommend that you start with a simple one. In this book, I develop two sample applications, both chosen to represent two types of applications frequently encountered on the Web: product catalogs and customer- or member-only sites that require the user to register and log in with a password. The sample applications are complicated enough to require more than one program and to use a variety of data and data manipulation techniques, yet simple enough to be easily understood and adapted to a variety of Web sites. After you master the simple applications, you can expand the basic design to include all the functionality that you can think of.

About This Book

Think of this book as your friendly guide to building a Web database application. This book is designed as a reference, not as a tutorial, so you don't have to read it from cover to cover. You can start reading at any point — in Chapter 1, Chapter 9, wherever. I divide the task of building a Web database application into manageable chunks of information, so check out the table of contents and locate the topic that you're interested in. If you need to know information from another chapter to understand the chapter you're reading, I reference that chapter number.

Here's a sample of the topics I discuss:

- Building and using a MySQL database
- Adding PHP to HTML files
- Using the features of the PHP language
- Using HTML forms to collect information from users
- Showing information from a database in a Web page
- Storing information in a database

Conventions Used in This Book

This book includes many examples of PHP programming statements, MySQL statements, and HTML. Such statements are shown in a different typeface, which looks like the following line:

```
A PHP program statement
```

In addition, snippets or key terms of PHP, MySQL, and HTML are sometimes shown in the text of a paragraph. When they are, the special text in the paragraph is also shown in the example typeface, different than the paragraph typeface. For instance, `this text` is an example of a PHP statement within the paragraph text.

In examples, you will often see some words in italic. Italicized words are general types that need to be replaced with the specific name appropriate for your data. For instance, when you see an example like the following:

```
SELECT field1,field2 FROM tablename
```

field1, *field2*, and *tablename* need to be replaced with real names because they are in italic. When you use this statement in your program, you might use it in the following form:

```
SELECT name,age FROM Customer
```

In addition, you might see three dots (. . .) following a list in an example line. You don't type the three dots. They just mean that you can have as many items in the list as you want. For instance, when you see

```
SELECT field1,field2,... FROM tablename
```

the three dots just mean that your list of fields can be longer than two. It means you can go on with *field3*, *field4*, and so forth. For example, your statement might be

```
SELECT name,age,height,shoesize FROM Customer
```

From time to time, you'll also see something in bold. Pay attention to these; they indicate something I want you to see or something you need to type.

What You're Not To Read

Some information in this book is flagged as *Technical Stuff* with an icon off to the left. Sometimes you'll see this technical stuff in a sidebar: Consider it information that you don't need to read to create a Web database application. This extra information might contain a further look under the hood or describe a technique that requires more technical knowledge to execute. Some readers may be interested in the extra technical information or techniques, but feel free to ignore them if you don't find them interesting or useful.

Foolish Assumptions

To write a focused book rather than an encyclopedia, I needed to assume some background for you, the reader. I assumed that you know HTML and have created Web sites with HTML. Consequently, although I use HTML in many examples, I do not explain the HTML. If you don't have an HTML background, this book will be more difficult to use. I suggest that you read an HTML book — such as *HTML 4 For Dummies,* 4th Edition, by Ed Tittel and Natanya Pitts (Wiley), or *HTML 4 For Dummies Quick Reference,* 2nd Edition, by Deborah S. Ray and Eric J. Ray (Wiley) — and build some practice Web pages before you start this book. In particular, some background in HTML forms and tables is useful. However, if you're the impatient type, I won't tell you it's impossible to proceed without knowing HTML. You may be able to glean enough HTML from this book to build your particular Web site. If you choose to proceed without knowing HTML, I suggest that you have an HTML book by your side to assist you.

If you are proceeding without any experience with Web pages, you might not know some required basics. You must know how to create and save plain text files with an editor such as Notepad or save the file as plain text from your word processor (not in the word processor format). You also must know where to put the text files containing the code (HTML or PHP) for your Web pages so that the pages are available to all users with access to your Web site, and you must know how to move the files to the appropriate location.

You do *not* need to know how to design or create databases or how to program. All the information that you need to know about databases and programming is included in this book.

How This Book Is Organized

This book is divided into six parts, with several chapters in each part. The content ranges from an introduction to PHP and MySQL to installing to creating and using databases to writing PHP programs.

Part I: Developing a Web Database Application Using PHP and MySQL

Part I provides an overview of using PHP and MySQL to create a Web database application. It describes and gives the advantages of PHP, of MySQL, and of their use together. You find out how to get started, including what you need, how to get access to PHP and MySQL, and how to test your software. You then find out about the process of developing the application.

Part II: MySQL Database

In Part II you find out the details of working with MySQL databases. You create a database, change a database, and move data into and out of a database.

Part III: PHP

Part III provides the details of writing PHP programs that enable your Web pages to insert new information, update existing information, or remove information from a MySQL database. You find out how to use the PHP features that are used for database interaction and forms processing.

Part IV: Applications

Part IV describes the Web database application as a whole. You find out how to organize the PHP programs into a functioning application that interacts with the database. Two complete sample applications are provided, described, and explained.

Part V: The Part of Tens

Part V provides some useful lists of important things to do and not to do when developing a Web database application.

Part VI: Appendixes

The final part, Part VI, provides instructions for installing PHP and MySQL for those who need to install the software themselves. Appendix C discusses the installation of the Apache Web server for those who need to install and administer the Web server themselves.

Icons Used in This Book



This icon is a sticky note of sorts, highlighting information that's worth committing to memory.



This icon flags information and techniques that are more technical than other sections of the book. The information here can be interesting and helpful, but you don't need to understand it to use the information in the book.



Tips provide extra information for a specific purpose. Tips can save you time and effort, so they're worth checking out.



You should always read warnings. Warnings emphasize actions that you must take or must avoid to prevent dire consequences.

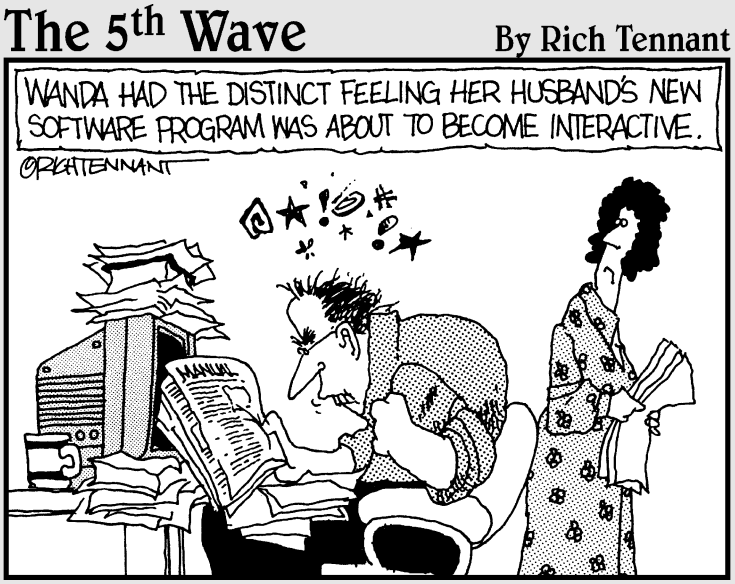
Where to Go from Here

This book is organized in the order in which things need to be done. If you're a newbie, you probably need to start with Part I, which describes how to get started, including how to design the pieces of your application and how the pieces will interact. When implementing your application, you need to create

the MySQL database first, so I discuss MySQL before PHP. After you understand the details of MySQL and PHP, you need to put them together into a complete application, which I describe in Part IV. If you're already familiar with any part of the book, you can go directly to the part that you need. For instance, if you're familiar with database design, you can go directly to Part II, which describes how to implement the design in MySQL. Or if you know MySQL, you can just read about PHP in Part III.

Part I

Developing a Web Database Application Using PHP and MySQL



In this part . . .

In this part, I provide an overview. I describe PHP and MySQL, how each one works, and how they work together to make your Web database application possible. After describing your tools, I show you how to set up your working environment. I present your options for accessing PHP and MySQL and point out what to look for in each environment.

After describing your tools and your options for your development environment, I provide an overview of the development process. I discuss planning, design, and building your application.

Chapter 1

Introduction to PHP and MySQL

In This Chapter

- Finding out what a Web database application is
- Discovering how MySQL works
- Taking a look at PHP
- Finding out how PHP and MySQL work together

So you need to develop an interactive Web site. Perhaps your boss just put you in charge of the company's online product catalog. Or you want to develop your own Web business. Or your sister wants to sell her paintings online. Or you volunteered to put up a Web site open only to members of your circus acrobats' association. Whatever your motivation might be, you can see that the application needs to store information (such as information about products or member passwords), thus requiring a database. You can see also that the application needs to interact *dynamically* with the user; for instance, the user selects a product to view or enters membership information. This type of Web site is a *Web database application.*

I assume that you've created static Web pages before, using HTML (HyperText Markup Language), but creating an interactive Web site is a new challenge, as is designing a database. You asked three computer gurus you know what you should do. They said a lot of things you didn't understand, but among the technical jargon, you heard "quick" and "easy" and "free" mentioned in the same sentence as PHP and MySQL. Now you want to know more about using PHP and MySQL to develop the Web site that you need.

PHP and MySQL work together very well; it's a dynamic partnership. In this chapter, you find out the advantages of each, how each one works, and how they work together to produce a dynamic Web database application.

What Is a Web Database Application?

An *application* is a program or a group of programs designed for use by an end user (for example, customers, members, or circus acrobats). If the end user interacts with the application via a Web browser, the application is a *Web based* or *Web application.* If the Web application requires the long-term storage of information using a database, it is a *Web database application.* This book provides you with the information that you need to develop a Web database application that can be accessed with Web browsers such as Internet Explorer and Netscape.

A Web database application is designed to help a user accomplish a task. It can be a simple application that displays information in a browser window (for example, current job openings when the user selects a job title) or a complicated program with extended functionality (for example, the book-ordering application at Amazon.com or the bidding application at eBay).

A Web database application consists of just two pieces:

- **Database:** The *database* is the long-term memory of your Web database application. The application can't fulfill its purpose without the database. However, the database alone is not enough.
- **Application:** The *application* piece is the program or group of programs that performs the tasks. Programs create the display that the user sees in the browser window; they make your application interactive by accepting and processing information that the user types in the browser window; and they store information in the database and get information out of the database. (The database is useless unless you can move data in and out.)

The Web pages that you've previously created with HTML alone are *static,* meaning the user can't interact with the Web page. All users see the same Web page. *Dynamic* Web pages, on the other hand, allow the user to interact with the Web page. Different users might see different Web pages. For instance, one user looking at a furniture store's online product catalog might choose to view information about the sofas, whereas another user might choose to view information about coffee tables. To create dynamic Web pages, you must use another language in addition to HTML.

One language widely used to make Web pages dynamic is JavaScript. JavaScript is useful for several purposes, such as mouse-overs (for example, to highlight a navigation button when the user moves the mouse pointer over it) or accepting and validating information that users type into a Web form. However, it's not useful for interacting with a database. You wouldn't use JavaScript to move the information from the Web form into a database. PHP, however, is a language particularly well suited to interacting with databases. PHP can accept and validate the information that users type into a Web form and can also move the information into a database. The programs in this book are written with PHP.

The database

The core of a Web database application is the *database,* which is the long-term memory (I hope more efficient than my long-term memory) that stores information for the application. A database is an electronic file cabinet that stores information in an organized manner so that you can find it when you need it. After all, storing information is pointless if you can't find it. A database can be small, with a simple structure — for example, a database containing the titles and authors' names of all the books that you own. Or a database can be huge, with an extremely complex structure — such as the database that Amazon.com has to hold all its information.

The information that you store in the database comes in many varieties. A company's online catalog requires a database to store information about all the company's products. A membership Web site requires a database to store information about members. An employment Web site requires a database (or perhaps two databases) to store information about job openings and information from résumés. The information that you plan to store could be similar to information that's stored by Web sites all over the Internet — or information that's unique to your application.

Technically, the term *database* refers to the file or group of files that holds the actual data. The data is accessed by using a set of programs called a DBMS (Database Management System). Almost all DBMSs these days are RDBMSs (Relational Database Management Systems), in which data is organized and stored in a set of related tables.

In this book, MySQL is the RDBMS used because it is particularly well suited for Web sites. MySQL and its advantages are discussed in the section, "MySQL, My Database," later in this chapter. You can find out how to organize and design a MySQL database in Chapter 3.

The application: Moving data into and out of the database

For a database to be useful, you need to be able to move data into and out of it. Programs are your tools for this because they interact with the database to store and retrieve data. A program connects to the database and makes a request: "Take this data and store it in the specified location." Another program makes the request: "Find the specified data and give it to me." The application programs that interact with the database run when the user interacts with the Web page. For instance, when the user clicks the submit button after filling in a Web form, a program processes the information in the form and stores it in a database.

E-mail discussion lists

Good technical support is available from *e-mail discussion lists,* which are groups of people discussing specific topics through e-mail. E-mail lists are available for pretty much any subject you can think of: Powerball, ancient philosophy, cooking, the Beatles, Scottish terriers, politics, and so on. The *list manager* maintains a distribution list of e-mail addresses for anyone who wants to join the discussion. When you send a message to the discussion list, your message is sent to the entire list so that everyone can see it. Thus, the discussion is a group effort, and anyone can respond to any message that interests him or her.

E-mail discussion lists are supported by various sponsors. Any individual or organization can run a list. Most software vendors run one or more lists devoted to their software. Universities run many lists for educational subjects. In addition, some Web sites manage discussion lists, such as Yahoo! Groups and Topica. Users can create a new list or join an existing list through the Web application.

Software-related e-mail lists are a treasure trove of technical support. Anywhere from a hundred to several thousand users of the software subscribe to the list. Often the developers, programmers, and technical support staff for the software vendor are on the list. You are unlikely to be the first person to ever experience your problem. Whatever your question or problem, someone on the list probably knows the answer or the solution. When you post a question to an e-mail list, the answer usually appears in your inbox within minutes. In addition, most lists maintain an archive of previous discussions so that you can search for answers. When you're new to any software, you can find out a great deal simply by joining the discussion list and reading the messages for a few days.

PHP and MySQL have e-mail discussion lists. Actually, each has several discussion lists for special topics, such as *databases and PHP.* You can find the names of the mailing lists and instructions for joining them on the PHP and MySQL Web sites.

MySQL, My Database

MySQL is a fast, easy-to-use RDBMS used on many Web sites. Speed was the developers' main focus from the beginning. In the interest of speed, they made the decision to offer fewer features than their major competitors (such as Oracle and Sybase). However, even though MySQL is less full-featured than its commercial competitors, it has all the features needed by the majority of database developers. It's easier to install and use than its commercial competitors, and the difference in price is strongly in MySQL's favor.

MySQL is developed, marketed, and supported by MySQL AB, which is a Swedish company. The company licenses it in two ways:

- **Open source software:** MySQL is available through the GNU GPL (General Public License). MySQL provides two versions of the open source software:

- **MySQL Community Edition:** A freely downloadable, open source edition of MySQL, released early and often with the most advanced features. Anyone who can meet the requirements of the GPL can use the software for free. If you're using MySQL as a database on a Web site (the subject of this book), you can use MySQL for free, even if you're making money with your Web site.
- **MySQL Network:** An enterprise-grade set of software and services available for a monthly subscription fee. MySQL Network provides certified software, thoroughly tested and optimized. Services include technical support, regular updates, access to a knowledge base of hundreds of technical articles, and other services useful to a large business. The subscription is available at four levels, from the Basic level, with a limit of two incidents, no phone support, and a two-day response time, to Platinum support, with unlimited incidents, 24/7 phone support, and a 30-minute response time.

✓ **Commercial license:** MySQL is available with a commercial license for those who prefer it to the GPL. If a developer wants to use MySQL as part of a new software product and wants to sell the new product rather than release it under the GPL, the developer needs to purchase a commercial license.

Finding technical support for MySQL Community Edition is not a problem. You can join one of several e-mail discussion lists offered on the MySQL Web site at `www.mysql.com`. You can even search the e-mail list archives, which contain a large archive of MySQL questions and answers.

Advantages of MySQL

MySQL is a popular database with Web developers. Its speed and small size make it ideal for a Web site. Add to that the fact that it's open source, which means free, and you have the foundation of its popularity. Here is a rundown of some of its advantages:

✓ **It's fast.** The main goal of the folks who developed MySQL was speed. Thus, the software was designed from the beginning with speed in mind.

✓ **It's inexpensive.** MySQL is free under the open source GPL license, and the fee for a commercial license is reasonable.

✓ **It's easy to use.** You can build and interact with a MySQL database by using a few simple statements in the SQL language, which is the standard language for communicating with RDBMSs. Check out Chapter 4 for the lowdown on the SQL language.

- **It can run on many operating systems.** MySQL runs on many operating systems — Windows, Linux, Mac OS, most varieties of Unix (including Solaris and AIX), FreeBSD, OS/2, Irix, and others.
- **Technical support is widely available.** A large base of users provides free support through mailing lists. The MySQL developers also participate in the e-mail lists. You can also purchase technical support from MySQL AB for a small fee.
- **It's secure.** MySQL's flexible system of authorization allows some or all database privileges (such as the privilege to create a database or delete data) to specific users or groups of users. Passwords are encrypted.
- **It supports large databases.** MySQL handles databases up to 50 million rows or more. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- **It's customizable.** The open source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

How MySQL works

The MySQL software consists of the MySQL server, several utility programs that assist in the administration of MySQL databases, and some supporting software that the MySQL server needs (but you don't need to know about). The heart of the system is the MySQL server.

The MySQL server is the manager of the database system. It handles all your database instructions. For instance, if you want to create a new database, you send a message to the MySQL server that says "create a new database and call it newdata." The MySQL server then creates a subdirectory in its data directory, names the new subdirectory `newdata`, and puts the necessary files with the required format into the `newdata` subdirectory. In the same manner, to add data to that database, you send a message to the MySQL server, giving it the data and telling it where you want the data to be added. You find out how to write and send messages to MySQL in Part II.

Before you can pass instructions to the MySQL server, it must be running and waiting for requests. The MySQL server is usually set up so that it starts when the computer starts and continues running all the time. This is the usual setup for a Web site. However, it's not necessary to set it up to start when the computer starts. If you need to, you can start it manually whenever you want to access a database. When it's running, the MySQL server listens continuously for messages that are directed to it.

Communicating with the MySQL server

All your interaction with the database is accomplished by passing messages to the MySQL server. You can send messages to the MySQL server several ways, but this book focuses on sending messages using PHP. The PHP software has specific statements that you use to send instructions to the MySQL server.

The MySQL server must be able to understand the instructions that you send it. You communicate by using *SQL (Structured Query Language),* which is a standard language understood by many RDBMSs. The MySQL server understands SQL. PHP doesn't understand SQL, but it doesn't need to: PHP just establishes a connection with the MySQL server and sends the SQL message over the connection. The MySQL server interprets the SQL message and follows the instructions. The MySQL server sends a return message, stating its status and what it did (or reporting an error if it was unable to understand or follow the instructions). For the lowdown on how to write and send SQL messages to MySQL, check out Part II.

PHP, a Data Mover

PHP, a scripting language designed specifically for use on the Web, is your tool for creating dynamic Web pages. Rich in features that make Web design and programming easier, PHP is in use on more than 20 million domains (according to the Netcraft survey at `www.php.net/usage.php`). Its popularity continues to grow, so it must be fulfilling its function pretty well.

PHP stands for *PHP: HyperText Preprocessor.* In its early development by a guy named Rasmus Lerdorf, it was called *Personal Home Page tools.* When it developed into a full-blown language, the name was changed to be more in line with its expanded functionality.



The PHP language's syntax is similar to the syntax of C, so if you have experience with C, you'll be comfortable with PHP. PHP is actually simpler than C because it doesn't use some of the more difficult concepts of C. PHP also doesn't include the low-level programming capabilities of C because PHP is designed to program Web sites and doesn't require those capabilities.

PHP is particularly strong in its ability to interact with databases. PHP supports pretty much every database you've ever heard of (and some you haven't). PHP handles connecting to the database and communicating with it. You don't need to know the technical details for connecting to a database or

for exchanging messages with it. You tell PHP the name of the database and where it is, and PHP handles the details. It connects to the database, passes your instructions to the database, and returns the database response to you.

Technical support is available for PHP. You can join one of several e-mail discussion lists offered on the PHP Web site (`www.php.net`), including a list for *databases and PHP*. In addition, a Web interface to the discussion lists is available at `news.php.net`, where you can browse or search the messages.

Advantages of PHP

The popularity of PHP is growing rapidly because of its many advantages:

- **It's fast.** Because it is embedded in HTML code, the response time is short.
- **It's inexpensive — free, in fact.** PHP is proof that free lunches do exist and that you can get more than you paid for.
- **It's easy to use.** PHP contains many special features and functions needed to create dynamic Web pages. The PHP language is designed to be included easily in an HTML file.
- **It can run on many operating systems.** It runs on a variety of operating systems — Windows, Linux, Mac OS, and most varieties of Unix.
- **Technical support is widely available.** A large base of users provides free support through e-mail discussion lists.
- **It's secure.** The user does not see the PHP code.
- **It's designed to support databases.** PHP includes functionality designed to interact with specific databases. It relieves you of the need to know the technical details required to communicate with a database.
- **It's customizable.** The open source license allows programmers to modify the PHP software, adding or modifying features as needed to fit their own specific environments.

How PHP works

PHP is an embedded scripting language when used in Web pages. This means that PHP code is embedded in HTML code. You use HTML tags to enclose the PHP language that you embed in your HTML file — the same way that you would use other HTML tags. You create and edit Web pages containing PHP the same way that you create and edit regular HTML pages.

The PHP software works with the Web server. The Web server is the software that delivers Web pages to the world. When you type a URL into your Web browser, you're sending a message to the Web server at that URL, asking it to send you an HTML file. The Web server responds by sending the requested file. Your browser reads the HTML file and displays the Web page. You also request the Web server to send you a file when you click a link in a Web page. In addition, the Web server processes a file when you click a Web page button that submits a form.

When PHP is installed, the Web server is configured to expect certain file extensions to contain PHP language statements. Often the extension is `.php` or `.phtml`, but any extension can be used. When the Web server gets a request for a file with the designated extension, it sends the HTML statements as-is, but PHP statements are processed by the PHP software before they're sent to the requester.

When PHP language statements are processed, only the output is sent by the Web server to the Web browser. The PHP language statements are not included in the output sent to the browser, so the PHP code is secure and transparent to the user. For instance, in this simple PHP statement:

```
<?php echo "<p>Hello World"; ?>
```

`<?php` is the PHP opening tag, and `?>` is the closing tag. `echo` is a PHP instruction that tells PHP to output the upcoming text. The PHP software processes the PHP statement and outputs this:

```
<p>Hello World
```

which is a regular HTML statement. This HTML statement is delivered to the user's browser. The browser interprets the statement as HTML code and displays a Web page with one paragraph — Hello World. The PHP statement is not delivered to the browser, so the user never sees any PHP statements. PHP and the Web server must work closely together.

PHP is not integrated with all Web servers but does work with many of the popular Web servers. PHP is developed as a project of the Apache Software Foundation — thus, it works best with Apache. PHP also works with Microsoft IIS/PWS, iPlanet (formerly Netscape Enterprise Server), and others.



Although PHP works with several Web servers, it works best with Apache. If you can select or influence the selection of the Web server used in your organization, select Apache. By itself, Apache is a good choice. It is free, open source, stable, and popular. It currently powers more than 60 percent of all Web sites, according to the Web server survey at `www.netcraft.com`. It runs on Windows, Linux, Mac OS, and most flavors of Unix.

MySQL and PHP, the Perfect Pair

MySQL and PHP are frequently used together. They are often called the *dynamic duo.* MySQL provides the database part, and PHP provides the application part of your Web database application.

Advantages of the relationship

MySQL and PHP as a pair have several advantages:

- **They're free.** It's hard to beat free for cost-effectiveness.
- **They're Web oriented.** Both were designed specifically for use on Web sites. Both have a set of features focused on building dynamic Web sites.
- **They're easy to use.** Both were designed to get a Web site up quickly.
- **They're fast.** Both were designed with speed as a major goal. Together they provide one of the fastest ways to deliver dynamic Web pages to users.
- **They communicate well with one another.** PHP has built-in features for communicating with MySQL. You don't need to know the technical details; just leave it to PHP.
- **A wide base of support is available for both.** Both have large user bases. Because they are often used as a pair, they often have the same user base. Many people are available to help, including those on e-mail discussion lists who have experience using MySQL and PHP together.
- **They're customizable.** Both are open source, thus allowing programmers to modify the PHP and MySQL software to fit their own specific environments.

How MySQL and PHP work together

PHP provides the application part, and MySQL provides the database part of a Web database application. You use the PHP language to write the programs that perform the application tasks. PHP can be used for simple tasks (such as displaying a Web page) or for complicated tasks (such as accepting and verifying data that a user typed into an HTML form). One of the tasks that your application must do is move data into and out of the database — and PHP has built-in features to use when writing programs that move data into and out of a MySQL database.

PHP statements are embedded in your HTML files with PHP tags. When the task to be performed by the application requires storing or retrieving data, you use specific PHP statements designed to interact with a MySQL database. You use one PHP statement to connect to the correct database, telling PHP where the database is located, its name, and the password needed to connect to it. The database doesn't need to be on the same machine as your Web site; PHP can communicate with a database across a network. You use another PHP statement to send an SQL message to MySQL, giving MySQL instructions for the task you want to accomplish. MySQL returns a status message that shows whether it successfully performed the task. If there was a problem, it returns an error message. If your SQL message asked to retrieve some data, MySQL sends the data that you asked for, and PHP stores it in a temporary location where it is available to you.

You then use one or more PHP statements to complete the application task. For instance, you can use PHP statements to display data that you retrieved. Or you might use PHP statements to display a status message in the browser, informing the user that the data was saved.

As an RDBMS, MySQL can store complex information. As a scripting language, PHP can perform complicated manipulations of data, on either data that you need to modify before saving it in the database or data that you retrieved from the database and need to modify before displaying or using it for another task. Together, PHP and MySQL can be used to build a sophisticated and complicated Web database application.

Keeping Up with PHP and MySQL Changes

PHP and MySQL are open source software. If you've used only software from major software publishers — such as Microsoft, Macromedia, or Adobe — you'll find that open source software is an entirely different species. It's developed by a group of programmers who write the code in their spare time, for fun and for free. There's no corporate office.

Open source software changes frequently, rather than once every year or two like commercial software does. It changes when the developers feel that it's ready. It also changes quickly in response to problems. When a serious problem is found — such as a security hole — a new version that fixes the problem can be released in days. You don't receive glossy brochures or see splashy magazine ads for a year before a new version is released. Thus, if you don't make the effort to stay informed, you could miss the release of a new version or be unaware of a serious problem with your current version.

Visit the PHP and MySQL Web sites often. You need to know the information that's published there. Join the mailing lists, which often are high in traffic. When you first get acquainted with PHP and MySQL, the large number of mail messages on the discussion lists brings valuable information into your e-mail box; you can pick up a lot by reading those messages. And soon, you might be able to help others based on your own experience. At the very least, subscribe to the announcement mailing list, which delivers e-mail only occasionally. Any important problems or new versions are announced here. The e-mail that you receive from the announcement list contains information you need to know. So, right now, before you forget, hop over to the PHP and MySQL Web sites and sign up for a list or two at `www.php.net/mailing-lists.php` and `lists.mysql.com`.



You should be aware of some significant changes in previous PHP versions because existing scripts that work fine on earlier versions could have problems when they're run on a later version and vice versa. The following are some changes that you should be aware of:

- **Version 6.0.0:** Removed configuration setting for `register_globals`. Removed configuration setting for magic quotes. Removed the long arrays, such as `$HTTP_POST_VARS`.
- **Version 5.1.0:** Added a configuration setting to set a local time zone. If a time zone is not set, a notice is displayed.
- **Version 5.0.0:** Added support for MySQL 4.1 and later versions. Support for MySQL is not included automatically; it must be included with an option when PHP is installed. Also changed the filename of the PHP interpreter used with a Web server from `.php` to `.php-cgi`.
- **Version 4.3.1:** Fixed a security problem in 4.3.0. It's not wise to continue to run a Web site using version 4.3.0 or earlier.
- **Version 4.2.0:** Changed the default setting for `register_globals` to `Off`. Scripts running under previous versions might depend on `register_globals` being set to `On` and could stop running with the new setting. It's best to change the coding of the script so that it runs with `register_globals` set to `Off`.
- **Version 4.1.0:** Introduced superglobal arrays. Scripts written with the superglobals won't run in earlier versions. Prior to 4.1.0, you must use old style arrays, such as `$HTTP_POST_VARS`.

Chapter 2

Setting Up Your Work Environment

In This Chapter

- Accessing PHP and MySQL through company Web sites and Web hosting companies
- Building your own Web site from scratch
- Testing PHP and MySQL

After you decide to use PHP and MySQL, your first task is to get access to them. A work setting already set up for Web application development might be ready and waiting for you with all the tools that you need. On the other hand, it might be part of your job to set up this work setting yourself. Perhaps your job is to create an entire new Web site. In this chapter, I describe the tools you need and how to get access to them.

The Required Tools

To put up your dynamic Web site, you need to have access to the following three software tools:

- **A Web server:** The software that delivers your Web pages to the world
- **MySQL:** The RDBMS (Relational Database Management System) that will store information for your Web database application
- **PHP:** The scripting language that you'll use to write the programs that provide the dynamic functionality for your Web site



I describe these three tools in detail in Chapter 1.

Finding a Place to Work

To create your dynamic Web pages, you need access to a Web site that provides your three software tools (see the preceding section). All Web sites include a Web server, but not all Web sites provide MySQL and PHP. These are the most common environments in which you can develop your Web site:

- **A Web site put up by a company on its own computer:** The company — usually the company's IT (Information Technology) department — installs and administers the Web site software. Your job, for the purposes of this book, is to program the Web site, either as an employee of the company or as a contractor.
- **A Web site hosted by a Web hosting company:** The Web site is located on the Web hosting company's computer. The Web hosting company installs and maintains the Web site software and provides space on its computer where you can install the HTML (HyperText Markup Language) files for a Web site.
- **A Web site that doesn't yet exist:** You plan to install and maintain the Web site software yourself. It could be a Web site of your own that you're building on your own computer, or it might be a Web site that you're installing for a client on the client's computer.

How much you need to understand about the administration and operation of the Web site software depends on the type of Web site access you have. In the next few sections, I describe these environments in more detail and explain how you gain access to PHP and MySQL.

A company Web site

When the Web site is run by the company, you don't need to understand the installation and administration of the Web site software at all. The company is responsible for the operation of the Web site. In most cases, the Web site already exists, and your job is to add to, modify, or redesign the existing Web site. In a few cases, the company might be installing its first Web site, and your job is to design the Web site. In either case, your responsibility is to write and install the HTML files for the Web site. You are not responsible for the operation of the Web site.

You access the Web site software through the company's IT department. The name of this department can vary in different companies, but its function is the same: It keeps the company's computers running and up to date.

If PHP or MySQL or both aren't available on the company's Web site, IT needs to install them and make them available to you. PHP and MySQL have many options, but IT might not understand the best options — and might have options set in ways that aren't well suited for your purposes. If you need PHP

or MySQL options changed, you need to request that IT make the change; you won't be able to make the change yourself. For instance, PHP must be installed with MySQL support enabled, so if PHP isn't communicating correctly with MySQL, IT might have to reinstall PHP with MySQL support enabled.

For the world to see the company's Web pages, the HTML files must be in a specific location on the computer. The Web server that delivers the Web pages to the world expects to find the HTML files in a specific directory. The IT department should provide you with access to the directory where the HTML files need to be installed. In most cases, you develop and test your Web pages in a test location and then transfer the completed files to their permanent home. Depending on the access that IT gives you, you might copy the files from the test location to the permanent location, or you might transfer the files via FTP (File Transfer Protocol), which is a method of copying a file from one computer to another on a network. In some cases, for security reasons, the IT folks won't give you access to the permanent location, preferring to install the files in their permanent location themselves.

To use the Web software tools and build your dynamic Web site, you need the following information from IT:

- **The location of Web pages:** You need to know where to put the files for the Web pages. IT needs to provide you with the name and location of the directory where the files should be installed. Also, you need to know how to install the files — copy them, FTP them, or use other methods. You might need a user ID and password to install the files.
- **The default filename:** When users point their browsers at a URL, a file is sent to them. The Web server is set up to send a file with a specific name when the URL points to a directory. The file that is automatically sent is the *default file.* Very often the default file is named `index.htm` or `index.html`, but sometimes other names are used, such as `default.htm`. Ask IT what you should name your default file.
- **A MySQL account:** Access to MySQL databases is controlled through a system of account names and passwords. IT sets up a MySQL account for you that has the appropriate permissions and also gives you the MySQL account name and password. (I explain MySQL accounts in detail in Chapter 5.)
- **The location of the MySQL databases:** MySQL databases need not be located on the same computer as the Web site. If the MySQL databases are located on a computer other than that of the Web site, you need to know the *hostname* (for example, `thor.companyname.com`) where the databases can be found.
- **The PHP file extension:** When PHP is installed, the Web server is instructed to expect PHP statements in files with specific extensions. Frequently, the extensions used are `.php` or `.phtml`, but other extensions can be used. PHP statements in files that don't have the correct extension won't be processed. Ask IT what extension to use for your PHP programs.



You will interact with the IT folks frequently as needs arise. For example, you might need options changed, you might need information to help you interpret an error message, or you might need to report a problem with the Web site software. So a good relationship with the IT folks will make your life much easier. Bring them tasty cookies and doughnuts often.

A Web hosting company

A *Web hosting company* provides everything that you need to put up a Web site, including the computer space and all the Web site software. You just create the files for your Web pages and move them to a location specified by the Web hosting company.

About a gazillion companies offer Web hosting services. Most charge a monthly fee (often quite small), and some are even free. (Most, but not all, of the free ones require you to display advertising.) Usually, the monthly fee varies depending on the resources provided for your Web site. For instance, a Web site with 2MB of disk space for your Web page files costs less than a Web site with 10MB of disk space.

When looking for a place to host your Web site, make sure that the Web hosting company offers the following:

- **PHP and MySQL:** Not all companies provide these tools. You might have to pay more for a site with access to PHP and MySQL; sometimes you have to pay an additional fee for MySQL databases.
- **A recent version of PHP:** Sometimes the PHP versions offered aren't the most recent versions. As of this writing, PHP 6 is close to being released. However, you may have trouble finding a Web hosting company that offers PHP 6. In fact, you may find that most Web hosting companies still offer PHP 4, although I hope that will change over time. It is worth the time to find a Web hosting company that offers at least PHP 5, if not PHP 6. Some Web hosting companies offer PHP 4 but have PHP 5 or 6 available for customers who request it.

Other considerations when choosing a Web hosting company are

- **Reliability:** You need a Web hosting company that you can depend on — one that won't go broke and disappear tomorrow, and one that isn't running on old computers, held together by chewing gum and baling wire, with more downtime than uptime.
- **Speed:** Web pages that download slowly are a problem because users will get impatient and go elsewhere. Slow pages could be a result of a Web hosting company that started its business on a shoestring and has a shortage of good equipment — or the Web hosting company might be so successful that its equipment is overwhelmed by new customers. Either way, Web hosting companies that deliver Web pages too slowly are unacceptable.

- **Technical support:** Some Web hosting companies have no one available to answer questions or troubleshoot problems. Technical support is often provided only through e-mail, which can be acceptable if the response time is short. Sometimes you can test the quality of the company's support by calling the tech support number, or test the e-mail response time by sending an e-mail.
- **The domain name:** Each Web site has a domain name that Web browsers use to find the site on the Web. Each domain name is registered for a small yearly fee so that only one Web site can use it. Some Web hosting companies allow you to use a domain name that you have registered independently of the Web hosting company, some assist you in registering and using a new domain name, and some require that you use their domain name. For instance, suppose that your name is Lola Designer and you want your Web site to be named LolaDesigner. Some Web hosting companies will allow your Web site to be `LolaDesigner.com`, but some will require that your Web site be named `LolaDesigner.`*`webhostingcompanyname`*`.com`, or *`webhostingcompanyname`*`.com/~LolaDesigner`, or something similar. In general, your Web site will look more professional if you use your own domain name.
- **Backups:** *Backups* are copies of your Web page files and your database that are stored in case your files or database are lost or damaged. You want to be sure that the company makes regular, frequent backup copies of your application. You also want to know how long it would take for backups to be put in place to restore your Web site to working order after a problem.
- **Features:** Select features based on the purpose of your Web site. Usually a hosting company bundles features together into plans — more features equal a higher cost. Some features to consider are
  - **Disk space:** How many MB or GB of disk space will your Web site require? Media files, such as graphics or music files, can be quite large.
  - **Data transfer:** Some hosting companies charge you for sending Web pages to users. If you expect to have a lot of traffic on your Web site, this cost should be a consideration.
  - **E-mail addresses:** Many hosting companies provide you with a number of e-mail addresses for your Web site. For instance, if your Web site is `LolaDesigner.com`, you could allow users to send you e-mail at `me@LolaDesigner.com`.
  - **Software:** Hosting companies offer access to a variety of software for Web development. PHP and MySQL are the software that I discuss in this book. Some hosting companies might offer other databases, and some might offer other development tools such as FrontPage extensions, shopping cart software, and credit card validation.
  - **Statistics:** Often you can get statistics regarding your Web traffic, such as the number of users, time of access, access by Web page, and so on.

One disadvantage of hosting your site with a commercial Web hosting company is that you have no control over your development environment. The Web hosting company provides the environment that works best for it — probably setting up the environment for ease of maintenance, low cost, and minimal customer defections. Most of your environment is set by the company, and you can't change it. You can only beg the company to change it. The company will be reluctant to change a working setup, fearing that a change could cause problems for the company's system or for other customers.

Access to MySQL databases is controlled via a system of accounts and passwords that must be maintained manually, thus causing extra work for the hosting company. For this reason, many hosting companies either don't offer MySQL or charge extra for it. Also, PHP has myriad options that can be set, unset, or given various values. The hosting company decides the option settings based on its needs, which might or might not be ideal for your purposes.



It's pretty difficult to research Web hosting companies from a standing start — a search at Google.com for *"Web hosting"* results in almost 400 million hits. The best way to research Web hosting companies is to ask for recommendations from people who have experience with those companies. People who have used a hosting company can warn you if the service is slow or the computers are down often. After you gather a few names of Web hosting companies from satisfied customers, you can narrow the list to the one that is best suited to your purposes and the most cost effective.

Setting up and running a Web site on your local computer

If you're starting a Web site from scratch, you need to understand the Web site software fairly well. You have to make several decisions regarding hardware and software. You have to install a Web server, PHP, and MySQL — as well as maintain, administer, and update the system yourself. Taking this route, rather than using a Web site provided by others, requires more work and more knowledge. The advantage is that you have total control over the Web development environment.

You may want to set up a Web site on your personal computer to be a public Web site, accessed from the World Wide Web. Or you may want to set up a Web site on your personal computer where you can develop your Web site before transferring the Web page files to your work computer or a Web hosting company. These two types of use require different Internet connections, as described in Step 1 next.

Here are the general steps that lead to your dynamic Web site (I explain these steps in more detail in the next few sections):

1. **Set up the computer.**
2. **Install the Web server.**
3. **Install MySQL.**
4. **Install PHP.**

If you're starting from scratch, with nothing but an empty space where the computer will go, start at Step 1. If you already have a running computer but no Software, start at Step 2. Or if you have an existing Web site that does not have PHP and MySQL installed, start with Step 3.



Domain names

Every Web site needs a unique address on the Web. The unique address used by computers to locate a Web site is the *IP address,* which is a series of four numbers between 0 and 255, separated by dots — for example, `172.17.204.2` or `192.163.2.33`.

Because IP addresses are made up of numbers and dots, they're not easy to remember. Fortunately, most IP addresses have an associated name that's much easier to remember, such as `amazon.com`, `www.irs.gov`, or `mycompany.com`. A name that's an address for a Web site is a *domain name.* A *domain* can be one computer or many connected computers. When a domain refers to several computers, each computer in the domain can have its own name. A name that includes an individual computer name, such as `thor.mycompany.com`, identifies a *subdomain.*

Each domain name must be unique in order to serve as an address. Consequently, a system of registering domain names ensures that no two locations use the same domain name. Anyone can register any domain name as long as the name isn't already taken. You can register a domain name on the Web. First, you test your potential domain name to find out whether it's available. If it's available, you register it in your name or a company name and pay the fee. The name is then yours to use, and no one else can use it. The standard fee for domain name registration is $35 per year. You should never pay more, but bargains are often available.

Many Web sites provide the ability to register a domain name, including the Web sites of many Web hosting companies. A search at Google (`www.google.com`) for *register domain name* results in more than 85 million hits. Shop around to be sure that you find the lowest price. Also, many Web sites allow you to enter a domain name and see whom it is registered to. These Web sites do a domain name database search using a tool called *whois.* A search at Google for *domain name whois* results in more than 17 million hits. A couple of places where you can do a whois search are Allwhois.com (`www.allwhois.com`) and BetterWhois.com (`www.betterwhois.com`).

Setting up the computer

Your first decision is to choose which hardware platform and operating system to use. In most cases, you'll choose a PC with either Linux or Windows as the operating system. Here are some advantages and disadvantages of these two operating systems:

- **Linux:** Linux is open source, so it's free. It also has advantages for use as a Web server: It runs for long periods without needing to be rebooted; and Apache, the most popular Web server, runs better on Linux than Windows. Running Linux on a PC is the lowest cost option. The disadvantage of running Linux is that many people find Linux more difficult to install, configure, administer, and install software on than Windows, although Linux is getting easier to install every day.
- **Windows:** Unlike Linux, Windows is not free. However, most people feel that Windows is easier to use, and because it's widely used, many people can help you if you have problems. If you plan a public Web site, with users accessing your Web site from the WWW, you need Windows 2000 or later.

I assume that you're buying a computer with the operating system and software installed, ready to use. It's easier to find a computer that comes with Windows installed on it than with Linux, but Linux computers are available. For instance, at this time, Dell, IBM, and Hewlett-Packard offer computers with Linux installed.

If you're building your own hardware, you need more information than I have room to provide in this book. If you have the hardware and plan to install an operating system, Windows is easier to install, but Linux is getting easier all the time. You can install Linux from a CD, like Windows, but you often must provide information or make decisions that require more knowledge about your system. If you already know how to perform system administration tasks (such as installing software and making backups) in Windows or in Linux, the fastest solution is to use the operating system that you already know.



For using PHP and MySQL, you should seriously consider Linux. PHP is a project of the Apache Software Foundation, so it runs best with the Apache server. And Apache runs better on Linux than on Windows. Therefore, if all other things are equal and the computer is mainly for running a Web site with a Web database application, Linux is well suited for your purposes.

Other solutions besides a PC with Windows or Linux are available, but they're less popular:

- **Unix-based:** Other free, Unix-based operating systems are available for PCs, such as FreeBSD (which some people prefer to Linux) or a version of Solaris provided by Sun for free download.

- **Mac:** Mac computers can be used as Web servers. Most newer Macs come with PHP installed. Installing PHP and MySQL on Mac OS X is fairly simple. There are fewer Mac users, however, so it can be difficult to find help when you need it. One good site is `www.phpmac.com`.

Your computer must be connected to the Internet. In most cases, you obtain an account from an Internet service provider (ISP). When you obtain an account from the ISP, be sure to discuss the type of use you intend. A simple user connection to the Internet is sufficient to transfer Web page files from your development computer to a Web hosting company. However, if you plan for users to access your Web site from the WWW, you need an Internet connection with more resources.

Installing the Web server

After you set up the computer, you need to decide which Web server to install. The answer is almost always Apache. Apache offers the following advantages:

- **It's free.** What else do I need to say?
- **It runs on a variety of operating systems.** Apache runs on Windows, Linux, Mac OS, FreeBSD, and most varieties of Unix.
- **It's popular.** Approximately 60 percent of Web sites on the Internet use Apache, according to surveys at `news.netcraft.com/archives/web_server_survey.html` and `www.securityspace.com/s_survey/data/`. This wouldn't be true if it didn't work well. Also, this means that a large group of users can provide help.
- **It's reliable.** After Apache is up and running, it should run as long as your computer runs. Emergency problems with Apache are rare.
- **It's customizable.** The open source license allows programmers to modify the Apache software, adding or modifying modules as needed to fit their own environment.
- **It's secure.** Free software is available that runs with Apache to make it into an SSL (Secure Sockets Layer) server. Security is an essential issue if you're using the site for e-commerce.



Apache is automatically installed when you install most Linux distributions. All recent Macs come with Apache installed. For most other Unix flavors, you have to download the Apache source code and compile it yourself, although some *binaries* (programs that are already compiled for specific operating systems) are available. For Windows, Apache provides an installer that asks you questions and installs and configures Apache for you. For a public Web site, you should run Windows NT/2000/XP, although Apache also runs on Windows 95/98/Me. As of this writing, Apache 1.3.36, 2.0.58, and 2.2.2 are the current stable releases. (Information on Apache versions and instructions for installing Apache are provided in Appendix C.) The Apache Web site (`httpd.apache.org`) provides information, software downloads, extensive documentation that is improving all the time, and installation instructions for various operating systems.

Other Web servers are available. Microsoft offers *IIS (Internet Information Server),* which is the second most popular Web server on the Internet with approximately 27 percent of Web sites. Sun offers a Web server, which serves less than 3 percent of the Internet. Other Web servers are available, but they have even smaller user bases.

Installing MySQL

After setting up the computer and installing the Web server, you're ready to install MySQL. You need to install MySQL before installing PHP because you may need to provide the path to the MySQL software when you install PHP.

But before installing MySQL, be sure that you actually need to install it. It might already be running on your computer, or it might be installed but not running. For instance, many Linux distributions automatically install MySQL. Here's how to check whether MySQL is currently running:



- **Linux/Unix/Mac:** At the command line, type the following:

  ```
  ps -ax
  ```

  The output should be a list of programs. Some operating systems (usually flavors of Unix) have different options for the `ps` command. If the preceding does not produce a list of programs that are running, type **man ps** to see which options you need to use.

  In the list of programs that appears, look for one called `mysqld`.

- **Windows:** If MySQL is running, it will be running as a service. To check this, choose Start➪Control Panel➪Administrative Tools➪Services and scroll down the alphabetical list of services. If MySQL is installed as a service, it will appear in the list. If it's currently running, its status displays *Started*.

Even if MySQL isn't currently running, it might be installed but just not started. Here's how to check to see whether MySQL is installed on your computer:

- **Linux/Unix/Mac:** Type the following:

  ```
  find / -name "mysql*"
  ```

  If a directory named `mysql` is found, MySQL has been installed.

- **Windows:** If you did not find MySQL in the list of current services, look for a MySQL directory or files. You can search at Start➪Search. The default installation directory is `C:\Program Files\MySQL\MySQL Server` *`versionnumber`* for recent versions or `C:\mysql` for older versions.

  If you found MySQL in the service list, as described, but it is not started, you can start it by highlighting MySQL in the service list and clicking Start the Service in the left panel.

If you find MySQL on your computer but did not find it in the list of running programs (Linux/Unix/Mac) or the list of current services (Windows), here's how to start it:

- **Linux/Unix/Mac:**



1. **Change to the directory `mysql/bin`.**

   This is the directory that you should have found when you were checking whether MySQL was installed.

2. **Type** safe_mysqld &**.**

   When this command finishes, the prompt is displayed.

3. **Check that the MySQL server started by typing** ps -ax**.**

   In the list of programs that appears, look for one called `mysqld`.

- **Windows:**

1. **Open a Command Prompt window.**

   In Windows XP, choose Start➪All Programs➪Accessories➪ Command Prompt.

2. **Change to the folder where MySQL is installed.**

   For example, type **cd C:\Program Files\MySQL\MySQL Server 5.0**. Your cursor is now located in the MySQL folder.

3. **Change to the bin subfolder by typing** cd bin**.**

   Your cursor is now located in the bin subfolder.

4. **Start the MySQL Server by typing** mysqld –install**.**

   The MySQL server starts as a Windows service. You can check the installation by going to the service list, as described previously, and making sure that MySQL now appears in the service list and its status is Started.



If MySQL isn't installed on your computer, you need to download it and install it from `www.mysql.com`. The Web site provides all the information and software that you need. (You can find detailed installation instructions in Appendix A.)

Installing PHP

After you install MySQL, you're ready to install PHP. As I mention earlier, you must install MySQL before you install PHP because you may need to provide the path to the MySQL software when you install PHP. If PHP isn't compiled with MySQL support when it is installed, it won't communicate with MySQL.

Before you install PHP, check whether it's already installed. For instance, some Linux and Mac distributions automatically install PHP. To see whether PHP is installed, search your disk for any PHP files:

- **Linux/Unix/Mac:** Type the following:

```
find / -name "php*"
```

- **Windows:** Use the Find feature (choose Start➪Find) to search for *php**.

If you find PHP files, PHP is already installed, and you might not need to reinstall it. For instance, even if you installed MySQL yourself after PHP was installed, you might have installed it in the location where PHP is expecting it. Better safe than sorry, however: Perform the testing that I describe in the next section to see whether MySQL and PHP are working correctly together.

If you don't find any PHP files, PHP is not installed. To install PHP, you need access to the Web server for your site. For instance, when you install PHP with Apache, you need to edit the Apache configuration file. All the information and software that you need is provided on the PHP Web site (`www.php.net`). I provide detailed installation instructions in Appendix B.

Testing, Testing, 1, 2, 3

Suppose you believe that PHP and MySQL are available for you to use, for one of the following reasons:

- The IT department at your company or your client company gave you all the information that you asked for and told you that you're good to go.
- The Web hosting company gave you all the information that you need and told you that you're good to go.
- You followed all the instructions and installed PHP and MySQL yourself.

Now you need to test to make sure that PHP and MySQL are working correctly.

Understanding PHP/MySQL functions

PHP can communicate with any version of MySQL. However, PHP needs to be installed differently, depending on which version of MySQL you're using. PHP provides one set of functions (`mysql` functions) that communicate with MySQL 4.0 or earlier and a different set of functions (`mysqli` functions) that

communicate with MySQL 4.1 or later. The `mysql` functions, which communicate with earlier versions of MySQL, can also communicate with the later versions of MySQL, but you may not be able to use some of the newer, advanced features that were added to MySQL in the later versions. The `mysqli` functions, which can take advantage of all the MySQL features, are available only with PHP 5 or later.

The programs in this book, including the test programs in this section, use MySQL 5.0 and the `mysqli` functions. If you're using PHP 4, you need to change the programs to use the `mysql` functions, rather than the `mysqli` functions. The functions are similar, but some have slight changes in syntax. Chapter 8 provides a table (Table 8-1) showing the differences between the functions used in this book. Versions of the programs that will run with PHP 4 are available for download at my Web site (`janet.valade.com`).

You might see an error message similar to the following:

```
Fatal error: Call to undefined function mysql_connect()
```

The message means that you're using a `mysql` function in your program, but the `mysql` functions are not enabled. MySQL support might not be enabled at all or `mysqli` support might be enabled instead of `mysql` support. Enabling MySQL support is explained in Appendix B.

Functions are explained later in the book and the PHP functions that communicate with MySQL are discussed at the beginning of Chapter 8. I mention them briefly here for those people who may be using PHP 4, because the test programs that follow this section will not run correctly with PHP 4.

Testing PHP

To test whether PHP is installed and working, follow these steps:

1. **Find the directory in which your PHP programs need to be saved.**

   This directory and the subdirectories under it are your *Web space.* Apache calls this directory the *document root.* The default Web space for Apache is `htdocs` in the directory where Apache is installed. For IIS, it's `Inetpub\wwwroot`. In Linux, it might be `/var/www/html`. The Web space can be set to a different directory by configuring the Web server (see Appendix C). If you're using a Web hosting company, the staff will supply the directory name.

2. **Create the following file somewhere in your Web space with the name `test.php`.**

```
<html>
<head>
<title>PHP Test</title>
</head>
<body>
<p>This is an HTML line
<p>
<?php
   echo "This is a PHP line";
   phpinfo();
?>
</body></html>
```



The file must be saved in your Web space for the Web server to find it.

3. **Point your browser at the `test.php` file created in Step 1. That is, type the name of your Web server into the browser address window, followed by the name of the file (for example, `www.myfinecompany.com/test.php`).**



If your Web server, PHP, and the `test.php` file are on the same computer that you're testing from, you can type **localhost/test.php**.

For the file to be processed by PHP, you need to access the file through the Web server — not by choosing File➪Open from your Web browser menu.

You should see the following in the Web browser:

```
This is an HTML line
This is a PHP line
```

Below these lines, you should see a large table that shows all the information associated with PHP on your system. It shows PHP information, pathnames and filenames, variable values, and the status of various options. The table is produced by the `phpinfo()` line in the test script. Anytime that you have a question about the settings for PHP, you can use the `phpinfo()` statement to display this table and check a setting.

4. **Check the PHP values for the settings you need.**

   For instance, you need MySQL support enabled. Looking through the listing, find the section for MySQL and make sure that MySQL support is `On`.

   PHP has many settings that can be changed. Various PHP settings are discussed throughout the book in the appropriate sections.

5. **Change values if necessary.**

   The general settings for PHP are stored in a file named `php.ini`. If you installed PHP yourself, you can edit `php.ini` and change settings. If your Web site is located on a company computer or a Web hosting company computer, you may not have access to `php.ini` to change settings. You can request the PHP administrator to change settings. For some settings, you can temporarily change a setting with a statement in your PHP program, but not all settings can be changed in a program. Changing PHP settings is discussed in Appendix B.

Testing MySQL

After you know that PHP is running okay, you can test whether you can access MySQL by using PHP. Just follow these steps:



1. **Create the following file somewhere in your Web space with the name `mysql_up.php`.**

   You can download the file from my Web site at `janet.valade.com`.

```
<?php
/* Program: mysql_up.php
 * Desc:    Connects to MySQL Server and
 *          outputs settings.
 */
 echo "<html>
       <head><title>Test MySQL</title></head>
       <body>";
 $host="host";
 $user="mysqlaccount";
 $password="mysqlpassword";

 $cxn = mysqli_connect($host,$user,$password);
 $sql="SHOW STATUS";
 $result = mysqli_query($cxn,$sql);
 if($result == false)
 {
    echo "<h4>Error: ".mysqli_error($cxn)."</h4>";
 }
 else
 {
   /* Table that displays the results */
   echo "<table border='1'>
         <tr><th>Variable_name</th>
             <th>Value</th></tr>";
   for($i = 0; $i < mysqli_num_rows($result); $i++)
   {
     echo "<tr>";
     $row_array = mysqli_fetch_row($result);
      for($j = 0;$j < mysqli_num_fields($result);$j++)
      {
         echo "<td>".$row_array[$j]."</td>\n";
      }
   }
   echo "</table>";
 }
?>
</body></html>
```

2. **The following lines 9, 10, and 11 of the program need to be changed:**

```
$host="host";
$user="mysqlaccount";
$password="mysqlpassword";
```

Change *host* to the name of the computer where MySQL is installed — for example, `databasehost.mycompany.com`. If the MySQL database is on the same computer as your Web site, you can use `localhost` as the hostname.

Change *mysqlaccountname* and *mysqlpassword* to the appropriate values. An account named `root` is installed when MySQL is installed, which may or may not have a password. (I discuss MySQL accounts and passwords in Chapter 5.) If your MySQL account doesn't require a password, type nothing between the quotes, as follows:

```
$password="";
```

3. **Point your browser at `mysql_up.php`.**

   You should see a table with a long list of variable names and values. You don't want to see an error message or a warning message. Don't worry about the contents of the table. It's only important that the table is displayed so that you know your connection to MySQL is working correctly.

   If no error or warning messages are displayed, MySQL is working fine. If you see an error or a warning message, you need to fix the problem that's causing the message.

The following is a common error message:

```
MySQL Connection Failed: Access denied for user:
        'user73@localhost' (Using password: YES)
```

This message means that MySQL did not accept your MySQL account number or your MySQL password. Notice that the message reads `YES` for `Using password` but doesn't show the actual password that you tried for security reasons. If you tried with a blank password, the message would read `NO`.

If you receive an error message, double-check your account number and password. Remember that this is your MySQL account number — not your account number to log on to the computer. If you can't connect with the account number and password that you have, you might need to contact the IT department or the Web hosting company that gave you the account number. (For a further discussion of MySQL accounts and passwords, see Chapter 5.)

Chapter 3

Developing a Web Database Application

In This Chapter

- Planning your application
- Selecting and organizing your data
- Designing your database
- Building your database: An overview
- Writing your application programs: An overview

Developing a Web database application involves more than just storing data in MySQL databases and typing in PHP programs. Development has to start with planning. Building the application pieces comes after planning. The development steps are

1. Develop a plan, listing the tasks that your application will perform.
2. Design the database needed to support your application tasks.
3. Build the MySQL database, based on the database design.
4. Write the PHP programs that perform the application tasks.

I discuss these steps in detail in this chapter.

Planning Your Web Database Application

Before you ever put finger to keyboard to write a PHP program, you need to plan your Web database application. This is possibly the most important step in developing your application. It's painful to discover, especially just after you finish the last program for your application, that you left something out and have to start over from the beginning. It's also hard on your computer (and your foot) when you take out your frustrations by drop-kicking it across the room.



Good planning prevents such painful backtracking. In addition, it keeps you focused on the functionality of your application, thus preventing you from writing pieces for the application that do really cool things but turn out to have no real purpose in the finished application. And if more than one person is working on your application, planning ensures that all the pieces will fit together in the end.

Identifying what you want from the application

The first step in the planning phase is to identify exactly why you're developing your application and what you want from it. For example, your main purpose might be to

- Collect names and addresses from users so that you can develop a customer list
- Deliver information about your products to users, as in a customer catalog
- Sell products online
- Provide technical support to people who already own your product

After you clearly identify the general purpose of your application, make a list of exactly what you want that application to do. For instance, if your goal is to develop a database of customer names and addresses for marketing purposes, the application's list of required tasks is fairly short:

- Provide a form for customers to fill out
- Store the customer information in a database

If your goal is to sell products online, the list is a little longer:

- Provide information about your products to the customer
- Motivate the customer to buy the product
- Provide a way for the customer to order the product online
- Provide a method for the customer to pay for the product online
- Validate the payment so you know that you'll actually get the money
- Send the order to the person responsible for filling the order and sending the product to the customer

At this point in the planning process, the tasks that you want your application to perform are still pretty general. You can accomplish each of these tasks in many different ways. So now you need to examine the tasks closely and detail exactly how the application will accomplish them. For instance, if your goal is to sell products online, you might expand the preceding list like this:

- **Provide information about products to the customer.**
  - Display a list of product categories. Each category is a link.
  - When the customer clicks a category link, the list of products in that category is displayed. Each product name is a link.
  - When a customer clicks a product link, the description of the product is displayed.
- **Motivate the customer to buy the product.**
  - Provide well-written descriptions of the products that communicate their obviously superior qualities.
  - Use flattering pictures of the products.
  - Make color product brochures available online.
  - Offer quantity discounts.
- **Provide a way for customers to order the product online.**
  - Provide a button that customers can click to indicate their intention to buy the product.
  - Provide a form that collects necessary information about the product the customer is ordering, such as size and color.
  - Provide forms for customers to enter shipping and billing addresses.
  - Compute and display the total cost for all items in the order.
  - Compute and display the shipping costs.
  - Compute and display the sales tax.
- **Provide a method for customers to pay for the product online.**
  - Provide a button that customers can click to pay with a credit card.
  - Display a form that collects customers' credit card information.
- **Validate the payment so you know that you'll actually get the money.**

  The usual method is to send the customer's credit card information to a credit card processing service.
- **Send the order to the person responsible for filling the order and sending the product to the customer.**

  E-mailing order information to the shipping department should do it.



At this point, you should have a fairly clear idea of what you want from your Web database application. However, this doesn't mean that your goals can't change. In fact, your goals are likely to change as you develop your Web database application and discover new possibilities. At the onset of the project, start with as comprehensive a plan as possible to keep you focused.

Taking the user into consideration

Identifying what *you* want your Web database application to do is only one aspect of planning. You must also consider what your users will want from it. For example, say your goal is to gather a list of names and addresses for marketing purposes. Will customers be willing to give up that information?

Your application needs to fulfill a purpose for the users as well as for yourself. Otherwise, they'll just ignore it. Before users will be willing to give you their names and addresses, for example, they need to perceive that they will benefit from giving you this information. Here are a few examples of why users might be willing to register their names and addresses at your site:

- **To receive a newsletter:** To be perceived as valuable, the newsletter should cover an industry related to your products. It should offer news and spot trends — and not just serve as marketing material about your products.
- **To enter a sweepstakes for a nice prize:** Who can turn down a chance to win an all-expense-paid vacation to Hawaii or a brand-new SUV?
- **To receive special discounts:** For example, you can periodically e-mail special discount opportunities to customers.
- **To be notified about new products or product upgrades when they become available:** For example, customers might be interested in being notified when a software update is available for downloading.
- **To get access to valuable information:** For instance, you must register at *The New York Times* Web site to gain access to its articles online.

Now add the customer tasks to your list of tasks that you want the application to perform. For example, consider this list of tasks that you identified for setting up an online retailer:

- Provide a form for customers to fill out
- Store the customer information in a database

If you take the customer's viewpoint into account, the list expands a bit:

- Present a description of the advantages customers receive by registering with the site
- Provide a form for customers to fill out

- Add customers' e-mail addresses to the newsletter distribution list
- Store the customer information in a database

After you have a list of tasks that you want and tasks that your users want, you have a plan for a Web application that is worth your time to develop and worth your users' time to use.

Making the site easy to use

In addition to planning what your Web application is going to do, you need to consider how it is going to do it. Making your application easy to use is important: If customers can't find your products, they aren't going to buy them. And if customers can't find the information they need in a short time, they will look elsewhere. On the Web, customers can easily go elsewhere.

Making your application easy to use is *usability engineering.* Web usability includes such issues as

- **Navigation:** What is on your site and where it is located should be immediately obvious to a user.
- **Graphics:** Graphics make your site attractive, but graphic files can be slow to display.
- **Access:** Some design decisions can make your application accessible or not accessible to users who have disabilities such as impaired vision.
- **Browsers:** Different browsers (even different versions of the same browser) can display the same HTML file differently.



Web usability is a large and important subject, and delving into the topic more deeply is beyond the scope of this book. But fear not, you can find lots of helpful information on Web usability on — you guessed it — the Web. Be sure to check out the Web sites of usability experts Jakob Nielsen (`www.useit.com`) and Jared Spool (`www.uie.com`). Vincent Flanders also has a fun site full of helpful information about Web design at `WebPagesThatSuck.com`. And books on the subject can be very helpful, such as *Web Design For Dummies* by Lisa Lopuck (Wiley).

Leaving room for expansion

One certainty about your Web application is that it will change over time. Down the line, you might think of new functions for it or just simply want to change something about it. Or maybe Web site software improves so that your Web application can do things that it couldn't do when you first put it up. Whatever the reason, your Web site will change. When you plan your application, you need to keep future changes in mind.

You can design your application in steps, taking planned changes into account. You can develop a plan in which you build an application today that meets your most immediate needs and make it available as soon as it's ready. Your plan can include adding functions to the application as quickly as you can develop them. For example, you can build a product catalog and publish it on your Web site as soon as it's ready. You can then begin work on an online ordering function for the Web site, which you will add when it's ready.



You can't necessarily foresee all the functions that you might want in your application. For instance, you might design your travel Web site with sections for all possible destinations today, but the future could surprise you. Trips to Mars? Alpha Centauri? An alternate universe? Plan your application with the flexibility needed to add functionality in the future.

Writing it down

Write your plan down. You will hear this often from me. I speak from the painful experience of not writing it down. When you develop your plan, it's foremost in your mind and perfectly clear. But in a few short weeks, you will be astonished to discover that it has gone absolutely hazy while your attention was on other pressing issues. Or you want to make some changes in the application a year from now and won't remember exactly how the application was designed. Or you're working with a partner to develop an application and you discover that your partner misunderstood your verbal explanation and developed functions for the application that don't fit in your plan. You can avoid these types of problems by writing everything down.

Presenting the Two Running Examples in This Book

In the next two sections, I introduce the two example Web database applications that I created for this book. I refer to these examples throughout the book to demonstrate aspects of application design and development.

Stuff for Sale

The first example is an online product catalog. You're the owner of a pet store, and you want your catalog to provide customers with information about the pets for sale. Selling the pets online is not feasible, although you're toying with the idea of allowing customers to reserve pets online — that is, before they come into the store to purchase them. Currently, the application is simply an online catalog. Customers can look through the catalog online

and then come into the store to buy the pet. The information about all the pets is stored in a database, and customers can search the database for information on specific pets or types of pets.

Here is your plan for this application:

- **Allow customers to select which pet information they want to see.**

  Offer two selection methods:

  - **Selecting from a list of links:** Display a list of links that are pet categories (dog, cat, dinosaur, and so on). When the customer clicks a category link, a list of pets is displayed. Each pet in the list is a link to a description of the pet.
  - **Typing search terms:** Display a search form in which customers can type words that describe the type of pet they're looking for. The application searches the database for matching words and displays the pet information for pets that match the search words. For example, a customer can type **cat** to see a list of all available cats. Each cat in the list is a link to a description of that cat.

- **Display a description of the pet when the customer clicks the link.**

  The description is stored in a database.

Members Only

The second example Web database application is related to the preceding pet store example. In addition to the online catalog, you also want to put up a section on your pet store Web site that's for members only. To access this area of the site, customers have to register — providing their names and addresses. In this Members Only section, customers can order pet food at a discount, find out about pets that are on order but haven't arrived yet, and gain access to articles with news and information about pets and pet care.

This is your plan for this application:

- **Display a description of what special features and information are available in the Members Only section.**
- **Provide an area where customers can register for the Members Only section.**
  - **Provide a link to the registration area.**
  - **Display a form in the registration area where customers can type their registration information.**

    The form should include space for a user login name and password as well as the information that you want to collect.

- **Validate the information that the user entered.**

  For example, verify that the zip code is the correct length and that the e-mail address is in the correct format.

- **Store the information in the database.**

- **Provide a login section for customers who are already registered for the Members Only section.**
  - **Display a login form that asks for the customer's user name and password.**
  - **Compare the user name and password that are entered with the user names and passwords in the database.**

    If no match is found, display an error message.
- **Display the Members Only Web page after the customer has successfully logged in.**

Designing the Database

After you determine exactly what the Web database application is going to do (see the beginning part of this chapter if you haven't done this yet), you're ready to design the database that holds the information needed by the application. Designing the database includes identifying the data that you need and organizing the data in the way required by the database software.

Choosing the data

First, you must identify what information belongs in your database. Look at the list of tasks that you want the application to perform and determine what information you need to complete each of those tasks.

Here are a few examples:

- An online catalog needs a database containing product information.
- An online order application needs a database that can hold customer information and order information.
- A travel Web site needs a database with information on destinations, reservations, fares, schedules, and so on.

In many cases, your application might include a task that collects information from the user. You'll have to balance your urge to collect all the potentially useful information that you can think of against your users' reluctance to give out personal information — as well as their avoidance of forms that look too time-consuming. One compromise is to ask for some optional information. Users who don't mind can enter it, but users who object can leave it blank. Another possibility is to offer an incentive: The longer the form, the stronger the incentive that you'll need to motivate the user to fill out the form. A user might be willing to fill out a short form to enter a sweepstakes that offers two sneak-preview movie tickets for a prize. But if the form is long and complicated, the prize needs to be more valuable, such as a free trip to California and a tour of a Hollywood movie studio.

In the first example application, your customers search the online catalog for information on pets that they might want to buy. You want customers to see information that will motivate them to buy a pet. The information that you want to have available in the database for the customer to see is as follows:

- **The name of the pet** (for example, poodle or unicorn)
- **A description of the pet**
- **A picture of the pet**
- **The cost of the pet**

In the second example application, the Members Only section, you want to store information about registered members. The information that you want to store in the database is as follows:

- **Member name**
- **Member address**
- **Member phone number**
- **Member fax number**
- **Member e-mail address**



Take the time to develop a comprehensive list of the information you need to store in your database. Although you can change and add information to your database after it's developed, including the information from the beginning is easier. Also, if you add information to the database later — after it's in use — the first users in the database will have incomplete information. For example, if you change your form so that it now asks for the user's age, you won't have the age for the people who have already filled out the form and are already in the database.

Organizing the data

MySQL is an RDBMS (Relational Database Management System), which means that the data is organized into tables. (See Chapter 1 for more on MySQL.) You can establish relationships between the tables in the database.

Organizing data in tables

RDBMS tables are organized like other tables that you're used to — in rows and columns, as shown in Figure 3-1. The place where a particular row and column intersect, the individual cell, is a *field*.

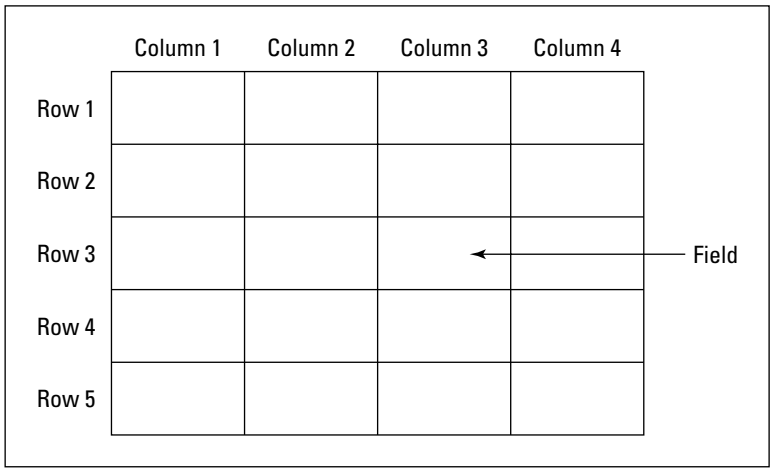


Figure 3-1: MySQL data is organized into tables.

The focus of each table is an *object* (a thing) that you want to store information about. Here are some examples of objects:

Customers	Shapes	Rooms
Companies	Projects	Computers
Cities	Products	Documents
Books	Animals	Weeks

You create a table for each object. The table name should clearly identify the objects that it contains with a descriptive word or term. The name must be a character string, containing letters, numbers, underscores, or dollar signs, with no spaces in it. It's customary to name the table in the singular. Thus, a name for a table of customers might be `Customer`, and a table containing customer orders might be named `CustomerOrder`. Uppercase and lowercase is significant on Linux and Unix but not on Windows: `CustomerOrder` and `Customerorder` are the same to Windows — but not to Linux or Unix.

In database talk, an object is an *entity,* and an entity has *attributes.* In the table, each row represents an entity, and the columns contain the attributes of each entity. For example, in a table of customers, each row contains information for a single customer. Some of the attributes contained in the columns might be first name, last name, phone number, and age.

Here are the steps for organizing your data into tables:

1. **Name your database.**

   Assign a name to the database for your application. For instance, a database containing information about households in a neighborhood might be named `HouseholdDirectory`.

2. **Identify the objects.**

   Look at the list of information that you want to store in the database (as discussed in the section, "Choosing the data," earlier in this chapter). Analyze your list and identify the objects. For instance, the `HouseholdDirectory` database might need to store the following:

   - Name of each family member
   - Address of the house
   - Phone number
   - Age of each household member
   - Favorite breakfast cereal of each household member

   When you analyze this list carefully, you realize that you're storing information about two objects: the household and the household members. That is, the address and phone number are for the household in general, but the name, age, and favorite cereal are for a particular household member.

3. **Define and name a table for each object.**

   For instance, the `HouseholdDirectory` database needs a table called `Household` and a table called `HouseholdMember`.

4. **Identify the attributes for each object.**

   Analyze your information list and identify the attributes you need to store for each object. Break the information to be stored into its smallest reasonable pieces. For example, when storing the name of a person in a table, you can break the name into first name and last name. Doing this enables you to sort by the last name, which would be more difficult if the first and last name were stored together. You can even break down the name into first name, middle name, and last name, although not many applications need to use the middle name separately.

5. **Define and name columns for each separate attribute that you identified in Step 4.**

   Give each column a name that clearly identifies the information in that column. The column names should be one word, with no spaces. For example, you might have columns named `firstName` and `lastName` or `first_name` and `last_name`.



   Some words are reserved by MySQL and SQL for their own use and can't be used as column names. The words are currently used in SQL statements or are reserved for future use. For example, `ADD`, `ALL`, `AND`, `CREATE`, `DROP`, `GROUP`, `ORDER`, `RETURN`, `SELECT`, `SET`, `TABLE`, `USE`, `WHERE`, and many, many more can't be used as column names. For a complete list of reserved words, see the online MySQL manual at `www.mysql.com/doc/en/Reserved_words.html`.

6. **Identify the primary key.**

   Each row in a table needs a unique identifier. No two rows in a table should be exactly the same. When you design your table, you decide which column holds the unique identifier, called the *primary key.* The primary key can be more than one column combined. In many cases, your object attributes will not have a unique identifier. For example, a customer table might not have a unique identifier because two customers can have the same name. When there is no unique identifier column, you need to add a column specifically to be the primary key. Frequently, a column with a sequence number is used for this purpose. For example, in Figure 3-2, the primary key is the `cust_id` field because each customer has a unique ID number.

cust_id	first_name	last_name	phone
27895	John	Smith	555-5555
44555	Joe	Lopez	555-5553
23695	Judy	Chang	555-5552
27822	Jubal	Tudor	555-5556
29844	Joan	Smythe	555-5559

Figure 3-2: A sample from the Customer table.

7. **Define the defaults.**

   You can define a default that MySQL will assign to a field when no data is entered into the field. A default is not required but is often useful. For example, if your application stores an address that includes a country, you can specify US as the default. If the user does not type a country, US will be entered.

8. **Identify columns that require data.**

   You can specify that certain columns are not allowed to be empty (also called `NULL`). For instance, the column containing your primary key can't be empty. That means that MySQL will not create the row and will return an error message if no value is stored in the column. The value can be a blank space or an empty string (for example, `""`), but some value must be stored in the column. Other columns, in addition to the primary key, can be set to require data.

Well-designed databases store each piece of information in only one place. Storing it in more than one place is inefficient and creates problems if information needs to be changed. If you change information in one place but forget to change it in another place, your database can have serious problems.



If you find that you're storing the same data in several rows, you probably need to reorganize your tables. For example, suppose you're storing data about books, including the publisher's address. When you enter the data, you realize that you're entering the same publisher's address in many rows. A more efficient way to store this data would be to store the book information in one table and the book publisher information in a separate table. You can define two tables: `Book` and `BookPublisher`. In the `Book` table, you would have the columns `title`, `author`, `pub_date`, and `price`. In the `BookPublisher` table, you would have columns such as `name`, `streetAddress`, and `city`.

Creating relationships between tables

Some tables in a database are related. Most often, a row in one table is related to several rows in another table. A column is needed to connect the related rows in different tables. In many cases, you include a column in one table to hold data that matches data in the primary key column of another table.

A common application that needs a database with two related tables is a customer order application. For example, one table contains the customer information, such as name, address, and phone number. Each customer can have from zero to many orders. You could store the order information in the table with the customer information, but a new row would be created each time

that the customer placed an order, and each new row would contain all the customer's information. It would be much more efficient to store the orders in a separate table, named perhaps `CustomerOrder`. (You can't name the table *Order* because that is a reserved word.) The `CustomerOrder` table would have a column that contains the primary key from a row in the `Customer` table so that the order is related to the correct row of the `Customer` table. The relationship is shown in the tables in Figures 3-2 and 3-3.

The `Customer` table in this example looks like Figure 3-2 (see the preceding section). Notice the unique `cust_id` for each customer. The related `CustomerOrder` table is shown in Figure 3-3. Notice that it has the same `cust_id` column that appears in the `Customer` table. In this way, the order information in the `CustomerOrder` table is connected to the related customer's name and phone number in the `Customer` table.

Order_no	cust_id	item_num	cost
87-222	27895	cat-3	200.00
87-223	27895	cat-4	225.00
87-224	44555	horse-1	550.00
87-225	44555	dog-27	210.00
87-226	27895	bird-1	50.00

Figure 3-3: A sample from the Customer Order table.

In this example, the columns that relate the `Customer` table and the `CustomerOrder` table have the same name. They could have different names as long as the data in the columns is the same.

Designing the Sample Databases

In the following two sections, I design the two databases for the two example applications used in this book.

Pet Catalog design process

You want to display the following list of information when customers search your pet catalog:

- **The name of the pet** (for example, poodle or unicorn)
- **A description of the pet**
- **A picture of the pet**
- **The cost of the pet**

In the Pet Catalog plan, a list of pet categories is displayed. This requires that each pet be classified into a pet category and that the pet category be stored in the database.

You design the `PetCatalog` database by following the steps presented in the "Organizing data in tables" section, earlier in this chapter:

1. **Name your database.**

   The database for the Pet Catalog is named `PetCatalog`.

2. **Identify the objects.**

   The information list is

   - **The name of the pet** (poodle, unicorn, and so on)
   - **A description of the pet**
   - **A picture of the pet**
   - **The cost of the pet**
   - **The category for the pet**

   All this information is about pets, so the only object for this list is `Pet`.

3. **Define and name a table for each object.**

   The Pet Catalog application needs a table called `Pet`.

4. **Identify the attributes for each object.**

   Now you look at the information in detail:

   - **Name of the pet:** A single attribute — for example, poodle or unicorn. However, it seems likely that your pet shop might have more than one poodle for sale at a time. Therefore, your table needs a unique identifier to serve as the primary key.
   - **Pet identification number:** A sequence number assigned to each pet when it's added to the table. This number is the primary key.

- **Description of the pet:** Two attributes — the written description of the pet as it would appear in print and the color of the pet.
- **Picture of the pet:** A path name to a graphic file containing a beautiful picture of the pet.
- **Cost of the pet:** The dollar amount that the store is asking for the pet.
- **Category for the pet:** Two attributes: a category name that includes the pet — for example, dog, horse, dragon — and a description of the category.

It would be inefficient to include two types of information in the `Pet` table:

- The category information includes a description of the category. Because each category can include several pets, including the category description in the `Pet` table would result in the same description appearing in several rows. It is more efficient to define the pet category as an object with its own table.
- If the pet comes in several colors, all the pet information will be repeated in a separate row for each color. It is more efficient to define the pet color as an object with its own table.

The added tables are named `PetType` and `PetColor`.

5. **Define and name columns.**

   The `Pet` table has one row for each pet. The columns for the `Pet` table are

   - `petID`: Unique sequence number assigned to each pet.
   - `petName`: Name of the pet.
   - `petType`: The category name. This is the column that connects the pet to the correct row in the `PetType` table.
   - `petDescription`: The description of the pet.
   - `price`: The price of the pet.
   - `pix`: The filename of a file that contains a picture of the pet.

   The `PetType` table has one row for each pet category. It has the following columns:

   - `petType`: The category name of a type of pet. This is the primary key for this table. Notice that the `Pet` table has a column with the same name. These columns link this table with the `Pet` table.
   - `typeDescription`: The description of the pet type.

   The `PetColor` table has one row for each pet color. It has the following columns:

- `petName`: The name of the pet. This is the column that connects the color row to the correct row in the `Pet` table.
- `petColor`: The color of the pet.
- `pix`: The filename of a file that contains a picture of the pet of the specified color.

6. **Identify the primary key.**
   - The primary key of the `Pet` table is `petID`.
   - The primary key of the `PetType` table is `petType`.
   - The primary key of the `PetColor` table is `petName` and `petColor` together.
7. **Define the defaults.**

   No defaults are defined for any of the tables.
8. **Identify columns with required data.**

   The following columns should never be allowed to be empty:
   - `petID`
   - `petName`
   - `petColor`
   - `petType`

   These columns are the primary key columns. A row without these values should never be allowed in the tables.

Members Only design process

You create the following list of information that you want to store when customers register for the Members Only section of your Web site:

- Member name
- Member address
- Member phone number
- Member fax number
- Member e-mail address

In addition, you would like to collect the date when the member registers and track how often the member goes into the Members Only section.

You design the Members Only database by following the steps presented in the "Organizing data in tables" section, earlier in this chapter:

1. **Name your database.**

   The database for the Members Only section is named `MemberDirectory`.

2. **Identify the objects.**

   The information list is

   - Member name
   - Member address
   - Member phone number
   - Member fax number
   - Member e-mail address
   - Member registration date
   - Member logins

   All this information pertains to members, so the only object for this list is `member`.

3. **Define and name a table for each object.**

   The `MemberDirectory` database needs a table called `Member`.

4. **Identify the attributes for each object.**

   Look at the information list in detail:

   - **Member name:** Two attributes: first name and last name.
   - **Member address:** Four attributes: street address, city, state, and zip code. Currently, you have pet stores only in the United States, so you can assume that the member address is an address in the U.S. mailing address format.
   - **Member phone number:** One attribute.
   - **Member fax number:** One attribute.
   - **Member e-mail address:** One attribute.
   - **Member registration date:** One attribute.

   Several pieces of information are related to member logins:

   - Logging in to the Members Only section requires a login name and a password. These two items need to be stored in the database.
   - The easiest way to keep track of member logins is to store the date and time when the user logged into the Members Only section.

Because each member can have many logins, many dates and times for logins need to be stored. Therefore, rather than defining the login time as an attribute of the member, define login as an object, related to the member, but requiring its own table.

The added table is named `Login`. The attribute of a login object is its login time (the time includes the date).

5. **Define and name the columns.**

   The `Member` table has one row for each member. The columns for the `Member` table are

   - `loginName`
   - `password`
   - `createDate`
   - `firstName`
   - `lastName`
   - `street`
   - `city`
   - `state`
   - `zip`
   - `email`
   - `phone`
   - `fax`

   The `Login` table has one row for each *login:* that is, each time a member logs into the Members Only section. It has the following columns:

   - `loginName`: The login name of the member who logged in. This is the column that links this table to the `Member` table. This value is unique in the `Member` table but not unique in this table.
   - `loginTime`: The date and time of login.

6. **Identify the primary key.**

   - The primary key for the `Member` table is `loginName`. Therefore, `loginName` must be unique.
   - The primary key for the `Login` table is both `loginName` and `loginTime` together.

7. **Define the defaults.**

   No defaults are defined for either table.

8. **Identify columns with required data.**

   The following columns should never be allowed to be empty:

   - `loginName`
   - `password`
   - `loginTime`

   These columns are the primary key columns. A row without these values should never be allowed in the tables.

Types of Data

MySQL stores information in different formats based on the type of information that you tell MySQL to expect. MySQL allows different types of data to be used in different ways. The main types of data are character, numerical, and date and time data.

Character data

The most common type of data is *character* data — data that is stored as strings of characters and can be manipulated only in strings. Most of the information that you store will be character data, such as customer name, address, phone, and pet description. Character data can be moved and printed. Two character strings can be put together (concatenated), a substring can be selected from a longer string, and one string can be substituted for another.

Character data can be stored in a fixed-length or variable-length format.

- **Fixed-length format:** In this format, MySQL reserves a fixed space for the data. If the data is longer than the fixed length, only the characters that fit are stored — the remaining characters on the end are not stored. If the string is shorter than the fixed length, the extra spaces are left empty and wasted.
- **Variable-length format:** In this format, MySQL stores the string in a field that is the same length as the string. You specify a string length, but if the string is shorter than the specified length, MySQL uses only the space required rather than leaving the extra space empty. If the string is longer than the space specified, the extra characters are not stored.

If a character string length varies only a little, use the fixed-length format. For example, a length of 10 works for all zip codes, including those with the zip+4 number. If the zip code does not include the zip+4 number, only five spaces are left empty. However, if your character string can vary more than a few characters, use a variable-length format to save space. For example, your pet description might be *Small bat* or might run to several lines of description. It would be better to store this description in a variable-length format.

Numerical data

Another common type of data is *numerical* data — data that is stored as a number. Decimal numbers (for example, 10.5, 2.34567, 23456.7) can be stored

as well as integers (for example, 1, 2, 248). When data is stored as a number, it can be used in numerical operations, such as adding, subtracting, and squaring. If data isn't used for numerical operations, however, storing it as a character string is better because the programmer will be using it as a character string. No conversion is required. For example, you probably won't want to add the digits in the users' phone numbers, so phone numbers should be stored as character strings.

MySQL stores positive and negative numbers, but you can tell MySQL to store only positive numbers. If your data is never negative, store the data as *unsigned* (without using a + or – sign before the number). For example, a city population or the number of pages in a document can never be negative.

MySQL provides a specific type of numeric column called an auto-increment column. This type of column is automatically filled with a sequential number when no specific number is provided. For example, when a table row is added with 5 in the auto-increment column, the next row is automatically assigned 6 in the column, unless a different number is specified. Auto-increment columns are useful when unique numbers are needed, such as a product number or an order number.

Date and time data

A third common type of data is date and time data. Data stored as a date can be displayed in a variety of date formats. It can also be used to determine the length of time between two dates or two times — or between a specific date or time and some arbitrary date or time.

Enumeration data

Sometimes data can have only a limited number of values. For example, the only possible values for a column might be `yes` or `no`. MySQL provides a data type called *enumeration* for use with this type of data. You tell MySQL what values can be stored in the column (for example, `yes`, `no`), and MySQL will not store any other values in the column.

MySQL data type names

When you create a database, you tell MySQL what kind of data to expect in a particular column by using the MySQL names for data types. Table 3-1 shows the MySQL data types used most often in Web database applications.

Table 3-1 MySQL Data Types

MySQL Data Type	*Description*
CHAR(*length*)	Fixed-length character string.
VARCHAR(*length*)	Variable-length character string. The longest string that can be stored is *length*, which must be between 1 and 255.
TEXT	Variable-length character string with a maximum length of 64K of text.
INT(*length*)	Integer with a range from –2147483648 to +2147483647. The number that can be displayed is limited by *length*. For example, if *length* is 4, only numbers from –999 to 9999 can be displayed, even though higher numbers are stored.
INT(*length*) UNSIGNED	Integer with a range from 0 to 4294967295. *length* is the size of the number that can be displayed. For example, if *length* is 4, only numbers up to 9999 can be displayed, even though higher numbers are stored.
BIGINT	A large integer. The signed range is –9223372036854775808 to 9223372036854775807. The unsigned range is 0 to 18446744073709551615.
DECIMAL (*length*,*dec*)	Decimal number where *length* is the number of characters that can be used to display the number, including decimal points, signs, and exponents, and *dec* is the maximum number of decimal places allowed. For example, 12.34 has a *length* of 5 and a *dec* of 2.
DATE	Date value with year, month, and date. Displays the value as YYYY-MM-DD (for example, 2006-04-03).
TIME	Time value with hour, minute, and second. Displays as HH:MM:SS.
DATETIME	Date and time are stored together. Displays as YYYY-MM-DD HH:MM:SS.
ENUM ("*val1*", "*val2*"...)	Only the values listed can be stored. A maximum of 65,535 values can be listed.
SERIAL	A shortcut name for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT.

MySQL allows many other data types, but they're needed less frequently. For a description of all the available data types, see the MySQL online manual at `dev.mysql.com/doc/refman/5.0/en/data-types.html`.

Writing it down

Here's my usual nagging: *Write it down.* You probably spent substantial time making the design decisions for your database. At this point, the decisions are firmly fixed in your mind. You don't believe that you can forget them. However, suppose that a crisis intervenes; you don't get back to this project for two months. You will have to analyze your data and make all the design decisions again. You can avoid this by writing down the decisions now.

Document the organization of the tables, the column names, and all other design decisions. A good format is a document that describes each table in table format, with a row for each column and a column for each design decision. For example, your columns would be *column name, data type,* and *description.*

Taking a Look at the Sample Database Designs

This section contains the database designs for the two example Web database applications.

Stuff for Sale database tables

The database design for the Pet Catalog application includes three tables: `Pet`, `PetType`, and `PetColor`. Tables 3-2 through 3-4 show the organization of these tables. The table definition is not set in concrete; MySQL allows you to change tables pretty easily. For example, if you set the data type for a variable to `CHAR(20)` and find that isn't long enough, you can easily change the data type. The database design follows.

Table 3-2 PetCatalog Database Table 1: Pet

Variable Name	*Type*	*Description*
petID	SERIAL	Sequence number for pet (primary key)
petName	CHAR(25)	Name of pet
petType	CHAR(15)	Category of pet
petDescription	VARCHAR(255)	Description of pet
price	DECIMAL(9,2)	Price of pet
pix	CHAR(15)	Path name to graphic file containing picture of pet

Table 3-3 PetCatalog Database Table 2: PetType

Variable Name	*Type*	*Description*
petType	CHAR(15)	Name of pet category (primary key)
typeDescription	VARCHAR(255)	Description of category

Table 3-4 PetCatalog Database Table 3: PetColor

Variable Name	*Type*	*Description*
petName	CHAR(25)	Name of pet (primary key 1)
petColor	CHAR(15)	Color name (primary key 2)
pix	CHAR(!5)	Path name to graphic file containing picture of pet

Members Only database tables

The database design for the Members Only application includes two tables — `Member` and `Login`. Tables 3-5 and 3-6 document the organization of these tables. The table definition is not set in concrete; MySQL allows you to change tables pretty easily. If you set the data type for a variable to `CHAR(5)` and find that it isn't long enough, it's easy to change the data type. The database design follows.

Table 3-5 MemberDirectory Database Table 1: Member

Variable Name	*Type*	*Description*
loginName	VARCHAR(20)	User-specified login name (primary key)
password	CHAR(255)	User-specified password
createDate	DATE	Date member registered and created login account
lastName	VARCHAR(50)	Member's last name
firstName	VARCHAR(40)	Member's first name
street	VARCHAR(50)	Member's street address
city	VARCHAR(50)	Member's city
state	CHAR(2)	Member's state
zip	CHAR(10)	Member's zip code
email	VARCHAR(50)	Member's e-mail address
phone	CHAR(15)	Member's phone number
fax	CHAR(15)	Member's fax number

Table 3-6 MemberDirectory Database Table 2: Login

Variable Name	*Type*	*Description*
loginName	VARCHAR(20)	Login name specified by user (primary key 1)
loginTime	DATETIME	Date and time of login (primary key 2)

Developing the Application

After you develop a plan listing the tasks that your application will perform and you develop a database design, you're ready to create your application. First you build the database, and then you write your PHP programs. You are moments away from a working Web database application. Well, perhaps that's an exaggeration. But you are making progress.

Building the database

Building the database means turning the paper database design into a working database. Building the database is independent of the PHP programs that your application uses to interact with the database. The database can be accessed using programming languages other than PHP, such as Perl, C, or Java. The database stands on its own to hold the data.



You should build the database before writing the PHP programs. The PHP programs are written to move data in and out of the database, so you can't develop and test them until the database is available.

The database design names the database and defines the tables that make up the database. To build the database, you communicate with MySQL by using the SQL language. You tell MySQL to create the database and to add tables to the database. You tell MySQL how to organize the data tables and what format to use to store the data. Detailed instructions for building the database are provided in Chapter 4.

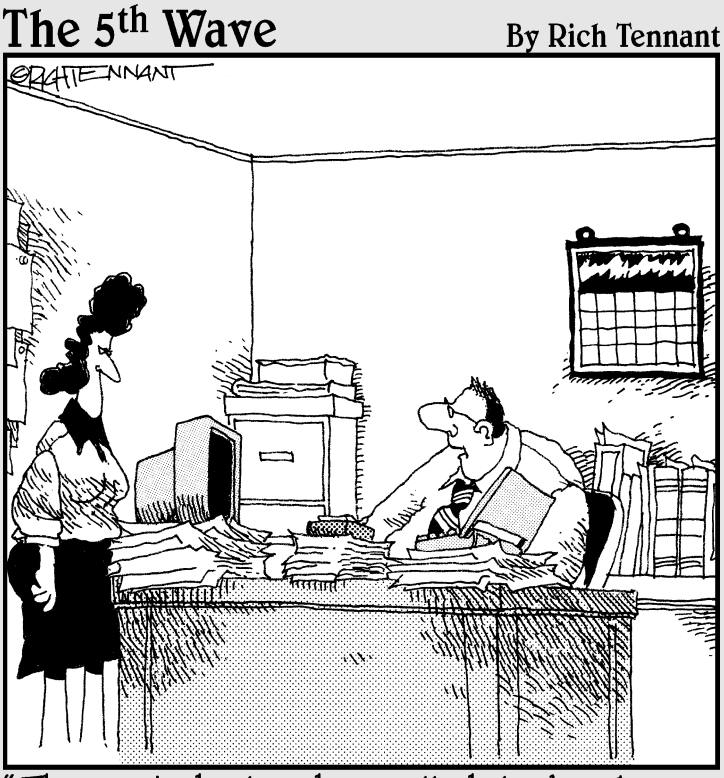
Writing the programs

Your programs perform the tasks for your Web database application. They create the display that the user sees in the browser window. They make your application interactive by accepting and processing information typed in the browser window by the user. They store information in the database and get information out of the database. The database is useless unless you can move data in and out of it.

The plan that you develop (as I discuss in the earlier sections in this chapter) outlines the programs that you need to write. In general, each task in your plan calls for a program. If your plan says that your application will display a form, you need a program that displays a form. If your plan says that your application will store the data from a form, you need a program that gets the data from the form and puts it in the database.

The PHP language was developed specifically to write interactive Web applications. It has the built-in functionality needed to make writing application programs as painless as possible. Methods were included in the language specifically to access data from forms, to put data into a MySQL database, and to get data from a MySQL database. Detailed instructions for writing PHP programs are provided in Part III.

Part II
MySQL Database



"The new technology has really helped me become organized. I keep my project reports under the PC, budgets under my laptop, and memos under my pager."

In this part . . .

This part provides the details of working with a MySQL database. You find out how to use SQL (Structured Query Language) to communicate with MySQL. In addition, you discover how to create a database, change a database, and move data into and out of a database.

Chapter 4

Building the Database

In This Chapter

- Using SQL to make requests to MySQL
- Creating a new database
- Adding information to an existing database
- Looking at information in an existing database
- Removing information from an existing database

After completing your database design (see Chapter 3 if you haven't done this yet), you're ready to turn it into a working database. In this chapter, you find out how to build a database based on your design — and how to move data into and out of it.

The database design names the database and defines the tables that make up the database. To build the database, you must communicate with MySQL, providing the database name and the table structure. Later, you must communicate with MySQL to add data to (or request information from) the database. The language that you use to communicate with MySQL is SQL. In this chapter, I explain how to create SQL queries and use them to build new databases and interact with existing databases.

Communicating with MySQL

The MySQL server is the manager of your database:

- It creates new databases.
- It knows where the databases are stored.
- It stores and retrieves information, guided by the requests, or *queries,* that it receives.

To make a request that MySQL can understand, you build an SQL query and send it to the MySQL server. (For a more complete description of the MySQL server, see Chapter 1.) The next two sections detail how to do this.

Building SQL queries

SQL (Structured Query Language) is the computer language that you use to communicate with MySQL. SQL is almost English; it is made up largely of English words, put together into strings of words that sound similar to English sentences. In general (fortunately), you don't need to understand any arcane technical language to write SQL queries that work.

The first word of each query is its name, which is an action word (a verb) that tells MySQL what you want to do. The queries that I discuss in this chapter are `CREATE`, `DROP`, `ALTER`, `SHOW`, `INSERT`, `LOAD`, `SELECT`, `UPDATE`, and `DELETE`. This basic vocabulary is sufficient to create — and interact with — databases on Web sites.

The query name is followed by words and phrases — some required and some optional — that tell MySQL how to perform the action. For instance, you always need to tell MySQL what to create, and you always need to tell it which table to insert data into or to select data from.

The following is a typical SQL query. As you can see, it uses English words:

```
SELECT lastName FROM Member
```

This query retrieves all the last names stored in the table named `Member`. More complicated queries, such as the following, are less English-like:

```
SELECT lastName,firstName FROM Member WHERE state="CA" AND
        city="Fresno" ORDER BY lastName
```

This query retrieves all the last names and first names of members who live in Fresno and then puts them in alphabetical order by last name. This query is less English-like but still pretty clear.



Here are some general points to keep in mind when constructing an SQL query, as illustrated in the preceding sample query:

- **Capitalization:** In this book, I put SQL language words in all caps; items of variable information (such as column names) are usually given labels that are all or mostly lowercase letters. I did this to make it easier for you to read — not because MySQL needs this format. The case of the SQL words doesn't matter; for example, `select` is the same as `SELECT`, and `from` is the same as `FROM`, as far as MySQL is concerned. On the other hand, the case of the table names, column names, and other variable information does matter if your operating system is Unix or Linux. When using Unix or Linux, MySQL needs to match the column names exactly, so the case for the column names has to be correct — for example, `lastname` is not the same as `lastName`. Windows, however, isn't as picky as Unix and Linux; from its point of view, `lastname` and `lastName` are the same.

- **Spacing:** SQL words must be separated by one or more spaces. It doesn't matter how many spaces you use; you could just as well use 20 spaces or just 1 space. SQL also doesn't pay any attention to the end of the line. You can start a new line at any point in the SQL statement or write the entire statement on one line.
- **Quotes:** Notice that `CA` and `Fresno` are enclosed in double quotes (`"`) in the preceding query. `CA` and `Fresno` are series of characters called *text strings,* or *character strings.* (I explain strings in detail later in this chapter.) You are asking MySQL to compare the text strings in the SQL query with the text strings already stored in the database. When you compare numbers (such as integers) stored in numeric columns, you don't enclose the numbers in quotes. (In Chapter 3, I explain the types of data that can be stored in a MySQL database.)

Sending SQL queries

This book is about PHP and MySQL as a pair. Consequently, I don't describe the multitude of ways in which you can send SQL queries to MySQL — many of which have nothing to do with PHP. Rather, I provide a simple PHP program that you can use to execute SQL queries. (For the lowdown on PHP and how to write PHP programs, check out Part III.)

The program `mysql_send.php` has one simple function: to execute queries and display the results. Enter the program into the directory where you're developing your Web application (or download it from my Web site at `janet.valade.com`), change the information in lines 13–15, and then point your browser at the program. Listing 4-1 shows the program.

Listing 4-1: PHP Program for Sending SQL Queries to MySQL

```
<?php
/*Program:  mysql_send.php
 *Desc:     PHP program that sends an SQL query to the
 *          MySQL server and displays the results.
 */
echo "<html>
      <head><title>SQL Query Sender</title></head>
      <body>";
if(ini_get("magic_quotes_gpc") == "1")
{
   $_POST['query'] = stripslashes($_POST['query']);
}
$host="hostname";
$user=" mysqlaccountname";
$password=" mysqlpassword";

/* Section that executes query and displays the results */
```

(continued)

Listing 4-1 *(continued)*

```
if(!empty($_POST['form']))
{
  $cxn = mysqli_connect($host,$user,$password,
                        $_POST['database']);
  $result = mysqli_query($cxn,$_POST['query']);
  echo "Database Selected: <b>{$_POST['database']}</b><br>
        Query: <b>{$_POST['query']}</b>
        <h3>Results</h3><hr>";
  if($result == false)
  {
     echo "<h4>Error: ".mysqli_error($cxn)."</h4>";
  }
  elseif(@mysqli_num_rows($result) == 0)
  {
     echo "<h4>Query completed.
           No results returned.</h4>";
  }
  else
  {
   /* Display results */
     echo "<table border='1'><thead><tr>";
     $finfo = mysqli_fetch_fields($result);
     foreach($finfo as $field)
     {
        echo "<th>".$field->name."</th>";
     }
     echo "</tr></thead>
           <tbody>";
     for ($i=0;$i < mysqli_num_rows($result);$i++)
     {
        echo "<tr>";
        $row = mysqli_fetch_row($result);
        foreach($row as $value)
        {
           echo "<td>".$value."</td>";
        }
        echo "</tr>";
     }
     echo "</tbody></table>";
  }
 /* Display form with only buttons after results */
  $query = str_replace("'","%&%",$_POST['query']);
  echo "<hr><br>
      <form action='{$_SERVER['PHP_SELF']}' method='POST'>
        <input type='hidden' name='query' value='$query'>
        <input type='hidden' name='database'
               value={$_POST['database']}>
        <input type='submit' name='queryButton'
               value='New Query'>
        <input type='submit' name='queryButton'
```

```
                value='Edit Query'>
      </form>";
  exit();
}

/* Displays form for query input */
if (@$_POST['queryButton'] != "Edit Query")
{
   $query = " ";
}
else
{
   $query = str_replace("%&%","'",$_POST['query']);
}
?>
<form action="<?php echo $_SERVER['PHP_SELF'] ?>"
      method="POST">
<table>
 <tr><td style='text-align: right; font-weight: bold'>
         Type in database name</td>
     <td><input type="text" name="database"
           value=<?php echo @$_POST['database'] ?> ></td>
 </tr>
 <tr><td style='text-align: right; font-weight: bold'
         valign="top">Type in SQL query</td>
     <td><textarea name="query" cols="60"
           rows="10"><?php echo $query ?></textarea></td>
 </tr>
 <tr><td colspan="2" style='text-align: center'>
       <input type="submit" value="Submit Query"></td>
 </tr>
</table>
<input type="hidden" name="form" value="yes">
</form>
</body></html>
```



The program in Listing 4-1 will not run correctly if you are using the `mysql` functions, rather than the `mysqli` functions. The difference between `mysql` and `mysqli` functions is discussed in Chapter 2. You must change the `mysqli` functions to `mysql` functions, with some small syntax changes as well, as discussed in the beginning of Chapter 8. In addition, this program uses the function `mysqli_fetch_fields`, which has no comparable `mysql` function. Consequently, you must replace lines 39–43, shown next:

```
$finfo = mysqli_fetch_fields($result);
foreach($finfo as $field)
{
   echo "<th>".$field->name."</th>";
}
```

with the following lines:

```
for($i = 0;$i < mysql_num_fields($result);$i++)
{
   echo "<th>".mysql_field_name($result,$i).
   "</th>";
}
```

Whether you are using the program as it is in Listing 4-1 or changing the program to use the `mysql` functions mentioned in the preceding warning, you need to change lines 13, 14, and 15 of the program before you can use it. These lines are

```
$host="hostname";
$user="mysqlaccountname";
$password="mysqlpassword";
```

Change *`hostname`* to the name of the computer where MySQL is installed, for example, `databasehost.mycompany.com`. If the MySQL database is installed on the same computer as your Web site, you can use `localhost`.

Change *`mysqlaccountname`* and *`mysqlpassword`* to the account name and password that you were given by the MySQL administrator to use to access your database. If you installed MySQL yourself, an account named `root` is automatically installed. The `root` account may be installed with no password or you may have been prompted to enter a password for `root` during the installation process. Sometimes an account with a blank account name and password is installed. You can use the `root` or the blank account, but it's much better if you install an account specifically for use with your Web database application. (I discuss MySQL accounts and passwords in detail in Chapter 5.)



An account named `root` with no password is not secure. You should give the root account a password right away. An account with a blank account name and password is even less secure because anyone can access your database without needing to know an account name or a password. You should delete this account if it exists (see Chapter 5).

If your MySQL account doesn't require a password, type nothing between the double quotes, as follows:

```
$password="";
```

After you enter the correct hostname, account name, and password in `mysqlsend.php`, these are the general steps that you follow to execute a query:

1. **Point your browser at `mysql_send.php`.**

   You see the Web page shown in Figure 4-1.

2. **In the large text box, type the SQL query.**

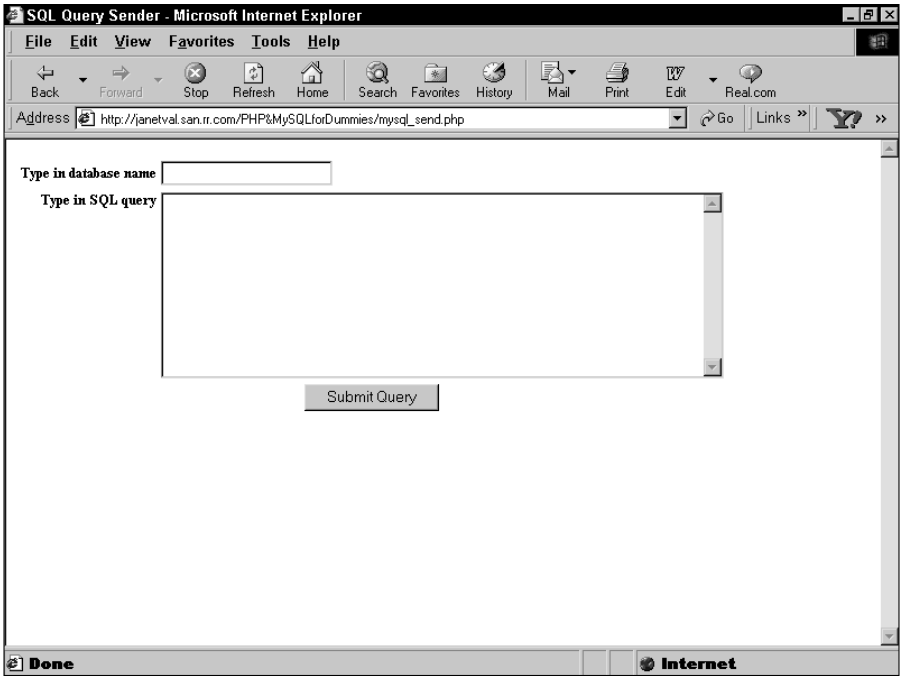


Figure 4-1: An SQL query Web page produced by mysql_send.php.

3. **In the first text box, enter a database name if the SQL query requires one.**

   I explain the details of writing specific SQL queries in the following sections of this chapter.

4. **Click the Submit Query button.**

   The query is executed, and a page is displayed, showing the results of the query. If your query had an error, the error message is displayed.

You can test the `mysql_send.php` program by entering the following test query in Step 2 of the preceding steps:

```
SHOW DATABASES
```

This query does not require you to enter a database name, so you can skip Step 3. When you click the Submit Query button in Step 4, a listing of existing databases is displayed. In most cases, you see a database called `Test`, which is installed automatically when MySQL is installed. Also, you'll probably see a database called `mysql`, which MySQL uses to store information that it needs, such as account names, passwords, and permissions. Even if there are no existing databases, your SQL query will execute correctly. If a problem occurs, an error message is displayed. MySQL error messages are usually pretty helpful in finding the problem.



A quicker way to send SQL queries to the MySQL server

When MySQL is installed, a simple, text-based program called `mysql` (or sometimes the *terminal monitor* or the *monitor*) is also installed. Programs that communicate with servers are *client software;* because this program communicates with the MySQL server, it's a client. When you enter SQL queries in this client, the response is returned to the client and displayed onscreen. The monitor program can send queries across a network; it doesn't have to be running on the machine where the database is stored.

To send SQL queries to MySQL by using the `mysql` client, follow these steps:

1. **Locate the `mysql` client.**

   By default, the `mysql` client program is installed in the subdirectory `bin`, under the directory where MySQL is installed. In Unix/Linux, the default is `/usr/local/mysql/bin` or `/usr/local/bin`. In Windows, the default is `c:\Program Files\MySQL\MySQL Server 5.0\bin`. However, the client might be installed in a different directory. Or, if you're not the MySQL administrator, you might not have access to the `mysql` client. If you don't know where MySQL is installed or can't run the client, ask the MySQL administrator to put the client somewhere where you can run it or to give you a copy that you can put on your own computer.

2. **Start the client.**

   In Unix and Linux, type the path/filename (for example, `/usr/local/mysql/bin/mysql`). In Windows, open a command prompt window and then type the path\filename (for example, `c:\ Program Files\MySQL\MySQL Server 5.0\bin\mysql`). This command will start the client if you don't need to use an account name or a password. If you need to enter an account or a password or both, use the following parameters:

   `-u` *`user`*: *`user`* is your MySQL account name.

   `-p`: This parameter prompts you for the password for your MySQL account.

   For instance, if you're in the directory where the `mysql` client is located, the command might look like this:

   ```
   mysql -u root -p
   ```

3. **If you're starting the `mysql` client to access a database across the network, use the following parameter after the `mysql` command:**

   `-h` *`host`*: *`host`* is the name of the machine where MySQL is located.

   For instance, if you're in the directory where the `mysql` client is located, the command might look like this:

   ```
   mysql -h mysqlhost.mycompany.com -u root -p
   ```

   Press Enter after typing the command.

4. **Enter your password when prompted for it.**

The `mysql` client starts, and you see something similar to this:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 459 to server version: 5.0.15
Type 'help;' or '\h' for help. Type '\c' to clear the
       buffer.
mysql>
```

5. **Select the database that you want to use.**

   At the `mysql` prompt, type the following: `use` *`databasename`*.

   Use the name of the database that you want to query.

6. **At the `mysql` prompt, type your SQL query followed by a semicolon (;), and then press the Enter key.**

   The `mysql` client continues to prompt for input and does not execute the query until you enter a semicolon. The response to the query is displayed onscreen.

7. **To leave the `mysql` client, type** quit **at the prompt and then press the Enter key.**

Building a Database

A database has two parts: a structure to hold the data and the data itself. In the following few sections, I explain how to create the database structure. First you create an empty database with no structure at all, and then you add tables to it.

The SQL queries that you use to work with the database structure are `CREATE`, `ALTER`, `DROP`, and `SHOW`. To use these queries, you must have a MySQL account that has permission to create, alter, and drop databases and tables. See Chapter 5 for more on MySQL accounts.

Creating a new database

To create a new, empty database, use the following SQL query:

```
CREATE DATABASE databasename
```

where *`databasename`* is the name that you give the database. For instance, these two SQL queries create the sample databases used in this book:

```
CREATE DATABASE PetCatalog
CREATE DATABASE MemberDirectory
```



Some Web hosting companies don't allow you to create a new database. You are given one database to use with MySQL, and you can create tables in only this one database. You can try requesting another database, but you need a good reason. MySQL and PHP don't care that all your tables are in one database instead of organized into databases with meaningful names. It's just easier for humans to keep track of projects when they're organized.

To see for yourself that a database was in fact created, use this SQL query:

```
SHOW DATABASES
```

After you create an empty database, you can add tables to it. (Adding tables to a database is described later in this chapter.)

Deleting a database

You can delete any database with this SQL query:

```
DROP DATABASE databasename
```



Use `DROP` carefully because it is irreversible. After a database is dropped, it is gone forever. And any data that was in it is gone as well.

Adding tables to a database

You can add tables to any database, whether it's a new, empty database that you just created or an existing database that already has tables and data in it. You use the `CREATE` query to add tables to a database.

In the sample database designs that I introduce in Chapter 3, the `PetCatalog` database is designed with three tables: `Pet`, `PetType`, and `PetColor`. The `MemberDirectory` database is designed with two tables: `Member` and `Login`. Because a table is created in a database, you must indicate the database name where you want the table created. That is, when using the form shown in Figure 4-1, you must type a database name in the top field. If you don't, you see the error message `No Database Selected`.

The query to add a table begins with

```
CREATE TABLE tablename
```

Next comes a list of column names with definitions. The information for each column is separated from the information for the next column by a comma. The entire list is enclosed in parentheses. Each column name is followed by its data type (I explain data types in detail in Chapter 3) and any other definitions required. Here are some definitions that you can use:

- `NOT NULL`: This column must have a value; it can't be empty.
- `DEFAULT value`: This value is stored in the column when the row is created if no other value is given for this column.
- `AUTO_INCREMENT`: You use this definition to create a sequence number. As each row is added, the value of this column increases by one integer from the last row entered. You can override the auto number by assigning a specific value to the column.
- `UNSIGNED`: You use this definition to indicate that the values for this numeric field will never be negative numbers.

The last item in a `CREATE TABLE` query indicates which column or combination of columns is the unique identifier for the row — the *primary key*. Each row of a table must have a field or a combination of fields that is different for each row. No two rows can have the same primary key. If you attempt to add a row with the same primary key as a row already in the table, you get an error message, and the row is not added. The database design identifies the primary key (as I describe in Chapter 3). You specify the primary key by using the following format:

```
CREATE TABLE Member (
  loginName     VARCHAR(20) NOT NULL,
  createDate    DATE        NOT NULL),
PRIMARY KEY(columnname) )
```

The *columnname* is enclosed in parentheses. If you're using a combination of columns as the primary key, include all the column names, separated by commas. For instance, you would designate the primary key for the `Login` table in the `MemberDirectory` database by using this in the `CREATE` query:

```
PRIMARY KEY (loginName,loginTime)
```

Listing 4-2 shows the `CREATE TABLE` query used to create the `Member` table of the `MemberDirectory` database. You could enter this query on a single line if you wanted to. MySQL doesn't care how many lines you use. However, the format shown in Listing 4-2 makes it easier to read. This human-friendly format also helps you spot typos.

Listing 4-2: An SQL Query for Creating a Table

```
CREATE TABLE Member (
  loginName     VARCHAR(20) NOT NULL,
  createDate    DATE        NOT NULL,
  password      CHAR(255)   NOT NULL,
  lastName      VARCHAR(50),
  firstName     VARCHAR(40),
  street        VARCHAR(50),
  city          VARCHAR(50),
  state         CHAR(2),
  zip           CHAR(10),
```

Listing 4-2 *(continued)*

```
   email          VARCHAR(50),
   phone          CHAR(15),
   fax            CHAR(15),
PRIMARY KEY(loginName) )
```

Notice that the list of column names in Listing 4-2 is enclosed in parentheses (one on the first line and one on the last line), and a comma follows each column definition.



Remember not to use any MySQL reserved words for column names, as I discuss in Chapter 3. If you do, MySQL gives you an error message that looks like this:

```
You have an error in your SQL syntax near 'order var(20))' at line 1
```

Note that this message shows the column definition that it didn't like and the line where it found the offending definition. However, the message doesn't tell you much about what the problem is. The `error in your SQL syntax` that it refers to is the use of the MySQL reserved word `order` as a column name.

After a table has been created, you can query to see it, review its structure, or remove it.

- **To see the tables that have been added to a database, use this query:**

  ```
  SHOW TABLES
  ```

- **To see the structure of a table, use this query:**

  ```
  DESCRIBE tablename
  ```

- **To remove any table, use this query:**

  ```
  DROP TABLE tablename
  ```

  Use `DROP` carefully because it is irreversible. After a table is dropped, it is gone forever. And any data that was in it is gone as well.



Changing the database structure

Your database isn't written in stone. By using the `ALTER` query, you can change the name of the table; add, drop, or rename a column; or change the data type or other attributes of the column.

The basic format for this query is `ALTER TABLE tablename`, followed by the specified changes. Table 4-1 shows the changes that you can make.

Table 4-1 Changes You Can Make with the ALTER Query

Change	*Description*
`ADD columnname definition`	Adds a column; `definition` includes the data type and optional definitions.
`ALTER columnname SET DEFAULT value`	Changes the default value for a column.
`ALTER columnname DROP DEFAULT`	Removes the default value for a column.
`CHANGE columnname newcolumnname definition`	Changes the definition of a column and renames the column; `definition` includes the data type and optional definitions.
`DROP columnname`	Deletes a column, including all the data in the column. The data cannot be recovered.
`MODIFY columnname definition`	Changes the definition of a column; `definition` includes the data type and optional definitions.
`RENAME newtablename`	Renames a table.

Changing a database is not a rare occurrence. You might want to change your database for many reasons. For example, suppose that you defined the column `lastName` with `VARCHAR(20)` in the `Member` table of the `MemberDirectory` database. At the time, 20 characters seemed sufficient for a last name. But now you just received a memo announcing the new CEO, John Schwartzheimer-Losertman. Oops. MySQL will truncate his name to the first 20 letters, a less-than-desirable new name for the boss. So you need to make the column wider — pronto. Send this query to change the column in a second:

```
ALTER TABLE Member MODIFY lastName VARCHAR(50)
```

Moving Data Into and Out of the Database

An empty database is like an empty cookie jar — it's not much fun. And searching an empty database is no more interesting or fruitful than searching an empty cookie jar. A database is only useful with respect to the information that it holds.

A database needs to be able to receive information for storage and to deliver information on request. For instance, the `MemberDirectory` database needs to be able to receive the member information, and it also needs to be able to deliver its stored information when you request it. If you want to know the address of a particular member, for example, the database needs to deliver that information when you request it.

Your MySQL database responds to four types of requests:

- **Adding information:** Adding a row to a table.
- **Updating information:** Changing information in an existing row. This includes adding data to a blank field in an existing row.
- **Retrieving information:** Looking at the data. This request does not remove data from the database.
- **Removing information:** Deleting data from the database.

Sometimes your question requires information from more than one table. For instance, the question, "How much does a green dragon cost?" requires information from the `Pet` table and from the `Color` table. You can ask this question easily in a single `SELECT` query by combining the tables.

In the following sections, I discuss how to receive and deliver information as well as how to combine tables.

Adding information

Every database needs data. For example, you might want to add data to your database so that your users can look at it — an example of this is the Pet Catalog that I introduce in Chapter 3. Or you might want to create an empty database for users to put data into, making the data available for your eyes only — an example of this is the Member Directory. In either scenario, data will be added to the database.

If your data is still on paper, you can enter it directly into a MySQL database, one row at a time, by using an SQL query. However, if you have a lot of data, this process could be tedious and involve a lot of typing. Suppose that you have information on 1000 products that must be added to your database. Assuming that you're greased lightening on a keyboard and can enter a row per minute, that's 16 hours of rapid typing — well, rapid editing, anyway. Doable, but not fun. On the other hand, suppose that you need to enter 5000 members of an organization into a database and that it takes five minutes to enter each member. Now you're looking at more than 400 hours of typing — who has time for that?

If you have a large amount of data to enter, consider some alternatives. Sometimes scanning in the data is an option. Or perhaps you need to beg, borrow, or hire some help. In many cases, it could be faster to enter the data into a big text file than to enter each row in a separate SQL query.

The SQL query LOAD can read data from a big text file (or even a small text file). So, if your data is already in a computer file, you can work with that file; you don't need to type all the data again. Even if the data is in a format other than a text file (for example, in an Excel, Access, or Oracle file), you can usually convert the file to a big text file, which can then be read into your MySQL database. If the data isn't yet in a computer file and there's a lot of data, it might be faster to enter that data into the computer in a big text file and transfer it into MySQL as a second step.

Most text files can be read into MySQL, but some formats are easier than others. If you're planning to enter the data into a big text file, read the section, "Adding a bunch of data," to find the best format. Of course, if the data is already on the computer, you have to work with the file as it is.

Adding one row at a time

You use the INSERT query to add a row to a database. This query tells MySQL which table to add the row to and what the values are for the fields in the row. The general form of the query is

```
INSERT INTO tablename (columnname, columnname,...,columnname)
     VALUES (value, value,...,value)
```

The following rules apply to the INSERT query:

- **Values must be listed in the same order in which the column names are listed.** The first value in the value list is inserted into the column that's named first in the column list; the second value in the value list is inserted into the column that's named second; and so on.
- **A partial column list is allowed.** You don't need to list all the columns. Columns that are not listed are given their default value or left blank if no default value is defined.
- **A column list is not required.** If you're entering values for all the columns, you don't need to list the columns at all. If no columns are listed, MySQL will look for values for all the columns, in the order in which they appear in the table.
- **The column list and value list must be the same length.** If the list of columns is longer or shorter than the list of values, you get an error message like this: `Column count doesn't match value count`.

The following INSERT query adds a row to the Member table:

```
INSERT INTO Member (loginName,createDate,password,lastName,
                    street,city,state,zip,email,phone,fax)
       VALUES ("bigguy","2001-Dec-2","secret","Smith",
               "1234 Happy St","Las Vegas","NV","88888",
               "gsmith@GSmithCompany.com","(555) 555-5555","")
```

Notice that `firstName` is not listed in the column name list. No value is entered into the `firstName` field. If `firstName` were defined as `NOT NULL`, MySQL would not allow this. Also, if the definition for `firstName` included a default, the default value would be entered, but because it doesn't, the field is left empty. Notice that the value stored for `fax` is an empty string.

To look at the data that you entered and ensure that you entered it correctly, use an SQL query that retrieves data from the database. I describe these SQL queries in detail in "Retrieving information," later in this chapter. In brief, the following query retrieves all the data in the `Member` table:

```
SELECT * FROM Member
```

Adding a bunch of data

If you have a large amount of data to enter and it's already in a computer file, you can transfer the data from the existing computer file to your MySQL database. The SQL query that reads data from a text file is `LOAD`. The `LOAD` query requires you to specify a database.

Because data in a database is organized in rows and columns, the text file being read must indicate where the data for each column begins and ends and where the end of a row is. To indicate columns, a specific character separates the data for each column. By default, MySQL looks for a tab character to separate the fields. However, if a tab doesn't work for your data file, you can choose a different character to separate the fields and tell MySQL in the query that a different character than the tab separates the fields. Also by default, the end of a line is expected to be the end of a row — although you can choose a character to indicate the end of a line if you need to. A data file for the `Pet` table might look like this:

```
Unicorn<TAB>horse<TAB>Spiral horn<Tab>5000.00<Tab>/pix/unicorn.jpg
Pegasus<TAB>horse<TAB>Winged<Tab>8000.00<Tab>/pix/pegasus.jpg
Lion<TAB>cat<TAB>Large; Mane on neck<Tab>2000.00<Tab>/pix/lion.jpg
```

A data file with tabs between the fields is a *tab-delimited* file. Another common format is a *comma-delimited* file, where commas separate the fields. If your data is in another file format, you need to convert it into a delimited file.



To convert data in another file format into a delimited file, check the manual for that software or talk to your local expert who understands the data's current format. Many programs, such as Excel, Access, and Oracle, allow you to output the data into a delimited file. For a text file, you might be able to convert it to delimited format by using the search-and-replace function of an editor or word processor. For a truly troublesome file, you might need to seek the help of an expert or a programmer.

The basic form of the `LOAD` query is

```
LOAD DATA INFILE "path/datafilename" INTO TABLE tablename
```

The query loads data from a text file located on your server. If the filename does not include a path, MySQL looks for the data file in the directory where your table definition file, called *tablename*.frm, is located. By default, this file is located in a directory named for your database, such as a directory named PetDirectory. This directory is located in your data directory, which is located in the main directory where MySQL is installed. For example, if the file was named data.dat, the LOAD command might look for the file at C:\Program Files\MySQL\MySQL Server 5.0\data\PetDirectory\data.dat.

This basic form can be followed by optional phrases if you want to change a default delimiter. The options are

```
FIELDS TERMINATED BY 'character'
FIELDS ENCLOSED BY 'character'
LINES TERMINATED BY 'character'
```

Suppose that you have the data file for the Pet table, shown previously in this section, except that the fields are separated by a comma rather than a tab. The name of the data file is pets.dat, and it's located in the same directory as the database. The SQL query to read the data into the table is

```
LOAD DATA INFILE "pets.dat" INTO TABLE Pet
          FIELDS TERMINATED BY ','
```



To use the LOAD DATA INFILE query, the MySQL account must have the FILE privilege on the server host. I discuss MySQL account privileges in Chapter 5.

You can also load data from a text file on your local computer by using the word LOCAL, as follows:

```
LOAD DATA LOCAL INFILE "path/datafilename"
    INTO TABLE tablename
```

You must include a path to the file. Use forward slashes for the path, even on a Windows computer, such as "C:/data/datafile1.txt". If you get an error message when sending this query, LOCAL may not be enabled. Enabling LOCAL is discussed in Chapter 5.

To look at the data that you loaded — to make sure that it's correct — use an SQL query that retrieves data from the database. I describe these types of SQL queries in detail in the next section. In brief, use the following query to look at all the data in the table so that you can check it:

```
SELECT * FROM Pet
```

Retrieving information

The only purpose in storing information is to have it available when you need it. A database lives to answer questions. What pets are for sale? Who are the members? How many members live in Arkansas? Do you have an alligator for sale? How much does a dragon cost? What is Goliath Smith's phone number? And on and on. You use the SELECT query to ask the database questions.

The simplest, basic SELECT query is

```
SELECT * FROM tablename
```

This query retrieves all the information from the table. The asterisk (*) is a wildcard meaning *all the columns.*

The SELECT query can be much more selective. SQL words and phrases in the SELECT query can pinpoint the information needed to answer your question. You can specify what information you want, how you want it organized, and the source of the information:

- **You can request only the information (the columns) that you need to answer your question.** For instance, you can request only the first and last names to create a list of members.
- **You can request information in a particular order.** For instance, you can request that the information be sorted in alphabetical order.
- **You can request information from selected objects (the rows) in your table.** (See Chapter 3 for an explanation of database objects.) For instance, you can request the first and last names for only those members whose addresses are in Florida.



In MySQL 4.1, MySQL added the ability to nest a SELECT query inside another query. The nested query is called a *subquery*. You can use a subquery in SELECT, INSERT, UPDATE, or DELETE queries or in SET clauses. A subquery can return a single value, a single row or column, or a table, which is used in the outer query. All the features of SELECT queries can be used in subqueries. See the MySQL online manual at dev.mysql.com/doc/refman/5.0/en/subqueries.html for detailed information on using subqueries.

Retrieving specific information

To retrieve specific information, list the columns containing the information you want. For example:

```
SELECT columnname,columnname,columnname,... FROM tablename
```

This query retrieves the values from all the rows for the indicated column(s). For instance, the following query retrieves all the last names and first names stored in the `Member` table:

```
SELECT lastName,firstName FROM Member
```

You can perform mathematical operations on columns when you select them. For example, you can use the following `SELECT` query to add two columns:

```
SELECT col1+col2 FROM tablename
```

Or you could use the following query:

```
SELECT price,price*1.08 FROM Pet
```

The result is the price and the price with the sales tax of 8 percent added. You can change the name of a column when selecting it, as follows:

```
SELECT price,price*1.08 AS priceWithTax FROM Pet
```

The `AS` clause tells MySQL to give the name `priceWithTax` to the second column retrieved. Thus, the query retrieves two columns of data: `price` and `priceWithTax`.

In some cases, you don't want to see the values in a column, but you want to know something about the column. For instance, you might want to know the lowest value in the column or the highest value in the column. Table 4-2 lists some of the information that is available about a column.

Table 4-2 Information That Can Be Selected

SQL Format	*Description of Information*
`AVG(`*`columnname`*`)`	Returns the average of all the values in *`columnname`*
`COUNT(`*`columnname`*`)`	Returns the number of rows in which *`columnname`* is not blank
`MAX(`*`columnname`*`)`	Returns the largest value in *`columnname`*
`MIN(`*`columnname`*`)`	Returns the smallest value in *`columnname`*
`SUM(`*`columnname`*`)`	Returns the sum of all the values in *`columnname`*

For example, the query to find out the highest price in the `Pet` table is

```
SELECT MAX(price) FROM Pet
```



SQL words that look like `MAX()` and `SUM()`, with parentheses following the name, are *functions.* SQL provides many functions in addition to those in Table 4-2. Some functions, like those in Table 4-2, provide information about a column. Other functions change each value selected. For example, `SQRT()` returns the square root of each value in the column, and `DAYNAME()` returns the name of the day of the week for each value in a date column, rather than the actual date stored in the column. More than 100 functions are available for use in a `SELECT` query. For descriptions of all the functions, see the MySQL online manual at `dev.mysql.com/doc/refman/5.0/en/functions.html`.

Retrieving data in a specific order

You might want to retrieve data in a particular order. For instance, in the `Member` table, you might want members organized in alphabetical order by last name. Or, in the `Pet` table, you might want the pets grouped by type of pet.

In a `SELECT` query, `ORDER BY` and `GROUP BY` affect the order in which the data is delivered to you:

- `ORDER BY`: To sort information, use the phrase

  ```
  ORDER BY columnname
  ```

  The data is sorted by *`columnname`* in ascending order. For instance, if *`columnname`* is `lastName`, the data is delivered to you in alphabetical order by the last name.

  You can sort in descending order by adding the word `DESC` before the column name. For example:

  ```
  SELECT * FROM Member ORDER BY DESC lastName
  ```

- `GROUP BY`: To group information, use the following phrase:

  ```
  GROUP BY columnname
  ```

  The rows that have the same value of *`columnname`* are grouped together. For example, use this query to group the rows that have the same value as `petType`:

  ```
  SELECT * FROM Pet GROUP BY petType
  ```



You can use `GROUP BY` and `ORDER BY` in the same query.

Retrieving data from a specific source

Frequently, you don't want all the information from a table. You want information from selected database objects, that is, *rows.* Three SQL words are frequently used to specify the source of the information:

- `WHERE`: Allows you to request information from database objects with certain characteristics. For instance, you can request the names of members who live in California, or you can list only pets that are cats.

- LIMIT: Allows you to limit the number of rows from which information is retrieved. For instance, you can request all the information from the first three rows in the table.
- DISTINCT: Allows you to request information from only one row of identical rows. For instance, in the Login table, you can request loginName but specify no duplicate names, thus limiting the response to one record for each member. This would answer the question, "Has the member ever logged in?" rather than the question "How many times has the member logged in?"

The WHERE clause of the SELECT query enables you to make complicated selections. For instance, suppose your boss asks for a list of all the members whose last names begin with *B,* who live in Santa Barbara, and who have an 8 in either their phone or fax number. I'm sure there are many uses for such a list. You can get this list for your boss with a SELECT query by using a WHERE clause.

The basic format of the WHERE clause is

```
WHERE expression AND|OR expression AND|OR expression ...
```

expression specifies a value to compare with the values stored in the database. Only the rows containing a match for the expression are selected. You can use as many expressions as needed, each one separated by AND or OR. When you use AND, both of the expressions connected by the AND (that is, both the expression before the AND *and* the expression after the AND) must be true in order for the row to be selected. When you use OR, only one of the expressions connected by the OR must be true for the row to be selected.

Some common expressions are shown in Table 4-3.

Table 4-3 Expressions for the WHERE Clause

Expression	*Example*	*Result*
column = value	zip="12345"	Selects only the rows where 12345 is stored in the column named zip
column > value	zip > "50000"	Selects only the rows where the zip code is 50001 or higher
column >= value	zip >= "50000"	Selects only the rows where the zip code is 50000 or higher

(continued)

Table 4-3 *(continued)*

Expression	*Example*	*Result*
column < *value*	zip < "50000"	Selects only the rows where the zip code is 49999 or lower
column <= *value*	zip <= "50000"	Selects only the rows where the zip code is 50000 or lower
column BETWEEN *value1* AND *value2*	zip BETWEEN "20000" AND "30000"	Selects only the rows where the zip code is greater than 19999 but less 30001
column IN (*value1*,*value2*,...)	zip IN ("90001", "30044")	Selects only the rows where the zip code is 90001 or 30044
column NOT IN (*value1*,*value2*,...)	zip NOT IN ("90001", "30044")	Selects only the rows where the zip code is any zip code except 90001 or 30044
column LIKE *value* — *value* can contain the wildcards % (which matches any string) and _ (which matches any character)	zip LIKE "9%"	Selects all rows where the zip code begins with 9
column NOT LIKE *value* — *value* can contain the wildcards % (which matches any string) and _ (which matches any character)	zip NOT LIKE "9%"	Selects all rows where the zip code does not begin with 9

You can combine any of the expressions in Table 4-3 with ANDs and ORs. In some cases, you need to use parentheses to clarify the selection criteria. For instance, you can use the following query to answer your boss's urgent need to find all people in the Member Directory whose names begin with *B,* who live in Santa Barbara, and who have an 8 in either their phone or fax number:

```
SELECT lastName,firstName FROM Member
       WHERE lastName LIKE "B%"
         AND city = "Santa Barbara"
         AND (phone LIKE "%8%" OR fax LIKE "%8%")
```

Notice the parentheses in the last line. You would not get the results that your boss asked for without the parentheses. Without the parentheses, each connector would be processed in order from the first to the last, resulting in a list that includes all members whose names begin with *B* and who live in Santa Barbara and whose phone numbers have an 8 in them *and* all members whose fax numbers have an 8 in them, whether they live in Santa Barbara or not and whether their name begins with a *B* or not. When the last `OR` is processed, members are selected whose characteristics match the expression before the `OR` *or* the expression after the `OR`. The expression before the `OR` is connected to previous expressions by the previous `AND`s and so does not stand alone, but the expression after the `OR` does stand alone, resulting in the selection of all members with an 8 in their fax number.

`LIMIT` specifies how many rows can be returned. The form for `LIMIT` is

```
LIMIT startnumber,numberofrows
```

The first row that you want to retrieve is *`startnumber`*, and the number of rows to retrieve is *`numberofrows`*. If *`startnumber`* is not specified, 1 is assumed. To select only the first three members who live in Texas, use this query:

```
SELECT * FROM Member WHERE state="TX" LIMIT 3
```



Some `SELECT` queries will find identical records, but in this example you want to see only one — not all — of the identical records. To prevent the query from returning all identical records, add the word `DISTINCT` immediately after `SELECT`.

Combining information from tables

In previous sections of this chapter, I assume that all the information you want is in a single table. However, you might want to combine information from different tables. You can do this easily in a single query.

Two words can be used in a `SELECT` query to combine information from two or more tables:

- `UNION`: Rows are retrieved from one or more tables and stored together, one after the other, in a single result. For example, if your query selected 6 rows from one table and 5 rows from another table, the result would contain 11 rows.
- `JOIN`: The tables are combined side by side, and the information is retrieved from both tables.

UNION

UNION is used to combine the results from two or more select queries. The results from each query are added to the result set following the results of the previous query. The format of the UNION query is as follows:

```
SELECT query UNION ALL SELECT query ...
```

You can combine as many SELECT queries as you need. A SELECT query can include any valid SELECT format, including WHERE clauses, LIMIT clauses, and so on. The rules for the queries are

- All the select queries must select the same number of columns.
- The columns selected in the queries must contain the same type of data.

The result set will contain all the rows from the first query followed by all the rows from the second query and so on. The column names used in the result set are the column names from the first SELECT query.

The series of SELECT queries can select different columns from the same table, but situations in which you want a new table with one column in a table followed by another column from the same table are unusual. It's much more likely that you want to combine columns from different tables. For example, you might have a table of members who have resigned from the club and a separate table of current members. You can get a list of all members, both current and resigned, with the following query:

```
SELECT lastName,firstName FROM Member UNION ALL
    SELECT lastName,firstName FROM OldMember
```

The result of this query is the last and first names of all current members, followed by the last and first names of all the members who have resigned.

Depending on how you organized your data, you might have duplicate names. For instance, perhaps a member resigned, and his name is in the OldMember table — but he joined again, so his name is added to the Member table. If you don't want duplicates, don't include the word ALL. If ALL is not included, duplicate lines are not added to the result.

You can use ORDER BY with each SELECT query, as I discuss in the previous section, or you can use ORDER BY with a UNION query to sort all the rows in the result set. If you want ORDER BY to apply to the entire result set, rather than just to the query that it follows, use parentheses as follows:

```
(SELECT lastName FROM Member UNION ALL
     SELECT lastName FROM OldMember) ORDER BY lastName
```



The UNION statement was introduced in MySQL 4.0. It is not available in MySQL 3.

Join

Combining tables side by side is a *join.* Tables are combined by matching data in a column — the column that they have in common. The combined results table produced by a join contains all the columns from both tables. For instance, if one table has two columns (memberID and height), and the second table has two columns (memberID and weight), a join results in a table with four columns: memberID (from the first table), height, memberID (from the second table), and weight.

The two common types of joins are an *inner join* and an *outer join*. The difference between an inner and outer join is in the number of rows included in the results table. The results table produced by an inner join contains only rows that existed in both tables. The combined table produced by an outer join contains all rows that existed in one table with blanks in the columns for the rows that did not exist in the second table. For instance, if table1 contains a row for Joe and a row for Sally, and table2 contains only a row for Sally, an inner join would contain only one row: the row for Sally. However, an outer join would contain two rows — a row for Joe and a row for Sally — even though the row for Joe would have a blank field for weight.

The results table for the outer join contains all the rows for one table. If any of the rows for that table don't exist in the second table, the columns for the second table are empty. Clearly, the contents of the results table are determined by which table contributes all its rows, requiring the second table to match it. Two kinds of outer joins control which table sets the rows and which match: a LEFT JOIN and a RIGHT JOIN.

You use different SELECT queries for an inner join and the two types of outer joins. The following query is an inner join:

```
SELECT columnnamelist FROM table1,table2
                      WHERE table1.col2 = table2.col2
```

And these queries are outer joins:

```
SELECT columnnamelist FROM table1 LEFT JOIN table2
       ON table1.col1=table2.col2
```

```
SELECT columnnamelist FROM table1 RIGHT JOIN table2
       ON table1.col1=table2.col2
```

In all three queries, `table1` and `table2` are the tables to be joined. You can join more than two tables. In both queries, `col1` and `col2` are the names of the columns being matched to join the tables. The tables are matched based on the data in these columns. These two columns can have the same name or different names. The two columns must contain the same type of data.

As an example of inner and outer joins, consider a short form of the `Pet Catalog`. One table is `Pet`, with the two columns `petName` and `petType` holding the following data:

```
petName         petType

Unicorn         Horse
Pegasus         Horse
Lion            Cat
```

The second table is `Color`, with two columns `petName` and `petColor` holding the following data:

```
petName         petColor

Unicorn         white
Unicorn         silver
Fish            Gold
```

You need to ask a question that requires information from both tables. If you do an inner join with the following query:

```
SELECT * FROM Pet,Color WHERE Pet.petName = Color.petName
```

you get the following results table with four columns: `petName` (from `Pet`), `petType`, `petName` (from `Color`), and `petColor`.

```
petName         petType         petName         petColor

Unicorn         Horse           Unicorn         white
Unicorn         Horse           Unicorn         silver
```

Notice that only `Unicorn` appears in the results table — because only `Unicorn` was in both of the original tables, before the join. On the other hand, suppose you do a left outer join with the following query:

```
SELECT * FROM Pet LEFT JOIN Color
         ON Pet.petName=Color.petName
```

You get the following results table, with the same four columns — `petName` (from `Pet`), `petType`, `petName` (from `Color`), and `petColor` — but with different rows:

petName	petType	petName	petColor
Unicorn	Horse	Unicorn	white
Unicorn	Horse	Unicorn	silver
Pegasus	Horse	<NULL>	<NULL>
Lion	Cat	<NULL>	<NULL>

This table has four rows. It has the same first two rows as the inner join, but it has two additional rows — rows that are in the `PetType` table on the left but not in the `Color` table. Notice that the columns from the table `Color` are blank for the last two rows.

And, on the third hand, suppose that you do a right outer join with the following query:

```
SELECT * FROM Pet RIGHT JOIN Color
         ON Pet.petName=Color.petName
```

You get the following results table, with the same four columns, but with still different rows:

petName	petType	petName	petColor
Unicorn	Horse	Unicorn	white
Unicorn	Horse	Unicorn	silver
<NULL>	<NULL>	Fish	Gold

Notice that these results contain all the rows for the `Color` table on the right but not for the `Pet` table. Notice the blanks in the columns for the `Pet` table, which doesn't have a row for `Fish`.

The joins that I've talked about so far find matching entries in tables. Sometimes it's useful to find out which rows in a table have no matching entries in another table. For example, suppose that you want to know who has never logged into your Members Only section. Because you have one table with the member's login name and another table with the login dates, you can ask this question by using the two tables. You can find out which login names do not have an entry in the Login table with the following query:

```
SELECT loginName from Member LEFT JOIN Login
       ON Member.loginName=Login.loginName
       WHERE Login.loginName IS NULL
```

This query will give you a list of all the login names in `Member` that are not in the `Login` table.

Updating information

Changing information in an existing row is *updating* the information. For instance, you might need to change the address of a member because she has moved, or you might need to add a fax number that a member left blank when he originally entered his information.

The UPDATE query is straightforward:

```
UPDATE tablename SET column=value,column=value,...
        WHERE clause
```

In the SET clause, you list the columns to be updated and the new values to be inserted. List all the columns that you want to change in one query. Without a WHERE clause, the values of the column(s) would be changed in all rows. But with the WHERE clause, you can specify which rows to update. For instance, to update an address in the Member table, use this query:

```
UPDATE Member SET street="3333 Giant St",
                  phone="555-555-5555"
              WHERE loginName="bigguy"
```

Removing information

Keep the information in your database up to date by deleting obsolete information. You can remove a row from a table with the DELETE query:

```
DELETE FROM tablename WHERE clause
```



Be extremely careful when using DELETE. If you use a DELETE query without a WHERE clause, it will delete all the data in the table. I mean *all the data.* I repeat, *all the data.* The data cannot be recovered. This function of the DELETE query is right at the top of my don't-try-this-at-home list.

You can delete a column from a table by using the ALTER query:

```
ALTER TABLE tablename DROP columnname
```

Or you could remove the whole thing and start over again with

```
DROP TABLE tablename
```

or

```
DROP DATABASE databasename
```

Chapter 5

Protecting Your Data

In This Chapter

- Understanding MySQL data security
- Adding new MySQL accounts
- Modifying existing accounts
- Changing passwords
- Making backups
- Restoring data

Your data is essential to your Web database application. You have spent valuable time developing your database, and it contains important information entered by you or by your users. You need to protect it. In this chapter, I show you how.

Controlling Access to Your Data

You need to control access to the information in your database. You need to decide who can see the data and who can change it. Imagine what would happen if your competitors could change the information in your online product catalog or copy your list of customers — you'd be out of business in no time flat. Clearly, you need to guard your data.

MySQL provides a security system for protecting your data. No one can access the data in your database without an account. Each MySQL account has the following attributes:

- A name
- A *hostname* — the machine from which the account can access the MySQL server
- A password
- A set of permissions

To access your data, someone must use a valid account name and know the password associated with that account. In addition, that person must be connecting from a computer that is permitted to connect to your database via that specific account.

After the user is granted access to the database, what he or she can do to the data depends on what permissions have been set for the account. Each account is either allowed or not allowed to perform an operation in your database, such as `SELECT`, `DELETE`, `INSERT`, `CREATE`, or `DROP`. The settings that specify what an account can do are *privileges,* or *permissions.* You can set up an account with all permissions, no permissions, or anything in between. For instance, for an online product catalog, you want the customer to be able to see the information in the catalog but not be able to change it.

When a user attempts to connect to MySQL and execute a query, MySQL controls access to the data in two stages:

1. **Connection verification:** MySQL checks the validity of the account name and password and checks whether the connection is coming from a host that is allowed to connect to the MySQL server by using the specified account. If everything checks out, MySQL accepts the connection.
2. **Request verification:** After MySQL accepts the connection, it checks whether the account has the necessary permissions to execute the specified query. If it does, MySQL executes the query.

Any query that you send to MySQL can fail either because the connection is rejected in the first step or because the query is not permitted in the second step. An error message is returned to help you identify the source of the problem.

In the following few sections, I describe accounts and permissions in detail.

Understanding account names and hostnames

Together, the account name and *hostname* (the name of the computer that is authorized to connect to the database) identify a unique account. Two accounts with the same name but different hostnames can exist and can have different passwords and permissions. However, you *cannot* have two accounts with the same name *and* the same hostname.

The MySQL server will accept connections from a MySQL account only when it is connecting from *hostname*. When you build the `GRANT` or `REVOKE` query (which I describe later in this chapter), you identify the MySQL account by using both the account name and the hostname in the following format: *accountname*`@`*hostname* (for instance, `root@localhost`).



The MySQL account name is completely unrelated in any way to the Unix, Linux, or Windows user name (also sometimes called the *login name*). If you're using an administrative MySQL account named `root`, it is not related to the Unix or Linux `root` login name. Changing the MySQL login name does not affect the Unix, Linux, or Windows login name — and vice versa.

MySQL account names and hostnames are defined as follows:

- **An account name can be up to 16 characters long.** You can use special characters in account names, such as a space or a hyphen (-). However, you cannot use wildcards in the account name.
- **An account name can be blank.** If an account exists in MySQL with a blank account name, any account name will be valid for that account. A user could use any account name to connect to your database, given that the user is connecting from a hostname that is allowed to connect to the blank account name and uses the correct password, if required. You can use an account with a blank name to allow anonymous users to connect to your database.
- **The hostname can be a name or an IP address.** For example, it can be a name such as `thor.mycompany.com` or an IP (Internet protocol) address such as `192.163.2.33`. The machine on which the MySQL server is installed is `localhost`.
- **The hostname can contain wildcards.** You can use a percent sign (`%`) as a wildcard; `%` matches any hostname. If you add an account for `george@%`, someone using the account named `george` can connect to the MySQL server from any computer.
- **The hostname can be blank.** A blank hostname is the same as using `%` for the hostname.

An account with a blank account name and a blank hostname is possible. Such an account would allow anyone to connect to the MySQL server by using any account name from any computer. An account with a blank name and a percent sign (`%`) for the hostname is the same thing. It is unlikely that you would want such an account. Such an account is sometimes installed when MySQL is installed, but it's given no privileges, so it can't do anything.



When MySQL is installed, it automatically installs an account with all privileges: `root@localhost`. Depending on your operating system, this account may be installed without a password. Anyone who is logged in to the computer on which MySQL is installed can access MySQL and do anything to it by using the account named `root`. (Of course, `root` is a well-known account name, so this account is not secure. If you're the MySQL administrator, you should add a password to this account immediately.)



On some operating systems, additional accounts besides `root@localhost` are automatically installed. For instance, on Windows, an account called `root@%` might be installed with no password protection. This `root` account with all privileges can be used by anyone from any machine. You should remove this account immediately or, at the very least, give it a password.

Finding out about passwords

A password is set up for every account. If no password is provided for the account, the password is blank, which means that no password is required. MySQL doesn't have any limit for the length of a password, but sometimes other software on your system limits the length to eight characters. If so, any characters after eight are dropped.

For extra security, MySQL encrypts passwords before it stores them. That means passwords are not stored in the recognizable characters that you entered. This security measure ensures that no one can look at the stored passwords and see what they are.

Unfortunately, some bad people out there might try to access your data by guessing your password. They use software that tries to connect rapidly in succession using different passwords — a practice called *cracking*. The following are some recommendations for choosing a password that is as difficult to crack as possible:

- Use six to eight characters.
- Include one or more of each of the following — uppercase letter, lowercase letter, number, and punctuation mark.
- Do not use your account name or any variation of your account name.
- Do not include any word in a dictionary, including foreign language dictionaries.
- Do not include a name.
- Do not use a phone number or a date.

A good password is hard to guess and easy to remember. If it's too hard to remember, you might need to write it down, which defeats the purpose of having a password. One way to create a good password is to use the first characters of a favorite phrase. For instance, you could use the phrase "All for one! One for all!" to make this password:

```
Afo!Ofa!
```

This password doesn't include any numbers, but you can fix that by using the numeral *4* instead of the letter *f*. Then your password is

```
A4o!O4a!
```

Or you could use the number *1* instead of the letter *o* to represent one. Then the password is

```
A41!14a!
```

This password is definitely hard to guess. Other ways to incorporate numbers into your passwords include substituting *1* (one) for the letter *l* or substituting *0* (zero) for the letter *o*.

Taking a look at account permissions

MySQL uses account permissions to specify who can do what. Anyone using a valid account can connect to the MySQL server, but he or she can only do those things that are allowed by the permissions for the account. For example, an account might be set up so that users can select data but cannot insert or update data.

Permissions can be granted for particular databases, tables, or columns. For instance, an account can be set up that allows the user to select data from all the tables in the database but insert data into only one table and update only a single column in a specific table.

Permissions are added by using the `GRANT` query and removed by using the `REVOKE` query. The `GRANT` or `REVOKE` query must be sent using an account that has permission to execute `GRANT` or `REVOKE` statements in the database. If you attempt to send a `GRANT` query or a `REVOKE` query using an account without grant permission, you get an error message. For instance, if you try to grant permission to use a select query, and you send the query using an account that does not have permission to grant permissions, you might see the following error message:

```
grant command denied
```

Permissions can be granted or removed individually or all at once. Table 5-1 lists some permissions that you might want to assign or remove.

Table 5-1 MySQL Account Permissions

Permission	*Description*
ALL	All permissions
ALTER	Can alter the structure of tables
CREATE	Can create new databases or tables
DELETE	Can delete rows in tables
DROP	Can drop databases or tables
FILE	Can read and write files on the server
GRANT	Can change the permissions on a MySQL account
INSERT	Can insert new rows into tables
SELECT	Can read data from tables
SHUTDOWN	Can shut down the MySQL server
UPDATE	Can change data in a table
USAGE	No permissions



Granting ALL is not a good idea because it includes permissions for administrative operations, such as shutting down the MySQL server. You are unlikely to want anyone other than yourself to have such sweeping privileges.

Setting Up MySQL Accounts

An *account* is identified by the account name and the name of the computer allowed to access MySQL using this account. When you create a new account, you specify it as *accountname*@*hostname*. You can specify a password when you create an account or you can add a password later. You can set up permissions when you create an account or add permissions later.

All the account information is stored in a database named mysql that is automatically created when MySQL is installed. To add a new account or change any account information, you must use an account that has the proper permissions on the mysql database.



The MySQL security database

When MySQL is installed, it automatically creates a database called `mysql`. All the information used to protect your data is stored in this database, including account names, hostnames, passwords, and permissions.

Permissions are stored in columns. The format of each column name is *`permission`*`_priv`, where *`permission`* is one of the permissions shown in Table 5-1. For instance, the column containing `ALTER` permissions is named `alter_priv`. The value in each permission column is `Y` or `N`, meaning *yes* or *no.* So, for instance, in the user table (described in the following list), there would be a row for an account and a column for `alter_priv`. If the account field for `alter_priv` contains `Y`, the account can be used to execute an `ALTER` query. If `alter_priv` contains `N`, the account doesn't have permission to execute an `ALTER` query.

The `mysql` database contains the following tables that store permissions:

- `user` table: This table stores permissions that apply to all the databases and tables. It contains a row for each valid account that includes the column's user name, hostname, and password. The MySQL server will reject a connection for an account that does not exist in this table.
- `db` table: This table stores permissions that apply to a particular database. It contains a row for the database, which gives permissions to an account name and a hostname. The account must exist in the `user` table for the permissions to be granted. Permissions that are given in the `user` table overrule permissions in this table. For instance, if the `user` table has a row for the account *designer* that gives `INSERT` privileges, *designer* can insert into all the databases. If a row in the `db` table shows `N` for `INSERT` for the *designer* account in the `PetCatalog` database, the `user` table overrules it, and *designer* can insert in the `PetCatalog` database.
- `host` table: This table controls access to a database depending on the host. The `host` table works with the `db` table. If a row in the `db` table has an empty field for the host, MySQL checks the `host` table to see whether the `db` has a row there. In this way, you can allow access to a `db` from some hosts but not from others. For instance, suppose you have two databases: `db1` and `db2`. The `db1` database has sensitive information, so you want only certain people to see it. The `db2` database has information that you want everyone to see. If you have a row in the `db` table for `db1` with a blank `host` field, you can have two rows for `db1` in the `host` table. One row can give all permissions to users connecting from a specific host, whereas another row can deny privileges to users connecting from any other host.
- `tables_priv` table: This table stores permissions that apply to specific tables.
- `columns_priv` table: This table stores permissions that apply to specific columns.

You can see and change the tables in `mysql` directly if you're using an account that has the necessary permissions. You can use SQL queries such as `SELECT`, `INSERT`, and `UPDATE`. If you're accessing MySQL through your employer, a client, or a Web hosting company, it is unlikely that you will be given an account that has the necessary permissions.

You need at least one account to access the MySQL server. When MySQL is installed, it automatically sets up some accounts, including an account called `root` that has all permissions. If you have MySQL access through a company Web site or a Web hosting company, the MySQL administrator for the company should give you the account; the account is probably not named `root`, and it might or might not have all permissions.

In the rest of this section, I describe how to add and delete accounts and change passwords and permissions for accounts. If you have an account that you received from your company IT department or from a Web hosting company, you might receive an error when you try to send any or some of the `GRANT` or `REVOKE` queries described. If your account is restricted from performing any of the necessary queries, you need to request an account with more permissions or ask the MySQL administrator to add a new account or make the changes you need.

Identifying what accounts currently exist

To see what accounts currently exist for your database, you need an account that has the necessary permissions. Try to execute the following query on a database named `mysql`:

```
SELECT * FROM user
```

You should get a list of all the accounts. However, if you're accessing MySQL through your company or a Web hosting company, you probably don't have the necessary permissions. In that case, you might get an error message like this:

```
No Database Selected
```

This message means that your account is not allowed to select the `mysql` database. Or you might get an error message saying that you don't have `SELECT` permission. Even though this message is annoying, it's a sign that the company has good security measures in place. However, it also means that you can't see what privileges your account has. You must ask your MySQL administrator or try to figure it out yourself by trying queries and seeing whether you're allowed to execute them.

Adding accounts

The preferred way to access MySQL from PHP is to set up an account specifically for this purpose with only the permissions that are needed. In this section,

I describe how to add accounts. If you're using an account given to you by a company IT department or a Web hosting company, it might or might not have all the permissions needed to create an account. If it doesn't, you won't be able to successfully execute the query to add an account, and you'll have to request a second account to use with PHP.



If you need to request a second account, get an account with restricted permission (if at all possible) because your Web database application will be more secure if the account used in your PHP programs doesn't have more privileges than are necessary.

To create one or more users, you can use the `CREATE USER` query added to MySQL in version 5.0.2, as follows:

```
CREATE USER accountname@hostname IDENTIFIED BY 'password',
accountname@hostname IDENTIFIED BY 'password',...
```

This query creates the specified new user account(s) with the specified password and no permissions. You do not need to specify a password. If you leave out `IDENTIFIED BY 'password'`, the account is created with no password. You can add or change a password for the account at a later time. Adding passwords and permissions is discussed in the following sections.

If you're using a version of MySQL before 5.0.2, you must use a `GRANT` query to create an account. The `GRANT` query is described in the "Changing permissions" section.

Adding and changing passwords

You can add or change a password for an existing account with the `SET PASSWORD` query, as follows:

```
SET PASSWORD FOR username@hostname = PASSWORD('password')
```

The account is set to *password* for the account *username@hostname*. If the account currently has a password, the password is changed. You do not need to specify the `FOR` clause. If you do not, the password is set for the account you are currently using.

You can remove a password by sending the `SET PASSWORD` query with an empty password, as follows:

```
SET PASSWORD FOR username@hostname = PASSWORD('')
```

Changing permissions

You can see the current permissions for an account with the following query:

```
SHOW GRANTS ON accountname@hostname
```

The output is a GRANT query that would create the current account. It shows all the current permissions. If you do not include the ON clause, you see the current permissions for the account that issued the SHOW GRANTS query.

You can change permissions for an account with the GRANT query, which has the following general format:

```
GRANT permission (columns) ON tablename
      TO accountname@hostname IDENTIFIED BY 'password'
```

You can also create a new account or change a password with the GRANT query. You need to fill in the following information:

- *permission* (*columns*): You must list at least one permission. You can limit each permission to one or more columns by listing the column name in parentheses following the permission. If no column name is listed, the permission is granted on all columns in the table(s). You can list as many permissions and columns as needed, separated by commas. The possible permissions are listed in Table 5-1. For instance, a GRANT query might start with this:

  ```
  GRANT select (firstName,lastName), update,
        insert (birthdate) ...
  ```

- *tablename*: This indicates which tables the permission is granted on. At least one table is required. You can list several tables, separated by commas. The possible values for *tablename* are

  - *tablename*: The entire table named *tablename* in the current database. You can use an asterisk (*) to mean all tables in the current database. If you use an asterisk and no current database is selected, the privilege will be granted to all tables on all databases.
  - *databasename.tablename*: The entire table named *tablename* in *databasename*. You can use an asterisk (*) for either the database name or the table name to mean *all*. Using *.* grants the permission on all tables in all databases.

- *accountname@hostname*: If the account already exists, it is given the indicated permissions. If the account doesn't exist, it's added. The account is identified by the *accountname* and the *hostname* as a pair. If an account exists with the specified account name but a different hostname, the existing account is not changed; a new one is created.

- *password*: This is the password that you're adding or changing. A password is not required. If you don't want to add or change a password for this account, leave out the phrase IDENTIFIED BY '*password*'.

The GRANT query to add a new account for use in the PHP programs for the PetCatalog database might be

```
GRANT select ON PetCatalog.* TO phpuser@localhost
             IDENTIFIED BY 'A41!14a!'
```

Removing accounts and permissions

To remove an account, you can use the DROP USER query, which was added in MySQL 4.1.1, as follows:

```
DROP USER accountname@hostname, accountname@hostname, ...
```

You must be using an account that has DELETE privileges on the mysql database to execute the DROP USER query.

The behavior of DROP USER has changed through MySQL versions. As of MySQL 5.0.2, it removes the account and all records related to the account, including records that give it permissions on specific databases or tables. However, before MySQL 5.0.2, DROP USER drops only accounts with no privileges. Therefore, in older versions, you must remove all the privileges from an account, including database or table permissions, before you can drop it.

To remove permissions, use the REVOKE query. The general format is

```
REVOKE permission (columns) ON tablename
       FROM accountname@hostname
```

You need to fill in the following information:

- *permission (columns)*: You must list at least one permission. You can remove each permission from one or more columns by listing the column name in parentheses following the permission. If no column name is listed, the permission is removed from all columns in the table(s). You can list as many permissions/columns as needed, separated by commas. The possible permissions are listed in Table 5-1. For instance, a REVOKE query might start like this:

  ```
  REVOKE select (firstName,lastName), update, insert
         (birthdate) ...
  ```

- *tablename*: Indicate which tables the permission is being removed from. At least one table is required. You can list several tables, separated by commas. The possible values for *tablename* are

  - *tablename*: The entire table named *tablename* in the current database. You can use an asterisk (*) to mean all tables. If you use an asterisk when no current database is selected, the privilege will be revoked on all tables in all databases.

- *databasename*.*tablename*: The entire table named *tablename* in *databasename*. You can use an asterisk (*) for either the database name or the table name to mean *all*. Using *.* revokes the permission on all tables in all databases.

- *accountname*@*hostname*: The account is identified by the *accountname* and the *hostname* as a pair. If an account exists with the specified account name but a different hostname, the REVOKE query will fail, and you will receive an error message.

You can remove all the permissions for an account with the following REVOKE query:

```
REVOKE all ON *.* FROM accountname@hostname
```

Backing Up Your Data

You need to have at least one copy of your valuable database. Disasters occur rarely, but they do occur. The computer where your database is stored can break down and lose your data, the computer file can become corrupted, the building can burn down, and so on. Backup copies of your database guard against data loss from such disasters.

You should have at least one backup copy of your database, stored in a location that is separate from the copy that is currently in use. More than one copy — perhaps as many as three — is usually a good idea:

- Store one copy in a handy location, perhaps even on the same computer, to quickly replace a working database that has been damaged.
- Store a second copy on another computer in case the computer breaks down, and the first backup copy isn't available.
- Store a third copy in a different physical location, for that remote chance that the building burns down. If the second backup copy is stored via a network on a computer at another physical location, this third copy isn't needed.



If you don't have access to a computer offsite where you can back up your database, you can copy your backup to a portable medium, such as a CD or DVD, and store it offsite. Certain companies will store your computer media at their location for a fee, or you can just put the media in your pocket and take it home.

If you use MySQL on someone else's computer, such as the computer of your employer or a Web hosting company, the people who provide your access are responsible for backups. They should have automated procedures in place that make backups of your database. When evaluating a Web hosting company, ask about their backup procedures. You want to know how often backup copies are made and where they are stored. If you aren't confident that your data is safe, you can discuss changes or additions to the backup procedures.

If you are the MySQL administrator, you are responsible for making backups. MySQL provides a program called `mysqldump` that you can use to make backup copies. The `mysqldump` program creates a text file that contains all the SQL statements needed to re-create your entire database. The file contains the `CREATE` statements for each table and `INSERT` statements for each row of data in the tables. You can restore your database by executing the set of MySQL statements. You can restore it in its current location, or you can restore it on another computer if necessary.

Follow these steps to make a backup copy of your database in Linux, in Unix, or on a Mac:

1. **Change to the `bin` subdirectory in the directory where MySQL is installed.**

   For instance, type **cd /usr/local/mysql/bin**.

2. **Type the following:**

   ```
   mysqldump --user=accountname --password=password
        databasename >path/backupfilename
   ```

   where

   - *`accountname`* is the name of the MySQL account that you're using to back up the database
   - *`password`* is the password for the account
   - *`databasename`* is the name of the database that you want to back up
   - *`path/backupfilename`* is the path to the directory where you want to store the backups and the filename the SQL output will be stored in



The account that you use needs to have select permission. If the account doesn't require a password, you can leave out the entire password option.

You can type the command on one line, without pressing Enter. Or you can type a backslash (\), press Enter, and continue the command on another line.

For example, to back up the `PetCatalog` database, the command might be

```
mysqldump --user=root --password=secret PetCatalog \
>/usr/local/mysql/backups/PetCatalogBackup
```

Note: With Linux or Unix, the account that you are logged into must have permission to write a file into the backup directory.

To make a backup copy of your database in Windows, follow these steps:

1. **Open a command prompt window.**

   For instance, choose Start➪All Programs➪Accessories➪Command prompt.

2. **Change to the `bin` subdirectory in the directory where MySQL is installed.**

   For instance, type **cd c:\Program Files\MySQL\MySQL Server 5.0\bin**.

3. **Type the following:**

   ```
   mysqldump --user=accountname --password=password
           databasename >path\backupfilename
   ```

   where

   - *`accountname`* is the name of the MySQL account that you're using to back up the database
   - *`password`* is the password for the account
   - *`databasename`* is the name of the database that you want to back up
   - *`path\backupfilename`* is the path to the directory where you want to store the backups and the filename the SQL output will be stored in

The account that you use needs to have select permission. If the account does not require a password, you can leave out the entire password option.



You must type the `mysqldump` command on one line without pressing Enter.

For example, to back up the `PetCatalog` database, the command might be

```
mysqldump --user=root PetCatalog >PetCatalogBackup
```

Backups should be made at certain times — at least once per day. If your database changes frequently, you might want to back up more often. For example, you might want to back up to the backup directory hourly but back up to another computer once a day.

Restoring Your Data

At some point, one of your database tables might become damaged and unusable. It's unusual, but it happens. For instance, a hardware problem or an unexpected shutdown of the computer can cause corrupted tables. Sometimes an anomaly in the data that confuses MySQL can cause corrupt tables. In some cases, a corrupt table can cause your MySQL server to shut down.

Here is a typical error message that signals a corrupted table:

```
Incorrect key file for table: 'tablename'.
```

You can replace the corrupted table(s) with the data stored in a backup copy. In some cases, the database might be lost completely. For instance, if the computer where your database resides breaks down and can't be fixed, your current database is lost, but your data isn't gone forever. You can replace the broken computer with a new computer and restore your database from a backup copy.

You can replace your current database table(s) with the database stored in a backup copy. The backup copy contains a snapshot of the data as it was when the copy was made. Any changes to the database since the backup copy was made are not included; you have to re-create those changes manually.

Again, if you access MySQL through an IT department or through a Web hosting company, you need to ask the MySQL administrator to restore your database from a backup. If you're the MySQL administrator, you can restore it yourself.

As I describe in Chapter 4, you build a database by creating the database and then adding tables to the database. The backup created by the `mysqldump` utility is a file that contains all the SQL statements necessary to rebuild the tables, but it does not contain the statements needed to create the database.

Your database might not exist, or it could exist with one or more corrupted tables. You can restore the entire database or any single table. Follow these steps to restore a single table:

1. **If the table currently exists, delete the table with the following SQL query:**

   ```
   DROP TABLE tablename
   ```

   where *`tablename`* is the table that you want to delete.

2. **Point your browser at `mysql_send.php`.**

   For a description of `mysql_send.php`, see Chapter 4.

3. **Copy the `CREATE` query for the table from the backup file into the form in the browser window.**

   For instance, choose Edit➪Copy and Edit➪Paste.

4. **Type the name of the database in which you are restoring the table.**

   The form shows where to type the database name.

5. **Click Submit.**

   A new Web page shows the results of the query.

6. **Click New Query.**

7. **Copy an `INSERT` query for the table from the backup file into the form in the browser window.**

   For instance, choose Edit➪Copy and Edit➪Paste.

8. **Type the name of the database in which you are restoring the table.**

   The form shows where to type the database name.

9. **Click Submit.**

   A new Web page shows the results of the query.

10. **Click New Query.**

11. **Repeat Steps 7–10 until all the `INSERT` queries from the backup file have been sent.**

If you have so many `INSERT` queries for the table that sending them one by one would take forever — or if there are just a lot of tables — you can send all the queries in the backup file at once. First, you may need to edit the backup file, as follows:

1. **Open the backup file in a text editor.**
2. **Locate the line that shows the Server Versions.**
3. **If you want to rebuild an entire database, add the following statement after the line located in Step 2:**

   ```
   CREATE DATABASE IF NOT EXISTS databasename
   ```

4. **After the line in Step 3, add a line specifying which database to add the tables to:**

   ```
   USE databasename
   ```

5. **Check the blocks of statements that rebuild the tables. If you do not want to rebuild a table, comment out the lines that rebuild the table by adding – (two hyphens) in front of each line.**
6. **Check the `INSERT` lines for each table. If you do not want to add data to any tables, comment out the lines that `INSERT` the data.**
7. **Save the edited backup file.**

After the backup file contains the statements that you want to use to rebuild your database or table(s), you need to perform a few more steps.

On Linux, Unix, and Mac:

1. **Change to the `bin` subdirectory in the directory where MySQL is installed.**

   Type a `cd` command to change to the correct directory. For instance, type **cd /usr/local/mysql/bin**.

2. **Type the command that sends the SQL queries in the backup file:**

   ```
   mysql -u accountname -p < path/backupfilename
   ```

   where *accountname* is an account that has create permission. If the account doesn't require a password, leave out the `-p`. If you use the `-p`, you will be asked for the password. Use the entire path and filename for the backup file. For instance, a command to restore the `PetCatalog` database might be

   ```
   mysql -u root -p < /usr/backupfiles/PetCatalog.bak
   ```

On Windows:

1. **Change to the `bin` subdirectory in the directory where MySQL is installed.**

   a. **Open a command prompt window.**

      For instance, choose Start➪All Programs➪Accessories➪Command Prompt.

   b. **Type a `cd` command to change to the correct directory.**

      For instance, type **cd c:\Program Files\MySQL\MySQL Server 5.0\bin**.

2. **Type the command that sends the SQL queries in the backup file:**

   ```
   mysql -u accountname -p < path\backupname
   ```

   where *accountname* is an account that has create permission. If the account doesn't require a password, leave out the `-p`. If you use the `-p`, you will be asked for the password. Use the entire path and filename for the backup file. For instance, a command to restore the `PetCatalog` database might be

   ```
   mysql -u root -p < c:\Program Files\MySQL\MySQL Server
           5.0\bin\bak\PetCatalog.bak
   ```

The tables might take a short time to restore. Wait for the command to finish. If a problem occurs, an error message is displayed. If no problems occur, you see no output. When the command is finished, the prompt appears.

Your database is now restored with all the data that was in it at the time the copy was made. If the data has changed since the copy was made, the changes are lost. For instance, if more data was added after the backup copy was made, the new data is not restored. If you know the changes that were made, you can make them manually in the restored database.

Upgrading MySQL

New versions of MySQL are released periodically, and you can upgrade from one version of MySQL to a newer version. Upgrading information is provided in the MySQL manual at `dev.mysql.com/doc/refman/5.0/en/upgrade.html`. However, there are special considerations when upgrading. As a precaution, it is wise to back up your current databases, including the grant tables in the `mysql` database, before upgrading.

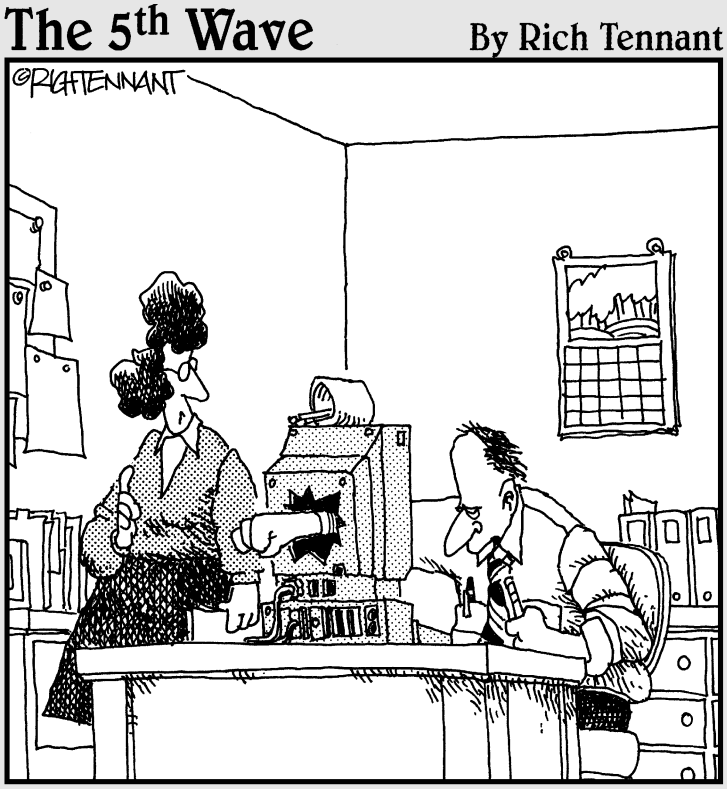
MySQL recommends that you do not skip versions. If you want to upgrade from one version to a version more than one version newer, such as from MySQL 4.0 to MySQL 5.0, you should upgrade to the next version first. After that version is working correctly, you can upgrade to the next version. And so on. In other words, upgrade from 4.0 to 4.1, then from 4.1 to 5.0.

Occasionally, incompatible changes are introduced in new versions of MySQL. Some releases introduce changes to the structure of the grant tables. For instance, MySQL 4.1 changed the method of encrypting passwords, requiring a longer password field in the grant tables.

After upgrading to the newer version, you should run the `mysql_upgrade` script. It checks your files, repairing them if needed, and upgrades the system tables if needed. Before MySQL version 5.0.19, the `mysql_upgrade` script does not run on Windows; it runs only on Unix. On Windows, you can run a script called `mysql_fix_privileges_tables` with MySQL versions prior to 5.0.19. The script upgrades the system tables but does not perform the complete table check and repair that `mysql_upgrade` performs.

Part III

PHP



"Do you want me to call the company and have them send another review copy of their database software system, or do you know what you are going to write?"

In this part . . .

In Part III, you find out how to use PHP for your Web database application. Here are some of the topics described:

- Adding PHP to HTML files
- PHP features that are useful for building a dynamic Web database application
- Using PHP features
- Using forms to collect information from users
- Showing information from a database on a Web page
- Storing data in a database
- Moving information from one Web page to the next

You find out everything you need to know to write PHP programs.

Chapter 6

General PHP

In This Chapter

- Adding PHP sections to HTML files
- Writing PHP statements
- Using PHP variables
- Comparing values in PHP variables
- Documenting your programs

Programs are the application part of your Web database application. Programs perform the tasks. Programs create and display Web pages, accept and process information from users, store information in the database, get information out of the database, and perform any other necessary tasks.

PHP, the language that you use to write your programs, is a scripting language designed for use on the Web. It has features to aid you in programming the tasks needed by dynamic Web applications.

In this chapter I describe the general rules for writing PHP programs — the rules that apply to all PHP statements. Consider these rules similar to general grammar and punctuation rules. In the remaining chapters in Part III, you find out about specific PHP statements and features and how to write PHP programs to perform specific tasks.

Adding a PHP Section to an HTML Page

PHP is a partner to HTML (HyperText Markup Language), enabling HTML to do things it can't do on its own. For example, HTML can display Web pages, and HTML has features that allow you to format those Web pages. HTML also allows you to display graphics in your Web pages and to play music files. But HTML alone does not allow you to interact with the person viewing the Web page.

HTML is almost interactive. That is, HTML forms allow users to type information that the Web page is designed to collect; however, you can't access that information without using a language other than HTML. PHP processes form information and allows other interactive tasks as well.

HTML tags are used to make PHP language statements part of HTML scripts. The file is named with a `.php` extension. (The PHP administrator can define other extensions, such as `.phtml` or `.php5`, but `.php` is the most common. In this book, I assume `.php` is the extension for PHP programs.) The PHP language statements are enclosed in PHP tags with the following form:

```
<?php        ?>
```



Sometimes you can use a shorter version of the PHP tags. You can try using `<?` and `?>` without the `php`. If short tags are enabled, you can save a little typing. However, if you use short tags, your programs will not run if they are moved to another Web host where PHP short tags are not activated.

PHP processes all statements between the two PHP tags. After the PHP section is processed, it's discarded. Or if the PHP statements produce output, the PHP section is replaced by the output. The browser doesn't see the PHP section — the browser sees only its output, if there is any. For more on this process, see the sidebar, "How the Web server processes PHP files."

As an example, I'll start with an HTML program that displays `Hello World!` in the browser window, shown in Listing 6-1. (It's a tradition that the first program you write in any language is the Hello World program. You might have written a Hello World program when you first learned HTML.)



How the Web server processes PHP files

When a browser is pointed to a regular HTML file with an `.html` or `.htm` extension, the Web server sends the file, as-is, to the browser. The browser processes the file and displays the Web page described by the HTML tags in the file.

When a browser is pointed to a PHP file (with a `.php` extension), the Web server looks for PHP sections in the file and processes them instead of just sending them as-is to the browser.

The Web server processes the PHP file as follows:

1. The Web server starts scanning the file in HTML mode. It assumes the statements are HTML and sends them to the browser without any processing.
2. The Web server continues in HTML mode until it encounters a PHP opening tag (`<?php`).
3. When it encounters a PHP opening tag, the Web server switches to PHP mode. This is sometimes called *escaping from HTML*. The Web server then assumes that all statements are PHP statements and executes the PHP statements. If there is output, the output is sent by the server to the browser.
4. The Web server continues in PHP mode until it encounters a PHP closing tag (`?>`).
5. When the Web server encounters a PHP closing tag, it returns to HTML mode. It resumes scanning, and the cycle continues from Step 1.

Listing 6-1: The Hello World HTML Program

```
<html>
<head><title>Hello World Program</title></head>
<body>
<p>Hello World!
</body>
</html>
```

If you point your browser at this HTML program, you see a Web page that displays

```
Hello World!
```

Listing 6-2 shows a PHP program that does the same thing — it displays `Hello World!` in a browser window.

Listing 6-2: The Hello World PHP Program

```
<html>
<head><title>Hello World Program</title></head>
<body>
<?php
  echo "<p>Hello World!"
?>
</body>
</html>
```

If you point your browser at this program, it displays the same Web page as the HTML program in Listing 6-1.



Don't look at the file directly with your browser. That is, don't choose File➪Open➪Browse from your browser menu to navigate to the file and click it. You must open the file by typing its URL, as I discuss in Chapter 2. If you see the PHP code displayed in the browser window instead of the output that you expect, you might not have pointed to the file by using its URL.

In this PHP program, the PHP section is

```
<?php
  echo "<p>Hello World!"
?>
```

The PHP tags enclose only one statement — an `echo` statement. The `echo` statement is a PHP statement that you will use frequently. It simply outputs the text is included between the double quotes.

There is no rule that says you must enter the PHP on separate lines. You could just as well include the PHP in the file on a single line, like this:

```
<?php echo "<p>Hello World!" ?>
```

When the PHP section is processed, it is replaced with the output. In this case, the output is

```
<p>Hello World!
```

If you replace the PHP section in Listing 6-2 with the preceding output, the program now looks exactly like the HTML program in Listing 6-1. If you point your browser at either program, you see the same Web page. If you look at the source code that the browser sees (in the browser, choose View➪Source), you see the same source code listing for both programs.

Writing PHP Statements

The PHP section that you add to your HTML file consists of a series of PHP statements. Each PHP statement is an instruction to PHP to do something. In the Hello World program shown in Listing 6-2, the PHP section contains only one simple PHP statement. The `echo` statement instructs PHP to output the text between the double quotes.



PHP statements end with a semicolon (`;`). PHP does not notice white space or the end of lines. It continues reading a statement until it encounters a semicolon or the PHP closing tag, no matter how many lines the statement spans. Leaving out the semicolon is a common error, resulting in an error message that looks something like this:

```
Parse error: expecting `','' or `';'' in /hello.php on
          line 6
```

Error messages and warnings

PHP tries to be helpful when problems arise. It provides error messages and warnings as follows:

- **Parse Error:** A Parse Error is a syntax error that PHP finds when it scans the script before executing it. A parse error is a fatal error, preventing the script from running at all. A parse error looks similar to the following:

  ```
  Parse error: parse error, error, in c:\test\test.php on line 6
  ```

  Often, you receive this error message because you've forgotten a semicolon, a parenthesis, or a curly brace. The error provides more information when possible. For instance, `error might be unexpected T_ECHO, expecting ',' or ';'` means that PHP found an `echo` statement where it was expecting a comma or a semicolon, which probably means you forgot the semicolon at the end of the previous line.

- **Error message:** You receive this message when PHP encounters a serious error during the execution of the program that prevents it from continuing to run. The message contains as much information as possible to help you identify the problem.
- **Warning message:** You receive this message when the program sees a problem but the problem is not serious enough to prevent the program from running. Warning messages do not mean that the program can't run; the program does continue to run. Rather, warning messages tell you that PHP believes that something is probably wrong. You should identify the source of the warning and then decide whether it needs to be fixed. It usually does.
- **Notice:** You receive a notice when PHP sees a condition that might be an error or might be perfectly okay. Notices, like warnings, do not cause the script to stop running. Notices are much less likely than warnings to indicate serious problems. Notices just tell you that you are doing something unusual and to take a second look at what you're doing to be sure that you really want to do it.

  One common reason why you might receive a notice is if you're echoing variables that don't exist. Here's an example of what you might see in that instance:

  ```
  Notice: Undefined variable: age in testing.php on line 9
  ```

- **Strict:** Strict messages, added in PHP 5, warn about language that is poor coding practice or has been replaced by better code.

All types of messages indicate the filename causing the problem and the line number where the problem was encountered.

You can specify which types of error messages you want displayed in the Web page. In general, when you are developing a program, you want to see all messages, but when the program is published on your Web site, you do not want any messages to be displayed to the user.

To change the error-message level for your Web site to show more or fewer messages, you must edit the `php.ini` file on your system. It contains a section that explains the error-message setting (`error_reporting`), error-message levels, and how to set them. Some possible settings are

```
error_reporting = E_ALL | E_STRICT
error_reporting = 0
error_reporting = E_ALL & ~ E_NOTICE
```

The first setting displays `E_ALL`, which is all errors, warnings, and notices except strict, and `E_STRICT`, which displays strict messages. The second setting displays no error messages. The third setting displays all error and warning messages, but not notices or stricts. After changing the `error_reporting` settings, save the edited `php.ini` file and restart your Web server.

If you don't have access to `php.ini`, you can add a statement to a program that sets the error reporting level for that program only. Add the following statement at the beginning of the program:

```
error_reporting(errorSetting);
```

For example, to see all errors except stricts, use the following:

```
error_reporting(E_ALL);
```

Notice that the error message gives you the line number where it encountered problems. This information helps you locate the error in your program. This error message probably means that the semicolon was omitted at the end of line 5.



I recommend writing your PHP programs with an editor that uses line numbers. If your editor doesn't let you specify which line you want to go to, you have to count the lines manually from the top of the file every time that you receive an error message. You can find information about many editors, including descriptions and reviews, at `www.php-editors.com`.

Sometimes groups of statements are combined into a *block.* A block is enclosed by curly braces, `{` and `}`. A block of statements execute together. A common use of a block is in a *conditional block,* in which statements are executed only when certain conditions are true. For instance, you might want your program to do the following:

```
if (the sky is blue)
{
  put leash on dragon;
  take dragon for a walk in the park;
}
```

These statements are enclosed in curly braces to ensure that they execute as a block. If the sky is blue, both `put leash on dragon` and `take dragon for a walk in the park` are executed. If the sky is not blue, neither statement is executed (no leash; no walk).

PHP statements that use blocks, such as `if` statements (which I explain in Chapter 7), are *complex statements.* PHP reads the entire complex statement, not stopping at the first semicolon that it encounters. PHP knows to expect one or more blocks and looks for the ending curly brace of the last block in complex statements. Notice that there is a semicolon before the ending brace. This semicolon is required, but no semicolon is required after the ending curly brace.

If you wanted to, you could write the entire PHP section in one long line, as long as you separated statements with semicolons and enclosed blocks with curly braces. However, a program written this way would be impossible for people to read. Therefore, you should put statements on separate lines, except for occasional, really short statements.



Notice that the statements inside the block are indented. Indenting is not necessary for PHP. Nevertheless, you should indent the statements in a block so that people reading the script can tell more easily where a block begins and ends.

In general, PHP doesn't care whether the statement keywords are in uppercase or lowercase. `Echo`, `echo`, `ECHO`, and `eCHo` are all the same to PHP.

Using PHP Variables

Variables are containers used to hold information. A variable has a name, and information is stored in the variable. For instance, you might name a variable `$age` and store the number 12 in it. After information is stored in a variable, it can be used later in the program. One of the most common uses for variables is to hold the information that a user types into a form.

Naming a variable

When you're naming a variable, keep the following rules in mind:

- All variable names have a dollar sign (`$`) in front of them. This tells PHP that it is a variable name.
- Variable names can be any length.
- Variable names can include letters, numbers, and underscores only.
- Variable names must begin with a letter or an underscore. They cannot begin with a number.
- Uppercase and lowercase letters are not the same. For example, `$firstname` and `$Firstname` are not the same variable. If you store information in `$firstname`, for example, you can't access that information by using the variable name `$firstName`.



When you name variables, use names that make it clear what information is in the variable. Using variable names like `$var1`, `$var2`, `$A`, or `$B` does not contribute to the clarity of the program. Although PHP doesn't care what you name the variable and won't get mixed up, people trying to follow the program will have a hard time keeping track of which variable holds what information. Variable names like `$firstName`, `$age`, and `$orderTotal` are much more descriptive and helpful.

Creating and assigning values to variables

Variables can hold either numbers or strings of characters. You store information in variables by using a single equal sign (`=`). For instance, the following four PHP statements assign information to variables:

```
$age = 12;
$price = 2.55;
$number = -2;
$name = "Goliath Smith";
```

Notice that the character string is enclosed in quotes but the numbers are not. I provide details about using numbers and characters later in this chapter, in the "Working with Numbers" and "Working with Character Strings" sections.

You can now use any of these variable names in an `echo` statement to see the value in that variable. For instance, if you use the following PHP statement in a PHP section:

```
echo $age;
```

the output is `12`. If you include the following line in an HTML file:

```
<p>Your age is <?php echo $age ?>.
```

the output on the Web page is

```
Your age is 12.
```

Whenever you put information into a variable that did not exist before, you create that variable. For instance, suppose you use the following PHP statement:

```
$firstname = "George";
```

If this statement is the first time that you've mentioned the variable `$firstname`, this statement creates the variable and sets it to `"George"`. If you have a previous statement setting `$firstname` to `"Mary"`, this statement changes the value of `$firstname` to `"George"`.

You can also remove information from a variable. For example, the following statement takes information out of the variable `$age`:

```
$age = "";
```

The variable `$age` exists but does not contain a value. It does not mean that `$age` is set to `0` (zero) because 0 is a value. It means that `$age` does not store any information. It contains a string of length 0.

You can go even further and uncreate the variable by using this statement:

```
unset($age);
```

After this statement is executed, the variable `$age` no longer exists.

A variable keeps its information for the entire program, not just for a single PHP section. If a variable is set to `"yes"` at the beginning of a file, it will still hold `"yes"` at the end of the page. For instance, suppose your file has the following statements:

```
<p>Hello World!</p>
<?php
```

```
    $age = 15;
    $name = "Harry";
?>
<p>Hello World again!</p>
<?php
    echo $name;
?>
```

The `echo` statement in the second PHP section will display `Harry`. The Web page resulting from these statements is

```
Hello World!

Hello World again!

Harry
```

Dealing with notices

If you use a statement that includes a variable that does not exist, you might get a notice. It depends on the error-message level that PHP is set to. Remember that notices aren't the same as error messages. With a notice, the program continues to run. A notice simply tells you that you're doing something unusual and to take a second look at what you're doing. (See the sidebar, "Error messages and warnings.") For instance, suppose you use the following statements:

```
unset($age);
echo $age;
$age2 = $age;
```

You might see two notices: one for the second statement and one for the third statement. The notices will look something like this:

```
Notice: Undefined variable: age in testing.php on line 9
```

Suppose that you definitely want to use these statements. The program works exactly the way you want it to. The only problems are the unsightly notices. You can prevent notices in a program by inserting an at sign (`@`) at the point where the notice would be issued. For instance, you can prevent the notices generated by the preceding statements if you change the statements to this:

```
unset($age);
echo @$age;
$age2 = @$age;
```

Instead of suppressing notices in the PHP code with an `@`, you can change the error-message level so that notices are not displayed. For details on how to do this, check out the sidebar, "Error messages and warnings," elsewhere in this chapter.

Using PHP Constants

PHP constants are similar to variables. Constants are given a name, and a value is stored in them. However, constants are constant; that is, they can't be changed by the program. After you set the value for a constant, it stays the same. If you used a constant for `age` and set it to `29`, for example, it can't be changed. Wouldn't that be nice — `29` forever?

Constants are used when a value is needed several places in the program and doesn't change during the program. The value is set in a constant at the start of the program. By using a constant throughout the program, instead of a variable, you make sure that the value won't get changed accidentally. By giving it a name, you know what the information is instantly. And by setting a constant once at the start of the program (instead of using the value throughout the program), you can change the value in one place if it needs changing instead of hunting for it in many places in the program to change it.

For instance, you might set one constant that's the company name and another constant that's the company address and use them wherever needed. Then, if the company moves, you could just change the value in the company address at the start of the program instead of having to find every place in your program that echoed the company name to change it.

Constants are set by using the `define` statement. The format is

```
define("constantname","constantvalue");
```

For instance, to set a constant with the company name, use the following statement:

```
define("COMPANY","ABC Pet Store");
```

Use the constant in your program wherever you need your company name:

```
echo COMPANY;
```

When you echo a constant, you can't enclose it in quotes. If you do, it will echo the constant name, instead of the value. You can echo it without anything, as shown in the preceding example, or enclosed with parentheses.

You can use any name for a constant that you can use for a variable. Constant names are not preceded by a dollar sign (`$`). By convention, constants are given names that are all uppercase, so you can easily spot constants, but PHP itself doesn't care what you name a constant. You can store either a string or a number in it. The following statement is perfectly okay with PHP:

```
define ("AGE",29);
```

Just don't expect Mother Nature to believe it.

Working with Numbers

PHP allows you to do arithmetic operations on numbers. You indicate arithmetic operations with two numbers and an arithmetic operator. For instance, one operator is the plus (+) sign, so you can indicate an arithmetic operation like this:

```
1 + 2
```

You can also perform arithmetic operations with variables that contain numbers, as follows:

```
$n1 = 1;
$n2 = 2;
$sum = $n1 + $n2;
```

Table 6-1 shows the arithmetic operators that you can use.

Table 6-1 Arithmetic Operators

Operator	*Description*
+	Add two numbers.
-	Subtract the second number from the first number.
*	Multiply two numbers.
/	Divide the first number by the second number.
%	Find the remainder when the first number is divided by the second number. This is called *modulus*. For instance, in `$a = 13 % 4`, `$a` is set to `1`.

You can do several arithmetic operations at once. For instance, the following statement performs three operations:

```
$result = 1 + 2 * 4 + 1;
```



The order in which the arithmetic is performed is important. You can get different results depending on which operation is performed first. PHP does multiplication and division first, followed by addition and subtraction. If other considerations are equal, PHP goes from left to right. Consequently, the preceding statement sets `$result` to `10`, in the following order:

```
$result = 1 + 2 * 4 + 1   (first it does the multiplication)
$result = 1 + 8 + 1       (next it does the leftmost addition)
$result = 9 + 1           (next it does the remaining addition)
$result = 10
```

You can change the order in which the arithmetic is performed by using parentheses. The arithmetic inside the parentheses is performed first. For instance, you can write the previous statement with parentheses like this:

```
$result = (1 + 2) * 4 + 1;
```

This statement sets `$result` to `13`, in the following order:

```
$result = (1 + 2) * 4 + 1  (first it does the math in the parentheses)
$result = 3 * 4 + 1        (next it does the multiplication)
$result = 12 + 1           (next it does the addition)
$result = 13
```



On the better-safe-than-sorry principle, it's best to use parentheses whenever more than one answer is possible.

Often, the numbers that you work with are dollar amounts, such as product prices. You want your customers to see prices in the proper format on Web pages. In other words, dollar amounts should always have two decimal places. However, PHP stores and displays numbers in the most efficient format. If the number is 10.00, it is displayed as `10`. To put numbers into the proper format for dollars, you can use `sprintf`. The following statement formats a number into a dollar amount:

```
$newvariablename = sprintf("%01.2f", $oldvariablename);
```

This statement reformats the number in *`$oldvariablename`* and stores it in the new format in *`$newvariablename`*. For example, the following statements display money in the correct format:

```
$price = 25;
$f_price = sprintf("%01.2f",$price);
echo "$f_price<br />";
```

You see the following on the Web page:

```
25.00
```

`sprintf` can do more than format decimal places. For more information on using `sprintf` to format values, see Chapter 14.

If you want commas to separate thousands in your number, you can use `number_format`. The following statement creates a dollar format with commas:

```
$price = 25000;
$f_price = number_format($price,2);
echo "$f_price";
```

You see the following on the Web page:

```
25,000.00
```

The `2` in the `number_format` statement sets the format to two decimal places. You can use any number to get any number of decimal places.

Working with Character Strings

A *character string* is a series of characters. Characters are letters, numbers, and punctuation. When a number is used as a character, it is just a stored character, the same as a letter. It can't be used in arithmetic. For instance, a phone number is stored as a character string because it needs to be only stored — not added or multiplied.

When you store a character string in a variable, you tell PHP where the string begins and ends by using double quotes or single quotes. For instance, the following two statements are the same:

```
$string = "Hello World!";
$string = 'Hello World!';
```

Suppose that you wanted to store a string as follows:

```
$string = 'It is Tom's house';
echo $string;
```

These statements won't work because when PHP sees the `'` (single quote) after `Tom`, it thinks that this is the end of the string, displaying the following:

```
It is Tom
```

You need to tell PHP to interpret the single quote (`'`) as an apostrophe instead of as the end of the string. You can do this by using a backslash (`\`) in front of the single quote. The backslash tells PHP that the single quote does not have any special meaning; it's just an apostrophe. This is *escaping* the character. Use the following statements to display the entire string:

```
$string = 'It is Tom\'s house';
echo $string;
```



Similarly, when you enclose a string in double quotes, you must also use a backslash in front of any double quotes in the string.

Single-quoted strings versus double-quoted strings

Single-quoted and double-quoted strings are handled differently. Single-quoted strings are stored literally, with the exception of `\'`, which is stored as an apostrophe. In double-quoted strings, variables and some special characters are evaluated before the string is stored. Here are the most important differences in the use of double or single quotes in code:

- **Handling variables:** If you enclose a variable in double quotes, PHP uses the value of the variable. However, if you enclose a variable in single quotes, PHP uses the literal variable name. For example, if you use the following statements:

```
$age = 12;
$result1 = "$age";
$result2 = '$age';
echo $result1;
echo "<br />";
echo $result2;
```

  the output is

```
12
$age
```

- **Starting a new line:** The special characters `\n` tell PHP to start a new line. When you use double quotes, PHP starts a new line at `\n`, but with single quotes, `\n` is a literal string. For instance, when using the following statements:

```
$string1 = "String in \ndouble quotes";
$string2 = 'String in \nsingle quotes';
```

  `string1` outputs as

```
String in
double quotes
```

  and `string2` outputs as

```
String in \nsingle quotes
```

- **Inserting a tab:** The special characters `\t` tell PHP to insert a tab. When you use double quotes, PHP inserts a tab at `\t`, but with single quotes, `\t` is a literal string. For instance, when using the following statements:

```
$string1 = "String in \tdouble quotes";
$string2 = 'String in \tsingle quotes';
```

  `string1` outputs as

```
String in     double quotes
```

  and `string2` outputs as

```
String in \tsingle quotes
```

The quotes that enclose the entire string determine the treatment of variables and special characters, even if other sets of quotes are inside the string. For example, look at the following statements:

```
$number = 10;
$string1 = "There are '$number' people in line.";
$string2 = 'There are "$number" people waiting.';
echo $string1,"<br>\n";
echo $string2;
```

The output is as follows:

```
There are '10' people in line.
There are "$number" people waiting.
```

Joining strings

You can join strings, a process called *concatenation,* by using a dot (`.`). For instance, you can join strings with the following statements:

```
$string1 = 'Hello';
$string2 = 'World!';
$stringall = $string1.$string2;
echo $stringall;
```

The `echo` statement outputs

```
HelloWorld!
```

Notice that no space appears between `Hello` and `World`. That's because no spaces are included in the two strings that are joined. You can add a space between the words by using the following concatenation statement rather than the earlier statement:

```
$stringall = $string1." ".$string2;
```

You can use `.=` to add characters to an existing string. For example, you can use the following statements in place of the preceding statements:

```
$stringall = "Hello";
$stringall .= " World!";
echo $stringall;
```

The `echo` statement outputs this:

```
Hello World!
```

You can also take strings apart. You can separate them at a given character or look for a substring in a string. You use functions to perform these and other operations on a string. I explain functions in Chapter 7.

Working with Dates and Times

Dates and times can be important elements in a Web database application. PHP has the ability to recognize dates and times and handle them differently than plain character strings. Dates and times are stored by the computer in a format called a *timestamp.* However, this is not a format in which you or I would want to see the date. PHP converts dates from your notation into a timestamp that the computer understands and from a timestamp into a format familiar to people. PHP handles dates and times by using built-in functions.



The timestamp format is a Unix Timestamp, which is an integer that is the number of seconds from January 1, 1970, 00:00:00 GMT (Greenwich Mean Time) to the time represented by the timestamp. This format makes it easy to calculate the time between two dates — just subtract one timestamp from the other.

Setting local time

With the release of PHP 5.1, PHP added a setting for a default local time zone to `php.ini`. If you do not set a default time zone, PHP will guess, which sometimes results in GMT. In addition, PHP displays a message advising you to set your local time zone.

To set a default time zone:

1. **Open php.ini in a text editor.**
2. **Scroll down to the section headed `[Date]`.**
3. **Find the setting: `date.timezone =`.**
4. **If the line begins with a semicolon (`;`), remove the semicolon.**
5. **Add a time zone code after the equal sign.**

You can see a list of time zone codes in Appendix H of the PHP online manual at `www.php.net/manual/en/timezones.php`. For example, you can set your default time zone to Pacific time with the setting:

```
date.timezone = America/Los_Angeles
```

If you do not have access to the `php.ini` file, you can set a default time zone in each program that applies to that program only, as follows:

```
date_default_timezone_set("timezonecode");
```

You can see which time zone is currently your default time zone as follows:

```
$def = date_default_timezone_get()
echo $def;
```

Formatting a date

The function that you will use most often is `date`, which converts a date or time from the timestamp format into a format that you specify. The general format is

```
$mydate = date("format",$timestamp);
```

`$timestamp` is a variable with a timestamp stored in it. You previously stored the timestamp in the variable, using a PHP function as I describe later in this section. If `$timestamp` is not included, the current time is obtained from the operating system and used. Thus, you can get today's date with the following:

```
$today = date("Y/m/d");
```

If today is August 10, 2006, this statements returns

```
2006/08/10
```

The *format* is a string that specifies the date format that you want stored in the variable. For instance, the format `"y-m-d"` returns 06-08-10, and `"M.d.Y"` returns Aug.10.2006. Table 6-2 lists some of the symbols that you can use in the format string. (For a complete list of symbols, see the documentation at `www.php.net/manual/en/function.date.php`.) The parts of the date can be separated by a hyphen (`-`), a dot (`.`), a forward slash (`/`), or a space.

Table 6-2 Date Format Symbols

Symbol	*Meaning*	*Example*
`F`	Month in text, not abbreviated	January
`M`	Month in text, abbreviated	Jan
`m`	Month in numbers with leading zeros	02, 12
`n`	Month in numbers without leading zeros	1, 12
`d`	Day of the month; two digits with leading zeros	01, 14
`j`	Day of the month without leading zeros	3, 30
`l`	Day of the week in text, not abbreviated	Friday
`D`	Day of the week in text, abbreviated	Fri
`w`	Day of the week in numbers	From 0 (Sunday) to 6 (Saturday)
`Y`	Year in four digits	2002

(continued)

Table 6-2 *(continued)*

Symbol	*Meaning*	*Example*
`y`	Year in two digits	02
`g`	Hour between 0 and 12 without leading zeros	2, 10
`G`	Hour between 0 and 24 without leading zeros	2, 15
`h`	Hour between 0 and 12 with leading zeros	01, 10
`H`	Hour between 0 and 24 with leading zeros	00, 23
`i`	Minutes	00, 59
`s`	Seconds	00, 59
`a`	am or pm in lowercase	am, pm
`A`	AM or PM in uppercase	AM, PM

Storing a timestamp in a variable

You can assign a timestamp with the current date and time to a variable with the following statements:

```
$today = time();
```

Another way to store a current timestamp is with the statement

```
$today = strtotime("today");
```

You can store specific timestamps by using `strtotime` with various keywords and abbreviations that are similar to English. For instance, you can create a timestamp for January 15, 2006, as follows:

```
$importantDate = strtotime("January 15 2006");
```

`strtotime` recognizes the following words and abbreviations:

- **Month names:** Twelve month names and abbreviations
- **Days of the week:** Seven days and some abbreviations
- **Time units:** `year`, `month`, `fortnight`, `week`, `day`, `hour`, `minute`, `second`, `am`, `pm`
- **Some useful English words:** `ago`, `now`, `last`, `next`, `this`, `tomorrow`, `yesterday`
- **Plus and minus:** + or –

- **All numbers**
- **Time zones:** For example, `gmt` (Greenwich Mean Time), `pdt` (Pacific Daylight Time), and `akst` (Alaska Standard Time)

You can combine the words and abbreviations in a wide variety of ways. The following statements are all valid:

```
$importantDate = strtotime("tomorrow"); #24 hours from now
$importantDate = strtotime("now + 24 hours");
$importantDate = strtotime("last saturday");
$importantDate = strtotime("8pm + 3 days");
$importantDate = strtotime("2 weeks ago"); # current time
$importantDate = strtotime("next year gmt");
$importantDate = strtotime("this 4am");    # 4 AM today
```

If you wanted to know how long ago `$importantDate` was, you could subtract it from `$today`. For instance:

```
$timeSpan = $today - $importantDate;
```

This gives you the number of seconds between the important date and today. Or use the statement

```
$timeSpan =(($today - $importantDate)/60)/60
```

to find out the number of hours since the important date.

Using dates with MySQL

Often you want to store a date in your MySQL database. For instance, you might want to store the date when a customer made an order or the time when a member logged in. MySQL also recognizes dates and times and handles them differently than plain character strings. However, MySQL also handles them differently than PHP. To use dates and times in your application, you need to understand both how PHP handles dates (which I describe in the previous few sections) and how MySQL handles dates.

I discuss the `DATE` and `DATETIME` data types for MySQL in detail in Chapter 3. The following is a summary:

- `DATE`: MySQL `DATE` columns expect dates with the year first, the month second, and the day last. The year can be `yyyy` or `yy`. The month can be `mm` or `m`. The day can be `dd` or `d`. The parts of the date can be separated by a hyphen (`-`), a forward slash (`/`), a dot (`.`), or a space.
- `DATETIME`: MySQL `DATETIME` columns expect both the date and the time. The date is formatted as I describe in the preceding bullet. The date is followed by the time in the format `hh:mm:ss`.

Dates and times must be formatted in the correct MySQL format to store them in your database. PHP functions can be used for formatting. For instance, you can format today's date into a MySQL format with this statement:

```
$today = date("Y-m-d");
```

You can format a specific date by using the statement

```
$importantDate = date("Y.m.d",strtotime("Jan 15 2006"));
```

You can then store the formatted date in a database with an SQL query like this:

```
UPDATE Member SET createDate="$today"
```

In some cases, MySQL date functions are easier to use than PHP statements to manipulate dates. For example, MySQL provides a function named `DATEDIFF` that computes the number of days between two dates, as follows:

```
DATEDIFF(date1,date2)
```

The function returns the number of days from *date2* to *date1*. For example, to determine the number of days between a date in a table and the current date, you can use the following:

```
SELECT DATEDIFF(NOW(),Birth_date) FROM Customer
```

`NOW()` is a MySQL function that returns the current date and time, and `Birth_date` is the name of a column in the `Customer` table.

You can also use the function to return the number of days between dates that you provide, as follows:

```
SELECT DATEDIFF('2004-1-15','1997-12-30')
```

MySQL provides many useful functions. All the date/time functions are described at `dev.mysql.com/doc/refman/5.0/en/date-and-time-functions.html`.

Comparing Values

In programs, you often use *conditional statements.* That is, if something is true, your program does one thing, but if something is not true, your program does something different. Here are two examples of conditional statements:

```
if user is a child
   show toy catalog
if user is not a child
   show electronics catalog
```

To know which conditions exist, the program must ask questions. Your program then performs tasks based on the answers. Some questions (conditions) that you might want to ask — and the actions that you might want taken — are

- Is the customer a child? If so, display a toy catalog.
- Which product has more sales? Display the most popular one first.
- Did the customer enter the correct password? If so, display the Members Only Web page.
- Does the customer live in Ohio? If so, display the map to the Ohio store location.

To ask a question in a program, you form a statement that compares values. The program tests the statement and determines whether the statement is true or false. For instance, you can state the preceding questions as

- The customer is less than 13 years of age. True or false? If true, display the toy catalog.
- Product 1 sales are higher than Product 2 sales. True or false? If true, display Product 1 first; if false, display Product 2 first.
- The customer's password is `secret`. True or false? If true, show the Members Only Web page.
- The customer lives in Ohio. True or false? If true, display a map to the Ohio store location.

Comparisons can be quite simple. For instance, is the first value larger than the second value? Or smaller? Or equal to? But sometimes you need to look at character strings to see whether they have certain characteristics instead of looking at their exact values. For instance, you might want to identify strings that begin with *S* or strings that look like phone numbers. For this type of comparison, you compare a string to a pattern, which I describe in the section "Matching character strings to patterns," later in this chapter.

Making simple comparisons

Simple comparisons compare one value to another value. PHP offers several ways to compare values. Table 6-3 shows the comparisons that are available.

Table 6-3	Comparing Values
Comparison	***Description***
==	Are the two values equal?
>	Is the first value larger than the second value?
>=	Is the first value larger than or equal to the second value?
<	Is the first value smaller than the second value?
<=	Is the first value smaller than or equal to the second value?
!=	Are the two values not equal to each other?
<>	Are the two values not equal to each other?

You can compare both numbers and strings. Strings are compared alphabetically, with all uppercase characters coming before any lowercase characters. For instance, *SS* comes before *Sa.* Characters that are punctuation also have an order, and one character can be found to be larger than another character. However, comparing a comma to a period doesn't have much practical value.



Strings are compared based on their ASCII (American Standard Code for Information Interchange) code. In the ASCII character set, each character is assigned an ASCII code that corresponds to a decimal number between 0 and 127. For instance, the number that represents the comma is 44. The period corresponds to 46. Therefore, if a period and a comma are compared, the period is seen as larger.

Comparisons are often used to execute statements only under certain conditions. For instance, in the following example, the block of statements is executed only when the comparison `$weather == "raining"` is true:

```
if ( $weather == "raining" )
{
    put up umbrella;
    cancel picnic;
}
```

PHP checks the variable `$weather` to see whether it is equal to `"raining"`. If it is, PHP executes the two statements. If `$weather` is not equal to `"raining"`, PHP does not execute the two statements.



The comparison sign is two equal signs (==). One of the most common mistakes is to use a single equal sign for a comparison. A single equal sign puts the value into the variable. Thus, a statement like `if ($weather = "raining")` would set `$weather` to `raining` rather than check whether it already equaled `raining` and would thus always be true.

For example, here's a solution to the programming problem presented at the beginning of this section. The problem is

```
if user is a child
   show toy catalog
if user is not a child
   show electronics catalog
```

To determine whether a customer is an adult, you compare the customer's age with the age when the customer is considered to be an adult. You need to decide at what age a customer would stop being interested in toy catalogs and start being more interested in electronic catalogs. Suppose you decide that 13 seems like the right age. You then ask whether the customer is younger than 13 by comparing the customer's age to 13. If the age is less than 13, show the toy catalog; if the age is 13 or over, show the electronics catalog. These comparisons would have the following format:

```
$age < 13      (is the customer's age less than 13?)
$age >= 13     (is the customer's age greater than or equal to 13?)
```

One way to program the conditional actions is to use the following statements:

```
if ($age < 13)
     $status = "child";
if ($age >= 13)
     $status = "adult";
```

These statements instruct PHP to compare the customer's age to 13. In the first statement, if the customer's age is less than 13, the customer's status is set to `"child"`. In the second statement, if the customer's age is greater than or equal to 13, the customer's status is set to `"adult"`. You then show the toy catalog to customers whose status is `child` and show the electronic catalog to those whose status is `adult`. Although you can write these `if` statements in a more efficient way, the statements shown will work. A full description of conditional statements is provided in Chapter 7.

Matching character strings to patterns

Sometimes you need to compare character strings to see whether they fit certain characteristics rather than match exact values. For instance, you might want to identify strings that begin with *S* or strings that have numbers in them. For this type of comparison, you compare the string to a pattern. These patterns are *regular expressions,* often called *regex.*

You've probably used some form of pattern matching in the past. When you use an asterisk (`*`) as a wildcard when searching for files (`dir s*.doc` or `ls s*.txt`), you are pattern matching. For instance, `c*.txt` is a pattern. Any string that begins with a `c` and ends with the string `.txt`, with any

characters in between the `c` and the `.txt`, matches the pattern. The strings `cow.txt`, `c3333.txt`, and `c3c4.txt` all match the pattern. Using regular expressions is just a more complicated variation of using wildcards.

The most common use for pattern matching on Web pages is to check the input from a form. If the information doesn't make sense, it's probably not something that you want to store in your database. For instance, if the user types a name into a form, you can check whether it seems like a real name by matching patterns. You know that a name consists mainly of letters and spaces. Other valid characters might be a hyphen (`-`) — for example, in the name *Smith-Kline* — and a single quote (`'`) — for example, O'Hara. You can check the name by setting up a pattern that's a string containing only letters, spaces, hyphens, and single quotes and then matching the name to the pattern. If the name doesn't match — that is, if it contains characters not in the pattern, such as numerals or a question mark (`?`) — it's not a real name.

Patterns consist of literal characters and special characters. *Literal characters* are normal characters, with no other special meaning. A *c* is a *c* with no meaning other than it's one of the 26 letters in the English alphabet. Special characters have special meaning in the pattern, such as the asterisk (`*`) when used as a wildcard. Table 6-4 shows the special characters used in patterns.

Table 6-4 Special Characters Used in Patterns

Character	*Meaning*	*Example*	*Match*	*Not a Match*
`^`	Beginning of line	`^c`	cat	my cat
`$`	End of line	`c$`	tic	stick
`.`	Any single character	`..`	Any string that contains at least two characters	a, I
`?`	Preceding character is optional	`mea?n`	mean, men	moan
`( )`	Groups literal characters into a string that must be matched exactly	`m(ea)n`	mean	men, mn
`[ ]`	Encloses a set of optional literal characters	`m[ea]n`	men, man	mean, mn
`-`	Represents all the characters between two characters	`m[a-c]n`	man, mbn, mcn	mdn, mun, maan

Character	*Meaning*	*Example*	*Match*	*Not a Match*
+	One or more of the preceding items	`door[1-3]+`	door111, door131	door, door55
*	Zero or more of the preceding items	`door[1-3]*`	door, door311	door4, door445
{ , }	The starting and ending numbers of a range of repetitions	`a{2,5}`	aa, aaaaa	a, xx3
\	The following character is literal	`m\*n`	m*n	men, mean
(| |)	A set of alternate strings	`(Tom|Tommy)`	Tom, Tommy	Thomas, To

Literal and special characters are combined to make patterns — sometimes long, complicated patterns. A string is compared to the pattern, and if it matches, the comparison is true. Some example patterns follow, with a breakdown of the pattern and some sample matching and nonmatching strings:

- `^[A-Z].*` **— Strings that begin with an uppercase letter**
  - `^[A-Z]` — Uppercase letter at the beginning of the string
  - `.*` — A string of characters that is one or more characters long

  Strings that match:
  - Play it again, Sam
  - I

  Strings that do not match:
  - play it again, Sam
  - i
- `Dear (son|daughter)` **— Two alternate strings**
  - `Dear` — Literal characters
  - `(son|daughter)` — Either son or daughter

  Strings that match:
  - Dear son
  - My Dear daughter

  Strings that do not match:
  - Dear Goliath
  - son

- `^[0-9]{5}(\-[0-9]{4})?$` **— Any zip code**
  - `^[0-9]{5}` — Any string of five numbers
  - `\-` — A literal
  - `[0-9]{4}` — A string of numbers that is four characters long
  - `( )?` — Groups the last two parts of the pattern and makes them optional

  Strings that match:
  - 90001
  - 90002–4323

  Strings that do not match:
  - 9001
  - 12–4321
- `^.+@.+\.com$` **— Any string with @ embedded that ends in `.com`**
  - `^.+` — Any string of one or more characters at the beginning
  - `@` — A literal @ (at sign); @ is not a special character
  - `.+` — Any string of one or more characters
  - `\.` — A literal dot
  - `com$` — A literal string `com` at the end of the string

  Strings that match:
  - mary@hercompany.com

  Strings that do not match:
  - mary@hercompany.net
  - @mary.com

You can compare a string to a pattern by using `ereg`. The general format is

```
ereg("pattern",string);
```

Either *`pattern`* or *`string`* can be a literal, as follows:

```
ereg("[0-9]*","1234");
```

or can be stored in variables, as follows:

```
ereg($pattern,$string);
```

To use `ereg` to check the name that a user typed in a form, compare the name to a pattern as follows:

```
ereg("^[A-Za-z' -]+$",$name)
```

The pattern in this statement does the following:

- Uses `^` and `$` to signify the beginning and end of the string. This means all the characters in the string must match the pattern.
- Encloses all the literal characters allowed in the string in `[ ]`. No other characters are allowed. The allowed characters are uppercase and lowercase letters, an apostrophe (`'`), a blank space, and a hyphen (-).

  You can specify a range of characters using a hyphen within the `[ ]`. When you do that, as in `A-Z` in the example, the hyphen does not represent a literal character. Because you want the hyphen included as a literal character that is allowed in your string, you need to add a hyphen that is not between any two other characters. In this case, the hyphen is included at the end of the list of literal characters.
- Follows the list of literal characters in the `[ ]` with a `+`. The plus sign means that the string can contain any number of the characters inside the `[ ]` but must contain at least one character.

Joining Comparisons with and/or/xor

Sometimes one comparison is sufficient to check for a condition, but often you need to ask more than one question. For instance, suppose that your company offers catalogs for different products in different languages. You need to know which product the customer wants to see *and* which language he or she needs to see it in. This is the general format for a series of comparisons:

```
comparison and|or|xor comparison and|or|xor comparison and|or|xor ...
```

Comparisons are connected by one of the three following words:

- `and`: Both comparisons are true.
- `or`: One comparison or both comparisons are true.
- `xor`: One of the comparisons is true, but both comparisons are not true.

Table 6-5 shows some examples of multiple comparisons.

Table 6-5 Multiple Comparisons	
Condition	***Is True If***
`$customer == "Smith" or $customer == "Jones"`	The customer is named Smith or Jones.
`$customer == "Smith" and $custState =="OR"`	The customer is named Smith, and the customer lives in Oregon.
`$customer == "Smith" or $custState == "OR"`	The customer is named Smith, or the customer lives in Oregon, or both.
`$customer == "Smith" xor $custState == "OR"`	The customer is named Smith, or the customer lives in Oregon — but not both.
`$customer != "Smith" and $custAge < 13`	The customer is named anything except Smith and is under 13 years of age.

You can string together as many comparisons as necessary. The comparisons that use `and` are tested first, the comparisons that use `xor` are tested next, and the comparisons that use `or` are tested last. For instance, the following is a condition that includes three comparisons:

```
$age == 200 or $age == 300 and $name == "Goliath"
```

If the customer's name is Goliath and he is 300 years old, this statement is true. The statement is also true if the customer is 200 years old, regardless of what his name is. This condition is not true if the customer is 300 years old but his name is not Goliath. You get these results because the program checks the condition as follows:

1. **The `and` is compared.** The program checks `$age` to see whether it equals `300`, and it checks `$name` to see whether it equals `Goliath`. If both match, the condition is true, and the program does not need to check `or`. If only one or neither of the variables equal the designated value, the testing continues.
2. **The `or` is compared.** The program checks `$age` to see whether it equals `200`. If it does, the condition is true. If it does not, the condition is false.

You can change the order in which comparisons are made by using parentheses. The word inside the parentheses is evaluated first. For instance, you can rewrite the previous statement with parentheses as follows:

```
( $age == 200 or $age == 300 ) and $name == "Goliath"
```

The parentheses change the order in which the conditions are checked. Now the `or` is checked first. This condition is true if the customer's name is Goliath and he is either 200 or 300 years old. You get these results because the program checks the condition as follows:

1. **The `or` is compared.** The program checks `$age` to see whether it equals either 200 or 300. If it does, this part of the condition is true. However, the comparison on the other side of the `and` must also be true, so the testing continues.
2. **The `and` is compared.** The program checks `$name` to see whether it equals Goliath. If it does, the condition is true. If it does not, the condition is false.



Use parentheses liberally, even when you believe that you know the order of the comparisons. Unnecessary parentheses can't hurt, but comparisons that have unexpected results can.



If you are familiar with other languages, such as C, you may have used `||` (for `or`) and `&&` (for `and`) in place of the words. The `||` and `&&` work in PHP as well. The statement `$a < $b && $c > $b` is just as valid as the statement `$a < $b and $c > $b`. The `||` is checked before `or`; the `&&` is checked before `and`.

Adding Comments to Your Program

Comments are notes embedded in the program itself. Adding comments in your programs that describe their purpose and what they do is essential. It's important for the lottery factor — that is, if you win the lottery and run off to a life of luxury on the French Riviera, someone else will have to finish the application. The new person needs to know what your program is supposed to do and how it does it. Actually, comments benefit you as well. You might need to revise the program next year when the details are long buried in your mind under more recent projects.

Use comments liberally. PHP ignores comments; comments are for humans. You can embed comments in your program anywhere as long as you tell PHP that they are comments. The format for comments is

```
/*  comment text
more comment text  */
```

Your comments can be as long or as short as you need. When PHP sees code that indicates the start of a comment (`/*`), it ignores everything until it sees the code that indicates the end of a comment (`*/`).

One possible format for comments at the start of each program is as follows:

```
/*  name:        catalog.php
    description: Program that displays descriptions of
                 products. The descriptions are stored
                 in a database. The product descriptions
                 are selected from the database based on
                 the category the user entered into a
           form.
    written by:  Lola Designer
    created:     2/1/06
    modified:    3/15/06
*/
```

You should use comments throughout the program to describe what the program does. Comments are particularly important when the program statements are complicated. Use comments such as the following frequently:

```
/* Get the information from the database */
/* Check whether the customer is over 18 years old */
/* Add shipping charges to the order total */
```

PHP also has a short comment format. You can specify that a single line is a comment by using the pound sign (#) or two forward slashes (//) in the following manner:

```
# This is comment line 1
// This is comment line 2
```

All text from the # or // to the end of the line is a comment. You can also use # or // in the middle of a line to signal the beginning of a comment. PHP will ignore everything from the # or // to the end of the line. This is useful for commenting a particular statement, as in the following example:

```
$average = $orderTotal/$nItems  // compute average price
```

Sometimes you want to emphasize a comment. The following format makes a comment very noticeable:

```
#####################################
##  Double-Check This Section      ##
#####################################
```

PHP comments are not included in the HTML code that is sent to the user's browser. The user does not see these comments.

Use comments as often as necessary in the script to make it clear. However, using too many comments is a mistake. Don't comment every line or everything you do in the script. If your script is too full of comments, the important comments can get lost in the maze. Use comments to label sections and to explain unusual or complicated code — not obvious code.

Chapter 7

PHP Building Blocks for Programs

In This Chapter

- Echoing output to Web pages
- Assigning values to variables
- Stopping and breaking out of programs
- Creating and using arrays
- Using conditional statements
- Building and using loops for repeated statements
- Using functions

PHP programs are a series of instructions in a file named with an extension that tells the Web server to look for PHP sections in the file. (The extension is usually `.php` or `.phtml`, but it can be anything that the Web server is configured to expect.) PHP begins at the top of the file and executes each instruction, in order, as it comes to it. Instructions are the building blocks of PHP programs.

The basic building blocks are simple *statements* — a single instruction followed by a semicolon. A simple program consists of a series of simple statements. For example, the Hello World program in Chapter 6 is a simple program. However, the programs that make up a Web database application are not that simple. They are dynamic and interact with both the user and the database. Consequently, the programs require more complex building blocks.

Here are some common programming tasks that require complex building blocks:

- **Storing groups of related values together:** You often have related information, such as the description, picture, and price of a product or a list of customers. Storing this information as a group that you can access under one name is efficient and useful. This PHP feature is an *array.*
- **Setting up statements that execute only when certain conditions are met:** Programs frequently need to do this. For instance, you may want to display a toy catalog to a child and an electronics catalog to an adult. This type of statement is a *conditional statement.* The PHP conditional statements are the `if` statement and the `case` statement.

- **Setting up a block of statements that is repeated:** You frequently need to repeat statements. For instance, you may want to create a list of all your customers. To do that, you might use two statements: one that gets the customer row from the database and a second one that stores the customer name in a list. You would need to repeat these two statements for every row in the customer database. The feature that enables you to do this is a *loop*. Three types of loops are `for` loops, `while` loops, and `do..while` loops.
- **Writing blocks of statements that can be reused many times:** Many tasks are performed in more than one part of the application. For instance, you might want to retrieve product information from the database and display it numerous times in an application. Getting and displaying the information might require several statements. Writing a block of statements that displays the product information and using this block repeatedly is much more efficient than writing the statements over again every time you need to display the product information. PHP allows you to reuse statement blocks by creating a *function*.

In this chapter, you find out how to use the building blocks of PHP programs. I describe the most frequently used simple statements and the most useful complex statements and variables. You find out how to construct the building blocks and what they are used for. Then in Chapter 8, you find out how to use these building blocks to move data in and out of a database.

Useful Simple Statements

A *simple statement* is a single instruction followed by a semicolon (`;`). Here are some useful simple statements used in PHP programs:

- `echo` statement: Produces output that browsers handle as HTML
- Assignment statement: Assigns values to variables
- Increment statement: Increases or decreases numbers in variables
- `exit` statement: Stops the execution of your program
- Function call: Uses stored blocks of statements at any location in a program

I discuss these simple statements and when to use them in this section.

Using echo statements

You use `echo` statements to produce output. The output from an `echo` statement is sent to the user's browser, which handles the output as HTML.

The general format of an echo statement is

```
echo outputitem,outputitem,outputitem,...
```

where the following rules apply:

- An *outputitem* can be a number, a string, or a variable. A string must be enclosed in quotes. The difference between double and single quotes is explained in Chapter 6.
- List as many *outputitem*s as you need, separated by commas.

Table 7-1 shows some `echo` statements and their output. For the purposes of the table, assume that `$string1` is set to `Hello` and `$string2` is set to `World!`.

Table 7-1 echo Statements

echo Statement	*Output*
`echo "Hello";`	`Hello`
`echo 123;`	`123`
`echo "Hello","World!";`	`HelloWorld!`
`echo Hello World!;`	Not valid; results in an error message
`echo "Hello World!";`	`Hello World!`
`echo 'Hello World!';`	`Hello World!`
`echo $string1;`	`Hello`
`echo $string1,$string2;`	`HelloWorld!`
`echo "$string1 $string2";`	`Hello World!`
`echo "Hello ",$string2;`	`Hello World!`
`echo "Hello"," ",$string2;`	`Hello World!`
`echo '$string1',"$string2";`	`$string1World!`



Double quotes and single quotes have different effects on variables. When you use single quotes, variable names are echoed as-is. When you use double quotes, variable names are replaced by the variable values.

You can separate variable names with curly braces (`{ }`). For instance, the following statements

```
$pet = "bird";
echo "The $petcage has arrived.";
```

will not output `bird` as the `$pet` variable. In other words, the output will not be `The birdcage has arrived.` Rather, PHP will look for the variable `$petcage` and won't be able to find it. You can echo the correct output by using curly braces to separate the `$pet` variable:

```
$pet = "bird";
echo "The {$pet}cage has arrived.";
```

The preceding statement will output

```
The birdcage has arrived.
```

`echo` statements output a line of text that is sent to a browser. The browser considers the text to be HTML and handles it that way. Therefore, you need to make sure that your output is valid HTML code that describes the Web page that you want the user to see.

When you want to display a Web page (or part of a Web page) by using PHP, you need to consider three stages in producing the Web page:

- **The PHP program:** PHP `echo` statements that you write.
- **The HTML source code:** The source code for the Web page that you see when you choose View➪Source in your browser. The *source code* is the output from the `echo` statements.
- **The Web page:** The Web page that your users see. The Web page results from the HTML source code.



The `echo` statements send exactly what you echo to the browser — no more, no less. If you do not echo any HTML tags, none are sent.

PHP allows some special characters that format output, but they are not HTML tags. The PHP special characters affect only the output from the `echo` statement — not the display on the Web page. For instance, if you want to start a new line in the PHP output, you must include a special character (`\n`) that tells PHP to start a new line. However, this special character just starts a new line in the output; it does *not* send an HTML tag to start a new line on the Web page. Table 7-2 shows examples of the three stages.

Table 7-2 Stages of Web Page Delivery

echo Statement	*HTML Source Code*	*Web Page Display*
`echo "Hello World!";`	`Hello World!`	`Hello World!`
`echo "Hello World!";` `echo "Here I am!";`	`Hello World` `Here I am!`	`Hello World!Here I am!`
`echo "Hello World!\n";` `echo "Here I am!";`	`Hello World!` `Here I am`	`Hello World!Here I am!`
`echo "Hello World!";` `echo " ";` `echo "Here I am!";`	`Hello World! ` `Here I am!"`	`Hello World!` `Here I am!`
`echo "Hello";` `echo " World! \n";` `echo "Here I am!";`	`Hello World! ` `Here I am!"`	`Hello World!` `Here I am!`

Table 7-2 summarizes the differences between the stages in creating a Web page with PHP. To look at these differences more closely, consider the following two `echo` statements:

```
echo "Line 1";
echo "Line 2";
```

If you put these lines in a program, you might *expect* the Web page to display

```
Line 1
Line 2
```

However, this is *not* the output that you would get. The Web page would display this:

```
Line 1Line 2
```

If you look at the source code for the Web page, you see exactly what is sent to the browser, which is this:

```
Line 1Line 2
```

Notice that the line that is output and sent to the browser contains exactly the characters that you echoed — no more, no less. The character strings that you echoed did not contain any spaces, so no spaces appear between the lines. Also notice that the two lines are echoed on the same line. If you want a new line to start, you have to send a signal indicating the start of a

new line. To signal that a new line starts here in PHP, echo the special character \n. Change the echo statements to the following:

```
echo "line 1\n";
echo "line 2";
```

Now you get what you want, right? Well, no. Now you see the following on the Web page:

```
line 1 line 2
```

If you look at the source code, you see this:

```
line 1
line 2
```

So, the \n did its job: It started a new line in the output. However, HTML displays the output on the Web page as one line. If you want HTML to display two lines, you must use a tag, such as the <br> tag. So, change the PHP end-of-line special character to an HTML tag, as follows:

```
echo "line 1<br>";
echo "line 2";
```

Now you see what you want on the Web page:

```
line 1
line 2
```

If you look at the source code for this output, you see this:

```
line 1<br>line 2
```



Use \n liberally. Otherwise, your HTML source code will have some really long lines. For instance, if you echo a long form, the whole thing might be one long line in the source code, even though it looks fine in the Web page. Use \n to break the HTML source code into reasonable lines. It's much easier to examine and troubleshoot the source code if it's not a mile-long line.

Using assignment statements

Assignment statements are statements that assign values to variables. The variable name is listed to the left of the equal sign; the value to be assigned to the variable is listed to the right of the equal sign. Here is the general format:

```
$variablename = value;
```

The *value* can be a single value or a combination of values, including values in variables. A variable can hold numbers or characters but not both at the same time. Therefore, a value cannot be a combination of numbers and characters. The following are valid assignment statements:

```
$number = 2;
$number = 2+1;
$number = (2 - 1) * (4 * 5) -17;
$number2 = $number + 3;
$string = "Hello World";
$string2 = $string." again!";
```

If you combine numbers and strings in a value, you won't get an error message; you'll just get unexpected results. For instance, the following statements combine numbers and strings:

```
$number = 2;
$string = "Hello";
$combined = $number + $string;
$combined2 = $number.$string;
echo $combined;
echo <br>;
echo $combined2;
```

The output of these statements is

```
2          ($string is evaluated as 0)
2Hello     ($number is evaluated as a character)
```

Using increment statements

Often a variable is used as a `counter`. For instance, suppose you want to be sure that everyone sees your company logo, so you display it three times. You set a variable to `0`. Each time that you display the logo, you add 1 to the variable. When the value of the variable reaches `3`, you know that it's time to stop showing the logo. The following statements show the use of a `counter`:

```
$counter=0;
$counter = $counter + 1;
echo $counter;
```

These statements would output `1`. Because counters are used so often, PHP provides shortcuts. The following statements have the same effect as the preceding statements:

```
$counter=0;
$counter++;
echo $counter;
```

This `echo` statement also outputs `1` because `++` adds 1 to the current value of `$counter`. Or you can use the following statement, which subtracts 1 from the current value of `$counter`.

```
$counter--;
```

Sometimes you may want to do a different arithmetic operation. You can use any of the following shortcuts:

```
$counter+=2;
$counter-=3;
$counter*=2;
$counter/=3;
```

These statements add 2 to `$counter`, subtract 3 from `$counter`, multiply `$counter` by 2, and divide `$counter` by 3, respectively.

Using exit

Sometimes you want the program to stop executing — just stop at some point in the middle of the program. For instance, if the program encounters an error, often you want it to stop rather than continue with more statements. The `exit` statement stops the program. No more statements are executed after the `exit` statement. The format of an `exit` statement is

```
exit("message");
```

The *`message`* is a message that is output when the program exits. For instance, you might use the statement

```
exit("The program is exiting");
```

You can also stop the program with the `die` statement, as follows:

```
die("The program is dying");
```

The `die` statement is the same as the `exit` statement. Sometimes it's just more fun to say *die*.

Using function calls

Functions are blocks of statements that perform certain specified tasks. You can think of functions as mini-programs or subprograms. The block of statements is stored under a function name, and you can execute the block of statements any place you want by *calling* the function by its name. (For details on how to use functions, check out the section, "Using Functions," later in this chapter.)

You can call a function by listing its name followed by parentheses, like this:

```
functionname();
```

For instance, you might have a function that gets all the names of customers that reside in a certain state from the database and displays the names in a list in the format *last name, first name*. You write the statements that do these tasks and store them as a function under the name `get_names`. Then when you call the function, you need to specify which state. You can use the following statement at any location in your program to get the list of customer names from the given state, which in this case is California:

```
get_names('CA');
```

The value in the parentheses is given to the function so it knows which state you're specifying. This is *passing* the value. You can pass one or more values.

PHP provides many built-in functions. For example, in Chapter 6, I discuss a built-in function called `unset`. You can uncreate a variable named `$testvar` by using this function call:

```
unset($testvar);
```

Using PHP Arrays

Arrays are complex variables. An *array* stores a group of values under a single variable name. An array is useful for storing related values. For instance, you can store information about a shirt (such as size, color, and cost) in a single array named `$shirtinfo`. Information in an array can be handled, accessed, and modified easily. For instance, PHP has several methods for sorting an array. This section gives you the lowdown on arrays.

Creating arrays

The simplest way to create an array is to assign a value to a variable with square brackets (`[ ]`) at the end of its name. For instance, assuming that you have not referenced `$pets` at any earlier point in the program, the following statement creates an array called `$pets`:

```
$pets[1] = "dragon";
```

At this point, the array named `$pets` has been created and has only one value: `dragon`. Next, you use the following statements:

```
$pets[2] = "unicorn";
$pets[3] = "tiger";
```

Now the array $pets contains three values: dragon, unicorn, and tiger.

An array can be viewed as a list of *key/value pairs.* To get a particular value, you specify the *key* in the brackets. In the preceding array, the keys are numbers — 1, 2, and 3. However, you can also use words for keys. For instance, the following statements create an array of state capitals:

```
$capitals['CA'] = "Sacramento";
$capitals['TX'] = "Austin";
$capitals['OR'] = "Salem";
```

You can use shortcuts rather than write separate assignment statements for each number. One shortcut uses the following statements:

```
$pets[] = "dragon";
$pets[] = "unicorn";
$pets[] = "tiger";
```

When you create an array using this shortcut, the values are automatically assigned keys that are serial numbers, starting with the number 0. For example, the following statement

```
echo "$pets[0]";
```

outputs dragon.



The first value in an array with a numbered index is 0 unless you deliberately set it to a different number. One common mistake when working with arrays is to think of the first number as 1 rather than 0.

An even better shortcut is to use the following statement:

```
$pets = array( "dragon","unicorn","tiger");
```

This statement creates the same array as the preceding shortcut. It assigns numbers as keys, starting with 0. You can use a similar statement to create arrays with words as keys. For example, the following statement creates the array of state capitals:

```
$capitals = array( "CA" => "Sacramento", "TX" => "Austin",
                   "OR" => "Salem" );
```

Viewing arrays

You can echo an array value like this:

```
echo $capitals['TX'];
```

If you include the array value in a longer `echo` statement enclosed by double quotes, you may need to enclose the array value name in curly braces:

```
echo "The capital of Texas is {$capitals['TX']}<br>";
```

You can see the structure and values of any array by using a `print_r` or a `var_dump` statement. To display the `$capitals` array, use one of the following statements:

```
print_r($capitals);

var_dump($capitals);
```

This `print_r` statement provides the following output:

```
Array
(
  [CA] => Sacramento
  [TX] => Austin
  [OR] => Salem
)
```

The `var_dump` statement provides the following output:

```
array(3) {
  ["CA"]=>
  string(10) "Sacramento"
  ["TX"]=>
  string(6) "Austin"
  ["OR"]=>
  string(5) "Salem"
}
```

The `print_r` output shows the key and the value for each element in the array. The `var_dump` output shows the data type, as well as the keys and values.



When you display the output from `print_r` or `var_dump` on a Web page, it displays with HTML, which means that it displays in one long line. To see the output on the Web in the useful format that I describe here, send HTML tags that tell the browser to display the text as received, without changing it, by using the following statements:

```
echo "<pre>";
print_r($capitals);
echo "</pre>";
```

Removing values from arrays

Sometimes you need to completely remove a value from an array. For example, suppose you have the following array:

```
$pets = array( "dragon", "unicorn", "tiger",
               "parrot", "scorpion" );
```

This array has five values. Now you decide that you no longer want to carry scorpions in your pet store, so you use the following statement to try to remove `scorpion` from the array:

```
$pets[4] = "";
```

Although this statement sets `$pets[4]` to an empty string, it does not remove it from the array. You still have an array with five values, with one of the five values being empty. To totally remove the item from the array, you need to unset it with the following statement:

```
unset($pets[4]);
```

Now your array has only four values in it.

Sorting arrays

One of the most useful features of arrays is that PHP can sort them for you. PHP originally stores array elements in the order in which you create them. If you display the entire array without changing the order, the elements will be displayed in the order in which they were created. Often, you want to change this order. For example, you may want to display the array in alphabetical order by value or by key.

PHP can sort arrays in a variety of ways. To sort an array that has numbers as keys, use a `sort` statement as follows:

```
sort($pets);
```

This statement sorts by the values and assigns new keys that are the appropriate numbers. The values are sorted with numbers first, uppercase letters next, and lowercase letters last. For instance, consider the `$pets` array created in the preceding section:

```
$pets[0] = "dragon";
$pets[1] = "unicorn";
$pets[2] = "tiger";
```

After the following `sort` statement

```
sort($pets);
```

the array becomes

```
$pets[0] = "dragon";
$pets[1] = "tiger";
$pets[2] = "unicorn";
```



If you use `sort()` to sort an array with words as keys, the keys will be changed to numbers, and the word keys will be thrown away.

To sort arrays that have words for keys, use the `asort` statement. This statement sorts the capitals by value but keeps the original key for each value instead of assigning a number key. For instance, consider the state capitals array created in the preceding section:

```
$capitals['CA'] = "Sacramento";
$capitals['TX'] = "Austin";
$capitals['OR'] = "Salem";
```

After the following sort statement

```
asort($capitals);
```

the array becomes

```
$capitals['TX'] = "Austin";
$capitals['CA'] = "Sacramento";
$capitals['OR'] = "Salem";
```

Notice that the keys stayed with the value when the elements were reordered. Now the elements are in alphabetical order, and the correct state key is still with the appropriate state capital. If the keys had been numbers, the numbers would now be in a different order. For example, if the original array was

```
$capitals[1] = "Sacramento";
$capitals[2] = "Austin";
$capitals[3] = "Salem";
```

after an `asort` statement, the new array would be

```
$capitals[2] = Austin
$capitals[1] = Sacramento
$capitals[3] = Salem
```

It's unlikely that you want to use `asort` on an array with numbers as a key.

Several other `sort` statements sort in other ways. Table 7-3 lists all the available `sort` statements.

Table 7-3 Ways You Can Sort Arrays

Sort Statement	*What It Does*
`sort($arrayname)`	Sorts by value; assigns new numbers as the keys
`asort($arrayname)`	Sorts by value; keeps the same key
`rsort($arrayname)`	Sorts by value in reverse order; assigns new numbers as the keys
`arsort($arrayname)`	Sorts by value in reverse order; keeps the same key
`ksort($arrayname)`	Sorts by key
`krsort($arrayname)`	Sorts by key in reverse order
`usort($arrayname, functionname)`	Sorts by a function (see "Using Functions," later in this chapter)

Getting values from arrays

You can retrieve any individual value in an array by accessing it directly. Here is an example:

```
$CAcapital = $capitals['CA'];
echo $CAcapital ;
```

The output from these statements is

```
Sacramento
```

If you use an array element that doesn't exist in a statement, a notice is displayed. (Read about notices in Chapter 6.) For example, suppose that you use the following statement:

```
$CAcapital = $capitals['CAx'];
```

If the array `$capitals` exists but no element has the key `CAx`, you see the following notice:

```
Notice: Undefined index: CAx in d:\testarray.php on line 9
```

Remember that a notice does not cause the script to stop. Statements after the notice will continue to execute. But because no value has been put into `$CAcapital`, any subsequent `echo` statements will echo a blank space. You can prevent the notice from being displayed by using the `@` symbol:

```
@$CAcapital = $capitals['CAx'];
```

You can get several values at once from an array using the `list` statement or all the values from an array by using the `extract` statement.

The `list` statement gets values from an array and puts them into variables. The following statements include a `list` statement:

```
$shirtInfo = array ("large", "blue", 12.00);
sort ($shirtInfo);
list($firstvalue,$secondvalue) = $shirtInfo;
echo $firstvalue,"<br>";
echo $secondvalue,"<br>";
```

The first line creates the `$shirtInfo` array. The second line sorts the array. The third line sets up two variables named `$firstvalue` and `$secondvalue` and copies the first two values in `$shirtInfo` into the two new variables, as if you had used the two statements

```
$firstvalue=$shirtInfo[0];
$secondvalue=$shirtInfo[1];
```

The third value in `$shirtInfo` is not copied into a variable because the `list` statement includes only two variables. The output from the `echo` statements is

```
blue
large
```

Notice that the output is in alphabetical order and not in the order in which the values were entered. It's in alphabetical order because the array was sorted after it was created.

You can retrieve all the values from an array with words as keys using `extract`. Each value is copied into a variable named for the key. For instance, the following statements get all the information from `$shirtInfo` and echo it:

```
extract($shirtInfo);
echo "size is $size; color is $color; cost is $cost";
```

The output for these statements is

```
size is large; color is blue; cost is 12.00;
```

Walking through an array

You will often want to do something to every value in an array. You might want to echo each value, store each value in the database, or add 6 to each value in the array. In technical talk, walking through each and every value in an array, in order, is *iteration.* It is also sometimes called *traversing.* Here are two ways to walk through an array:

- Manually: Move a pointer from one array value to another
- Using `foreach`: Automatically walk through the array, from beginning to end, one value at a time

Manually walking through an array

You can walk through an array manually by using a pointer. To do this, think of your array as a list. Imagine a pointer pointing to a value in the list. The pointer stays on a value until you move it. After you move it, it stays there until you move it again. You can move the pointer with the following instructions:

- `current($`*arrayname*`)`: Refers to the value currently under the pointer; does not move the pointer
- `next($`*arrayname*`)`: Moves the pointer to the value after the current value
- `previous($`*arrayname*`)`: Moves the pointer to the value before the current pointer location
- `end($`*arrayname*`)`: Moves the pointer to the last value in the array
- `reset($`*arrayname*`)`: Moves the pointer to the first value in the array

The following statements manually walk through an array containing state capitals:

```
$value = current ($capitals);
echo "$value<br>";
$value = next ($capitals);
echo "$value<br>";
$value = next ($capitals);
echo "$value<br>";
```

Unless you have moved the pointer previously, the pointer is located at the first element when you start walking through the array. If you think that the array pointer may have been moved earlier in the script or if your output from the array seems to start somewhere in the middle, use the `reset` statement before you start walking, as follows:

```
reset($capitals);
```

When using this method to walk through an array, you need an assignment statement and an `echo` statement for every value in the array — for each of the 50 states. The output is a list of all the state capitals.

This method gives you flexibility. You can move through the array in any manner — not just one value at a time. You can move backwards, go directly to the end, skip every other value by using two `next` statements in a row, or whatever method is useful. However, if you want to go through the array from beginning to end, one value at a time, PHP provides `foreach`, which does exactly what you need much more efficiently. `foreach` is described in the next section.

Using foreach to walk through an array

`foreach` walks through the array one value at a time. The current key and value of the array can be used in the block of statements each time the block executes. The general format is

```
foreach( $arrayname as $keyname => $valuename  )
{
    block of statements;
}
```

Fill in the following information:

- *arrayname*: The name of the array that you're walking through.
- *keyname*: The name of the variable where you want to store the key. *keyname* is optional. If you leave out `$keyname =>`, only the value is put into a variable that can be used in the block of statements.
- *valuename*: The name of the variable where you want to store the value.

For instance, the following `foreach` statement walks through the sample array of state capitals and echoes a list:

```
$capitals = array("CA" => "Sacramento", "TX" => "Austin",
                  "OR" => "Salem" );
ksort($capitals);
foreach( $capitals as $state => $city )
{
    echo "$city, $state<br>";
}
```

The preceding statements give the following Web page output:

```
Sacramento, CA
Salem, OR
Austin, TX
```

You can use the following line in place of the `foreach` line in the previous statements:

```
foreach( $capitals as $city )
```

When using this `foreach` statement, only the city is available for output. You would then use the following `echo` statement:

```
echo "$city<br>";
```

The output with these changes is

```
Sacramento
Salem
Austin
```

When `foreach` starts walking through an array, it moves the pointer to the beginning of the array. You don't need to reset an array before walking through it with `foreach`.

Multidimensional arrays

In the earlier sections of this chapter, I describe arrays that are a single list of key/value pairs. However, on some occasions, you might want to store values with more than one key. For instance, suppose you want to store these product prices together in one variable:

- shirt, 20.00
- pants, 22.50
- blanket, 25.00
- bedspread, 50.00
- lamp, 44.00
- rug, 75.00

You can store these products in an array as follows:

```
$productPrices['shirt'] = 20.00;
$productPrices['pants'] = 22.50;
$productPrices['blanket'] = 25.00;
$productPrices['bedspread'] = 50.00;
$productPrices['lamp'] = 44.00;
$productPrices['rug'] = 75.00;
```

Your program can easily look through this array whenever it needs to know a price. But suppose that you have 3000 products. Your program would need to look through 3000 products to find the one with *shirt* or *rug* as the key.

Notice that the list of products and prices includes a wide variety of products that can be classified into groups: clothing, linens, and furniture. If you classify the products, the program would need to look through only one classification to find the correct price. Classifying the products would be much more efficient. You can classify the products by putting the costs in a multidimensional array as follows:

```
$productPrices['clothing']['shirt'] = 20.00;
$productPrices['clothing']['pants'] = 22.50;
$productPrices['linens']['blanket'] = 25.00;
$productPrices['linens']['bedspread'] = 50.00;
$productPrices['furniture']['lamp'] = 44.00;
$productPrices['furniture']['rug'] = 75.00;
```

This kind of array is a *multidimensional* array because it's like an array of arrays. Figure 7-1 shows the structure of `$productPrices` as an array of arrays. The figure shows that `$productPrices` has three key/value pairs. The keys are clothing, linens, and furniture. The value for each key is an array with two key/value pairs. For instance, the value for the key `clothing` is an array with the two key/value pairs: `shirt/20.00` and `pants/22.50`.

$productPrices	**key**	**value**	
		key	*value*
	clothing	shirt	20.00
		pants	22.50
	linens	blanket	25.00
		bedspread	50.00
	furniture	lamp	44.00
		rug	75.00

Figure 7-1: An array of arrays.

`$productPrices` is a two-dimensional array. PHP can also understand multidimensional arrays that are four, five, six, or more levels deep. However, my head starts to hurt if I try to comprehend an array that is more than three levels deep. The possibility of confusion increases when the number of dimensions increases.

You can get values from a multidimensional array by using the same procedures that you use with a one-dimensional array. For instance, you can access a value directly with this statement:

```
$shirtPrice = $productPrices['clothing']['shirt'];
```

You can also echo the value:

```
echo $productPrices['clothing']['shirt'];
```

However, if you combine the value within double quotes, you need to use curly braces to enclose the variable name. The $ that begins the variable name must follow the { immediately, without a space, as follows:

```
echo "The price of a shirt is \${$productPrices['clothing']['shirt']}";
```

Notice the backslash (\) in front of the first dollar sign ($). The backslash tells PHP that $ is a literal dollar sign and not the beginning of a variable name. The output is

```
The price of a shirt is $20
```

You can walk through a multidimensional array by using `foreach` statements (described in the preceding section). You need a `foreach` statement for each array. One `foreach` statement is inside the other `foreach` statement. Putting statements inside other statements is called *nesting*.

Because a two-dimensional array, such as `$productPrices`, contains two arrays, it takes two `foreach` statements to walk through it. The following statements get the values from the multidimensional array and output them in an HTML table:

```
echo "<table border=1>";
foreach( $productPrices as $category )
{
   foreach( $category as $product => $price )
   {
     $f_price = sprintf("%01.2f", $price);
     echo "<tr><td>$product:</td>
               <td>\$$f_price</td></tr>";
   }
}
echo "</table>";
```

Figure 7-2 shows the Web page produced with these PHP statements.

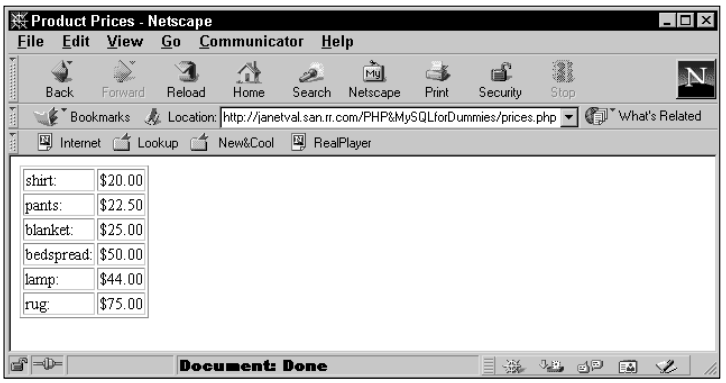


Figure 7-2: The Web page output for the multidimensional array.

Here is how the program interprets these statements:

1. Outputs the `table` tag.
2. Gets the first key/value pair in the `$productPrices` array and stores the value in the variable `$category`. The value is an array.
3. Gets the first key/value pair in the `$category` array. Stores the key in `$product` and stores the value in `$price`.
4. Formats the value in `$price` into the correct format for money.
5. Echoes one table row for the product and its price.
6. Goes to the next key/value pair in the `$category` array.
7. Formats the price and echoes the next table row for the product and its price.
8. Because there are no more key/value pairs in `$category`, the inner `foreach` statement ends.
9. Goes to the next key/value pair in the outer `foreach` statement. Puts the next value in `$category`, which is an array.
10. Repeats the procedure in Steps 2–9 until the last key/value pair in the last `$category` array is reached. The inner `foreach` statement ends. The outer `foreach` statement ends.
11. Outputs the `/table` tag to end the table.

In other words, the outer `foreach` starts with the first key/value pair in the array. The key is `clothing`, and the value of this pair is an array that is put into the variable `$category`. The inner `foreach` then walks through the array in `$category`. When it reaches the last key/value pair in `$category`, it ends. The program is then back in the outer loop, which goes on to the second key/value pair . . . and so on until the outer `foreach` reaches the end of the array.

Useful Conditional Statements

A *conditional statement* executes a block of statements only when certain conditions are met. Here are two useful types of conditional statements:

- `if` statement: Sets up a condition and tests it. If the condition is true, a block of statements is executed.
- `switch` statement: Sets up a list of alternative conditions. Tests for the true condition and executes the appropriate block of statements.

I describe these statements in more detail in the following two sections.

Using if statements

An `if` statement asks whether certain conditions exist. A block of statements executes depending on which conditions are met. The general format of an `if` conditional statement is

```
if ( condition ... )
{
   block of statements
}
elseif  ( condition ... )
{
   block of statements
}
else
{
   block of statements
}
```

The `if` statement consists of three sections:

- **`if`:** This section is required. It tests a condition.
  - **If condition is true:** The block of statements is executed. After the statements are executed, the program moves to the next instruction following the conditional statement; if the conditional statement contains any `elseif` or `else` sections, the program skips over them.
  - **If condition is not true:** The block of statements is not executed. The program skips to the next instruction, which can be an `elseif`, an `else`, or the next instruction after the `if` conditional statement.
- **`elseif`:** This section is optional. It tests a condition. You can use more than one `elseif` section if you want.
  - **If condition is true:** The block of statements is executed. After executing the block of statements, the program goes to the next instruction following the conditional statement; if the `if` statement contains any additional `elseif` sections or an `else` section, the program skips over them.
  - **If condition is not true:** The block of statements is not executed. The program skips to the next instruction, which can be an `elseif`, an `else`, or the next instruction after the `if` conditional statement.
- **`else`:** This section is optional. Only one `else` section is allowed. This section does not test a condition; rather, it executes the block of statements. If the program has entered this section, it means that the `if` section and all the `elseif` sections are not true.

Each section of the `if` conditional statement tests a condition that consists of one or more comparisons. A comparison asks a question that can be true or false. Some conditions are

```
$a == 1;
$a < $b
$c != "Hello"
```

The first comparison asks whether `$a` is equal to `1`; the second comparison asks whether `$a` is smaller than `$b`; the third comparison asks whether `$c` is not equal to `"Hello"`. You can use two or more comparisons in a condition by connecting the comparisons with `and`, `or`, or `xor`. I discuss comparing values and using more than one comparison in detail in Chapter 6.

The following example uses all three sections of the `if` conditional statement. Suppose that you have German, French, Italian, and English versions of your product catalog. You want your program to display the correct language version, based on where the customer lives. The following statements set a variable to the correct catalog version (depending on the country where the customer lives) and set a message in the correct language. You can then display a message in the appropriate language.

```
if ($country == "Germany" )
{
   $version = "German";
   $message = " Sie sehen unseren Katalog auf Deutsch";
}
elseif ($country == "France" )
{
   $version = "French";
   $message = " Vous verrez notre catalogue en francais";
}
elseif ($country == "Italy" )
{
   $version = "Italian";
   $message = " Vedrete il nostro catalogo in Italiano";
}
else
{
  $version = "English";
  $message = "You will see our catalog in English";
}
echo "$message<br>";
```

The `if` conditional statement proceeds as follows:

1. **Compares the variable `$country` to `"Germany"`.** If they are the same, `$version` is set to `"German"`, `$message` is set in German, and the program skips to `echo`. If `$country` does *not* equal `Germany`, `$version` and `$message` are *not* set, and the program skips to the `elseif` section.
2. **Compares the variable `$country` to `"France"`.** If they are the same, `$version` and `$message` are set, and the program skips to the `echo` statement. If `$country` does *not* equal `France`, `$version` and `$message` are *not* set, and the program skips to the second `elseif` section.

3. **Compares the variable `$country` to `"Italy"`.** If they are the same, `$version` is set to `"Italian"`, and the program skips to the `echo` statement. If `$country` does *not* equal `Italy`, `$version` and `$message` are *not* set, and the program skips to the `else` section.
4. **`$version` is set to English, and `$message` is set in English.** The program continues to the `echo` statement.

Notice that only the message is echoed in this example. However, the variable `$version` is stored because the version is useful information that can be used later in the program.



When the block to be executed by any section of the `if` conditional statement contains only one statement, the curly braces are not needed. For instance, if the preceding example had only one statement in the blocks

```
if ($country == "France")
{
   $version = "French";
}
```

you could write it as follows:

```
if ($country == "France" )
   $version = "French";
```

This shortcut can save some typing, but it can lead to confusion when several `if` statements are used.

You can have an `if` conditional statement inside another `if` conditional statement. Putting one statement inside another is *nesting*. For instance, suppose that you need to contact all your customers who live in Idaho. You plan to send e-mail to those who have an e-mail address and send a letter to those who do not have an e-mail address. You can identify the groups of customers by using the following nested `if` statements:

```
if ( $custState == "ID" )
{
   if ( $EmailAdd != "" )
   {
     $contactMethod = "email";
   }
   else
   {
     $contactMethod = "letter";
   }
}
else
{
   $contactMethod = "none needed";
}
```

These statements first check to see whether the customer lives in Idaho. If the customer does live in Idaho, the program tests for an e-mail address. If the e-mail address is not blank, the contact method is set to `email`. If the e-mail address is blank, the contact method is `letter`. If the customer does not live in Idaho, the `else` section sets the contact method to indicate that the customer will not be contacted at all.

Using switch statements

For most situations, the `if` conditional statement works best. Sometimes, however, you have a list of conditions and want to execute different statements for each of the conditions. For instance, suppose that your program computes sales tax. How do you handle the different state sales tax rates? The `switch` statement was designed for such situations.

The `switch` statement tests the value of one variable and executes the block of statements for the matching value of the variable. The general format is

```
switch ( $variablename )
{
  case value :
     block of statements;
     break;
  case value :
     block of statements;
     break;
  ...
  default:
     block of statements;
     break;
}
```

The `switch` statement tests the value of *`$variablename`*. The program then skips to the `case` section for that value and executes statements until it reaches a `break` statement or the end of the `switch` statement. If there is no `case` section for the value of *`$variablename`*, the program executes the `default` section. You can use as many `case` sections as you need. The `default` section is optional. If you use a `default` section, it's customary to put the `default` section at the end, but it can go anywhere.

The following statements set the sales tax rate for different states:

```
switch ( $custState )
{
  case "OR" :
     $salestaxrate = 0;
     break;
  case "CA" :
     $salestaxrate = 1.0;
```

```
      break;
   default:
      $salestaxrate = .5;
      break;
}
$salestax = $orderTotalCost * $salestaxrate;
```

In this case, the tax rate for Oregon is 0, the tax rate for California is 100 percent, and the tax rate for all the other states is 50 percent. The `switch` statement looks at the value of `$custState` and skips to the section that matches the value. For instance, if `$custState` is `TX`, the program executes the `default` section and sets `$salestaxrate` to `.5`. After the `switch` statement, the program computes `$salestax` at `.5` times the cost of the order.



The `break` statements are essential in the `case` section. If a `case` section does not include a `break` statement, the program does *not* stop executing at the end of the `case` section. The program continues executing statements past the end of the `case` section, on to the next `case` section, and continues until it reaches a `break` statement in a later `case` section or the end of the `switch` statement.



The last `case` section in a `switch` statement doesn't actually require a `break` statement. You can leave it out, but it's a good idea to include it for clarity.

Using Loops

Loops, which are used frequently in programs, set up a block of statements that repeat. Sometimes, the loop repeats a specified number of times. For instance, a loop to echo all the state capitals needs to repeat 50 times. Sometimes, the loop repeats until a certain condition exists. For instance, a loop that displays product information for all the products needs to repeat until it has displayed all the products, regardless of how many products there are. Here are three types of loops:

- Basic `for` loop: Sets up a counter; repeats a block of statements until the counter reaches a specified number
- `while` loop: Sets up a condition; checks the condition; and if it is true, repeats a block of statements
- `do..while` loop: Sets up a condition; executes a block of statements; checks the condition; if the condition is true, repeats the block of statements

I describe each of these loops in detail in the following few sections.

Using for loops

The most basic `for` loops are based on a counter. You set the beginning value for the counter, set the ending value, and set how the counter is incremented. The general format is

```
for(startingvalue;endingcondition;increment)
{
    block of statements;
}
```

Fill in the following values:

- *startingvalue*: A statement that sets up a variable to be your counter and sets it to your starting value. For instance, the statement `$i=1;` sets `$i` as the counter variable and sets it equal to `1`. Frequently, the counter variable is started at `0` or `1`. The starting value can be a combination of numbers (`2 + 2`) or a variable.
- *endingcondition*: A statement that sets your ending value. As long as this statement is true, the block of statements keeps repeating. When this statement is not true, the loop ends. For instance, the statement `$i<10;` sets the ending value for the loop to `10`. When `$i` is equal to `10`, the statement is no longer true (because `$i` is no longer less than `10`), and the loop stops repeating. The statement can include variables, such as `$i<$size;`.
- *increment*: A statement that increments your counter. For instance, the statement `$i++;` adds `1` to your counter at the end of each block of statements. You can use other increment statements, such as `$I+=1;` or `$i--;`.

The basic `for` loop sets up a variable — for example, a variable called `$i,` — that is a counter. This variable has a value during each loop. The variable `$i` can be used in the block of statements that is repeating. For instance, the following simple loop displays `Hello World!` three times:

```
for($i=1;$i<=3;$i++)
{
  echo "$i. Hello World!<br>";
}
```



PHP doesn't care whether the statements in the block are indented. However, indenting the blocks makes it much easier for you to understand the program.

The output from these statements is

```
1. Hello World!
2. Hello World!
3. Hello World!
```

`for` loops are particularly useful for looping through an array. Suppose that you have an array of customer names and want to display them all. You can do this easily with a loop:

```
for($i=0;$i<100;$i++)
{
  echo "$customerNames[$i]<br>";
}
```

The output displays a Web page with a list of all customer names, one on each line. In this case, you know that you have 100 customer names. But suppose that you don't know how many customers are in this list. You can ask PHP how many values are in the array and use that value in your `for` loop. For example, you can use the following statements:

```
for($i=0;$i<sizeof($customerNames);$i++)
{
   echo "$customerNames[$i]<br>";
}
```

Notice that the ending value is `sizeof($customerNames)`. This statement finds out the number of values in the array and uses that number. That way, your loop repeats exactly the number of times that there are values in the array.



The first value in an array with a numbered index is `0` unless you deliberately set it to a different number. One common mistake when working with arrays is to think of the first number as 1 rather than 0.

Using while loops

A `while` loop continues repeating as long as certain conditions are true. The loop works as follows:

1. You set up a condition.
2. The condition is tested at the top of each loop.
3. If the condition is true, the loop repeats. If the condition is not true, the loop stops.

The general format of a while loop is

```
while( condition )
{
   block of statements
}
```

A *condition* is any expression that can be found to be true or false. Comparisons, such as the following, are often used as conditions. (For detailed information on using comparisons, see Chapter 6.)

```
$test <= 10
$test1 == $test2
$a == "yes" and $b != "yes"
$name != "Smith"
```

As long as the condition is found to be true, the loop repeats. When the condition tests false, the loop stops. The following statements set up a `while` loop that looks through an array for a customer named `Smith`:

```
$customers = array( "Huang", "Smith", "Jones" );
$testvar = "no";
$k = 0;
while ( $testvar != "yes" )
{
  if ($customers[$k] == "Smith" )
  {
    $testvar = "yes";
    echo "Smith<br>";
  }
  else
  {
    echo "$customers[$k], not Smith<br>";
  }
  $k++;
}
```

These statements display the following on a Web page:

```
Huang, not Smith
Smith
```

The program executes the previous statements as follows:

1. Sets the variables before starting the loop: `$customers` (an array with three values), `$testvar` (a test variable set to `"no"`), and `$k` (a counter variable set to `0`).
2. Starts the loop by testing whether `$testvar != "yes"` is true. Because `$testvar` was set to `"no"`, the statement is true, so the loop continues.
3. Tests the `if` statement. Is `$customers[$k] == "Smith"` true? At this point, `$k` is `0`, so the program checks `$customers[0]`. Because `$customers[0]` is `"Huang"`, the statement is not true. The statements in the `if` block are not executed, so the program skips to the `else` statement.

4. Executes the statement in the `else` block. The `else` block outputs the line `"Huang, not Smith"`. This is the first line of the output.
5. Adds 1 to `$k`, which now becomes equal to 1.
6. Reaches the bottom of the loop.
7. Goes to the top of the loop.
8. Tests the condition again. Is `$testvar != "yes"` true? Because `$testvar` has not been changed and is still set to `"no"`, it is true, so the loop continues.
9. Tests the `if` statement. Is `$customers[$k] == "Smith"` true? At this point, `$k` is 1, so the program checks `$customers[1]`. Because `$customers[1]` is `"Smith"`, the statement is true. So the loop enters the `if` block.
10. Executes the statements in the `if` block. Sets `$testvar` to `"yes"`. Outputs `"Smith"`. This is the second line of the output.
11. Adds 1 to `$k` which now becomes equal to 2.
12. Reaches the bottom of the loop.
13. Goes to the top of the loop.
14. Tests the condition again. Is `$testvar != "yes"` true? Because `$testvar` has been changed and is now set to `"yes"`, it is *not* true. The loop stops.

It's possible to write a `while` loop that is infinite — that is, a loop that loops forever. Without intending to, you can easily write a loop in which the condition is always true. If the condition never becomes false, the loop never ends. For a discussion of infinite loops, see the "Infinite loops" section, later in this chapter.

Using do..while loops

A `do..while` loop is similar to a `while` loop. A `do..while` loop continues repeating as long as certain conditions are true. You set up a condition. The condition is tested at the bottom of each loop. If the condition is true, the loop repeats. When the condition is not true, the loop stops.

The general format for a `do..while` loop is

```
do
{
   block of statements
} while( condition );
```

The following statements set up a loop that looks for the customer named `Smith`. This program does the same thing as a program in the preceding section using a `while` loop:

```
$customers = array( "Huang", "Smith", "Jones" );
$testvar = "no";
$k = 0;
do
{
  if ($customers[$k] == "Smith" )
  {
    $testvar = "yes";
    echo "Smith<br>";
  }
  else
  {
    echo "$customers[$k], not Smith<br>";
  }
  $k++;
} while ( $testvar != "yes" );
```

The output of these statements in a browser is

```
Huang, not Smith
Smith
```

This is the same output shown for the `while` loop example. The difference between a `while` loop and a `do..while` loop is where the condition is checked. In a `while` loop, the condition is checked at the top of the loop. Therefore, the loop will never execute if the condition is never true. In the `do..while` loop, the condition is checked at the bottom of the loop. Therefore, the loop always executes at least once even if the condition is never true.

For instance, in the preceding loop that checks for the name `Smith`, suppose the original condition is set to `yes`, instead of `no`, by using this statement:

```
$testvar = "yes";
```

The condition would test false from the beginning. It would never be true. In a `while` loop, there would be no output. The statement block would never run. However, in a `do..while` loop, the statement block would run once before the condition was tested. Thus, the `while` loop would produce no output, but the `do..while` loop would produce the following output:

```
Huang, not Smith
```

The `do..while` loop produces one line of output before the condition is tested. It does not produce the second line of output because the condition tests false.

Infinite loops

You can easily set up loops so that they never stop. These are *infinite loops*. They repeat forever. However, seldom does anyone create an infinite loop intentionally. It is usually a mistake in the programming. For instance, a slight change to the program that sets up a `while` loop can make it into an infinite loop.

Here is the program shown in the "Using while loops" section, earlier in this chapter:

```
$customers = array ( "Huang", "Smith", "Jones" );
$testvar = "no";
$k = 0;
while ( $testvar != "yes" )
{
  if ($customers[$k] == "Smith" )
  {
    $testvar = "yes";
    echo "Smith<br>";
  }
  else
  {
    echo "$customers[$k], not Smith<br>";
  }
  $k++;
}
```

Here is the program with a slight change:

```
$customers = array ( "Huang", "Smith", "Jones" );
$testvar = "no";
while ( $testvar != "yes" )
{
  $k = 0;
  if ($customers[$k] == "Smith" )
  {
    $testvar = "yes";
    echo "Smith<br>";
  }
  else
  {
    echo "$customers[$k], not Smith<br>";
  }
  $k++;
}
```

The small change is moving the statement `$k = 0;` from outside the loop to inside the loop. This small change makes it into an endless loop. The output of this changed program is

```
Huang, not Smith
Huang, not Smith
Huang, not Smith
Huang, not Smith
...
```

This will repeat forever. Every time the loop runs, it resets `$k` to `0`. Then it gets `$customers[0]` and echoes it. At the end of the loop, `$k` is incremented to `1`. However, when the loop starts again, `$k` is set back to `0`. Consequently, only the first value in the array, `Huang`, is ever read. The loop never gets to the name `Smith`, and `$testvar` is never set to `"yes"`. The loop is endless.

Don't be embarrassed if you write an infinite loop. I guarantee that the best programming guru in the world has written many infinite loops. It's not a big deal. If you are testing a program and get output in your Web page repeating endlessly, it will stop by itself in a short time. The default time is 30 seconds, but the timeout period may have been changed by the PHP administrator. You can also click the Stop button on your browser to stop the display in your browser. Then figure out why the loop is repeating endlessly and fix it.

A common mistake that can result in an infinite loop is using a single equal sign (=) when you mean a double equal sign (==). The single equal sign stores a value in a variable; the double equal sign tests whether two values are equal. If you write the following condition with a single equal sign:

```
while ($testvar = "yes")
```

it is always true. The condition simply sets `$testvar` equal to `"yes"`. This is not a question that can be false. What you probably meant to write is this:

```
while ($testvar == "yes")
```

This is a question asking whether `$testvar` is equal to `"yes"`, which can be answered either true or false.



You can bulletproof your programs against this error by changing the condition to `"yes" == $testvar`. It's less logical to read but protects against the single-equal-sign problem. If you use a single equal sign instead of a double equal sign in this condition, you get an error and your program fails to run.

Another common mistake is to leave out the statement that increments the counter. For instance, in the program earlier in this section, if you leave out the statement `$k++;`, `$k` is always `0` and the result is an infinite loop.

Breaking out of a loop

Sometimes you want your program to break out of a loop. PHP provides two statements for this purpose:

- `break`: Breaks completely out of a loop and continues with the program statements after the loop.
- `continue`: Skips to the end of the loop where the condition is tested. If the condition tests positive, the program continues from the top of the loop.

`break` and `continue` are usually used in a conditional statement. `break`, in particular, is used most often in `switch` statements, as I discuss earlier in the chapter.

The following two sets of statements show the difference between `continue` and `break`. The first statements use the `break` statement:

```
$counter = 0;
while( $counter < 5 )
{
  $counter++;
  If( $counter == 3 )
  {
      echo "break<br>";
      break;
  }
  echo "End of while loop: counter=$counter<br>";
}
echo "After the break loop<p>";
```

The following statements use the `continue` statement:

```
$counter = 0;
while( $counter < 5 )
{
  $counter++;
  if( $counter == 3 )
  {
```

```
      echo "continue<br>";
      continue;
  }
  echo "End of while loop: counter=$counter<br>";
}
echo "After the continue loop<br>";
```

These statements build two loops that are the same, except the first uses `break` and the second uses `continue`. The output from the first set of statements that uses the `break` statement displays in your browser as follows:

```
End of while loop: counter=1
End of while loop: counter=2
break
After the break loop
```

The output from the second set of statements, with the `continue` statement, is

```
End of while loop: counter=1
End of while loop: counter=2
continue
End of while loop: counter=4
End of while loop: counter=5
After the continue loop
```

The first loop ends at the `break` statement. It stops looping and jumps immediately to the statement after the loop. The second loop does not end at the `continue` statement. It just stops the third repeat of the loop and jumps back up to the top of the loop. It then finishes the loop, with the fourth and fifth repeats, before it goes to the statement after the loop.

One use for `break` statements is insurance against infinite loops. The following statements inside a loop can stop it at a reasonable point:

```
$test4infinity++;
if ($test4infinity > 100 )
{
   break;
}
```

If you're sure that your loop should never repeat more than 100 times, these statements will stop the loop if it becomes endless. Use whatever number seems reasonable for the loop that you're building.

Using Functions

Applications often perform the same task at different points in the program or in different programs. For instance, your application might display the company logo on several Web pages or in different parts of the program. Suppose that you use the following statements to display the company logo:

```
echo "<hr style='width: 50' align='left' />","\n";
echo "<img src='/images/logo.jpg' width='50'
           height='50' />","\n";
echo "<hr style='width: 50' align='left' />","\n";
```

You can create a function that contains the preceding statements and name it `display_logo`. Then whenever the program needs to display the logo, you can just call the function `display_logo` with a simple function call, as follows:

```
display_logo();
```



Notice the parentheses after the function name. These are required in a function call because they tell PHP that this is a function.

Using a function offers several advantages:

- **Less typing:** You have to type the statements only once — in the function. Forever after, you just use the function call and never have to type the statements again.
- **Easier to read:** The line `display_logo()` is much easier for a person to understand at a glance.
- **Fewer errors:** After you have written your function and fixed all its problems, it runs correctly wherever you use it.
- **Easier to change:** If you decide to change how the task is performed, you need to change it in only one place. You just change the function instead of finding all the different places in your program where you performed the task and changing the code in all those places. For instance, suppose that you changed the name of the graphics file that holds the company logo. You just change the filename in one place — the function — and it works correctly everywhere.

You can create a function by putting the code into a function block. The general format is as follows:

```
function functionname()
{
   block of statements;
   return;
}
```

For instance, you create the function to display the company logo with the following statements:

```
function display_logo()
{
   echo "<hr style='width: 50' align='left' />","\n";
   echo "<img src='/images/logo.jpg' width='50'
           height='50' />","\n";
   echo "<hr style='width: 50' align='left' />","\n";
   return;
}
```

The `return` statement stops the function and returns to the main program. The `return` statement at the end of the function is not required, but it makes the function easier to understand. The `return` statement is often used for a conditional end to a function.

Suppose that your function displays an electronics catalog. You might use the following statement at the beginning of the function:

```
if ( $age < 13 )
   return;
```

If the customer's age is less than 13, the function stops, and the electronics catalog isn't displayed.

You can put functions anywhere in the program, but the usual practice is to put all the functions at the beginning or the end of the program file. Functions that you plan to use in more than one program can be in a separate file. Each program accesses the functions from the external file. For more on organizing applications into files and accessing separate files, see Chapter 10.

Notice that the sample function is quite simple. It doesn't use variables, and it doesn't share any information with the main program. It just performs an independent task when called. You can use variables in functions and pass information between the function and the main program as long as you know the rules and limitations. The remaining sections in this chapter explain how to use variables and pass values.

Using variables in functions

You can create and use variables that are local to the function. That is, you can create and use a variable inside your function. However, the variable is not available outside the function; it's not available to the main program. You can make the variable available at any location in the program by using a special statement called `global`. For instance, the following function creates a variable:

```
function format_name()
{
   $first_name = "Goliath";
   $last_name = "Smith";
   $name = $last_name.", ".$first_name;
}
format_name();
echo "$name";
```

These statements produce no output. In the `echo` statement, `$name` doesn't contain any value. The variable `$name` was created inside the function, so it doesn't exist outside the function.

To create a variable inside a function that does exist outside the function, you use the `global` statement. The following statements contain the same function with a `global` statement added:

```
function format_name()
{
   $first_name = "Goliath";
   $last_name = "Smith";
   global $name;
   $name = $last_name.", ".$first_name;
}
format_name();
echo "$name";
```

The program now echoes this:

```
Smith, Goliath
```

The `global` statement makes the variable available at any location in the program. You must make the variable global before you can use it. If the `global` statement follows the `$name` assignment statement, the program does not produce any output.

The same rules apply when you're using a variable created in the main program. You can't use a variable in a function that was created outside the function unless the variable is global, as shown in the following statements:

```
$first_name = "Goliath";
$last_name = "Smith";
function format_name()
{
   global $first_name, $last_name;
   $name = $last_name.", ".$first_name;
   echo "$name";
}
format_name();
```

If you don't use the `global` statement, `$last_name` and `$first_name` inside the function are different variables, created when you name them. They have no values. The program would produce no output without the `global` statement.

Passing values between a function and the main program

You can pass values into the function and receive values from the function. For instance, you might write a function to add the correct sales tax to an order. The function would need to know the cost of the order and which state the customer resides in. The function would need to send back the amount of the sales tax.

Passing values to a function

You can pass values to a function by putting the values between the parentheses when you call the function, as follows:

```
functionname(value,value,...);
```

Of course, the variables can't just show up. The function must be expecting them. The `function` statement includes variable names for the values that it's expecting, as follows:

```
function functionname($varname1,$varname2,...)
{
    statements
    return;
}
```

For example, the following function computes the sales tax:

```
function compute_salestax($amount,$custState)
{
  switch ( $custState )
  {
    case "OR" :
      $salestaxrate = 0;
      break;
    case "CA" :
      $salestaxrate = 1.0;
      break;
    default:
      $salestaxrate = .5;
      break;
  }
  $salestax = $amount * $salestaxrate;
  echo "$salestax<br>";
}
$cost = 2000.00;
$custState = "CA";
compute_salestax($cost,$custState);
```

The first line shows that the function expects two values, as follows:

```
function compute_salestax($amount,$custState)
```

The last line is the function call, which passes two values to the function `compute_salestax`, as it expects. The amount of the order and the state in which the customer resides are passed. The output from this program is `2000` because the tax rate for California is 100 percent.

You can pass as many values as you need to. Values can be variables or values, including computed values. The following function calls are valid:

```
compute_salestax(2000,"CA");
compute_salestax(2*1000,"");
compute_salestax(2000,"C"."A");
```

Values can be passed in an array. The function receives the variable as an array. For instance, the following statements pass an array:

```
$arrayofnumbers = array( 100, 200);
addnumbers($arrayofnumbers);
```

The function receives the entire array. For instance, suppose the function starts with the following statement:

```
function addnumbers($numbers)
```

The variable $numbers is an array. The function can include statements such as

```
return $numbers[0] + $numbers[1];
```

The values passed are passed by position. That is, the first value in the list that you pass is used as the first value in the list that the function expects, the second is used for the second, and so forth. If your values aren't in the same order, the function uses the wrong value when performing the task. For instance, for compute_salestax, you might call compute_salestax passing values in the wrong order:

```
compute_salestax($custState,$orderCost);
```

The function uses the state as the cost of the order, which it sets to 0 because the value passed is a string. It sets the state to the number in $orderCost, which would not match any of its categories. The output would be 0.

If you do not send enough values, the function sets the missing value to an empty string for a string variable or to 0 for a number. If you send too many values, the function ignores the extra values.

If you pass the wrong number of values to a function, you might get a warning message, depending on the error message level that PHP is set to:

```
Warning: Missing argument 2 for compute_salestax() in /test7.php on line 5
```

For the lowdown on warning messages, check out Chapter 6.

You can set default values to be used when a value isn't passed. The defaults are set when you write the function by assigning a default value for the value(s) that it is expecting, as follows:

```
function add_2_numbers($num1=1,$num2=1)
{
   $total = $num1 + $num2;
   return $total;
}
```

If one or both values are not passed, the function uses the assigned defaults. But if a value is passed, it is used instead of the default. For example, you could use one of the following calls:

```
add_2_numbers(2,2);
add_2_numbers(2);
add_2_numbers();
```

The results, in consecutive order, are as follows:

```
$total = 4
$total = 3
$total = 2
```

Getting a value from a function

When you call a function, you can pass values as just described. The function can also pass a value back to the program that called it. Use the `return` statement to pass a value back to the calling program. The program can store the value in a variable or use the value directly, such as using it in a conditional statement. The `return` statement also returns control to the main program; that is, it stops the function.

The general format of the return statement is

```
return value;
```

For instance, in the tax program from the preceding section, I echo the sales tax by using the following statements:

```
$salestax = $amount * $salestaxrate;
echo "$salestax<br>";
```

I could return the sales tax to the main program, rather than echoing it, by using the following statement:

```
$salestax = $amount * $salestaxrate;
return $salestax;
```

In fact, I could use a shortcut and send it back to the main program with one statement:

```
return $amount * $salestaxrate;
```

The `return` statement sends the `salestax` back to the main program and ends the function. The main program can use the value in any of the usual ways. The following statements use the function call in valid ways:

```
$salestax = compute_salestax($cost,$custState);
```

```
$totalcost = $cost + compute_salestax($cost,$custState);
```

```
if( compute_salestax($cost,$custState) > 100000.00 )
        $echo "Thank you very, very, very much";
```

```
foreach($customerOrder as $amount)
{
   $total = $amount +
                compute_salestax($amount,$custState);
   echo "Your total is $total";
}
```

A `return` statement can return only one value. However, the value returned can be an array, so you can actually return many values from a function.

You can use `return` statements in a conditional statement to return different values for different conditions. For example, the following function returns one of two different strings:

```
function compare_values($value1,$value2)
{
  if($value1 < $value2)
  {
     return "less than";
  }
  else
  {
     return "not less than";
  }
}
```

Although the function contains two `return` statements, only one is going to be executed, depending on the values in `$value1` and `$value2`.

Using built-in functions

PHP's many built-in functions are one reason why PHP is so powerful and useful for Web pages. The functions included with PHP are normal functions. They are no different than functions that you create yourself. It's just that PHP already did all the work for you.

I discuss some of the built-in functions in this chapter and the earlier chapters. For example, see Chapter 6 for more on the functions `unset` and `number_format`. Some useful functions for interacting with your MySQL database are discussed in Chapter 8. Other useful functions are listed in Part V. And all the functions are listed and described in the PHP documentation on the PHP Web site at `www.php.net/docs.php`.

Chapter 8

Data In, Data Out

In This Chapter

- Connecting to the database
- Getting information from the database
- Using HTML forms with PHP
- Getting data from an HTML form
- Processing the information that users type into HTML forms
- Storing data in the database
- Using functions to move data into and out of the database

PHP and MySQL work well together. This dynamic partnership is what makes PHP and MySQL so attractive for Web database application development. Whether you have a database full of information that you want to make available to users (such as a product catalog) or a database waiting to be filled up by users (for example, a membership database), PHP and MySQL work together to implement your application.

One of PHP's strongest features is its ability to interact with databases. It provides functions that make communicating with MySQL extremely simple. You use PHP functions to send SQL queries to the database. You don't need to know the details of communicating with MySQL; PHP handles the details. You only need to know the SQL queries and how to use the PHP functions.

In previous chapters, I describe the tools that you use to build your Web database application. You find out how to build SQL queries in Chapter 4 and how to construct and use the building blocks of the PHP language in Chapters 6 and 7. In this chapter, you find out how to use these tools for the specific tasks that a Web database application needs to perform.

PHP and MySQL Functions

You use built-in PHP functions to interact with MySQL. These functions connect to the MySQL server, select the correct database, send SQL queries, and perform other communication with MySQL databases. You don't need to

know the details of interacting with the database because PHP handles all the details. You need to know only how to use the functions.

The MySQL Web site, as of this writing, offers versions 4.1, 5.0, and 5.1. Version 5.0 is the current stable version — the version most people should install and use. MySQL versions 4.0 and 3.23 may still be in use on some Web sites. As of PHP 5, PHP offers two sets of functions for communicating with MySQL: one set of functions (the `mysqli` functions) for use with MySQL 4.1 or later and another set of functions (the `mysql` functions) for use with MySQL 4.0 and earlier versions.

If you are using Windows with PHP 5 or 6, you can enable the correct function set by activating the correct extension in your `php.ini` file. See the "Configuring PHP" section in Appendix B. If you are using Linux, Unix, or Mac, you need to use a configuration option to activate the correct set of functions, as explained in Step 10 of the installation instructions in Appendix B.

If you are using PHP 4, the `mysqli` functions are not available. Instead, you use the `mysql` functions, even with later versions of MySQL. The `mysql` functions can communicate with the later versions of MySQL, but they cannot access some of the new features added in the later versions of MySQL. The `mysql` functions are activated automatically in PHP 4.

Throughout the book, my examples and programs use MySQL 5.0 and use the `mysqli` functions to communicate with MySQL. The PHP functions for use with MySQL 5.0 have the following general format:

```
mysqli_function(value,value,...);
```

The `i` in the function name stands for *improved* (MySQL Improved). The second part of the function name is specific to the function, usually a word that describes what the function does. In addition, the function requires one or more values to be passed, specifying things such as the database connection or the data location. Following are two of the functions discussed in this chapter:

```
mysqli_connect(connection information);
mysqli_query($cxn,"SQL statement");
```

If you are using PHP 4 or are communicating with MySQL 4.0 or earlier, the corresponding `mysql` functions are

```
mysql_connect(connection information);
mysql_query("SQL statement");
```

The functionality and syntax of the functions are similar but not identical for all functions. If you need to use the `mysql` functions rather than the `mysqli` functions, you will need to edit the programs in this book, replacing the `mysqli` functions with `mysql` functions. Table 8-1 shows the equivalent `mysql` functions and their syntax.

Table 8-1 Syntax for mysql and mysqli Functions

mysqli Function	*mysql Function*
`mysqli_connect($host, $user,$passwd,$dbname)`	`mysql_connect($host, $user,$passwd)` followed by `mysql_select_db($dbname)`
`mysqli_errno($cxn)` `mysql_errno($cxn)`	`mysql_errno()` or
`mysqli_error($cxn)` `mysql_error($cxn)`	`mysql_error()` or
`mysqli_fetch_array($result)`	`mysql_fetch_array($result)`
`mysqli_fetch_assoc($result)`	`mysql_fetch_assoc($result)`
`mysqli_fetch_row($result)`	`mysql_fetch_row($result)`
`mysqli_insert_id($cxn)`	`mysql_insert_id($cxn)`
`mysqli_num_rows($result)`	`mysql_num_rows($result)`
`mysqli_query($cxn,$sql)`	`mysql_query($sql)` or `mysql_query($sql,$cxn)`
`mysqli_select_db ($cxn,$dbname)`	`mysql_select_db($dbname)`
`mysqli_real_escape_string ($cxn,$data)`	`mysql_real_escape_string ($data)`

Making a Connection

Before you can store any data or get any data, you need to connect to the database. The database might be on the same computer with your PHP programs, or it might be on a different computer. You don't need to know the details of connecting to the database because PHP handles all the details. All you need to know is the name and location of the database. Think of a database connection in the same way that you think of a telephone connection. You don't need to know the details about how the connection is made — that is, how your words move from your telephone to another telephone — you need to know only the area code and phone number. The phone company handles the details.

After connecting to the database, you send SQL queries to the MySQL database by using a PHP function designed for this purpose. You can send as many queries as you need. The connection remains open until you close it or the program ends. Similarly, in a telephone conversation, the connection remains open until you terminate it by hanging up the phone.

Connecting to the MySQL server

The first step in communicating with your MySQL database is connecting to the MySQL server. To connect to the server, you need to know the name of the computer where the database is located, the name of your MySQL account, and the password to your MySQL account. To open the connection, use the `mysql_connect` function as follows:

```
$cxn=mysqli_connect("addr","acct","password","dbname")
     or  die ("message");
```

Fill in the following information:

- *addr*: The name of the computer where MySQL is installed — for example, `databasehost.mycompany.com`. If the MySQL database is on the same computer as your Web site, you can use `localhost` as the computer name. If this information is blank (`""`), PHP assumes `localhost`.
- *acct*: The name of any valid MySQL account. (I discuss MySQL accounts in detail in Chapter 5.)
- *password*: The password for the MySQL account specified by *acct*. If the MySQL account does not require a password, don't type anything between the quotes: `""`.
- *dbname*: The name of the database you want to communicate with. This parameter is optional. You can select the database later, with a separate command if you prefer. You can select a different database at any point in your program.

  If you are using the `mysql` functions, you cannot select the database in the `connect` function. You must use a separate function — `mysql_select_db` — to select the database.



- *message*: The message sent to the browser if the connection fails. The connection fails if the computer or network is down or the MySQL server isn't running. It also may fail if the information provided isn't correct — for example, if the password contains a typo.

  You might want to use a descriptive *message* during development, such as `Couldn't connect to server`, but use a more general message suitable for customers after the application is in use, such as `The Pet Catalog is not available at the moment. Please try again later.`





The *addr* includes a port number that is needed for the connection. Almost always, the port number is 3306. On rare occasions, the MySQL administrator needs to set up MySQL to connect on a different port. In these cases, the port number is required for the connection. The port number is specified as *hostname:portnumber*. For instance, you might use `localhost:8808`.

With these statements, `mysqli_connect` attempts to open a connection to the named computer, using the account name and password provided. If the connection fails, the program stops running at this point and sends *`message`* to the browser.

The following statement connects to the MySQL server on the local computer by using a MySQL account named `catalog` that does not require a password:

```
$cxn = mysqli_connect("localhost","catalog",
                      "","PetCatalog")
     or  die ("Couldn't connect to server.");
```

For security reasons, it's a good idea to store the connection information in variables and use the variables in the connection statement, as follows:

```
$host="localhost";
$user="catalog";
$password="";
$dbname = "PetCatalog";
$cxn = mysqli_connect($host,$user,$password,$dbname)
     or  die ("Couldn't connect to server.");
```

For even more security, you can put the assignment statements for the connection information in a separate file in a hidden location so that the account name and password aren't even in the program. I explain how to do this in Chapter 10.

The variable `$cxn` contains information that identifies the connection. You can have more than one connection open at a time by using more than one variable name. A connection remains open until you close it or until the program ends. You close a connection as follows:

```
mysqli_close($connectionname);
```

For instance, to close the connection in the preceding example, use this statement:

```
mysqli_close($cxn);
```

Selecting the right database

If you do not select the database with the connect function, you can select the database using the `mysqli_select_db` function. You can also use this function to select a different database at any time in your program. The format is

```
mysqli_select_db($connectionname,"databasename")
      or die ("message");
```



Handling MySQL errors

You use the `mysqli` functions of the PHP language, such as `mysqli_connect` and `mysqli_query`, to interact with the MySQL database. If one of these functions fails to execute correctly, a MySQL error message is returned with information about the problem. However, this error message isn't sent to the browser unless the program deliberately sends it. Here are the three usual ways to call the `mysqli` functions:

- **Calling the function without error handling.** The function is called without any statements that provide error messages. For instance, the `mysqli_connect` function can be called as follows:

  ```
  $cxn = mysqli_connect($host,$user,$password,$dbname);
  ```

  If this statement fails (for instance, the account is not valid), the connection is not made, but the remaining statements in the program continue to execute. In most cases, this isn't useful because some of the statements in the rest of the program might depend on having an open connection, such as getting or storing data in the database.

- **Calling the function with a `die` statement.** The function is called with a `die` statement that sends a message to the browser. For instance, the `mysqli_connect` function can be called as follows:

  ```
  $cxn = mysqli_connect($host,$user,$password,$dbname)
      or die ("Couldn't connect to server");
  ```

  If this statement fails, the connection is not made, and the `die` statement is executed. The `die` statement stops the program and sends the message to the browser. If the connection can't be established, no more statements are executed. You can put any message that you want in the `die` statement.

- **Calling the function in an `if` statement.** The function is called by using an `if` statement that executes a block of statements if the connection fails. For instance, the `mysqli_connect` function can be called as follows:

  ```
  if (!$cxn = mysqli_connect($host,$user,$password,$dbname))
  {
      $message = mysqli_error($cxn);
      echo "$message";
      die();
  }
  ```

  If this statement fails, the statements in the `if` block are executed. The `mysqli_error` function returns the MySQL error message and saves it in the variable `$message`. The error message is then echoed. The `die` statement ends the program so that no more statements are executed. Notice the `!` (exclamation point) in the `if` statement. `!` means `"not"`. In other words, the `if` statement is true if the assignment statement is not true.

The type of error handling you want to include in your program depends on what you expect to happen in the program. When you're developing the program, you expect some errors to happen. Therefore, during development, you probably want error handling that is more descriptive, such as

the third method in the preceding list. For instance, suppose that you're using an account called `root` to access your database and that you make a typo as in the following statements:

```
$host = "localhost";
$user = "rot";
$password = "";
if (!$cxn = mysqli_connect($host,$user,$password))
      {
          $message = mysqli_error($cxn);
          echo "$message";
          die();
      }
```

Because you typed `"rot"` instead of `"root"`, you would see an error message similar to the following one:

```
Access denied for user: 'rot@localhost' (Using password: NO)
```

This error message has the information that you need to figure out what the problem is; it shows your account name with the typo. However, after your program is running and customers are using it, you probably don't want your users to see a technical error message like the preceding one. Instead, you probably want to use the second method with a general statement in the `die` message, such as `The Pet Catalog is not available at the moment. Please try again later.`

Fill in the following information:

- *`connectionname`*: The variable that contains the connection information.
- *`databasename`*: The name of the database.
- *`message`*: The message that is sent to the browser if the database can't be selected. The selection might fail because the database can't be found, which is usually the result of a typo in the database name.

For instance, you can select the database `PetCatalog` with the following statement:

```
mysqli_select_db($cxn,"PetCatalog")
      or die ("Couldn't select database.");
```

If `mysqli_select_db` is unable to select the database, the program stops running at this point, and the message `Couldn't select database.` is sent to the browser.

For security reasons, it's a good idea to store the database name in a variable and use the variable in the connection statement, as follows:

```
$database = "PetCatalog";
mysql_select_db($cxn,$database)
      or die ("Couldn't select database.");
```

For more security, you can put the assignment statement for the database name in a separate file in a hidden location — as suggested for the assignment statements for the connection information — so that the database name isn't in the program. I explain how to do this in Chapter 10.

The database stays selected until you select a different database. To select a different database, just use a new `mysqli_select_db` function statement.

Sending SQL queries

After you have an open connection to the MySQL server and PHP knows which database you want to interact with, you send your SQL query. The *query* is a request to the MySQL server to store some data, update some data, or retrieve some data. (See Chapter 4 for more on the SQL language and how to build SQL queries.)

To interact with the database, put your SQL query into a variable and send it to the MySQL server by using the function `mysqli_query`, as in the following example:

```
$query = "SELECT * FROM Pet";
$result = mysqli_query($cxn,$query)
     or    die ("Couldn't execute query.");
```

The query is executed on the currently selected database for the specified connection.

The variable `$result` holds information on the result of executing the query. The information depends on whether or not the query gets information from the database:

- **For queries that don't get any data:** The variable `$result` contains information on whether the query executed successfully or not. If it's successful, `$result` is set to `TRUE`; if it's not successful, `$result` is set to `FALSE`. Some queries that don't return data are `INSERT` and `UPDATE`.
- **For queries that return data:** The variable `$result` contains a result identifier that identifies where the returned data is located, not the returned data itself. Some queries that do return data are `SELECT` and `SHOW`.



Beginning with MySQL 4.1, if you use PHP 5 and the `mysqli` functions, you can send multiple queries to the server at once, separated by semicolons. You use the `mysqli_multiple_query` function for this purpose. However, sending more than one query at once can make your program less secure. Use multiple queries seldom and carefully.



The use of single and double quotes can be a little confusing when assigning the query string to `$query`. You are actually using quotes on two levels: the quotes needed to assign the string to `$query` and the quotes that are part of the SQL language query itself. The following rules will help you avoid any problems with quotes:

- Use double quotes at the beginning and end of the string.
- Use single quotes before and after variable names.
- Use single quotes before and after literal values.

The following are examples of assigning query strings:

```
$query = "SELECT firstName FROM Member";
$query = "SELECT firstName FROM Member WHERE lastName='Smith'";
$query = "UPDATE Member SET lastName='$last_name'";
```



The query string itself does not include a semicolon (`;`), so don't put a semicolon inside the final quote. The only semicolon is at the very end; this is the PHP semicolon that ends the statement.

Getting Information from a Database

Getting information from a database is a common task for Web database applications. Here are two common uses for information from the database:

- **Use the information to conditionally execute statements.** For instance, you might get the state of residence from the Member Directory and send different messages to members who live in different states.
- **Display the information in a Web page.** For instance, you might want to display product information from your database.

To use the database information in a program, you need to put the information in variables. Then you can use the variables in conditional statements, `echo` statements, or other statements. Getting information from a database is a two-step process:

1. You build a `SELECT` query and send the query to the database. When the query is executed, the selected data is stored in a temporary location.
2. You move the data from the temporary location into variables and use it in your program.

Sending a SELECT query

You use the SELECT query to get data from the database. SELECT queries are written in the SQL language. (I discuss the SELECT query in detail in Chapter 4.)

To get data from the database, build the SELECT query that you need, storing it in a variable, and then send the query to the database. The following statements select all the information from the Pet table in the PetCatalog database:

```
$query = "SELECT * FROM Pet";
$result = mysqli_query($cxn,$query)
     or    die ("Couldn't execute query.");
```

The mysqli_query function gets the data requested by the SELECT query and stores it in a temporary location. You can think of this data as being stored in a table, similar to a MySQL table, with the information in rows and columns.

The function returns a result identifier that contains the information needed to find the temporary location where the data is stored. In the preceding statements, the result identifier is put into the variable $result. If the function fails (because, for example, the query is incorrect), $result contains FALSE.

The next step after executing the function is to move the data from its temporary location into variables that can be used in the program.

Getting and using the data

You use the mysqli_fetch_assoc function or the mysqli_fetch_row function to get the data from the temporary location. The mysqli_fetch_assoc function returns the data in an associative array; mysqli_fetch_row returns the data in a numeric array. Occasionally, you might need to fetch the data in both an associative and a numeric array, which you can do with mysqli_fetch_array.

The functions get one row of data from the temporary location. The temporary data table might contain only one row of data or, more likely, your SELECT query resulted in more than one row of data. If you need to fetch more than one row of data from the temporary location, you use the mysqli_fetch_assoc or mysqli_fetch_row function in a loop.

Getting one row of data

To move the data from its temporary location and put it into variables that you can use in your program, you use the PHP function mysqli_fetch_assoc or mysql_fetch_row. The general format for these functions is

```
$row = mysqli_fetch_assoc($resultidentifier);
```

This statement gets one row from the data table in the temporary location and puts it in an array variable called `$row`. *`resultidentifier`* is the variable that points to the temporary location of the results.

The `mysql_fetch_array` function gets one row of data from the temporary location. In some cases, one row is all you selected. For instance, to check the password entered by a user, you only need to get the user's password from the database and compare it with the password that the user entered. The following statements check a password:

```
$userEntry = "secret";   // password user entered in form
$query = "SELECT password FROM Member
                 WHERE loginName='gsmith'";
$result = mysqli_query($cxn,$query)
     or    die ("Couldn't execute query.");
$row = mysqli_fetch_assoc($result);
if ( $userEntry == $row['password'] )
{
    echo "Login accepted";
    statements that display Members Only Web pages
}
else
{
    echo "Invalid password";
    statements that allow user to try another password
}
```

Note the following points about the preceding statements:

- The `SELECT` query requests only one field (`password`) from one row (row for `gsmith`).
- The `mysqli_fetch_assoc` function returns an array called `$row` with column names as keys.
- The `if` statement uses two equal signs (`==`) to compare the password that the user typed in (`$userEntry`) with the password obtained from the database (`$row['password']`) to see whether they are the same.
- If the comparison is `true`, the passwords match, and the `if` block (which displays the Members Only Web pages) is executed.
- If the comparison is not `true`, the user did not enter a password that matches the password stored in the database, and the `else` block is executed. The user sees an error message stating that the password is not correct and is returned to the login Web page.



PHP provides a convenient shortcut for using the variables retrieved with the `mysqli_fetch_assoc` function. You can use the `extract` function, which splits the array into variables that have the same name as the key. For instance, you can use the `extract` function to rewrite the previous statements that test the password. Here's how:

```
$userEntry = "secret"; #password entered in a form
$query = "SELECT password FROM Member
                  WHERE loginName='gsmith'";
$result = mysqli_query($cxn,$query)
     or    die ("Couldn't execute query.");
$row = mysqli_fetch_assoc($result);
extract($row);
if ( $userEntry == $password )
{
    echo "Login accepted<br>";
    statements that display Members Only Web pages
}
else
{
    echo "Invalid password<br>";
    statements that allow user to try another password
}
```

Using a loop to get all the rows of data

If you selected more than one row of data, use a loop to get all the rows from the temporary location. The statements in the loop block get one row of data and process it. The loop repeats until all rows have been retrieved. You can use a `while` loop or a `for` loop to retrieve this information. (For more on `while` loops and `for` loops, check out Chapter 7.)

The most common way to process the information is to use a `while` loop as follows:

```
while ( $row = mysqli_fetch_assoc($result))
{
   block of statements
}
```

This loop repeats until it has fetched the last row. If you just want to echo all the data, for example, you would use a loop similar to the following:

```
while ( $row = mysqli_fetch_assoc($result))
{
   extract($row);
   echo "$petType: $petName<br>";
}
```

Now, take a look at an example of how to get information for the Pet Catalog application. Assume the Pet Catalog has a table called `Pet` with four columns: `petName`, `petType`, `petDescription`, and `price`. Table 8-2 shows a sample set of data in the `Pet` table.

Table 8-2 Sample Data in Pet Table

petName	*petType*	*petDescription*	*price*
Unicorn	Horse	Spiral horn centered in forehead	10000
Pegasus	Horse	Flying; wings sprouting from back	15000
Pony	Horse	Very small; half the size of standard horse	500
Asian dragon	Dragon	Serpentine body	30000
Medieval dragon	Dragon	Lizard-like body	30000
Lion	Cat	Large; maned	2000
Gryphon	Cat	Lion body; eagle head; wings	25000

The `petDisplay.php` program in Listing 8-1 selects all the horses from the `Pet` table and displays the information in an HTML table in the Web page. The variable `$pettype` contains information that a user typed into a form.

Listing 8-1: Displaying Items from the Pet Catalog

```
/* Program: petDisplay.php
 * Desc:    Displays all pets in selected category.
 */
?>
<html>
<head><title>Pet Catalog</title></head>
<body>
<?php
  $user="catalog";
  $host="localhost";
  $password="";
  $database = "PetCatalog";
  $cxn = mysqli_connect($host,$user,$password,$database)
         or die ("couldn't connect to server");
  $pettype = "horse";  //horse was typed in a form by user
$query = "SELECT * FROM Pet WHERE petType='$pettype'";
  $result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");

  /* Display results in a table */
  $pettype = ucfirst($pettype)."s";
  echo "<h1>$pettype</h1>";
  echo "<table cellspacing='15'>";
  echo "<tr><td colspan='3'><hr /></td></tr>";
  while($row = mysqli_fetch_assoc($result))
  {
     extract($row);
     $f_price = number_format($price,2);
```

(continued)

Listing 8-1 *(continued)*

```
      echo "<tr>\n
            <td>$petName</td>\n
            <td>$petDescription</td>\n
            <td style='text-align: right'>\$$f_price</td>\n
            </tr>\n";
      echo "<tr><td colspan='3'><hr /></td></tr>\n";
   }
   echo "</table>\n";
?>
</body></html>
```

Figure 8-1 shows the Web page displayed by the program in Listing 8-1. The Web page shows the `Pet` items for the `petType` `horse`, with the display formatted in an HTML table.

The program in Listing 8-1 uses a `while` loop to get all the rows from the temporary location. In some cases, you might need to use a `for` loop. For instance, if you need to use a number in your loop, a `for` loop is more useful than a `while` loop. To use a `for` loop, you need to know how many rows of data were selected. You can find out how many rows are in temporary storage by using the PHP function `mysqli_num_rows`:

```
$nrows = mysqli_num_rows($result);
```

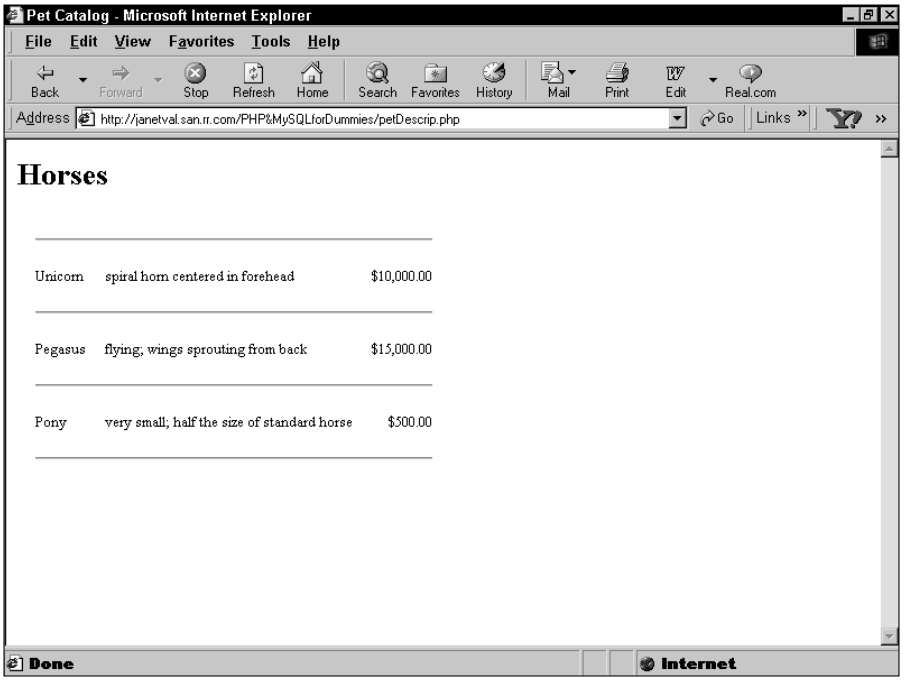


Figure 8-1: The Web page resulting from petDisplay.php.

The variable `$nrows` contains the number of rows in the temporary storage location. By using this number, you can build a `for` loop to get all the rows:

```
for ($i=0;$i<$nrows;$i++)
{
   $row = mysqli_fetch_assoc($result))
   block of statements;
}
```

For instance, the program in Listing 8-1 displays the `Pet` items of the type `horse`. Suppose that you want to number each item. Listing 8-2, the `petDescripFor.php` program, displays a numbered list with a `for` loop.

Listing 8-2: Displaying a Numbered List of Items from the Pet Catalog

```
/* Program: petDescripFor.php
 * Desc:    Displays a numbered list of all pets in
 *          selected category.
 */
?>
<html>
<head><title>Pet Catalog</title></head>
<body>
<?php
  $user="catalog";
  $host="localhost";
  $password="";
  $database = "PetCatalog";
  $cxn = mysqli_connect($host,$user,$password,$database)
         or die ("Couldn't connect to server");
  $pettype = "horse";  //horse was typed in a form by user
  $query = "SELECT * FROM Pet WHERE petType='$pettype'";
  $result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");
  $nrows = mysqli_num_rows($result);

  /* Display results in a table */
echo "<h1>Horses</h1>";
  echo "<table cellspacing='15'>";
  echo "<tr><td colspan='4'><hr /></td></tr>";
  for ($i=0;$i<$nrows;$i++)
  {
     $n = $i + 1;   #add 1 so that numbers don't start
          with 0
     $row = mysqli_fetch_assoc($result);
     extract($row);
     $f_price = number_format($price,2);
      echo "<tr>\n
           <td>$n.</td>\n
           <td>$petName</td>\n
           <td>$petDescription</td>\n
```

(continued)

Listing 8-2 *(continued)*

```
            <td style='text-align: right'>\$$f_price</td>\n
            </tr>\n";
      echo "<tr><td colspan='4'><hr></td></tr>\n";
   }
   echo "</table>\n";
?>
</body></html>
```

Figure 8-2 shows the Web page that results from using the `for` loop in this program. Notice that a number appears before the listing for each `Pet` item on this Web page.

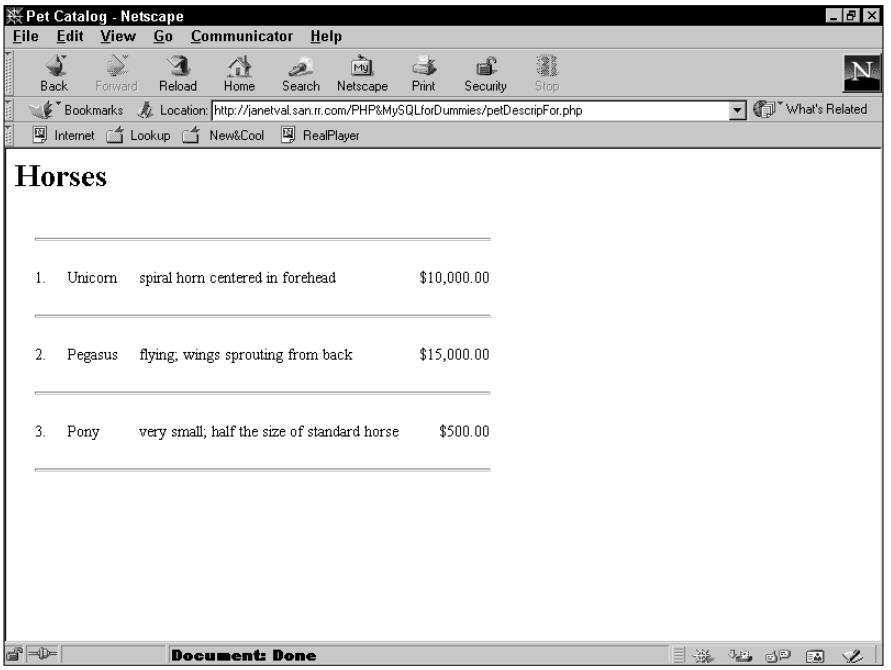


Figure 8-2: The Web page resulting from petDescripFor.php.

Using functions to get data

In most applications, you get data from the database. Often you get the data in more than one location in your program or more than one program in your application. *Functions* — blocks of statements that perform specified tasks — are designed for such situations. (I explain functions in detail in Chapter 7.)

A function to get data from the database can be really useful. Whenever the program needs to get data, you call the function. Functions not only save you a lot of typing but also make the program easier for you to follow. For example,

consider a product catalog, such as the Pet Catalog. You will need to get information about a specific product many times. You can write a function that gets the data and then use that function whenever you need data.

Listing 8-3 for program `getdata.php` shows how to use a function to get data. The function in Listing 8-3 will get the information for any single pet in the Pet Catalog. The pet information is put into an array, and the array is returned to the main program. The main program can then use the information any way that it wants. In this case, it echoes the pet information to a Web page.

Listing 8-3: Using a Function to Get Data from a Database

```
/* Program: getdata.php
 * Desc:    Gets data from a database using a function
 */
?>
<html>
<head><title>Pet Catalog</title></head>
<body>
<?php

  $petInfo = getPetInfo("Unicorn");      //call function

  $f_price = number_format($petInfo['price'],2);
  echo "<p><b>{$petInfo['petName']}</b><br>\n
        Description: {$petInfo['petDescription']}<br>\n
        Price: \${$petInfo['price']}\n"
?>
</body></html>

<?php
function getPetInfo($petName)
{
  $user="catalog";
  $host="localhost";
  $password="";
  $dbname = "PetCatalog";
  $cxn = mysqli_connect($host,$user,$password,$dbname)
         or die ("Couldn't connect to server");
  $query = "SELECT * FROM Pet WHERE petName='$petName'";
$result = mysqli_query($cxn,$query)
           or die ("Couldn't execute query.");
  return mysqli_fetch_assoc($result);
}
?>
```

The Web page displays

```
Unicorn
Description: spiral horn centered in forehead
Price: $10,000.00
```

Note the following about the program in Listing 8-3:

- The program is easier to read with the function call than it would be if all the statements in the function were in the main program.
- The function call sends the string `"Unicorn"`. In most cases, the function call will use a variable name.
- The program creates the variable `$petInfo` to receive the data from the function. `$petInfo` is an array because the information stored in it is an array.

The preceding function is very simple — it returns one row of the results as an array. But functions can be more complex. The preceding section provides a program to get all the pets of a specified type. The program `getPets.php` in Listing 8-4 uses a function for the same purpose. The function returns a multidimensional array with the pet data for all the pets of the specified type.

Listing 8-4: Using a Function to Display a Numbered List of Pets

```
/* Program: getPets.php
 * Desc:    Displays list of items from a database.
 */
?>
<html>
<head><title>Pet Catalog</title></head>
<body>
<?php

  $type = "Horse";
  $petInfo = getPetsOfType($type);       //call function

  /* Display results in a table */
  echo "<h1>{$type}s</h1>";
  echo "<table cellspacing='15'>";
  echo "<tr><td colspan='4'><hr /></td></tr>";
  for($i=1;$i<=sizeof($petInfo);$i++)
  {
      $f_price = number_format($petInfo[$i]['price'],2);
      echo "<tr>\n
           <td>$i.</td>\n
           <td>{$petInfo[$i]['petName']}</td>\n
           <td>{$petInfo[$i]['petDescription']}</td>\n
           <td style='text-align: right'>\$$f_price</td>\n
           </tr>\n";
     echo "<tr><td colspan='4'><hr /></td></tr>\n";
  }
  echo "</table>\n";
?>
</body></html>

<?php
function getPetsOfType($petType)
```

```
{
  $user="catalog";
  $host="localhost";
  $passwd="";
  $cxn = mysqli_connect($host,$user,$passwd,"PetCatalog")
         or die ("Couldn't connect to server");
  $query = "SELECT * FROM Pet WHERE petType='$petType'";
  $result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");

  $j = 1;
  while ($row=mysqli_fetch_assoc($result))
  {
    foreach ($row as $colname => $value)
    {
       $array_multi[$j][$colname] = $value;
    }
    $j++;
  }
  return $array_multi;
}
?>
```

The program in Listing 8-4 proceeds as follows:

1. **It calls the function `getPetsOfType`.** It passes `"horse"` in a variable `$type` containing the type of pet. It also sets up `$petInfo` to receive the data returned by the function.
2. **The function connects to the database and selects the database `PetCatalog`.**
3. **The function sends a query to get all the rows with `$petType` in the `petType` column.** `$petType` is passed to the function in the function call. The data is stored in a table in a temporary location. The variable `$result` identifies the location of the temporary table.
4. **It sets up a counter.** `$j` is a counter that is incremented in each loop. It starts at 1 before the loop.
5. **It starts a `while` loop.** The function attempts to get a row from the temporary data table and is successful. If there were no rows to get in the temporary location, the `while` loop would end.
6. **It starts a `foreach` loop.** The loop walks through the row, processing each field.
7. **It stores values in a multidimensional array.** `$array_multi` is a multidimensional array. Its first key is a number, which is set by the counter. Because this is the first time through the `while` loop, the counter — `$j` — is now equal to `1`. All the fields in the row are stored in `$array_multi` with the column name as the key. (I explain multidimensional arrays in detail in Chapter 7.)
8. **It increments the counter.** `$j` is incremented by 1.

9. **It reaches the end of the `while` loop.**
10. **It returns to the top of the `while` loop.**
11. **It repeats Steps 5–10 for every row in the results.**
12. **It returns `$array_multi` to the main program.** `$array_multi` contains all the data for all the selected rows.
13. **`$petInfo` receives data from the function.** All the data is passed. Figure 8-3 shows the structure of `$petInfo` after the function has finished executing.
14. **The main program sends Pet Descriptions to the browser in an HTML table.**

    The appropriate data is inserted from the `$petInfo` array.

The Web page that results from the program in Listing 8-4 is identical to the Web page shown in Figure 8-2, which is produced by a program that does not use a function. Functions do not produce different output. Any program that you can write that includes a function, you can also write without using a function. Functions just make programming easier.

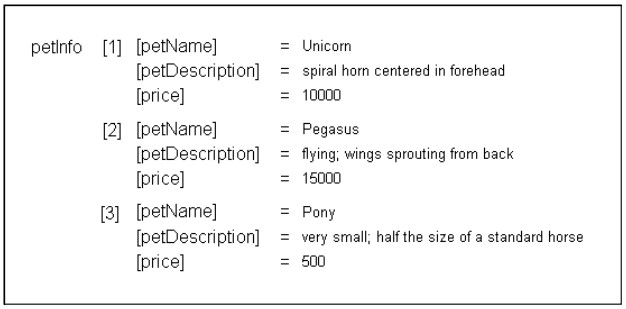


Figure 8-3: The structure of the multi-dimensional array $petInfo.

Getting Information from the User

Many applications are designed to ask questions that users answer by typing information. Sometimes the information is stored in a database; sometimes the information is used in conditional statements to deliver an individual Web page. Some of the most common application tasks that require users to answer questions are

- **Online ordering:** Customers need to select products and enter shipping and payment information.
- **Registering:** Many sites require users to provide some information before they receive certain benefits, such as access to special information or downloadable software.

- **Logging in:** Many sites restrict access to their pages. Users must enter an account name and password before they can see the Web pages.
- **Viewing selected information:** Many sites allow users to specify what information they want to see. For instance, an online catalog might allow users to type the name of the product or select a product category that they want to see.

You ask questions by displaying HTML forms. The user answers the questions by typing information into the form or selecting items from a list. The user then clicks a button to submit the form information. When the form is submitted, the information in the form is passed to a second, separate program, which processes the information.

In the next few sections, I don't tell you about the HTML required to display a form; I assume that you already know HTML. (If you don't know HTML or need a refresher, check out *HTML 4 For Dummies,* 4th Edition, by Ed Tittel and Natanya Pitts; Wiley. What I do tell you is how to use PHP to display HTML forms and to process the information that users type into the form.

Using HTML forms

HTML forms are very important for interactive Web sites. (If you are unfamiliar with HTML forms, you need to read the forms section of an HTML book.) To display a form with PHP, you can do one of the following:

- **Use `echo` statements to echo the HTML for a form.** For example:

```
<?php
  echo "<form action='processform.php'
              method='POST'>\n
     <input type='text' name='name'>\n
     <input type='submit' value='Submit Name'>\n
     </form>\n";
?>
```

- **Use plain HTML outside the PHP sections.** For a plain static form, there is no reason to include it in a PHP section. For example:

```
<?php
   statements in PHP section
?>
<form action="processform.php" method="POST">
  <input type="text" name="fullname">
  <input type="submit" value="Submit Name">
</form>
<?php
  statements in PHP section
?>
```

Either of these methods produces the form displayed in Figure 8-4.

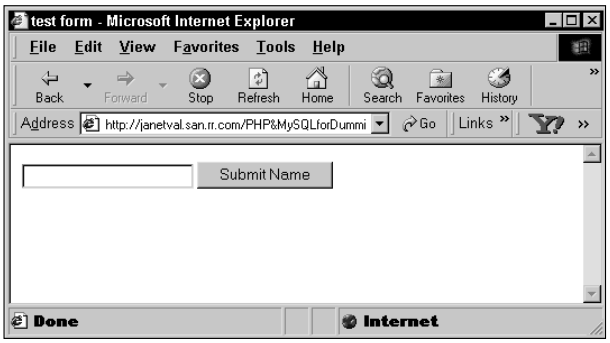


Figure 8-4: A form produced by HTML statements.

Joe Customer fills in the HTML form. He clicks the submit button. You now have the information that you wanted — his name. So where is it? How do you get it?

You get the form information by running a program that receives the form information. When the submit button is clicked, PHP automatically runs a program. The `action` parameter in the form tag tells PHP which program to run. For instance, in the preceding program, the parameter `action=processform.php` tells PHP to run the program `processform.php` when the user clicks the submit button. The program `processform.php` can display, store, or otherwise use the form data it receives when the form is submitted.

When the user clicks the submit button, the program specified in the `action` attribute runs, and statements in this program can get the form information from PHP built-in arrays and use the information in PHP statements. The built-in arrays that contain form information are `$_POST`, `$_GET`, and `$_REQUEST`, which are *superglobal arrays*. When the form uses the `POST` method, the information from the form fields is stored in the `$_POST` array. The `$_GET` array contains the variables passed as part of the URL, including fields passed from a form using the `GET` method. The `$_REQUEST` array contains all the array elements together that are contained in the `$_POST`, `$_GET`, and `$_COOKIES` arrays. Cookies are explained in Chapter 9.

When the form is submitted, the program that runs can get the form information from the appropriate built-in array. In these built-in arrays, each array index is the name of the input field in the form. For instance, if the user typed **Goliath Smith** in the input field shown in Figure 8-4 and clicked the submit button, the program `processform.php` runs and can use an array variable in the following format:

```
$_POST['fullname']
```

Notice that the name typed into the form is available in the $_POST array because the form tag specified method='POST'. Also, note that the array key is the name given the field in the HTML form with the name attribute name="fullname".



The superglobal arrays, including $_POST and $_GET, were introduced in PHP 4.1. Up until that time, form information was passed in old arrays named $HTTP_POST_VARS and $HTTP_GET_VARS. If you are using PHP 4.0 or earlier, you must use the long arrays. Both types of built-in arrays exist up until PHP 5. The long arrays no longer exist in PHP 6. If you are working with some old programs that use the long array names, you need to change the array names from the long names, such as $HTTP_POST_VARS, to the superglobal array names, such as $_POST. In most cases, a search-and-replace in a text editor will make the change with one command per array.

A program that displays all the fields in a form is a useful program for testing a form. You can see what values are passed from the form to be sure that your form is formatted properly and sends the field names and values that you expect. All the fields in a POST type form are displayed by the program in Listing 8-5, named processform.php. When the form shown in Figure 8-4 is submitted, the following program is run.

Listing 8-5: A Script That Displays All the Fields from a Form

```
<?php
/*  Script name:  processform.php
 *  Description:  Script displays all the information
 *                passed from a form.
 */
  echo "<html>
        <head><title>Customer Address</title></head>
        <body>";
  foreach ($_POST as $field => $value)
  {
     echo "$field = $value<br>";
  }
?>
</body></html>
```

If the user types the name *Goliath Smith* into the form in Figure 8-4, the following output is displayed:

```
fullname = Goliath Smith
```

The output displays only one line because there is only one field in the form in Figure 8-4.

The program in Listing 8-5 is written to process the form information from any form that uses the POST method. Suppose that you have a slightly more complicated form, such as the program in Listing 8-6, which displays a form with several fields.

Listing 8-6: Displaying a Phone Number Form

```
/*  Program name: displayForm
 *  Description:  Script displays a form that asks for the
 *                customer phone number.
 */
  echo "<html>
        <head><title>Customer Phone Number</title></head>
        <body>";
  $labels = array ( "first_name" => "First Name",
                    "middle_name" => "Middle Name",
                    "last_name" => "Last Name",
                    "phone" => "Phone");
  echo "<h3>Please enter your phone number below.</h3>";
  echo "<form action='processform.php' method='POST'>
        <table>\n";
  /* Loop that displays the form fields */
  foreach($labels as $field => $label)
  {
    echo "<tr>
          <td style='text-align: right;
                     font-weight: bold'>
      $label</td>
          <td><input type='text' name='$field' size='65'
                 maxlength='65' ></td>
          </tr>";
  }
  echo "<tr>
       <td colspan='2' style='text-align: center'>
            <input type='submit'
                   value='Submit Phone Number'>";
  echo "</td></tr></table>
        </form>";
?>
</body></html>
```

Notice the following in `displayForm.php`, as shown in Listing 8-6:

- **An array is created that contains the labels used in the form.** The keys are the field names. Setting up your fields in an array at the top of the program makes it easy to see what fields are displayed in the form and to add, remove, or modify fields.
- **The script `processform.php` is named as the script that runs when the form is submitted.** The information in the form is sent to `processform.php`, which processes the information.
- **The form is formatted with an HTML table.** Tables are an important part of HTML. If you're not familiar with HTML tables, check out *HTML 4 For Dummies,* 4th Edition, by Ed Tittel and Natanya Pitts (Wiley).
- **The script loops through the `$labels` array with a `foreach` statement.** The HTML code for a table row is output in each loop. The appropriate array values are used in the HTML code.



For security reasons, always include `maxlength` — which defines the number of characters that users are allowed to type into the field — in your HTML statement. Limiting the number of characters helps prevent the bad guys from typing malicious code into your form fields. If the information will be stored in a database, set `maxlength` to the same number as the width of the column in the database table.

When Goliath Smith fills in the form shown in Figure 8-5 (created by the program in Listing 8-6) and submits it, the program `processform.php` runs and produces the following output:

```
firstName = Goliath
midName =
lastName = Smith
phone = 555-5555
```

In `processform.php`, all elements of the `$_POST` built-in array are displayed because both of the forms shown in this section used the `POST` method, as do most forms.

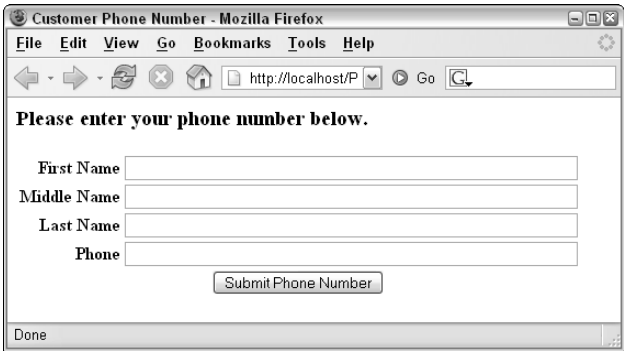


Figure 8-5: A form for entering a customer's phone number.

Making forms dynamic

PHP brings new capabilities to HTML forms. Because you can use variables in PHP forms, your forms can now be dynamic. Here are the major capabilities that PHP brings to forms:

- Using variables to display information in input text fields
- Using variables to build dynamic lists for users to select from
- Using variables to build dynamic lists of radio buttons
- Using variables to build dynamic lists of check boxes

Displaying dynamic information in form fields

When you display a form on a Web page, you can put information into the fields rather than just displaying a blank field. For example, if most of your customers live in the United States, you might automatically enter **US** in the country field when you ask customers for their address. If the customer does indeed live in the United States, you've saved the customer some typing. And if the customer doesn't live in the United States, he or she can just replace *US* with the appropriate country. Also, the text automatically entered into the field doesn't have any typos — well, unless you included some yourself.

To display a text field that contains information, you use the following format for the input field HTML statements:

```
<input type="text" name="country" value="US">
```

By using PHP, you can use a variable to display this information with either of the following statements:

```
<input type="text" name="country"
       value="<?php echo $country ?>">
echo "<input type='text' name='country'
           value='$country'>";
```

The first example creates an input field in an HTML section, using a short PHP section for the value only. The second example creates an input field by using an `echo` statement inside a PHP section. If you're using a long form with only an occasional variable, using the first format is more efficient. If your form uses many variables, it's more efficient to use the second format.

If you have user information stored in a database, you might want to display the information from the database in the form fields. For instance, you might show the information to the user so that he or she can make any needed changes. Or you might display the shipping address for the customer's last online order so that he or she doesn't need to retype the address. Listing 8-7 shows the program `displayAddress.php`, which displays a form with information from the database. This form is similar to the form shown in Figure 8-5, except that this form has information in it (retrieved from the database) and the fields in the form in Figure 8-5 are blank.

Listing 8-7: Displaying an HTML Form with Information

```
<?php
/*  Program name: displayAddress
 *  Description:  Script displays a form with address
 *                information obtained from the database.
```

```
 */
  echo "<html>
        <head><title>Customer Address</title></head>
        <body>";
  $labels = array( "firstName"=>"First Name:",
                   "lastName"=>"Last Name:",
                   "street"=>"Street Address:",
                   "city"=>"City:",
"state"=>"State:",
                   "zip"=>"Zipcode:");
  $user="admin";
  $host="localhost";
  $password="";
  $database = "MemberDirectory";
  $loginName = "gsmith";     // user login name

  $cxn = mysqli_connect($host,$user,$password,$database)
         or die ("couldn't connect to server");
  $query = "SELECT * FROM Member
                     WHERE loginName='$loginName'";
  $result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");
  $row = mysqli_fetch_assoc($result);

  echo "<div style='text-align: center'>
        <h1>Address for $loginName</h1>\n";
  echo "<p style='font-size: large; font-weight: bold'>
           Please check the information below and change
           any information that is incorrect.</p><hr />";
  echo "<form action='processAddress.php' method='POST'>
        <p><table align='center'>\n";
  foreach($labels as $field=>$label)
  {
    echo "<tr>
          <td style='text-align: right;
                     font-weight: bold'>$label</td>
          <td><input type='text' name='$field'
                 value='$row[$field]' size='65'
                 maxlength='65'>
          </td></tr>";
  }
  echo "<tr><td> </td>
         <td style='text-align: center'>
           <input type='submit' value='Submit Address'>";
  echo "</td></tr></table>
        </div>
        </form>";
?>
</body></html>
```



Registering long arrays

A `php.ini` setting, introduced in PHP 5, allows you to prevent the older, long arrays from being created automatically by PHP. It's very unlikely that you will need to use them unless you're using some old scripts containing the long variables.

The following line in `php.ini` controls this setting:

```
register_long_arrays = On
```

In PHP 5, this setting is `On` by default. Unless you're running old scripts that need the old arrays, you should change the setting to `Off` so that PHP doesn't do this extra work.

In PHP 6, the `register_long_arrays` setting is removed from `php.ini`. The long arrays no longer exist. If you're using old scripts, you must change the long array names, such as `$HTTP_POST_VARS`, to the newer global array names, such as `$_POST`.

Notice the following in the program in Listing 8-7:

- **The form statement transfers the action to the program `processAddress.php`.** This program processes the information in the form and updates the database with any information that the user changed. This is a program that you write yourself. Checking data in a form and saving information in the database are discussed later in this chapter in the sections "Checking the information" and "Putting Information into a Database," respectively.
- **Each input field in the form is given a name.** The information in the input field is stored in a variable that has the same name as the input field.
- **The program gives the field names in the form the same names as the columns in the database.** This simplifies moving information between the database and the form, requiring no transfer of information from one variable to another.
- **The values from the database are displayed in the form fields with the `value` parameter in the input field statement.** The `value` parameter displays the appropriate value from the array `$row`, which contains data from the database.



For security reasons, always include `maxlength` in your HTML statement. `maxlength` defines the number of characters that a user is allowed to type into the field. If the information is going to be stored in a database, set `maxlength` to the same number as the width of the column in the database table.

Figure 8-6 shows the Web page resulting from the program in Listing 8-7. The information in the form is the information stored in the database.

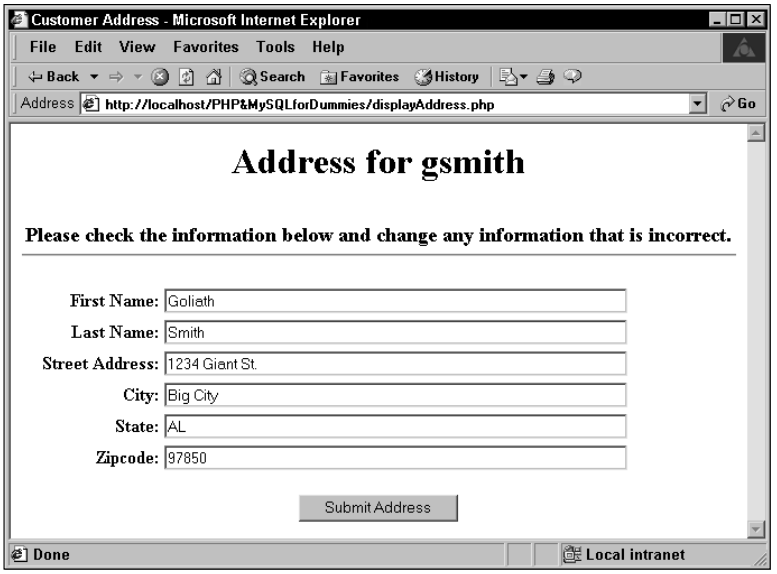


Figure 8-6: A form showing the user's address.

Building selection lists

One type of field that you can use in an HTML form is a *selection list.* Instead of typing into a field, your users select from a list. For instance, in a product catalog, you might provide a list of categories from which users select what they want to view. Or the form for users' addresses might include a list of states that users can select. Or users might enter a date by selecting a month, day, and year from a list.



Use selection lists whenever feasible. When the user selects an item from a list, you can be sure that the item is accurate, with no misspellings, odd characters, or other problems introduced by users' typing errors.

An HTML selection list for the categories in the Pet Catalog is formatted as follows:

```
<form action="processform.php" method="POST">
<select name="petType">
  <option value="horse">horse
  <option value="cat" selected>cat
  <option value="dragon">dragon
</select>    
<input type="submit" value="Select Type of Pet">
</form>;
```

Figure 8-7 shows the selection list that these HTML statements produce. Notice that *cat* is the choice that is selected when the field is first displayed. You determine this default selection by including `selected` in the option tag.

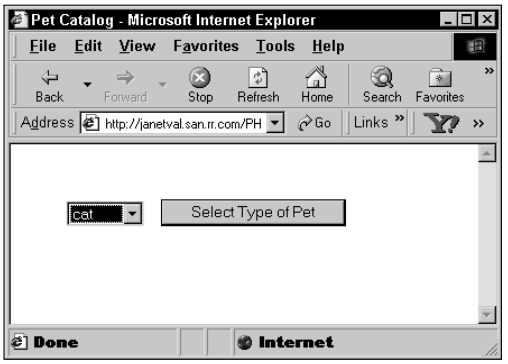


Figure 8-7: A selection field for the Pet Catalog.

When the user clicks the arrow on the select drop-down list box, the entire list drops down, as shown in Figure 8-8, and the user can select any item in the list. Notice that *cat* is selected until the user selects a different item.

When using PHP, your options can be variables. This capability allows you to build dynamic selection lists. For instance, you must maintain the static list of pet categories shown in the preceding example. If you add a new pet category, you must add an `option` tag manually. However, with PHP variables, you can build the list dynamically from the categories in the database. When you add a new category to the database, the new category is automatically added to your selection list without your having to change the PHP program. Listing 8-8 for program `buildSelect.php` builds a selection list of pet categories from the database.

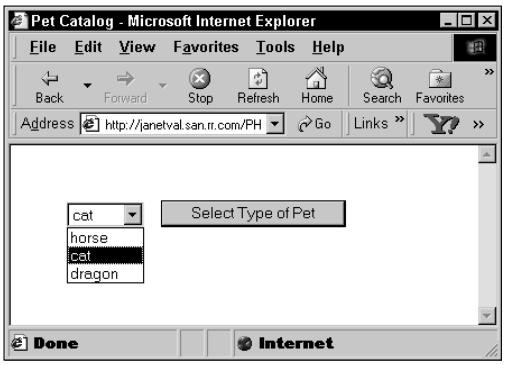


Figure 8-8: A selection field for the Pet Catalog with a drop-down list.

Listing 8-8: Building a Selection List

```
/*  Program name: buildSelect.php
 *  Description:  Program builds a selection list
 *                from the database.
 */
?>
<html>
<head><title>Pet Types</title></head>
<body>
<?php
  $user="catalog";
  $host="localhost";
  $password="";
  $database = "PetCatalog";

  $cxn = mysqli_connect($host,$user,$password,$database)
         or die ("couldn't connect to server");
  $query = "SELECT DISTINCT petType FROM Pet ORDER BY petType";
  $result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");

 /* create form containing selection list */
  echo "<form action='processform.php' method='POST'>
        <select name='petType'>\n";

  while ($row = mysqli_fetch_assoc($result))
  {
     extract($row);
     echo "<option value='$petType'>$petType\n";
  }
  echo "</select>\n";
  echo "<input type='submit' value='Select Type of Pet'>
        </form>\n";
?>
</body></html>
```

Notice the following in the program in Listing 8-8:

- **Using `DISTINCT` in the query:** `DISTINCT` causes the query to get each pet type only once. Without `DISTINCT`, the query would return each pet type several times if it appeared several times in the database.
- **Using `ORDER BY` in the query:** The pet types are sorted alphabetically.
- **`echo` statements before the loop:** The `form` and `select` tags are echoed before the `while` loop starts because they are echoed only once.
- **`echo` statements in the loop:** The `option` tags are echoed in the loop — one for each pet type in the database. No item is marked as selected, so the first item in the list is selected automatically.
- **`echo` statements after the loop:** The end `form` and `select` tags are echoed after the loop because they are echoed only once.

The selection list produced by this program is initially the same as the selection list shown in Figure 8-7, with cat selected. However, cat is selected in this program because it is the first item in the list — not because it's specifically selected as it is in the HTML tags that produce Figure 8-7. The drop-down list produced by this program is in alphabetical order, as shown in Figure 8-9.

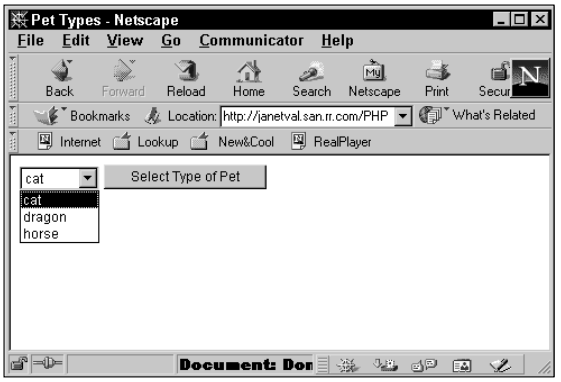


Figure 8-9: A selection field for the Pet Catalog produced by the program buildSelect.php.

You can use PHP variables also to set up which option is selected when the selection box is displayed. For instance, suppose that you want the user to select a date from month, day, and year selection lists. You believe that most people will select today's date, so you want today's date to be selected by default when the box is displayed. Listing 8-9 shows the program `dateSelect.php`, which displays a form for selecting a date and selects today's date automatically.

Listing 8-9: Building a Date Selection List

```
/*  Program name: dateSelect.php
 *  Description:  Program displays a selection list that
 *                customers can use to select a date.
 */
 echo "<html>
       <head><title>Select a date</title></head>
       <body>";

 $monthName = array(1=> "January", "February", "March",
                        "April", "May", "June", "July",
                        "August", "September", "October",
                        "November", "December");

 $today = time();                    //stores today's date
 $f_today = date("M-d-Y",$today);    //formats today's date

 echo "<div style = 'text-align: center'>\n";
```

```
 /* display today's date */
 echo "<h3>Today is $f_today</h3><hr>\n";

 /* create form containing date selection list */
 echo "<form action='processform.php' method='POST'>\n";

 /* build selection list for the month */
 $todayMO = date("n",$today);  //get the month from $today
 echo "<select name='dateMO'>\n";
 for ($n=1;$n<=12;$n++)
 {
   echo "<option value=$n\n";
   if ($todayMO == $n)
   {
     echo " selected";
   }
   echo "> $monthName[$n]\n";
 }
 echo "</select>";

 /* build selection list for the day */
 $todayDay= date("d",$today);    //get the day from $today
 echo "<select name='dateDay'>\n";
 for ($n=1;$n<=31;$n++)
 {
   echo " <option value=$n";
   if ($todayDay == $n )
   {
     echo " selected";
   }
   echo "> $n\n";
 }
 echo "</select>\n";

/* build selection list for the year */
 $startYr = date("Y", $today);  //get the year from $today
 echo "<select name='dateYr'>\n";
 for ($n=$startYr;$n<=$startYr+3;$n++)
 {
   echo " <option value=$n";
   if ($startYr == $n )
   {
     echo " selected";
   }
   echo "> $n\n";
 }
 echo "</select>\n";

 echo "</form>\n";
?>
</body></html>
```

The Web page produced by the program in Listing 8-9 is shown in Figure 8-10. The date appears above the form so that you can see that the select list shows the correct date. The selection list for the month shows all 12 months when it drops down. The selection list for the day shows 31 days when it drops down. The selection list for year shows four years.

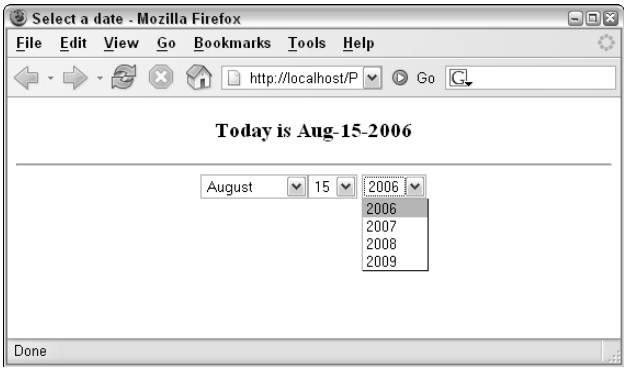


Figure 8-10: A selection field for the date with today's date selected.

The program in Listing 8-9 produces the Web page in Figure 8-10 by following these steps:

1. **Creates an array containing the names of the months.** The keys for the array are the numbers. The first month, January, starts with the key `1` so that the keys of the array match the numbers of the months.
2. **Creates variables containing the current date.** `$today` contains the date in a system format and is used in the form. `$f-today` is a formatted date that is used to display the date in the Web page.
3. **Displays the current date at the top of the Web page.**
4. **Builds the selection field for the month:**
    i. Creates a variable containing today's month.
    ii. Echoes the `select` tag, which should be echoed only once.
    iii. Starts a `for` loop that repeats 12 times.
    iv. Inside the loop, echoes the `option` tag by using the first value from the `$monthName` array.
    v. If the number of the month being processed is equal to the number of the current month, adds the word `"selected"` to the `option` tag.
    vi. Repeats the loop 11 more times.
    vii. Echoes the closing `select` tag for the selection field, which should be echoed only once.

5. **Builds the selection field for the day.** Uses the procedure described in Step 4 for the month. However, only numbers are used for this selection list. The loop repeats 31 times.
6. **Builds the selection field for the year:**
   i. Creates the variable `$startYr`, containing today's year.
   ii. Echoes the `select` tag, which should be echoed only once.
   iii. Starts a `for` loop. The starting value for the loop is `$startYr`. The ending value for the loop is `$startYr+3`.
   iv. Inside the loop, echoes the `option` tag, using the starting value of the `for` loop, which is today's year.
   v. If the number of the year being processed is equal to the number of the current year, adds the word `"selected"` to the option tag.
   vi. Repeats the loop until the ending value equals `$startYr+3`.
   vii. Echoes the closing `select` tag for the selection field, which should be echoed only once.
7. **Echoes the ending tag for the form.**

Building lists of radio buttons

You might want to use radio buttons instead of selection lists. For instance, you can display a list of radio buttons for your Pet Catalog and have users select the button for the pet category that they're interested in.

The format for radio buttons in a form is

```
<input type="radio" name="name" value="value">
```

You can build a dynamic list of radio buttons representing all the pet types in your database in the same manner that you build a dynamic selection list in the preceding section. Listing 8-10 shows the program `buildRadio.php`, which creates a list of radio buttons based on pet types.

Listing 8-10: Building a List of Radio Buttons

```
/*  Program name: buildRadio.php
 *  Description:  Program displays a list of radio
 *                buttons from database info.
 */
 echo "<html>
       <head><title>Pet Types</title></head>
       <body>";
 $user="catalog";
 $host="localhost";
 $password="";
```

(continued)

Listing 8-10 *(continued)*

```
$database = "PetCatalog";

$cxn = mysqli_connect($host,$user,$password,$database)
       or die ("Couldn't connect to server");
$query = "SELECT DISTINCT petType FROM Pet
                  ORDER BY petType";
$result = mysqli_query($cxn,$query)
          or die ("Couldn't execute query.");

echo "<div style='margin-left: .5in; margin-top: .5in'>
      <p style='font-weight: bold'>
       Which type of pet are you interested in?</p>
      <p>Please choose one type of pet from the
         following list:</p>\n";

/* create form containing radio buttons */
echo "<form action='processform.php' method='POST'>\n";

while ($row = mysqli_fetch_assoc($result))
{
   extract($row);
   echo "<input type='radio' name='interest'
                value='$petType'>$petType\n";
   echo "<br>\n";
}
echo "<p><input type='submit' value='Select Type of Pet'>
     </form>\n";
?>
</div></body></html>
```

This program is similar to the program in Listing 8-9. The Web page produced by this program is shown in Figure 8-11.

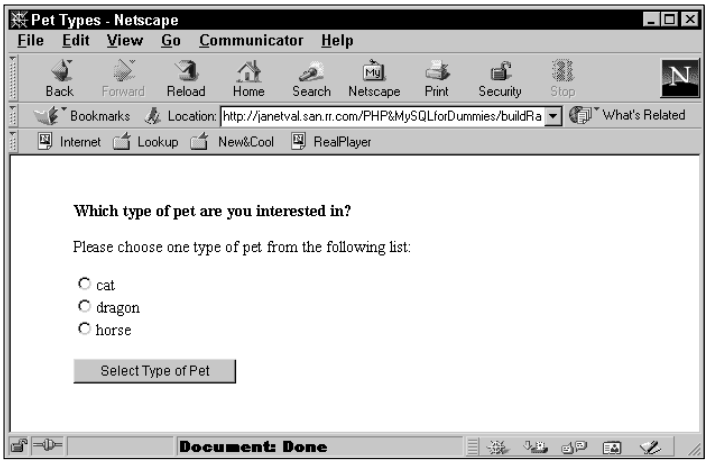


Figure 8-11: List of radio buttons produced by the program in buildRadio.php.

Building lists of check boxes

You might want to use check boxes in your form. Check boxes are different from selection lists and radio buttons because they allow users to select more than one option. For instance, if you display a list of pet categories by using check boxes, a user can select two or three or more pet categories. The program `buildCheckbox.php` in Listing 8-11 creates a list of check boxes.

Listing 8-11: Building a List of Check Boxes

```
<?php
/*  Program name: buildCheckbox.php
 *  Description:  Program displays a list of
 *                check boxes from database info.
 */
  echo "<html>
        <head><title>Pet Types</title></head>
        <body>";
  $user="catalog";
  $host="localhost";
  $password="";
  $database = "PetCatalog";

  $cxn = mysqli_connect($host,$user,$password,$database)
         or die ("couldn't connect to server");
  $query = "SELECT DISTINCT petType FROM Pet
                   ORDER BY petType";
  $result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");

  echo "<div style='margin-left: .5in; margin-top: .5in'>
      <p style='font-weight: bold'>
         Which type of pet are you interested in?</p>
<p>Choose as many types of pets as you want:</p>\n";

  /* create form containing checkboxes */
  echo "<form action='processform.php' method='POST'>\n";

  while ($row = mysqli_fetch_assoc($result))
  {
     extract($row);
     echo "<input type='checkbox'
                  name='interest[$petType]'
                  value='$petType'>$petType\n";
     echo "<br>\n";
  }
  echo "<p><input type='submit'
                  value='Select Type of Pet'>
        </form>\n";
?>
</div></body></html>
```

This program is similar to the program in Listing 8-10, which builds a list of radio buttons. However, notice that the input field uses an array `$interest` as the name for the field. This is because more than one check box can be selected. This program will create an element in the array with a key/value pair for each check box that's selected. For instance, if the user selects both *horse* and *dragon,* the following array is created:

```
$interest[horse]=horse
$interest[dragon]=dragon
```

The program that processes the form has the selections available in the `POST` array, as follows:

```
$_POST['interest']['horse']
$_POST['interest']['dragon']
```

Figure 8-12 shows the Web page produced by `buildCheckbox.php`.

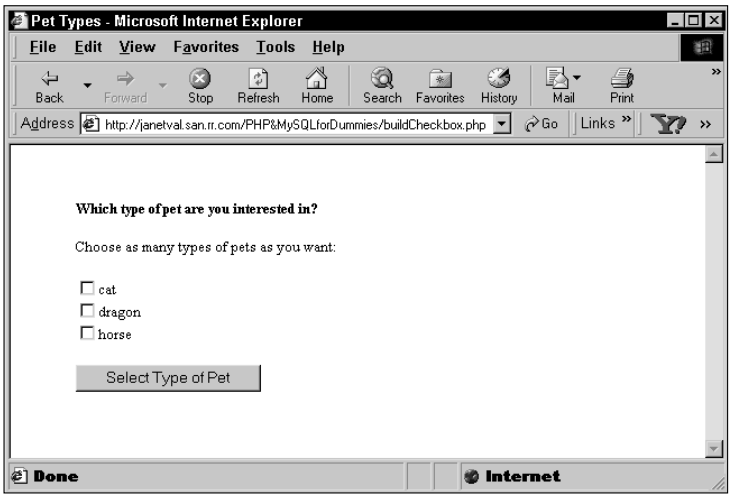


Figure 8-12: A list of check boxes produced by the program in buildCheck box.php.

Using the information from the form

As I discuss earlier in this section, Joe Customer fills in an HTML form, selecting from lists and typing information into text fields. He clicks the submit button. In the `form` tag, you tell PHP which program to run when the submit button is clicked. You do this by including `action="programname"` in the `form` tag. For instance, in most of the example listings in this chapter, I use `action="processform.php"`. When the user clicks the submit button, the

program runs and receives the information from the form. Handling form information is one of PHP's best features. You don't need to worry about the form data — just get it from one of the built-in arrays and use it.

The form data is available in the processing program in arrays, such as `$_POST` or `$_GET`. The key for the array element is the name of the input field in the form. For instance, if you echo the following field in your form

```
echo "<input type='text' name='firstName'>";
```

the processing program can use the variable `$_POST[firstName]`, which contains the text that the user typed into the field. The information that the user selects from selection drop-down lists or radio buttons is similarly available for use. For instance, if your form includes the following list of radio buttons

```
echo "<input type='radio' name='interest'
          value='dog'>dog\n";
echo "<input type='radio' name='interest'
          value='cat'>cat\n";
```

you can access the variable `$_POST[interest]`, which contains either `dog` or `cat`, depending on what the user selected.

You handle check boxes in a slightly different way because the user can select more than one check box. As shown in Listing 8-11, the data from a list of check boxes can be stored in an array so that all the check boxes are available. For instance, if your form includes the following list of check boxes

```
echo "<input type='checkbox' name='interest[dog]'
             value='dog'>dog\n";
echo "<input type='checkbox' name='interest[cat]'
             value='cat'>cat\n";
```

you can access the data by using the multidimensional variable `$_POST [interest]`, which contains the following:

```
$_POST[interest][dog] = dog
$_POST[interest][cat] = cat
```

In some cases, you might want to access all the fields in the form. Perhaps you want to check them all to make sure that the user didn't leave any fields blank. As shown in the program `processform.php`, earlier in this chapter (see Listing 8-5), you can use `foreach` to walk through the `$_POST` or `$_GET` built-in array. Most of the sample programs and statements in this book use the `POST` method. The keys are the field names. See the sidebar "Post versus get" for more on the two methods.

For instance, suppose your program includes the following statements to display a form:

```
echo "<form action='processform.php' method='POST'>\n";
echo "<input type='text' name='lname'
            value='Smith'><br>\n";
echo "<input type='radio' name='interest'
            value='dog'>dog\n";
echo "<input type='radio' name='interest'
            value='cat'>cat\n";
echo "<input type='hidden' name='hidvar' value='3'>\n";
echo "<br><input type='submit' value='Select Type of Pet'>
     </form>\n";
```

The program `processform.php` contains the following statements, which will list all the variables received from the form:

```
foreach($_POST as $field => $value)
{
   echo "$field, $value<br>";
}
```

The output from the `foreach` loop is

```
lname, Smith
interest, dog
hidvar, 3
```

The output shows three variables with these three values for the following reasons:

- **The user didn't change the text in the text field.** The value `"Smith"` that the program displayed is still the text in the text field.
- **The user selected the radio button for dog.** The user can select only one radio button.
- **The program passed a hidden field named `hidvar`.** The program sets the value for hidden fields. The user can't affect the hidden fields.

Checking the information

Joe Customer fills in an HTML form, selecting from lists and typing information into text fields. He clicks the submit button. You now have all the information that you wanted. Well, maybe. Joe might have typed information that contains a typo. Or he might have typed nonsense. Or he might even have typed malicious information that can cause problems for you or other people using your Web site. Before you use Joe's information or store it in your database, you want to check it to make sure it's the information you asked for. Checking the data is *validating* the data.

Validating the data includes the following:

- **Checking for empty fields:** You can require users to enter information in a field. If the field is blank, the user is told that the information is required, and the form is displayed again so the user can type the missing information.
- **Checking the format of the information:** You can check the information to see that it is in the correct format. For instance, *ab3&*xx* is clearly not a valid zip code.

Checking for empty fields

When you create a form, you can decide which fields are required and which are optional. Your decision is implemented in the PHP program. You check the fields that require information. If a required field is blank, you send a message to the user, indicating that the field is required, and you then redisplay the form.

The general procedure to check for empty fields is

```
if ($last_name == "")
{
    echo "You did not enter your last name.
          Last name is required.<br>\n";
    display the form;
    exit();
}
echo " Welcome to the Members Only club.
       You may select from the menu below.<br>\n";
display the menu;
```

Notice the `exit` statement, which ends the program. Without the `exit` statement, the program would continue to the statements after the `if` statement. In other words, without the `exit` statement, the program would display the form and then continue to echo the welcome statement and the menu as well.

In many cases, you want to check all the fields in the form. You can do this by looping through the array `$ _POST`. The following statements check the array for any empty fields:

```
foreach($_POST as $value)
{
   if ( $value == "" )
   {
     echo "You have not filled in all the fields<br>\n";
     display the form;
     exit();
   }
}
echo "Welcome";
```



Post versus get

You use one of two methods to submit form information. The methods pass the form data differently and have different advantages and disadvantages.

- **`get` method:** The form data is passed by adding it to the URL that calls the form-processing program. For instance, the URL might look like this:

  ```
  processform.php?lname=Smith
          &fname=Goliath
  ```

  The advantages of this method are simplicity and speed. The disadvantages are that less data can be passed and that the information is displayed in the browser, which can be a security problem in some situations.

- **`post` method:** The form data is passed as a package in a separate communication with the processing program. The advantages of this method are unlimited information passing and security of the data. The disadvantages are the additional overhead and slower speed.

For CGI programs that are not PHP, the program that processes the form must find the information and put the data into variables. In this case, the `get` method is much simpler and easier to use. Many programmers use the `get` method for this reason. However, PHP does all this work for you. The `get` and `post` methods are equally easy to use in PHP programs. Therefore, when using PHP, it's almost always better to use the `post` method because you have the advantages of the `post` method (unlimited data passing, better security) without its main disadvantage (more difficult to use).



When you redisplay the Web form, make sure that it contains the information that the user already typed. If users have to retype correct information, they are likely to get frustrated and leave your Web site.

In some cases, you might require the user to fill in most but not all fields. For instance, you might request a fax number in the form or provide a field for a middle name, but you don't really mean to restrict registration on your Web site to users with middle names and faxes. In this case, you can make an exception for fields that are not required, as follows:

```
foreach($_POST as $field => $value)
{
  if( $field != "fax" and $field != "middle_name" )
  {
     if( $value == "" )
     {
       echo "You have not filled in all the fields<br>\n";
       display the form;
       exit();
     }
  }
}
echo "Welcome";
```

Notice that the outside `if` conditional statement is true only if the field is not the fax field and is not the middle name field. For those two fields, the program does not reach the inside `if` statement, which checks for blank fields.

In most cases, the program should create two arrays: one that contains the names of the fields that are inappropriately blank and one that contains the data that is correct, so you can display or store it. The need for an array of correct data becomes clearer later in this section, when I discuss checking the format of data and cleaning data.

In some cases, you might want to tell the user exactly which fields need to be filled in. The `checkBlank.php` program in Listing 8-12 processes the form produced by the program `displayForm`, shown in Listing 8-6, which has four fields: `first_name`, `middle_name`, `last_name`, and `phone`. All the fields are required except `middle_name`.

To use the program in Listing 8-12, first edit the `displayForm` program, shown in Listing 8-6, so that `checkBlank.php` is shown in the `action` attribute in the `form` tag. Replace `processform.php` with `checkBlank.php`, as follows:

```
echo "<form action='checkBlank.php' method='POST'>
```

Then run the `displayForm.php` program, fill in the form, and click the submit button, which runs the `checkBlank.php` program.

Listing 8-12: Checking for Blank Fields

```
<?php
/*  Program name: checkBlank.php
 *  Description:  Program checks all the form fields for
 *                blank fields.
 */
?>
<html>
<head><title>Empty fields</title></head>
<body>
<?php
  /* set up array with all the fields */
  $labels = array( "first_name" => "First Name",
                   "middle_name" => "Middle Name",
                   "last_name" => "Last Name",
                   "phone" => "Phone");
  /* check each field except middle name for
     blank fields */
  foreach($_POST as $field => $value)
  {
    if($field != "middle_name")
    {
      if( $value == "" )
      {
```

(continued)

Listing 8-12 ***(continued)***

```
          $blank_array[] = $field;
        }
      }
    }  // end of foreach loop for $_POST
    /* if any fields were blank, display error message and
       redisplay form */
    if(@sizeof($blank_array) > 0) //blank fields are found
    {
      echo "<b>You didn't fill in one or more required
              fields. You must enter:</b><br>";
      /* display list of missing information */
      foreach($blank_array as $value)
      {
        echo "   {$labels[$value]}<br>";
      }
      /* redisplay form */
      echo "<p>";
      echo "<form action='$_SERVER[PHP_SELF]' method='POST'>
            <table>";
      foreach($labels as $field => $label)
      {
        $good_data[$field]=strip_tags(trim($_POST[$field]));
        echo "<tr>
               <td style='text-align: right;
                   font-weight: bold'>
                   {$labels[$field]}</td>
               <td><input type='text' name='$field'
                          size='65' maxlength='65'
                          value='$good_data[$field]'></td>
             </tr>";
      }
      echo "<tr>
             <td colspan='2' style='text-align: center'>
               <input type='submit'
                      value='Submit Name and Phone Number'>";
      echo "</td></tr></table>
            </form>";
      exit();
    }
    echo "All required fields contain information";
?>
</body></html>
```

To check for blanks, the program does the following:

1. **Sets up an array of field labels.** These labels are used as labels in the form, to display the list of missing information and to display the form.
2. **Loops through all the variables passed from the form, checking for blanks.** The variables are in the array `$ _POST`. The field `middle_name` is not checked for blanks because it is not a required field. Any blank fields are added to an array of blank fields, `$blank_array`.

3. **Checks whether any blank fields were found.** Checks the number of items in `$blank_array`.
4. **If zero blank fields were found, jumps to the message; all required fields contain information.**
5. **If one or more blank fields were found:**
   i. Displays an error message. This message explains to the user that some required information is missing.
   ii. Displays a list of missing information. Loops through `$blank_array` and displays the label(s).
   iii Creates an array of good data. The data is cleaned so it can be safely displayed in the form.
   iv. Redisplays the form. Because the form includes variable names in the `value` attribute, the information that the user previously entered is retrieved from `$good_data` and displayed.
   v. Exits. Stops after the form displays. The user must click the submit button to continue.



Remember, programs that process forms use the information from the form. If you run them by themselves, they don't have any information passed from the form and will not run correctly. These programs are intended to run when the user clicks the submit button for a form.

Don't forget the `exit` statement. Without the `exit` statement, the program would continue and would display the welcome message after displaying the form.

Figure 8-13 shows the Web page that results if the user didn't enter a first or a middle name. Notice that the list of missing information doesn't include Middle Name because Middle Name is not required. Also, notice that the information the user originally typed into the form is still displayed in the form fields.

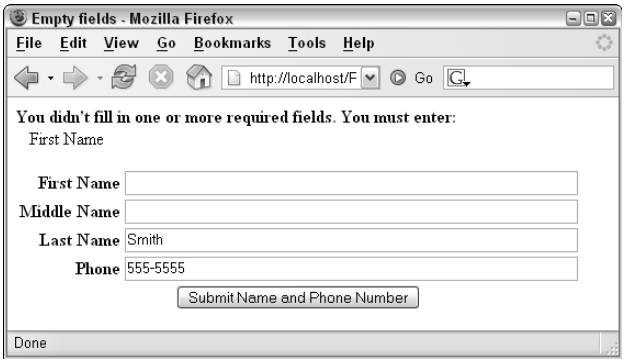


Figure 8-13: The result of processing a form with missing information.

Checking the format of the information

Whenever users must type information in a form, you can expect a certain number of typos. You can detect some of these errors when the form is submitted, point out the error(s) to the user, and then request that he or she retype the information. For instance, if the user types **8899776** in the zip code field, you know this is not correct. This information is too long to be a zip code and too short to be a zip+4 code.



You also need to protect yourself from malicious users — users who might want to damage your Web site or your database or steal information from you or your users. You don't want users to enter HTML tags into a form field — something that might have unexpected results when sent to a browser. A particularly dangerous tag would be a script tag that allows a user to enter a program into a form field.

If you check each field for its expected format, you can catch typos and prevent most malicious content. However, checking information is a balancing act. You want to catch as much incorrect data as possible, but you don't want to block any legitimate information. For instance, when you check a phone number, you might limit it to numbers. The problem with this check is that it would screen out legitimate phone numbers in the form 555-5555 or (888) 555-5555. So you also need to allow hyphens (-), parentheses ( ), and spaces. You might limit the field to a length of 14 characters, including parentheses, spaces, and hyphens, but this screens out overseas numbers or numbers that include an extension. The bottom line: You need to think carefully about what information you want to accept or screen out for any field.

You can check field information by using *regular expressions,* which are patterns. You compare the information in the field against the pattern to see whether it matches. If it doesn't match, the information in the field is incorrect, and the user must type it over. (See Chapter 6 for more on regular expressions.)

In general, these are the statements that you use to check fields:

```
if( !ereg("pattern",$variablename) )
{
    echo error message;
    redisplay form;
    exit();
}
echo "Welcome";
```

Notice that the condition in the `if` statement is negative. That is, the `!` (exclamation mark) means `"not"`. So, the `if` statement actually says: If the variable does *not* match the pattern, execute the `if` block.

For example, suppose that you want to check an input field that contains the user's last name. You can expect names to contain letters, not numbers, and possibly apostrophe and hyphen characters (as in *O'Hara* and *Smith-Jones*) and also spaces (as in *Van Dyke*). Also, it's difficult to imagine a name longer than 50 characters. Thus, you can use the following statements to check a name:

```
if( !ereg("[A-Za-z' -]{1,50}",$last_name)
{
    echo error message;
    redisplay form;
    exit();
}
echo "Welcome";
```



If you want to list a hyphen (-) as part of a set of allowable characters that are surrounded by square brackets (`[ ]`), you must list the hyphen at the beginning or at the end of the list. Otherwise, if you put it between two characters, the program will interpret it as the range between the two characters, such as *A–Z*.

You also need to check multiple-choice fields. Although multiple choice prevents honest users from entering mistakes, it doesn't prevent clever users with malicious intentions from entering unexpected data into the fields. You can check multiple-choice fields for acceptable output with the following type of `regex`:

```
if( !ereg("(male|female)",$gender)
```

If the field contains anything except the value `male` or the value `female`, the `if` block executes.

In the preceding section, you find out how to check every form field to ensure that it isn't blank. In addition to that, you will probably also want to check all the fields that have data to be sure the data is in an acceptable format. You can check the format by making a few simple changes to the program in Listing 8-12. Listing 8-13 shows the modified program, called `checkAll.php`.

The program in Listing 8-13, like the program in Listing 8-12, processes data submitted from the form produced by the `displayForm` program in Listing 8-6. To use the program in Listing 8-13, first edit the `displayForm` program, shown in Listing 8-6, so that `checkAll.php` is shown in the `action` attribute in the `form` tag. Replace `processform.php` with `checkAll.php`, as follows:

```
echo "<form action='checkAll.php' method='POST'>
```

Then run the `displayForm.php` program, fill in the form, and click the submit button, which runs the `checkAll.php` program.

Listing 8-13: Checking All the Data in Form Fields

```
<?php
/*  Program name: checkAll.php
 *  Description:  Program checks all the form fields for
 *                blank fields and incorrect format.
 */
?>
<html>
<head><title>Check fields</title></head>
<body>
<?php
  /* set up array containing all the fields */
  $labels = array ( "first_name" => "First Name",
                    "middle_name" => "Middle Name",
                    "last_name" => "Last Name",
                    "phone" => "Phone");
  foreach ($_POST as $field => $value)
  {
    /* check each field except middle name for blank fields */
    if ( $value == "" )
    {
       if ($field != "middle_name")
       {
          $blank_array[] = $field;
       }
    }
    /* check names for invalid formats. */
    elseif ($field == "first_name" or $field == "middle_name"
            or $field == "last_name" )
    {
        if (!ereg("^[A-Za-z' -]{1,50}$",$_POST[$field]) )
        {
             $bad_format[] = $field;
        }
    }
    /* check phone for invalid format. */
    elseif ($field == "phone")
    {
      if(!ereg("^[0-9)( -]{7,20}(([xX]|(ext)|(ex))?[ -]?[0-9]{1,7})?$",$value))
      {
           $bad_format[] = $field;
      }
    }
  }
  /* if any fields are not okay, display error message and form */
  if(@sizeof($blank_array) > 0 or @sizeof($bad_format) > 0)
  {
    if(@sizeof($blank_array) > 0)
    {
        /* display message for missing information */
        echo "<b>You didn't fill in one or more required
                 fields. You must enter:</b><br>";
        /* display list of missing information */
        foreach($blank_array as $value)
```

```
        {
           echo "   {$labels[$value]}<br>";
        }
    }
    if(@sizeof($bad_format) > 0)
    {
        /* display message for bad information */
        echo "<b>One or more fields have information that appears to be
                 incorrect. Correct the format for:</b><br>";
        /* display list of bad information */
        foreach($bad_format as $value)
        {
           echo "   {$labels[$value]}<br>";
        }
    }
    /* redisplay form */
    echo "<p>";
    echo "<form action='$_SERVER[PHP_SELF]' method='POST'>
          <table>";
    foreach($labels as $field => $label)
    {
      $good_data[$field]=strip_tags(trim($_POST[$field]));
      echo "<tr>
             <td style='text-align: right; font-weight: bold'>
                {$labels[$field]}</td>
             <td><input type='text' name='$field' size='65'
                 maxlength='65' value='$good_data[$field]'></td>
            </tr>";
    }
    echo "<tr>
           <td colspan='2' style='text-align: center'>
             <input type='submit' value='Submit Name and Phone Number'>";
    echo "</td></tr></table>
</form>";
    exit();
  }
  /* if data is good */
  echo "All data is good";
?>
</body></html>
```

Here are the differences between this program and the program in Listing 8-12:

- **This program creates two arrays for problem data.** It creates `$blank_array`, as did the previous program. But this program also creates `$bad_format` for fields that contain information that is not in an acceptable format.
- **This program loops through `$bad_format` to create a separate list of problem data.** If any fields are blank, it creates one error message and a list of problem fields, as did the previous program. If any fields are in an unacceptable format, this program also creates a second error message and a list of problem fields.

The Web page in Figure 8-14 results when the user accidentally types his or her first name into the Middle Name field and also types nonsense for his or her phone number. Notice that two error messages appear, showing that the First Name field is blank and that the Phone field contains incorrect information.

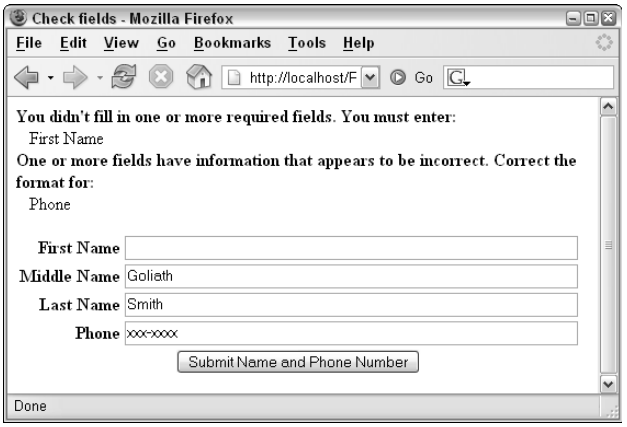


Figure 8-14: The result of processing a form with both missing and incorrect information.

Giving users a choice with multiple submit buttons

You can use more than one submit button in a form. For instance, in a customer order form, you might use a button that reads *Submit Order* and another button that reads *Cancel Order*. However, you can list only one program in the `action=programname` part of your `form` tag, meaning that the two buttons run the same program. PHP solves this problem. By using PHP, you can process the form differently, depending on which button the user clicks. The program in Listing 8-14 displays a form with two buttons.

Listing 8-14: Displaying a Form with Two Submit Buttons

```
<?php
/*  Program name: displayTwoButtons.php
 *  Description:  Program displays a form with two
 *                buttons.
 */
?>
<html>
<head><title>Two Buttons</title></head>
<body>
<?php
echo "<form action='processTwoButtons.php' method='POST'>
          Last Name: <input type='text' name='last_name'
                            maxlength='50'><br>
```

```
          <input type='submit' name='display_button'
                 value='Show Address'>
          <input type='submit' name='display_button'
                 value='Show Phone Number'>
      </form>";
?>
</body></html>
```

Notice that the submit button fields have a name: `display_button`. The fields each have a different value. Whichever button the user clicks sets the value for `$display_button`. The program `processTwoButtons.php` in Listing 8-15 processes the preceding form.

Listing 8-15: Processing Two Submit Buttons

```
<?php
/*  Program name: processTwoButtons.php
 *  Description:  Program displays different information
 *                depending on which submit button was
 *                pushed.
 */
?>
<html>
<head><title>Member Address or Phone Number</title></head>
<body>
<?php
  $user="admin";
  $host="localhost";
  $password="";
  $database = "MemberDirectory";
  $cxn = mysqli_connect($host,$user,$password,$database)
         or die ("Couldn't connect to server");
  if ($_POST['display_button'] == "Show Address")
  {
     $query = "SELECT street,city,state,zip FROM Member
                 WHERE lastName='$_POST[last_name]'";
$result = mysqli_query($cxn,$query)
               or die ("Couldn't execute query.");
     $row = mysqli_fetch_assoc($result);
     extract($row);
     echo "$street<br>$city, $state  $zip<br>";
  }
  else
  {
     $query = "SELECT phone FROM Member
                   WHERE lastName='$_POST[last_name]'";
     $result = mysqli_query($cxn,$query)
               or die ("Couldn't execute query.");
     $row = mysqli_fetch_assoc($result);
     echo "Phone: {$row['phone']}";
  }
?>
</body></html>
```

The program executes different statements, depending on which button is clicked. If the user clicks the button for the address, the program outputs the address for the name submitted in the form; if the user clicks the Show Phone Number button, the program outputs the phone number.

Putting Information into a Database

Your application probably needs to store data in your database. For example, your database might store information that a user typed into a form for your use — a Member Directory is an example of this. Or your database might store data temporarily during the application. Either way, you store data by sending SQL queries to MySQL. (I explain SQL queries in detail in Chapter 4.)

Preparing the data

You need to prepare the data before storing it in the database. Preparing the data includes the following:

- Putting the data into variables
- Making sure that the data is in the format expected by the database
- Cleaning the data
- Escaping the data

Putting the data into variables

You store the data by sending it to the database in an SQL query. You add the data to the query by including the variable names in the query. Most of the data that you want to store is typed by the user into a form. As I discuss earlier in this chapter, PHP stores the form data in a built-in array, with the name of the form field as the array key. You just use the PHP built-in array elements in the query. Occasionally, you'll want to store information that you generate yourself, such as today's date or a customer order number. You just need to assign this data to a variable so that you can include it in a query.

Using the correct format

When you design your database, you set the data type for each column. The data that you want to store must match the data type of the column that you want to store it in. For instance, if the column expects a data type integer, the data sent must be numbers. Or if the column expects data that's a date, the data that you send must be in a format that MySQL recognizes as a date. If you send incorrectly formatted data, MySQL still stores the data, but it might not store the value that you expected. Here's a rundown of how MySQL stores data for the most frequently used data types:

- CHAR or VARCHAR: Stores strings. MySQL stores pretty much any data sent to a character column, including numbers and dates, as strings. When you created the column, you specified a length. For example, if you specified CHAR(20), only 20 characters can be stored. If you send a string longer than 20 characters, only the first 20 characters are stored. The remaining characters are dropped.

  Set the maxlength for any text input fields in a form to the same length as the column width in the database where the data will be stored. That way, the user can't enter any more characters than the database can store.

- INT or DECIMAL: Stores numbers. MySQL will try to interpret any data sent to a number column as a number, whether it makes sense or not. For instance, it might interpret a date as a number, and you could end up with a number like 2001.00. If MySQL is unable to interpret the data sent as a number, it stores 0 (zero) in the column.

- DATE: Stores dates. MySQL expects dates as numbers, with the year first, month second, and day last. The year can be two or four digits (2006 or 06). The date can be a string of numbers, or each part can be separated by a hyphen (-), a period (.), or a forward slash (/). Some valid date formats are 20061203, 980103, 2006-3-2, and 2000.10.01. If MySQL cannot interpret the data sent as a date, it stores the date as 0000-00-00.

- ENUM: Stores only the values that you allowed when you created the column. If you send data that is not allowed, MySQL stores a 0.

In many cases, the data is collected in a form and stored in the database as is. For instance, users type their names in a form, and the program stores them. However, in some cases, the data needs to be changed before you store it. For instance, if a user enters a date into a form in three separate selection lists for month, day, and year (as I describe in the section, "Building selection lists," earlier in this chapter), the values in the three fields must be put together into one variable. The following statements put the fields together:

```
$expDate = $_POST['expYear']."-";
$expDate .= $_POST['expMonth']."-";
$expDate .= $_POST['expDay'];
```

Another case in which you might want to change the data before storing it is when you're storing phone numbers. Users enter phone numbers in a variety of formats, using parentheses, dashes, dots, or spaces. Rather than storing these varied formats in your database, you might just store the numbers. Then when you retrieve a phone number from the database, you can format the number however you want before you display it. The following statement removes characters from the string:

```
$phone = ereg_replace("[ )(.-]","",$_POST['phone']);
```

The function `ereg_replace` uses regular expressions to search for a pattern. The first string passed is the regular expression to match. If any part of the string matches the pattern, it is replaced by the second string. In this case, the regular expression is `[ )(.-]`, which means any one of the characters in the square brackets. The second string is `""`, which is a string with nothing in it. Therefore, any spaces, parentheses, dots, or hyphens in the string (characters that you might consider valid and allow when checking the data) are replaced by nothing.

Cleaning the data

The earlier section "Getting Information from the User," which describes the use of HTML forms, discusses checking the data in forms. Users can type data into a text field, either accidentally or maliciously, that can cause problems for your application, your database, or your users. Checking the data and accepting only the characters expected for the information requested can prevent many problems. However, you can miss something. Also, in some cases, the information that the user enters needs to allow pretty much anything. For instance, you normally wouldn't allow the characters < and > in a field. However, there might be a situation in which the user needs to enter these characters — perhaps the user needs to enter a technical formula or specification that requires them.

PHP has two functions that can clean the data, thus rendering it harmless:

- `strip_tags`: This function removes all text enclosed by < and > from the data. It looks for an opening < and removes it and everything else, until it finds a closing > or reaches the end of the string. You can include specific tags that you want to allow. For instance, the following statement removes all tags from a character string except `<b>` and `<i>`:

  ```
  $last_name = strip_tags($last_name,"<b><i>");
  ```

- `htmlspecialchars`: This function changes some special characters with meaning to HTML into an HTML format that allows them to be displayed without any special meaning. The changes are

  - < becomes `<`
  - > becomes `>`
  - & becomes `&`

  In this way, the characters < and > can be displayed on a Web page without HTML interpreting them as tags. The following statement changes these special characters:

  ```
  $last_name = htmlspecialchars($last_name);
  ```

If you're positive that you don't want to allow your users to type any < or > characters into a form field, use `strip_tags`. However, if you want to allow < or > characters, you can safely store them after they have been processed by `htmlspecialchars`.

Another function that you should use before storing data in your database is `trim`. Users often type spaces at the beginning or end of a text field without meaning to. `Trim` removes any leading or trailing spaces so they don't get stored. Use the following statement to remove these spaces:

```
$last_name = trim($_POST['last_name']);
```

Escaping the data

A user can type information into your form that, when used in your query, changes your query so that it operates differently than you expect. Some of these damaging queries are created by manipulating the quotes in your query. You can protect against this kind of attack, called an *SQL injection,* by escaping any quotes sent in form fields. *Escaping* special characters, such as quotes, means to place a backslash (`\`) in front of the character. The special character is then treated as any other character, not as a special character with special meaning, rendering the query safe. Escaping characters is discussed in Chapter 6.

PHP versions before version 6 provide a feature called *magic quotes* that automatically escapes all strings in the `$_POST` and `$_GET` arrays. Single quotes, double quotes, backslashes, and null characters are escaped. This feature, designed to help beginning users, is controlled by the `magic_quotes-gpc` setting in `php.ini` and is turned on by default in PHP 4 and PHP 5. In PHP 6, the magic quotes feature is no longer available.

The magic quotes feature is convenient and protects beginning users from SQL injection attacks that they may be unaware of. However, all `$_POST` and `$_GET` data is escaped, even if it is not going to be stored in a database. This unnecessary escaping is inefficient. In addition, if you just display the form data or use it in an e-mail, the backslashes in front of the quotes are displayed or added to the e-mail, unless you remove them first.

Most experienced users turn off magic quotes and escape quotes using PHP functions. Even if you use magic quotes in programs you run on PHP 4 or 5, you must modify your programs before they run correctly on PHP 6.

PHP provides the `mysqli_real_escape_string()` function (and the `mysql_real_escape_string` function) to escape form data for use in a MySQL query. The function is used after a connection is made to the MySQL server. The connection is passed to the function, along with the unescaped string, and the function escapes the string with respect to the connection. If magic quotes is on when you use the function, the string will already be escaped by magic quotes, resulting in a double escaped string.

In this book, all escaping is accomplished using the PHP function `mysqli_real_escape_string`. To use the programs in this book with PHP 4 or 5, turn `magic_quotes-gpc` off in your `php.ini` file.



If you plan to use your scripts on other computers, it may not be safe to assume that magic quotes is turned on or off. To write portable code, you need to test whether magic quotes is on or off in the script and then use the code that fits the status. You can use the PHP escape functions if magic quotes is turned off or just store the data as if magic quotes is turned on. You can test whether magic quotes is on or off using the `get_magic_quotes_gpc()` function in a conditional statement. The function returns 0 if magic quotes is off and 1 if magic quotes is turned on.

Adding new information

You use the `INSERT` query (described in Chapter 4) to add new information to the database. `INSERT` adds a new row to a database table. The general format is

```
$query = "INSERT INTO tablename (col,col,col...)
                 VALUES ('var','var','var'...)";
$result = mysqli_query($cxn,$query)
       or die ("Couldn't execute query.");
```

For instance, the statements to store the name and phone number that a user enters in a form are

```
$query = "INSERT INTO Member (lastName,firstName,phone)
          VALUES ('$_POST[lastName]','$_POST[firstName]',
                  '$_POST[phone]')";
$result = mysqli_query($query)
          or die ("Couldn't execute query.");
```



You would never insert data directly from the form field in the `$_POST` array. Always check its format first and clean it, as discussed earlier in this chapter.

Listing 8-16 shows a program that displays a form, and Listing 8-17 lists a program called `savePhone.php` that processes the form in Listing 8-16 and stores a name and a phone number from the form in a database.

Listing 8-16: Displaying a Form

```
<?php
/*  Program name: displayPhone
 *  Description:  Script displays a form that asks for the
 *                customer phone number.
 */
  echo "<html>
        <head><title>Customer Phone Number</title></head>
        <body>";
```

```
  $labels = array ( "first_name" => "First Name",
                    "last_name" => "Last Name",
                    "phone" => "Phone");
  echo "<h3>Please enter your phone number below.</h3>";
  echo "<form action='savePhone.php' method='POST'>
        <table>\n";
  /* Loop that displays the form fields */
  foreach($labels as $field => $label)
  {
    echo "<tr>
           <td style='text-align: right;
                     font-weight: bold'> $label</td>
           <td><input type='text' name='$field' size='65'
                     maxlength='65' ></td>
         </tr>";
  }
  echo "<tr>
       <td colspan='2' style='text-align: center'>
            <input type='submit'
                   value='Submit Phone Number'>";
  echo "</td></tr></table>
        </form>";
?>
</body></html>
```

The displayed form provides three fields: `first_name`, `last_name`, and `phone`.

Listing 8-17: Storing Data from a Form

```
<?php
/*  Program name: savePhone.php
 *  Description:  Program checks all the form fields for
 *                blank fields and incorrect format. Saves the
 *                correct fields in a database.
 */
?>
<html>
<head><title>Member Phone Number</title></head>
<body>
<?php
  /* set up array of field labels */
  $labels = array( "first_name" => "First Name",
                   "last_name" => "Last Name",
                   "phone" => "Phone");
/* Check information from form */
  foreach($_POST as $field => $value)
  {
    /* check each field for blank fields */
    if( $value == "" )
    {
```

(continued)

Listing 8-17 *(continued)*

```
      $blank_array[] = $field;
    }
              /* check format of each field */
    elseif( ereg("(name)",$field) )
{
      if(!ereg("^[A-Za-z' -]{1,50}$",$value) )
      {
          $bad_format[] = $field;
      }
    }
    elseif($field == "phone")
    {
      if(!ereg("^[0-9)( -]{7,20}(([xX]|(ext)|(ex))?[ -]?[0-9]{1,7})?$",$value) )
      {
          $bad_format[] = $field;
      }
    }
  } // end of foreach for $_POST
  /* if any fields were not okay, display error message and form */
  if(@sizeof($blank_array) > 0 or @sizeof($bad_format) > 0)
  {
    if(@sizeof($blank_array) > 0)
    {
        /* display message for missing information */
        echo "<b>You didn't fill in one or more required fields.
                You must enter:</b><br>";
        /* display list of missing information */
        foreach($blank_array as $value)
        {
           echo "   {$labels[$value]}<br>";
        }
    }
    if(@sizeof($bad_format) > 0)
    {
        /* display message for bad information */
        echo "<b>One or more fields have information that appears to
                be incorrect. Correct the format for:</b><br>";
        /* display list of bad information */
        foreach($bad_format as $value)
        {
           echo "   {$labels[$value]}<br>";
        }
    }
    /* redisplay form */
    echo "<p><hr />";
    echo "<h3>Please enter your phone number below.</h3>";
    echo "<form action='savePhone.php' method='POST'>
          <table>";
    foreach($labels as $field => $label)
```

```
    {
      $good_data[$field]=strip_tags(trim($_POST[$field]));
      echo "<tr>
             <td style='text-align: right; font-weight: bold'>
                $label</td>
             <td><input type='text' name='$field' size='65'
                 maxlength='65' value='$good_data[$field]'></td>
            </tr>";
    }
    echo "<tr>
           <td colspan='2' style='text-align: center'>
              <input type='submit' value='Submit Phone Number'>";
    echo "</td></tr></table>
          </form>";
    exit();
  }
  else   //if data is okay
  {
    $user="admin";
    $host="localhost";
    $password="";
    $database = "MemberDirectory";
    $cxn = mysqli_connect($host,$user,$password,$database)
           or die ("couldn't connect to server");

    $fields_all = array_keys($labels);
    foreach($fields_all as $field)
    {
      $good_data[$field] = strip_tags(trim($_POST[$field]));
      if($field == "phone")
      {
         $good_data[$field] = ereg_replace("[)( .-]","",$good_data[$field]);
      }
      $good_data[$field] = mysqli_real_escape_string($cxn,$good_data[$field]);
    }

    $query = "INSERT INTO Phone (lastName,firstName,phone)
              VALUES ('$good_data[last_name]','$good_data[first_name]',
                      '$good_data[phone]')";
    $result = mysqli_query($cxn,$query)
              or die ("Couldn't execute query.");
    echo "<h4>New Member added to database</h4>";
  }
?>
</body></html>
```

This program builds on the program `checkAll.php` in Listing 8-13. It checks the data from the form for blank fields and incorrect formats, asking the user to retype the data when it finds a problem. If the data is okay, the program trims the data, cleans it, and stores it in the database.

Your application might need to store data in several places. A function that stores data from a form can be very useful. The following is a function that stores all the data in a form:

```
function storeForm($formdata,$tablename,$cxn)
{
   if(!is_array($formdata))
   {
     return FALSE;
     exit();
   }
   foreach($formdata as $field => $value)
   {
      $formdata[$field] = trim($formdata[$field]);
      $formdata[$field] = strip_tags($formdata[$field]);
      if($field == "phone")
      {
          $formdata[$field] =
             ereg_replace("[)( .-]","",$formdata[$field]);
      }
      $field_array[]=$field;
      $value_array[]=$formdata[$field];
   }
   $fields=implode(",",$field_array);
   $values=implode('","',$value_array);
   $query = "INSERT INTO $tablename ($fields)
                    VALUES (\"$values\")";
   if($result = mysqli_query($cxn,$query))
      return TRUE;
   else
      return FALSE;
}
```

The function returns `TRUE` if it finishes inserting the data without an error or `FALSE` if it is unable to insert the data. At the beginning, the function checks that the data passed to it is actually an array. If `$formdata` is not an array, the function stops and returns `FALSE`.

Notice that this function works only if the field names in the form are the same as the column names in the database table. Also notice that this function assumes you're already connected to the MySQL server and have selected the correct database. The database connection is passed to the function.

The following code shows how you can call the function:

```
else   //if data is okay
{
  $user="admin";
  $host="localhost";
  $password="";
  $database = "MemberDirectory";
  $cxn = mysqli_connect($host,$user,$password,$database)
```

```
          or die ("couldn't connect to server");
  if(storeForm($good_data,"Phone",$cxn))
      echo "New Member added to database<br>";
  else
      echo "New Member was not added to the database<br>";
}
?>
</body></html>
```

Notice how much easier this program is to read with the majority of the statements in the function. Furthermore, this function works for any form as long as the field names in the form are the same as the column names in the database table. If the function is unable to execute the query, it stops execution at that point and prints the error message `"Couldn't execute query"`. If the query might fail in certain circumstances, you need to take these into consideration.

Updating existing information

You update existing information with the `UPDATE` query, as I describe in Chapter 4. Updating means changing data in the columns of rows that are already in the database — not adding new rows to the database table. The general format is

```
$query = "UPDATE tablename SET col=value WHERE col=value";
$result = mysql_query($query)
       or die ("Couldn't execute query.");
```

For instance, the statements to update the phone number for Goliath Smith are

```
$query = "UPDATE Member SET phone='$_POST['phone],
                WHERE lastName='$_POST[lastName]',
                AND firstName='$_POST[firstName]'";
$result = mysqli_query($cxn,$query)
       or die ("Couldn't execute query.");
```



If you don't use a `WHERE` clause in an `UPDATE` query, the field that is `SET` is set for all the rows. That is seldom what you want to do.



You would never update data using data directly from the form field in the `$_POST` array. Always check its format first and clean it, as discussed earlier.

Listing 8-18 shows a program called `updatePhone.php`, which updates a phone number in an existing database record. `updatePhone.php` processes data from the same form as `storePhone.php` — the form displayed by `displayPhone.php` listed in Listing 8-16. You just need to change the `form` tag so that the program in the `action` attribute is `updatePhone.php`, as follows:

```
echo "<form action='updatePhone.php' method='POST'>
```

Listing 8-18: Updating Data

```
<?php
/*  Program name: updatePhone.php
 *  Description:  Program checks the phone number for incorrect format. Updates
 *                the phone number in the database for the specified name.
 */
?>
<html>
<head><title>Update Member Phone Number</title></head>
<body>
<?php
  /* set up array of field labels */
  $labels = array ( "first_name" => "First Name",
                    "last_name" => "Last Name",
                    "phone" => "Phone");

  /* check each field for blank fields */
  foreach ($_POST as $field => $value)
  {
    if ( $value == "" )
    {
        $blank_array[] = $field;
    }
  }
  /* check format of phone number */
  if(!ereg("^[0-9)( -]{7,20}(([xX]|(ext)|(ex))?[ -]?[0-9]{1,7})?$",
       $_POST['phone']))
  {
         $bad_format[] = "phone";
  }
  /* if any fields were not okay, display error message and form */
  if (@sizeof($blank_array) > 0 or @sizeof($bad_format) > 0)
  {
    if (@sizeof($blank_array) > 0)
    {
        /* display message for missing information */
        echo "<b>You didn't fill in one or more required
                 fields. You must enter:</b><br>";
        /* display list of missing information */
        foreach($blank_array as $value)
        {
         echo "   {$labels[$value]}<br>";
        }
    }
    if (@sizeof($bad_format) > 0)
    {
        /* display message for bad phone number */
        echo "<b>Your phone number appears to be incorrect.<br>";
    }
```

```
    /* redisplay form */
    echo "<p><hr>";
    echo "<h3>Please enter your phone number below.</h3>";
    echo "<form action='updatePhone.php' method='POST'>
          <table>";
    foreach($labels as $field=>$label)
    {
      $good_data[$field] = strip_tags(trim($_POST[$field]));
      echo "<tr>
             <td style='text-align: right; font-weight: bold'>
                {$labels[$field]}</td>
             <td><input type='text' name='$field' size='65'
                 maxlength='65' value='$_POST[$field]'></td>
            </tr>";
    }
    echo "<tr>
           <td colspan='2' style='text-align: center'>
              <input type='submit' value='Submit Phone Number'>";
    echo "</td></tr></table>
          </form>";
    exit();
  }
  else   //if data is okay
  {
    $good_data['phone'] = strip_tags(trim($_POST['phone']));
    $good_data['phone'] = ereg_replace("[)( .-]","",$good_data['phone']);

    $user="admin";
    $host="localhost";
    $password="";
    $database = "MemberDirectory";
$cxn = mysqli_connect($host,$user,$password,$database)
          or die ("Couldn't connect to server");
    $query = "UPDATE Phone SET phone='$good_data[phone]'
                    WHERE lastName='$_POST[last_name]'
                      AND firstName='$_POST[first_name]'";
    $result = mysqli_query($cxn,$query)
              or die ("Couldn't execute query: ".mysqli_error($cxn));
    if(mysqli_affected_rows($cxn) > 0)
    {
       echo "<h3>The phone number for {$_POST['first_name']}
              {$_POST['last_name']} has been updated</h3>";
    }
    else
    echo "No record updated";
  }
?>
</body></html>
```

The program in Listing 8-18, which updates the database, is almost identical to the program in Listing 8-17, which adds new data. Using an `UPDATE` query in this program — instead of the `INSERT` query you used to add new data — is the major difference. Both programs check the data and then clean it because both programs store the data in the database.



If you see backslashes (\) in the database after you have inserted or updated the record, your data was escaped twice. This probably means you have magic quotes turned on and also used `mysqli_real_escape_quotes`. Turn off magic quotes. When your strings are escaped correctly, the escapes make sure the query is executed correctly, but the escapes are not stored in the database.

Getting Information in Files

Sometimes you want to receive an entire file of information from a user, such as user résumés for your job-search Web site or pictures for your photo album Web site. Or, suppose you're building the catalog from information supplied by the Sales department. In addition to descriptive text about the product, you want Sales to provide a picture of the product. You can supply a form that Sales can use to upload an image file.

Using a form to upload the file

You can display a form that allows a user to upload a file by using an HTML form designed for that purpose. The general format of the form is as follows:

```
<form enctype="multipart/form-data"
          action="processfile.php" method="POST">
  <input type="hidden" name="MAX_FILE_SIZE" value="30000">
  <input type="file" name="user_file">
  <input type="submit" value="Upload File">
</form>
```

Notice the following points regarding the form:

- **The `enctype` attribute is used in the `form` tag.** You must set this attribute to `multipart/form-data` when uploading a file to ensure that the file arrives correctly.
- **A hidden field is included that sends a value (in bytes) for `MAX_FILE_SIZE`.** If the user tries to upload a file that is larger than this value, it won't upload. You can set this value as high as 2MB. If you need to upload a file larger than that, you must change the default setting for `upload_max_filesize` in `php.ini` to a larger number before sending a value larger than 2MB for `MAX_FILE_SIZE` in the hidden field.

- **The input field that uploads the file is of type `file`.** Notice that the field has a name — `user_file` — as do other types of fields in a form. The filename that the user enters into the form is sent to the processing program and is available in the built-in array called `FILES`. I explain the structure and information in `FILES` in the following section.

When the user submits the form, the file is uploaded to a temporary location. The script that processes the form needs to copy the file to another location because the temporary file is deleted as soon as the script is finished.

Processing the uploaded file

Information about the uploaded file is stored in the PHP built-in array called `$_FILES`. An array of information is available for each file that was uploaded, resulting in `$_FILES` being a multidimensional array. As with any other form, you can obtain the information from the array by using the name of the field. The following is the array available from `$_FILES` for each uploaded file:

```
$_FILES['fieldname']['name']
$_FILES['fieldname']['type']
$_FILES['fieldname']['tmp_name']
$_FILES['fieldname']['size']
```

For example, suppose that you use the following field to upload a file, as shown in the preceding section:

```
<input type="file" name="user_file">
```

If the user uploads a file named `test.txt` by using the form, the resulting array that can be used by the processing program looks something like this:

```
$_FILES[user_file][name] = test.txt
$_FILES[user_file][type] = text/plain
$_FILES[user_file][tmp_name] = D:\WINNT\php92C.tmp
$_FILES[user_file][size] = 435
```

In this array, `name` is the name of the file that was uploaded, `type` is the type of file, `tmp_name` is the path/filename of the temporary file, and `435` is the size of the file. Notice that `name` contains only the filename, but `tmp_name` includes the path to the file as well as the filename.

If the file is too large to upload, the `tmp_name` in the array is set to `none`, and the size is set to `0`. The processing program must move the uploaded file from the temporary location to a permanent location. The general format of the statement that moves the file is as follows:

```
move_uploaded_file(path/tempfilename,path/permfilename);
```

The *path/tempfilename* is available in the built-in array element `$_FILES['fieldname']['tmp_file']`. The *path/permfilename* is the path to the file where you want to store the file. The following statement moves the file uploaded in the input field, given the name `user_file`, shown earlier in this section:

```
move_uploaded_file($_FILES['user_file']['tmp_name'],
    'c:\data\new_file.txt');
```

The destination directory (in this case, `c:\data`) must exist before the file can be moved to it. This statement doesn't create the destination directory.

Security can be an issue when uploading files. Allowing strangers to load files onto your computer is risky; malicious files are possible. You want to check the files for as many factors as possible after they're uploaded, using conditional statements to check file characteristics, such as expected file type and size. In some cases, for even more security, it might be a good idea to change the name of the file to something else so that users don't know where their files are or what they're called.

Putting it all together

A complete example script is shown in Listing 8-19. This program displays a form for the user to upload a file, saves the uploaded file, and then displays a message after the file has been successfully uploaded. That is, this program both displays the form and processes the form. This program expects the uploaded file to be an image file and tests to make sure that it is an image file, but any type of file can be uploaded. The HTML code that formats and displays the form is in a separate file — the include file shown in Listing 8-20. A Web page displaying the form is shown in Figure 8-15.

Listing 8-19: Uploading a File with a POST Form

```
<?php
 /* Script name: uploadFile.php
  * Description: Uploads a file via HTTP with a POST form.
  */
  if(!isset($_POST['Upload']))                                    #5
  {
    include("form_upload.inc");
  }
  else                                                            #9
  {
    if($_FILES['pix']['tmp_name'] == "none")                      #11
    {
```

```
      echo "<p style='font-weight: bold'>
        File did not successfully upload. Check the
            file size. File must be less than 500K.</p>";
      include("form_upload.inc");
      exit();
    }
    if(!ereg("image",$_FILES['pix']['type']))                    #19
    {
      echo "<p style='font-weight: bold'>
        File is not a picture. Please try another
            file.</p>";
      include("form_upload.inc");
      exit();
    }
    else                                                         #27
    {
      $destination='c:\data'."\\".$_FILES['pix']['name'];
      $temp_file = $_FILES['pix']['tmp_name'];
      move_uploaded_file($temp_file,$destination);
      echo "<p style='font-weight: bold'>
        The file has successfully uploaded:
            {$_FILES['pix']['name']}
            ({$_FILES['pix']['size']})</p>";
    }
  }
?>
```

I have added line numbers at the end of some of the lines in the script. The script is discussed with reference to these line numbers:

5 This line is an `if` statement that tests whether the form has been submitted. If not, the form is displayed by including the file containing the form code. The include file is shown in Listing 8-20.

9 This line starts an `else` block that executes if the form has been submitted. This block contains the rest of the script and processes the submitted form and uploaded file.

11 This line begins an `if` statement that tests whether the file was successfully uploaded. If not, an error message is displayed, and the form is redisplayed.

19 This line is an `if` statement that tests whether the file is a picture. If not, an error message is displayed, and the form is redisplayed.

27 This line starts an `else` block that executes if the file has been successfully uploaded. The file is moved to its permanent destination, and a message is displayed that the file has been uploaded.

Listing 8-20 shows the include file used to display the upload form.

Listing 8-20: An Include File That Displays the File Upload Form

```
<!-- Program Name: form_upload.inc
     Description:  Displays a form to upload a file -->
<html>
<head><title>File Upload</title></head>
<body>
<ol><li>Enter the file name of the product picture you
        want to upload or use the browse button
        to navigate to the picture file.</li>
    <li>When the path to the picture file shows in the
        text field, click the Upload Picture button.</li>
</ol>
<div align="center"><hr />
<form enctype="multipart/form-data"
        action="uploadFile.php" method="POST">
  <input type="hidden" name="MAX_FILE_SIZE"
         value="500000">
  <input type="file" name="pix" size="60">
  <p><input type="submit" name="Upload"
        value="Upload Picture">
</form>
</div></body></html>
```

Notice that the include file contains no PHP code — just HTML code.

The form that allows users to select a file to upload is shown in Figure 8-15. The form has a text field for inputting a filename and a Browse button that enables the user to navigate to the file and select it.

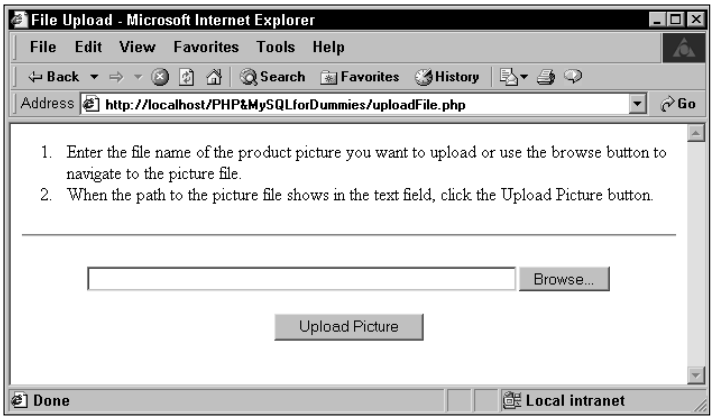


Figure 8-15: A form that allows users to upload an image file.

Chapter 9

Moving Information from One Web Page to the Next

In This Chapter

- Moving your user from one page to the next
- Moving information from one page to the next
- Adding information to a URL
- Taking a look at cookies
- Using hidden form fields
- Discovering PHP sessions

Most Web sites consist of more than one Web page. This includes the static Web pages that you may have developed in the past. With static Web pages, users click links to move from one page to the next. Users click a link in one Web page, and a new Web page appears in their browser. When users move from page to page this way, no information is transferred from the first page to the second. Each new page that is sent to the user's browser is independent of any other pages the user may have seen previously. With dynamic Web pages, you may need to transfer information from one page to the next. If you are an advanced HTML developer, you may have experience with limited methods for transferring information from one page to the next using HTML forms and CGI (Common Gateway Interface) or cookies. However, PHP is a more powerful method for passing information from Web page to Web page.

Moving Your User from One Page to Another

When using only HTML, you provide links so that a visitor can go from one page to another in your Web site. When using PHP, you have three options for moving your user from one page to the next:

- **Links:** You can echo the HTML tags that display a link. The general format of an HTML statement that displays a link is

  ```
  <a href="newpage.php">Text user sees as a link</a>
  ```

  When users click the link, the program `newpage.php` is sent to their browser. This method is used extensively in HTML Web pages. You're likely familiar with creating links from your HTML experience, but if you need a refresher, find out more about links in any HTML book, such as *HTML 4 For Dummies Quick Reference,* 2nd Edition, by Deborah S. Ray and Eric J. Ray (Wiley).

- **Form submit buttons:** You can use an HTML form with one or more submit buttons. When the user clicks a submit button, the program in the `form` tag runs and sends a new Web page to the user's browser. You can create a form with no fields — only a submit button — but the user must click the submit button to move to the next page. I discuss forms and submit buttons thoroughly in Chapter 8.

- **The `header` function:** You can send a message to the Web server with the PHP header function that tells the server to send a new page. When using this method, you can display a new page in the user's browser without the user having to click a link or a button.

The PHP `header` function can be used to send a new page to the user's browser. The program uses a `header` statement and displays the new Web page without needing any user action. When the `header` statement is executed, the new page is displayed. The format of the `header` function that requests a new page is

```
header("Location: URL");
```

The file located at *URL* is sent to the user's browser. Either of the following statements are valid `header` statements:

```
header("Location: newpage.php");
header("Location: http://company.com/cat/catalog.php");
```



URLs

A *URL* (Uniform Resource Locator) is an address on the Web. Every Web page has its own URL or address. The URL is used by the Web server to find the Web page and send it to a browser.

The format of a URL is

```
HTTP://servername:portnumber/
   path#target?string=string
```

Here's a breakdown of the parts that make up the URL:

- `HTTP://servername`: This tells the server that the address is a Web site and gives the name of the computer where the Web site is located. Other types of transfer can be specified, such as FTP (File Transfer Protocol), but these aren't related to the subject of this book. If this part of the URL is left out, the Web server assumes that the computer is the same computer that the URL is typed on. Valid choices for this part might be `HTTP://amazon.com` or `HTTP://localhost`. ***Note:*** HTTP doesn't have to be in uppercase letters.
- `:portnumber`: The Web server exchanges information with the Internet at a particular port on the computer. Most of the time, the Web server is set up to communicate via port 80. If the port number isn't specified, port 80 is assumed. In some unusual circumstances, a Web server may use a different port number, in which case the port number must be specified. The most common reason for using a different port number is to set up a test Web site on another port that's available only to developers and testers, not customers. When the site is ready for customers, it is made available on port 80.
- `path`: This is the path to the file, which follows the rules of any path. The root of the path is the main Web site directory. If the path points to a directory, rather than a file, the Web server searches for a default filename, such as `default.html` or `index.html`. The person who administers the Web site sets the default filename. The path `/catalog/show.php` indicates a directory called `catalog` in the main Web site directory and a file named `show.php`. The path `catalog/show.php` indicates a directory called `catalog` in the current directory.
- `#target`: An HTML tag defines a target. This part of the URL displays a Web page at the location where the target tag is located. For instance, if the tag `<a name="target"></a>` is in the middle of the file somewhere, the Web page will be displayed at the tag rather than at the top of the file.
- `?string=string`: The question mark allows information to be attached to the end of the URL. The information in forms that use the `get` method is passed at the end of the URL in the format `fieldname=value`. You can add information to the end of a URL to pass it to another page. PHP automatically gets information from the URL and puts it into built-in arrays. You can pass more than one `string=string` pair by separating each pair with an ampersand (`&`): for example, `?state=CA&city=home`.

Statements that must come before output

Some PHP statements can only be used before sending any output. `header` statements, `setcookie` statements, and `session` functions, all described in this chapter, must all come before any output is sent. If you use one of these statements after sending output, you may see the following message:

```
Cannot add header information - headers already sent
```

The message will also provide the name of the file and indicate which line sent the previous output. Or you might not see a message at all; the new page might just not appear. (Whether you see an error message depends on what error message level is set in PHP; see Chapter 6 for details.) The following statements will fail because the `header` message is not the first output:

```
<body>
<?php
   header("Location: http://company.com");
?>
</body>
```

One line of HTML code is sent before the `header` statement. The following statements will work, although they don't make much sense:

```
<?php
   header("Location: http://company.com");
?>
<body>
</body>
```

The following statements will fail:

```
 <?php
   header("Location: http://company.com");
?>
<html>
```

The reason why these statements fail is not easy to see, but if you look closely, you'll notice a single blank space before the opening PHP tag. This blank space is output to the browser, although the resulting Web page looks empty. Therefore, the `header` statement fails because there is output before it. This is a common mistake and difficult to spot.

The `header` function has a major limitation, however. The `header` statement can only be used *before* any other output is sent. You cannot send a message requesting a new page in the middle of a program after you have echoed some output to the Web page. See the sidebar "Statements that must come before output" for a discussion.

In spite of its limitation, the `header` function can be useful. You can have as many PHP statements as you want before the `header` function as long as they don't send output. Therefore, the following statements will work:

```
<?php
   if ($customer_age < 13)
   {
      header("Location: ToyCatalog.php");
   }
   else
   {
      header("Location: ElectronicsCatalog.php");
   }
?>
```

These statements run a program that displays a toy catalog if the customer's age is less than 13 but run a program that displays an electronics catalog if the customer's age is 13 or older.

Moving Information from Page to Page

HTML pages are independent from one another. When a user clicks a link, the Web server sends a new page to the user's browser, but the Web server doesn't know anything about the previous page. For static HTML pages, this process works fine. However, many dynamic applications need information to pass from page to page. For instance, you might want to store a user's name and refer to that person by name on another Web page.

Dynamic Web applications often consist of many pages and expect the user to view several different pages. The period beginning when a user views the first page and ending when a user leaves the Web site is a *session.* Often you want information to be available for a complete session. The following are examples of sessions that necessitate sharing information among pages:

- **Restricting access to a Web site:** Suppose that your Web site is restricted and users log in with a password to access the site. You don't want users to have to log in on every page. You want them to log in once and then be able to see all the pages that they want. You want users to bring information with them to each page showing that they have logged in and are authorized to view the page.
- **Providing Web pages based on the browser:** Because browsers interpret some HTML features differently, you might want to provide different versions of your Web pages for different browsers. You want to check the user's browser when the user views the first page and then deliver all the other pages based on the user's browser type and version.

With PHP, you can move information from page to page by using any of the following methods:

- **Adding information to the URL:** You can add certain information to the end of the URL of the new page, and PHP will put the information into built-in arrays that you can use in the new page. This method is most appropriate when you need to pass only a small amount of information.
- **Storing information via cookies:** You can store *cookies* — small amounts of information containing `variable=value` pairs — on the user's computer. After the cookie is stored, you can get it from any Web page. However, users can refuse to accept cookies. Therefore, this method works only in environments where you know for sure that the user will have cookies turned on.
- **Passing information using HTML forms:** You can pass information to a specific program by using a `form` tag. When the user clicks the submit button, the information in the form is sent to the next program. This method is useful when you need to collect information from users.
- **Using PHP session functions:** Beginning with PHP 4, PHP functions are available that set up a user session and store session information on the server; this information can be accessed from any Web page. This method is useful when you expect users to view many pages in a session.

Adding information to the URL

A simple way to move information from one page to the next is to add the information to the URL. Put the information in the following format:

```
variable=value
```

The *`variable`* is a variable name, but do not use a dollar sign (`$`) in it. The *`value`* is the value to be stored in the variable. You can add the `variable=value` pair anywhere that you use a URL. You signal the start of the information with a question mark (`?`). The following statements are all valid ways of passing information in the URL:

```
<form action="nextpage.php?state=CA" method="POST">
```

```
<a href="nextpage.php?state=CA">go to next page</a>
```

```
header("Location: nextpage.php?state=CA");
```

You can add several `variable=value` pairs, separating them with ampersands (`&`) as follows:

```
<form action="next.php?state=CA&city=home" method="POST">
```

Here are two reasons why you might not want to pass information in the URL:

- **Security:** The URL is shown in the address line of the browser, which means that the information that you attach to the URL is also shown. If the information needs to be secure, you don't want it shown so publicly. For example, if you're moving a password from one page to the next, you probably don't want to pass it in the URL. Also, the URL can be bookmarked by the user. There may be reasons why you don't want your users to save the information that you add to the URL.
- **Length of the string:** There is a limit on the length of the URL. The limit differs for various browsers and browser versions, but there is always a limit. Therefore, if you're passing a lot of information, there may not be room for it in the URL.

Adding information to the URL is useful for quick, simple data transfer. For instance, suppose that you want to provide a Web page where users can update their phone numbers. You want the form to behave as follows:

1. When the user first displays the form, the phone number from the database is shown in the form so that the user can see what number is currently stored in the database.
2. When the user submits the form, the program checks the phone number to see whether the field is blank or whether the field is in a format that could not possibly be a phone number.
3. If the phone number checks out okay, it is stored in the database.
4. If the phone number is blank or has bad data, the program redisplays the form. However, this time you don't want to show the data from the database. Instead, you want to show the bad data that the user typed and submitted in the form field.

The `changePhone.php` program in Listing 9-1 shows how to use the URL to determine whether this is the first showing of the form or a later showing. The program displays the phone number for the user's login name and allows the user to change the phone number.

Listing 9-1: Displaying a Phone Number in a Form

```
<?php
/*  Program name: changePhone.php
 *  Description:  Displays a phone number retrieved from the database
 *                and allows the user to change the phone number.
*/
?>
<html>
<head><title>Change phone number</title></head>
<body>
<?php
  $host="localhost";
  $user="admin";
  $password="";
  $database="MemberDirectory";
  $loginName = "gsmith";  // passed from previous page
  $cxn = mysqli_connect($host,$user,$password,$database)
         or die ("couldn't connect to server");

  if (@$_GET['first'] == "no")                                          #19
  {
    $phone = trim($_POST['phone']);
    if (!ereg("^[0-9)( -]{7,20}$",$phone) or $phone == "")              #22
    {
        echo "<h3 style='text-align: center'>
                Phone number does not appear to be valid.</h3>";
        display_form($loginName,$phone);                                #26
    }
    else // phone number is okay                                       #28
    {
        $query = "UPDATE Member SET phone='$phone'
                          WHERE loginName='$loginName'";
        $result = mysqli_query($cxn,$query)
                  or die ("Couldn't execute query.");
        echo "<h3>Phone number has been updated.</h3>";
        exit();
    }
  }
  else // first time form is displayed                                  #38
  {
    $query = "SELECT phone FROM Member WHERE loginName='$loginName'";
    $result = mysqli_query($cxn,$query)
              or die ("Couldn't execute query.");
    $row = mysqli_fetch_row($result);
    $phone = $row[0];
    display_form($loginName,$phone);                                    #45
  }
```

```
function display_form($loginName,$phone)                                    #48
{
  echo "<div style='text-align: center'>";
  echo "<form action='changePhone.php?first=no' method='POST'>
          <h4>Please check the phone number
              below and correct it if necessary.</h4><hr />
          <p><b>$loginName</b> <input type='text' name='phone'
                        maxlength='20' value='$phone'></p>
          <p><input type='submit' value='Submit phone number'></p>
        </form>";
  echo "</div>";
}
?>
</body></html>
```

Notice the following key points about this program:

- **The same program displays and processes the form.** The name of this program is `changePhone.php`. The `form` tag on line 51 includes `action=changePhone.php`, meaning that when the user clicks the submit button, the same program runs again.
- **Information is added to the URL.** The `form` tag on line 51 includes `action=changePhone.php?first=no`. When the user clicks the submit button and `changePhone.php` runs the second time, a variable `$first` is passed with the value `"no"`.
- **The value that was passed for `first` in the built-in `$_GET` array is checked at the beginning of the program on line 19.** This checks whether this is the first time the program has run.
- **If `$_GET[first]` equals `"no"`, the phone number is checked.** `$_GET[first]` equals `no` only if the form is being submitted. `$_GET[first]` does not equal `no` if this is the first time through the program.
  - If the phone number is not okay, an error message is printed and the form is redisplayed. This block of code starts on line 22.
  - If the phone number is okay, it is stored in the database and the program ends. This block of code starts on line 28.
- **If `$_GET[first]` does *not* equal `"no"`, the phone number is retrieved from the database.** In other words, if `$_GET[first]` doesn't equal `no`, it *is* the first time that the program has run. The program should get the phone number from the database. This block of code starts on line 38.
- **The program includes a function that displays the form.** The function is defined beginning on line 48. Whenever the form needs to be displayed, the function is called (lines 26 and 45).

The form displayed by the program in Listing 9-1 is shown in Figure 9-1. This shows what the Web page looks like the first time it's displayed. The URL in the browser address field doesn't have any added information.

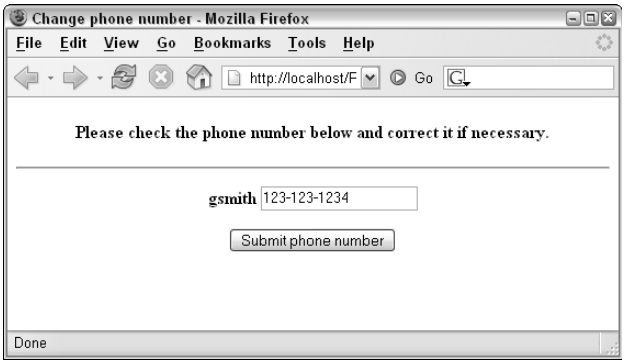


Figure 9-1: HTML form to update a phone number.

Figure 9-2 shows the results when a user types a nonsense phone number in the form in Figure 9-1 and clicks the submit button. Notice that the URL in the browser address field now has `?first=no` added to the end of it.



Figure 9-2: HTML form when a user submits a nonsense phone number.

Storing information via cookies

You can store information as cookies. *Cookies* are small amounts of information containing `variable=value` pairs, similar to the pairs that you can add to a URL. The user's browser stores cookies on the user's computer. Your application can then get the cookie from any Web page. Why these are called cookies is one of life's great mysteries. Perhaps they're called cookies because they seem at first glance to be a wonderful thing, but on closer examination, you realize that they aren't that good for you. For some people in some situations, cookies are not helpful at all.

At first glance, cookies seem to solve the entire problem of moving data from page to page. Just stash a cookie on the user's computer and get it whenever you need it. In fact, the cookie can be stored so that it remains there after the user leaves your site and will still be available when the user enters your Web site again a month later. Problem solved! Well, not exactly. Cookies are not under your control: They're under the user's control. The user can at any time delete the cookie. In fact, users can set their browsers to refuse to allow any cookies. And many users do refuse cookies or routinely delete them. Many users aren't comfortable with the whole idea of a stranger storing things on their computers, especially files that remain after they leave the stranger's Web site. It's an understandable attitude. However, it definitely limits the usefulness of cookies. If your application depends on cookies and the user has turned off cookies, your application won't work for that user.



Cookies were originally designed for storing small amounts of information for short periods of time. Unless you specifically set the cookie to last a longer period of time, the cookie will disappear when the user closes his or her browser. Although cookies are useful in some situations, you're unlikely to need them for your Web database application for the following reasons:

- **Users may set their browsers to refuse cookies.** Unless you know for sure that all your users will have cookies turned on or you can request that they turn on cookies (and expect them to follow your request), cookies are a problem. If your application depends on cookies, it won't run if cookies are turned off.
- **PHP has features that work better than cookies.** Beginning with PHP 4, PHP includes functions that create sessions and store information that's available for the entire session. The session feature is more reliable and much easier to use than cookies for making information available to all the Web pages in a session. Sessions don't work for long-term storage of information, but MySQL databases can be used for that.
- **You can store data in your database.** Your application includes a database where you can store and retrieve data, which is usually a better solution than a cookie. Users can't delete the data in your database unexpectedly. Because you're using a database in this application, you can use it for any data storage needed, especially long-term data storage. Cookies are more useful for applications that don't make use of a database.

You store cookies by using the `setcookie` function. The general format is

```
setcookie("variable","value");
```

The *`variable`* is the variable name, but do not include the dollar sign (`$`). This statement stores the information only until the user leaves your Web site. For instance, the following statement

```
setcookie("state","CA");
```

stores CA in a cookie variable named state. After you set the cookie, the information is available to your other PHP programs in the element of a built-in array as $_COOKIE[state]. You don't need to do anything to get the information from the cookie. PHP does this automatically. The cookie is not available in the program where it is set. The user must go to another page or redisplay the current page before the cookie information can be used.



If you are using a version of PHP earlier than PHP 4.1, you must get the data from the long array called $HTTP_COOKIE_VARS. However, long arrays are no longer available in PHP 6. To run old scripts in PHP 6, you must change the array name in your code from $HTTP_COOKIE_VARS to $_COOKIE.

If you want the information stored in a cookie to remain in a file on the user's computer after the user leaves your Web site, set your cookie with an expiration time, as follows:

```
setcookie("variable","value",expiretime);
```

The *expiretime* value sets the time when the cookie will expire. *expiretime* is usually set by using the time or mktime function, as follows:

- time: This function returns the current time in a format that the computer can understand. You use the time function plus a number of seconds to set the expiration time of the cookie, as follows:

  ```
  setcookie("state","CA",time()+3600); //expires in 1 hour
  setcookie("Name",$Name,time()+(3*86400)) // exp in 3 days
  ```

- mktime: This function returns a date and time in a format that the computer can understand. You must provide the desired date and time in the following order: hour, minute, second, month, day, and year. If any value is not included, the current value is used. You use the mktime function to set the expiration time of the cookie, as follows:

  ```
  setcookie("state","CA",mktime(3,0,0,4,1,2003));
       //expires at 3:00 AM on April 1, 2003.
  setcookie("state","CA",mktime(12,0,0,,,));
      //expires at noon today
  ```

You can remove a cookie by setting its value to nothing. Either of the following statements removes the cookie:

```
setcookie("name");
setcookie("name","");
```



The setcookie function has a major limitation. The setcookie function can only be used *before* any other output is sent. You *cannot* set a cookie in the middle of a program after you have echoed output to the Web page. See the sidebar "Statements that must come before output" elsewhere in this chapter.

Passing information with HTML forms

The most common way to pass information from one page to another is with HTML forms. An HTML form is displayed with a submit button. When the user clicks the submit button, the information in the form fields is passed to the program designated in the `form` tag. The general format is

```
<form action="processform.php" method="POST">
    tags for one or more fields
    <input type="submit" value="string">
</form>
```

The most common use of a form is to collect information from users (which I discuss in detail in Chapter 8). However, forms can also be used to pass other types of information using *hidden fields* — fields that are not displayed in the form. In fact, you can create a form that has only hidden fields. You always need a submit button, and the new page doesn't display until the user clicks the submit button, but you don't need to include any fields for the user to fill in.

For instance, the following statements pass the user's preferred background color to the next page when the user clicks a button named Next Page:

```
<?php
   $color = "blue"; //passed to this program via a user
            form
   echo "<form action='nextpage.php' method='POST'>
         <input type='hidden' name='color' value='$color'>
         <input type='submit' value='Next Page'>
         </form>\n";
?>
```

The Web page shows a submit button labeled Next Page, but it doesn't ask the user for any information. When the user clicks the button, `nextpage.php` runs and can use the array element `$_POST[color]`, which contains `"blue"`.

Using PHP Sessions

A session is the time that a user spends at your Web site. Users can view many Web pages between the time they enter your site and leave it. Often you want information to follow the user around your site so that it's available on every page. PHP, beginning with version 4.0, provides a way to do this.

PHP enables you to set up a session on one Web page and save variables as session variables. Then you open the session in any other page, and the session variables are available for your use in the built-in array `$_SESSION`. To do this, PHP does the following:

1. **Assigns a session ID number.** The number is a long, nonsense number that is unique for the user and that no one could possibly guess. The session ID is stored in a PHP system variable named `PHPSESSID`.
2. **Stores session variables in a file on the server.** The file is named with the session ID number. The file is stored in `\tmp` on Unix and Linux; in Windows, it's stored in a directory called `sessiondata` under the directory where PHP is installed.

   If you have access to edit `php.ini`, you can change the location where the session files are stored by changing the setting for `session.save_path`. Change the path to the location where you want to store the files.
3. **Passes the session ID number to every page.** If the user has cookies turned on, PHP passes the session ID using cookies. If the user has cookies turned off, PHP passes the session ID in the URL for links or in a hidden variable for forms that use the `post` method.
4. **Gets the variables from the session file for each new session page.** Whenever a user opens a new page that is part of the session, PHP gets the variables from the file, using the session ID number that was passed from the old page, and puts them into the built-in array `$_SESSION`. You can use the array elements with the variable name as the key, and they have the value that you assigned in the previous page.



Sessions do not work unless `track_vars` is enabled. As of PHP 4.0.3, `track-vars` is always turned on. For versions before 4.0.3, the option `--enable-track-vars` should be used when installing PHP.



If users have cookies turned off, sessions do not work unless `trans-sid` is turned on. You find out how to turn `trans-sid` on and off later, in the "Using PHP session variables" section.

Opening sessions

You should open a session on each Web page. Open the session with the `session_start` function, as follows:

```
session_start();
```

The function first checks for an existing session ID number. If it finds one, it sets up the `$_SESSION` array. If it doesn't find one, it starts a new session by creating a new session ID number.

Because sessions use cookies if the user has them turned on, `session_start` is subject to the same limitation as cookies. That is, the `session_start` function must be called before any output is sent. For complete details, see the sidebar "Statements that must come before output," elsewhere in this chapter.

Using PHP session variables

When you want to save a variable as a session variable — that is, available to other Web pages that the user might visit — save it in the `$_SESSION` array as follows:

```
$_SESSION['variablename'] = value;
```

The value is then available in `$_SESSION` on other Web pages. For example, you can store the state where the user lives with the following statement:

```
$_SESSION['state'] = "CA";
```

You can then use `$_SESSION['state']` in any other Web page, and it will have the value `CA`.

The following two programs show how to use sessions to pass information from one page to the next. The first program, `sessionTest1.php` in Listing 9-2, shows the first page where the session begins. Listing 9-3 shows the program `sessionTest2.php` for the second page in a session.

Listing 9-2: Starting a Session

```
<?php
  session_start();
?>
<html>
<head><title>Testing Sessions page 1</title></head>
<body>
<?php
  $_SESSION['session_var'] = "testing";
  echo "This is a test of the sessions feature.
       <form action='sessionTest2.php' method='POST'>
       <input type='hidden' name='form_var'
              value='testing'>
       <input type='submit' value='go to next page'>
       </form>";
?>
</body></html>
```

Note that this program sets two variables to be passed to the second page. The session variable `session_var` is created. In addition, a form is displayed with a hidden variable `form_var`, which is also passed to the second page when the submit button is pressed. Both variables are set to `"testing"`.

Listing 9-3: The Second Page of a Session

```
<?php
  session_start();
?>
<html>
<head><title>Testing Sessions page 2</title></head>
<body>
<?php
  echo "session_var = {$_SESSION['session_var']}<br>\n";
  echo "form_var = {$_POST['form_var']}<br>\n";
?>
</body></html>
```

Point your browser at `sessionTest1.php` and then click the submit button that reads Go to Next Page. You will then see the following output from `sessionTest2.php`:

```
session_var = testing
form_var = testing
```

Because sessions work differently for users with cookies turned on and for users with cookies turned off, you should test the two programs in both conditions. To turn off cookies in your browser, you change the settings for options or preferences.

To disable cookies in Internet Explorer, follow these steps:

1. **Choose Tools➪Internet Options.**
2. **Click the Security tab in IE 5.5 or the Privacy tab in IE 6.**
3. **Move the slider to the higher level, which says "Block All Cookies," and then click OK.**

To disable cookies in Firefox, follow these steps:

1. **Choose Tools➪Options.**
2. **Click the Cookies tab.**
3. **Deselect the Allow Sites to Set Cookies option, and then click OK.**

If the output from `sessionTest2` shows a blank value for `$session_var` when you turn off cookies in your browser, `trans-sid` is probably not turned on. You can turn on `trans-sid` in your `php.ini` file. Find the following line:

```
session.use_trans_sid = 0
```

Change the `0` to `1` to turn on `trans-sid`. If you can't get this problem fixed, you can still use sessions, but you must pass the session ID number in your programming statements; PHP won't pass the session ID number automatically when cookies are turned off. For details on how to use sessions when `trans-sid` is not turned on, check out the next section.



For PHP 4.1.2 or earlier, `trans-sid` is not available unless it was enabled by using the option `--enable-trans-sid` when PHP was compiled.

Sessions without cookies

Many users turn off cookies in their browsers. PHP checks the user's browser to see whether cookies are allowed and behaves accordingly. If the user's browser allows cookies, PHP does the following:

- Sets the variable `$PHPSESSID` equal to the session ID number
- Uses cookies to move `$PHPSESSID` from one page to the next

If the user's browser is set to refuse cookies, PHP does the following:

- **Sets a constant called `SID`.** The constant contains a `variable=value` pair that looks like `PHPSESSID=`*`longstringofnumbers`*.
- **Might or might not move the session ID number from one page to the next, depending on whether `trans-sid` is turned on.** If it is turned on, PHP passes the session ID number; if it is not turned on, PHP does not pass the session ID number.

Turning on `trans-sid` has advantages and disadvantages. The advantages are that sessions work seamlessly even when users turn off cookies and it's much easier to program sessions. The disadvantage is that the session ID number is often passed in the URL. In some situations, the session ID number should not be shown in the browser address. Also, when the session ID number is in the URL, it can be bookmarked by the user. Then, if the user returns to your site by using the bookmark with the session ID number in it, the new session ID number from the current visit can get confused with the old session ID number from the previous visit and possibly cause problems.

Sessions with trans-sid turned on

When `trans-sid` is turned on and the user has cookies turned off, PHP automatically sends the session ID number in the URL or as a hidden form field. If the user moves to the next page by using a link, a `header` function, or a form with the `get` method, the session ID number is added to the URL. If the user moves to the next page by using a form with the `post` method, the session ID number is passed in a hidden field. PHP recognizes `$PHPSESSID` as the session ID number and handles the session without any special programming on your part.



The session ID number is added only to the URLs for pages on your own Web site. If the URL of the next page includes a server name, PHP assumes that the URL is on another Web site and doesn't add the session ID number. For instance, here are two link statements:

```
<a href="newpage.php">
<a href="HTTP://www.company.com/newpage.php">
```

PHP adds the session ID number to the first link, but *not* to the second link.

Sessions without trans-sid turned on

When `trans-sid` is *not* turned on, PHP does *not* send the session ID number to the next page when users have cookies turned off. Rather, you must send the session ID number yourself.

Fortunately, PHP provides a constant that you can use to send the session ID yourself. A *constant* is a variable that contains information that can't be changed. (Constants are described in Chapter 6.) The constant that PHP provides is named `SID` and contains a `variable=value` pair that you can add to the URL, as follows:

```
<a href="nextpage.php?<?php echo SID?>" > next page </a>
```

This link statement adds a question mark (`?`) and the constant `SID` to the URL. `SID` contains the session ID number formatted as `variable=value`. Therefore, the URL that is sent is

```
<a href="nextpage.php?PHPSESSID=877c22163d8df9deb342c7333cfe38a7>
   next page </a>
```

For one of several reasons (which I discuss in the section "Adding information to the URL," earlier in this chapter), you may not want the session ID number to appear in the URL shown by the browser. To prevent that, you can send the session ID number in a hidden field in a form that uses the `post` method. First, get the session ID number; then send it in a hidden field. The statements to do this are

```
<?php
  $PHPSESSID = session_id();
  echo "<form action='nextpage.php' method='POST'>
          <input type='hidden' name='PHPSESSID'
                 value='$PHPSESSID'>
          <input type='submit' value='Next Page'>
        </form>";
?>
```

These statements do the following:

1. Store the session ID number in a variable called `$PHPSESSID`. Use the function `session_id`, which returns the current session ID number.
2. Send `$PHPSESSID` in a hidden form field.

On the new page, PHP automatically uses `$PHPSESSID` to get any session variables without any special programming needed from you.

Making sessions private

PHP session functions are ideal for restricted Web sites that require users to log in with a login name and password. Those Web sites undoubtedly have many pages, and you don't want the user to have to log in to each page. PHP sessions can keep track of whether the user has logged in and refuse access to users that aren't logged in. You can use PHP sessions to do the following:

1. Show users a login page.
2. If a user logs in successfully, set and store a session variable.
3. Whenever a user goes to a new page, check the session variable to see whether the user has logged in.
4. If the user has logged in, show the page.
5. If the user has not logged in, bring up the login page.

To check whether a user has logged in, add the following statements to the top of every page:

```
<?php
  session_start()
  if ( @$_SESSION['login'] != "yes" )
  {
    header("Location: loginPage.php");
    exit();
  }
?>
```

In these statements, `$_SESSION[login]` is a session variable that's set to `"yes"` when the user logs in. The statements check whether `$_SESSION[login]` is equal to `"yes"`. If it is not, the user is not logged in and is sent to the login page. If `$_SESSION[login]` equals `"yes"`, the program proceeds with the rest of the statements on the Web page.

Closing PHP sessions

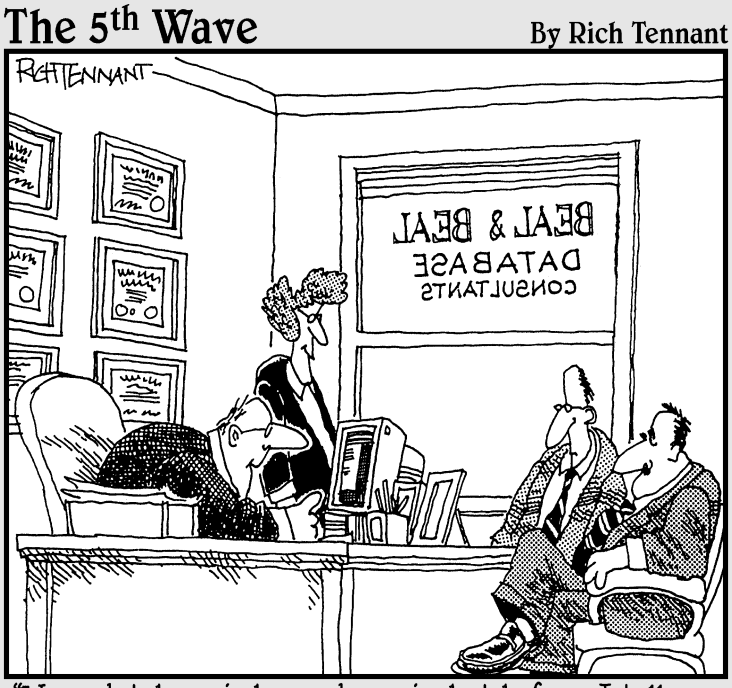
For restricted sessions that users log into, you often want users to log out when they're finished. To close a session, use the following statement:

```
session_destroy();
```

This statement gets rid of all the session variable information stored in the session file. PHP no longer passes the session ID number to the next page. However, the statement does *not* affect the variables currently set on the current page: They still equal the same values. If you want to remove the variables from the current page — as well as prevent them from being passed to the next page — unset them with this statement:

```
unset($_SESSION );
```

Part IV
Applications



"Your database is beyond repair, but before I tell you our backup recommendation, let me ask you a question. How many index cards do you think will fit on the walls of your computer room?"

In this part . . .

In this part, you find out how to take the planning and getting started information from Part I, the MySQL information from Part II, and the PHP information from Part III and put it all together into a dynamic Web database application. Chapters 11 and 12 present two sample applications, complete with their databases and all their PHP programs.

Chapter 10

Putting It All Together

In This Chapter

- Organizing your whole application
- Organizing individual programs
- Making your application secure
- Documenting your application

The previous chapters provide you with the tools you need to build your Web database application. In Part I, you find out how PHP and MySQL work and how to get access to them. In addition, you discover what you need to do to build your application and in what order. In Part II, you find out how to build and use a MySQL database. In Part III, you discover what features PHP has and how to use them. In addition, this part also explains how to show information in a Web page, collect information from users, and store information in a database. Now here, in the first chapter in Part IV, you're ready to put all the pieces together into a complete application. To do this, you need to

- Organize the application
- Make sure that the application is secure
- Document the application

I describe each of these steps in detail.

Organizing the Application

Organizing the application is for your benefit. As far as PHP is concerned, the application could be 8 million PHP statements all on one line of one computer file. PHP doesn't care about lines, indents, or files. However, humans write

and maintain the programs for the application, and humans need organization. Applications require two levels of organization:

- **The application level:** Most applications need more than one program to deliver complete functionality. You must divide the functions of the application into an organized set of programs.
- **The program level:** Most programs perform more than one specific task. You must divide the tasks of the program into sections within the program.

Organizing at the application level

In general, Web database applications consist of one program per Web page. For instance, you might have a program that provides a form to collect information and a program that stores the information in a database and tells the user that the data has been stored.

Another basis for organization is one program per major task. For instance, you might have a program to present the form and a program that stores the data in a database. For Web applications, most major tasks involve sending a Web page. Collecting data from the user requires a Web page for the HTML form; providing product information to customers requires Web pages; and when you store data in a database, you usually want to send a confirmation page to the user that the data was stored.

One program per Web page or one program per major task is not a rule but merely a guideline. The only rule regarding organization is that it must be clear and easy to understand. And that's subjective. Still, the organization of an application such as the Pet Catalog need not be overly complicated. Suppose that the Pet Catalog design calls for the first page to list all the pet types — such as cat, dog, and bird — that the user can select from. Then, after the user selects a type, all the pets in the catalog for that type are shown on the next Web page. A reasonable organization would be two programs: one to show the page of pet types and one to show the pets based on the pet type that was chosen.



Here are a few additional pointers for organizing your programs:

- **Choose descriptive names for the programs in your application.** Program names are part of the documentation that makes your application understandable. For instance, useful names for the Pet Catalog programs might be `ShowPetTypes.php` and `ShowPets.php`. It's usual, but not a requirement, to begin program names with an uppercase letter. Case isn't important for program names on Windows computers, but it's important on Unix and Linux computers. Pay attention to the uppercase and lowercase letters so that your programs can run on any computer if needed.

- **Put program files into subdirectories with meaningful names.** For instance, put all the graphic files into a directory called `images`. If you have only three files, you may be okay with only one directory, but looking through dozens of files for a specific file can waste a lot of time.

Organizing at the program level

A well-organized individual program is important for the following reasons:

- **It's easier for you to write.** The better organized your program is, the easier it is for you to read and understand it. You can see what the program is doing and find and correct problems faster.
- **It's easier for others to understand.** Others may need to understand your program. After you claim that big inheritance and head off to the South Sea Island that you purchased, someone else will have to maintain your application.
- **It's easier for you to maintain.** No matter how thoroughly you test your application, it's likely to have a problem or two. The better organized your program is, the easier it is for you to find and correct problems, especially later.
- **It's easier to change.** At some point, you or someone else will need to change the program. The needs of the user may change. The needs of the business may change. The technology may change. The ozone layer may change. Figuring out what the program does and how it does it so that you can change it is much easier if it is well organized. I guarantee that you won't remember the details; you just need to be able to understand the program.

The following rules will produce well-organized programs. I hesitate to call them *rules* because there can be reasons in a specific environment to break one or more of them — but I strongly recommend that you think carefully before doing so.

- **Divide the statements into sections for each specific task.** Start each section with a comment describing what the section does. Separate sections from each other by adding blank lines. For instance, for the Pet Catalog, the first program might have three sections for three tasks:

  1. **Echo introductory text, such as the page heading and instructions.** The comment before the section might be `/* opening text */`. If the program echoes a lot of complicated text and graphics, you might make it into more than one section, such as `/* title and logo */` and `/* instructions */`.

2. **Get a list of pet types from the database.** If this section is long and complicated, you can divide it into smaller sections, such as a) connect to database; b) execute `SELECT` query; and c) put data into variables.

3. **Create a form that displays a selection list of the pet types.** Forms are often long and complicated. It can be useful to have a section for each part of the form.

- **Use indents.** Indent blocks in the PHP statements. For instance, indent `if` blocks and `while` blocks as I did in the sample code for this book. If blocks are nested inside other blocks, indent the nested block even further. It's much easier to see where blocks begin and end when they're indented, which in turn makes it easier to understand what the program does. Indenting the HTML statements can also be helpful. For instance, if you indent the lines between the open and close tags for a form or between the `<table>` and `</table>` tags, you can more easily see what the statements are doing.
- **Use comments liberally.** Definitely add comments at the beginning that explain what the program does. And add comments for each section. Also, comment any statements that aren't obvious or where you may have done something in an unusual way. If it took you a while to figure out how to do it, it's probably worth commenting. Don't forget short comments on the end of lines; sometimes just a word or two can help.
- **Use simple statements.** Sometimes programmers get carried away with the idea of concise code to the detriment of readability. Nesting six function calls inside each other may save some lines and keystrokes, but it also makes the program more difficult to read.
- **Reuse blocks of statements.** If you find yourself typing the same ten lines of PHP statements in several places in the program, you can move that block of statements into another file and call it when you need it. One line in your program that reads `getData()` is much easier to read than ten lines that get the data. Not only that, if you need to change something within those lines, you can change it in one external file instead of having to find and change it a dozen different places in your program. You can reuse statements in two ways: functions and `include` statements. Chapter 7 explains how to write and use functions. The following two sections explain the use of functions and `include` statements in program organization.
- **Use constants.** If your program uses the same value many times, such as the sales tax for your state, you can define a constant in the beginning of the program that creates a constant called `CA_SALES_TAX` that is `.97` and use it whenever it's needed. Defining a constant that gives the number a name helps anyone reading the program understand what the number is — plus, if you ever need to change it, you have to change it in only one place. Constants are described in detail in Chapter 6.

Using include statements

PHP allows you to put statements into an *external* file — that is, a file separate from your program — and insert the file wherever you want in the program by using an `include` statement. `include` files are useful for storing a block of statements that is repeated. You add an `include` statement wherever you want to use the statements instead of adding the entire block of statements at several locations. It makes your program shorter and easier to read. The format for an `include` statement is

```
include("filename");
```

The file can have any name. I like to use the extension `.inc`. The statements in the file are included, as-is, at the point where the `include` statement is used. The statements are included as HTML, not PHP. Therefore, if you want to use PHP statements in your `include` file, you must include PHP tags in the `include` file. Otherwise, all the statements in the `include` file are seen as HTML and output to the Web page as-is.

Here are some ways to use `include` files to organize your programs:

- **Put all or most of your HTML into `include` files.** For instance, if your program sends a form to the browser, put the HTML for the form into an external file. When you need to send the form, use an `include` statement. Putting the HTML into an `include` file is a good idea if the form is shown several times. It is even a good idea if the form is shown only once because it makes your program much easier to read. The programs in Chapters 11 and 12 put HTML code for forms into separate files and `include` the files when the forms are displayed.
- **Store the information needed to access the database in a file separate from your program.** Store the variable names in the file as follows:

  ```
  <?php
  $host="localhost";
  $user="phpuser";
  $password="secret";
  ?>
  ```

  Notice that this file needs the `php` tags in it because the `include` statement inserts the file as HTML. Include this file at the top of every program that needs to connect to the database. If any of the information (such as the password) changes, just change the password in the `include` file. You don't need to search through every program file to change the password. For a little added security, use a misleading file-name, rather than something obvious like `secret_passwords.inc`.

- **Put your functions in `include` files.** You don't need the statements for functions in the program; you can put them in an `include` file. If you have a lot of functions, organize related functions into several `include` files, such as `data_functions.inc` and `form_functions.inc`. Use `include` statements at the top of your programs, reading in the functions that are used in the program.
- **Store statements that all the files on your Web site have in common.** Most Web sites have many Web pages with many elements in common. For instance, all Web pages start with `<html>`, `<head>`, and `<body>` tags. If you store the common statements in an `include` file, you can include them in every Web page, ensuring that all your pages look alike. For instance, you might have the following statements in an `include` file:

```
<html>
<head><title><?php echo $title ?></title></head>
<body topmargin="0">
<p style="text-align: center">
    <img src="logo.gif" width="100" height="200">
<hr color="red" />
```

  If you include this file at the top of every program on your Web site, you save a lot of typing, and you know that all your pages match. In addition, if you want to change anything about the look of all your pages, you only have to change it in one place — in the `include` file.

PHP provides a related statement — the `include_once` statement. If the specified file has already been included in a previous statement, the file is not included again. The format is as follows:

```
include_once("filename");
```

This statement prevents include files with similar variables from overwriting each other. Use `include_once` when you include your functions.

You can use a variable name for the filename as follows:

```
include("$filename");
```

For example, you might want to display different messages on different days. You might store these messages in files that are named for the day on which the message should appear. For instance, you could have a file named `Sun.inc` with the following content

```
<p>Go ahead. Sleep in. No work today.</p>
```

and similar files for all days of the week. The following statements can be used to display the correct message for the current day:

```
$today  = date("D");
include("$today".".inc");
```

After the first statement, `$today` contains the day of the week, in abbreviation form. The `date` statement is discussed in Chapter 6. The second statement includes the correct file, using the day stored in `$today`. If `$today` contains `Sun`, the statement includes a file called `Sun.inc`.

Protecting your `include` files is important. The best way to protect them is to store them in a directory outside your Web space so they can't be accessed by visitors to your Web site.

You can set up an `include` directory where PHP looks for any files specified in an `include` statement. If you are the PHP administrator, you can set up an `include` directory in the `php.ini` file (the PHP configuration file in your system directory, as I describe in Appendix B). Find the setting for `include_path` and change it to the path to your preferred directory. If a semicolon appears at the beginning of the line, before `include_path`, remove it. The following are examples of `include_path` settings in the `php.ini` file:

```
include_path=".;d:\include";            # for Windows
```

```
include_path=".:/user/include";  # for Unix/Linux/Mac
```

Both statements specify two directories where PHP looks for `include` files. The first directory is . (dot) (meaning the current directory), followed by the second directory path. You can specify as many include directories as you want, and PHP will search them for the `include` file in the order in which they are listed. The directory paths are separated by a semicolon for Windows and a colon for Unix and Linux.

If you don't have access to `php.ini`, you can set the path in each individual script by using the following statement:

```
ini_set("include_path","d:\hidden");
```

This statement sets the `include_path` to the specified directory only while the program is running. It doesn't set the directory for your entire Web site.

To access a file from an `include` directory, just use the filename, as follows. You don't need to use the full path name.

```
include("secretpasswords.inc");
```

If your `include` file is not in an `include` directory, you may need to use the entire path name in the `include` statement. If the file is in the same directory as the program, the filename alone is sufficient. However, if the file is located in another directory, such as a subdirectory of the directory that the program is in or a hidden directory outside the Web space, you need to use the full path name to the file, as follows:

```
include("d:/hidden/secretpasswords.inc");
```

Using functions

Make frequent use of functions to organize your programs. (In Chapter 7, I discuss creating and using functions.) Functions are useful when your program needs to perform the same task at repeated locations in a program or in different programs in the application. After you write a function that does the task and you know it works, you can use it anywhere that you need it.

Look for opportunities to use functions. Your program is much easier to read and understand with a line like this:

```
getMemberData();
```

than with 20 lines of statements that actually get the data. In fact, after you've been writing PHP programs for a while, you will have a stash of functions that you've written for various programs. Very often the program that you're writing can use a function that you wrote for an application two jobs ago. For instance, I often have a need for a list of the states. Rather than include a list of all 50 states every time I need it, I have a function called `getStateNames()` that returns an array that holds the 50 state names in alphabetical order and a function called `getStateCodes()` that returns an array with all 50 two-letter state codes in the same order.

Use descriptive function names. The function calls in your program should tell you exactly what the functions do. Long names are okay. You don't want to see a line in your program that reads

```
function1();
```

Even a line like the following is less informative than it could be:

```
getData();
```

You want to see a line like this:

```
getAllMemberNames();
```

Keeping It Private

You need to protect your Web database application. People out there may have nefarious designs on your Web site for purposes such as

- **Stealing stuff:** They hope to find a file sitting around full of valid credit card numbers or the secret formula for eternal youth.
- **Trashing your Web site:** Some people think this is funny. Some people do it to prove that they can.
- **Harming your users:** A malicious person can add things to your Web site that harm or steal from the people who visit your site.

This is not a security book. Security is a large, complex issue, and I am not a security expert. Nevertheless, I want to call a few issues to your attention and make some suggestions. The following measures will increase the security of your Web site, but if your site handles important, secret information, read some security books and talk to some experts:

- **Ensure the security of the computer that hosts your Web site.** This is probably not your responsibility, but you may want to talk to the people responsible and discuss your security concerns. You'll feel better if you know that someone is worrying about security.
- **Don't let the Web server display filenames.** Users don't need to know the names of the files on your Web site.
- **Hide things.** Store your information so that it can't be easily accessed from the Web.
- **Don't trust information from users.** Always clean any information that you didn't generate yourself.
- **Use a secure Web server.** This requires extra work, but it's important if you have top-secret information.

Ensure the security of the computer

First, the computer itself must be secure. The system administrator of the computer is responsible for keeping unauthorized visitors and vandals out of the system. Security measures include such things as firewalls, encryption, password shadowing, and scan detectors. In most cases, the system administrator is not you. If it is, you need to do some serious investigation into security issues. If you are using a Web-hosting company, you may want to discuss security with those folks to reassure yourself that they are using sufficient security measures.

Don't let the Web server display filenames

You may have noticed that sometimes you get a list of filenames when you point at a URL. If you point at a directory (rather than a specific file) and the directory doesn't contain a file with the default filename (such as `index.html`), the Web server may display a list of files for you to select from. You probably don't want your Web server to do this; your site won't be very secure if a visitor can look at any file on your site. On other Web sites, you may have seen an error message that reads

```
Forbidden
You don't have permission to access /secretdirectory on this server.
```

On those sites, the Web server is set so that it doesn't display a list of filenames when the URL points to a directory. Instead, it delivers this error message. This is more secure than listing the filenames. If the filename is being sent from your Web site, a setting for the Web server needs to be changed. If you aren't the administrator for your Web server, request a change. If you are the administrator, it's up to you to change this behavior. For instance, in Apache, this behavior is controlled by an option called *Indexes,* which can be turned on or off in the `httpd.conf` file as follows:

```
Options Indexes          // turns it on
Options -Indexes         // turns it off
```

See the documentation for your Web server to allow or not allow directory listings in the user's Web browser.

Hide things

Keep information as private as possible. Of course, the Web pages that you want visitors to see must be stored in your public Web space directory. But not everything needs to be stored there. For instance, you can store `include` files in another location altogether — in space on the computer that can't be accessed from the Web. Your database certainly isn't stored in your Web space, but it might be even more secure if it was stored on a different computer.

Another way to hide things is to give them misleading names. For instance, the include file containing the database variables shouldn't be called `passwords.inc`. A better name might be `UncleHenrysChickenSoupRecipe.inc`. I know this suggestion violates other sections of the book where I promote informative filenames, but this is a special case. Malicious people sometimes do obvious things like typing `www.yoursite.com/passwords.html` into their browser to see what happens.

Don't trust information from users

Malicious users can use the forms in your Web pages to send dangerous text to your Web site. Therefore, never store information from forms directly into a database without checking, cleaning, and escaping it first. Check the information that you receive for reasonable formats and dangerous characters. In particular, you don't want to accept HTML tags, such as `<script>` tags, from forms. By using script tags, a user could enter an actual script — perhaps a malicious one. If you accept the form field without checking it and store it in your database, you could have any number of problems, particularly if the stored script was sent in a Web page to a visitor to your Web site. For more on checking, cleaning, and escaping data from forms, see Chapter 8.

Use a secure Web server

Communication between your Web site and its visitors is not totally secure. When the files on your Web site are sent to the user's browser, someone on the Internet between you and the user can read the contents of these files as they pass by. For most Web sites, this isn't an issue; however, if your site collects or sends credit card numbers or other secret information, use a secure Web server to protect this data.

Secure Web servers use Security Sockets Layer (SSL) to protect communication sent to and received from browsers. This is similar to the scrambled telephone calls that you hear about in spy movies. The information is *encrypted* (translated into coded strings) before it is sent across the Web. The receiving software decrypts it into its original content. In addition, your Web site uses a certificate that verifies your identity. Using a secure Web server is extra work, but it's necessary for some applications.



You can tell when you are communicating using SSL. The URL begins with *HTTPS,* rather than *HTTP.*

Information about secure Web servers is specific to the Web server that you're using. To find out more about using SSL, look at the Web site for your Web server. For instance, if you are using Apache, check out two open source projects that implement SSL for Apache at `www.modssl.org` and `www.apache-ssl.org`. Commercial software is also available that provides a secure server based on the Apache Web server. If you're using Microsoft Internet Information Server (IIS), search for *SSL* on the Microsoft Web site at `www.microsoft.com`.

Completing Your Documentation

I'm making one last pitch here. Documenting your Web database application is essential. You start with a plan describing what the application is supposed to do. Based on your plan, you create a database design. Keep the plan and the design up to date. Often, as a project moves along, changes are made. Make sure that your documentation changes to match the new decisions.

While you design your programs, associate the tasks in the application plan with the programs that you plan to write. List the programs and what each one will do. If the programs are complicated, you may want to include a brief description of how the program will perform its tasks. If this is a team effort, list who is responsible for each program. When you complete your application, you should have the following documents:

- **Application plan:** Describes what the application is supposed to do, listing the tasks that it will perform
- **Database design:** Describes the tables and fields in the database
- **Program design:** Describes how the program(s) will perform the tasks in the application plan
- **Program comments:** Describe the details of how the individual program works

Pretend that it's five years in the future and you're about to do a major rewrite of your application. What will you need to know about the application to change it? Be sure that you include all the information that you need in your documentation.

Chapter 11

Building an Online Catalog

In This Chapter

- Designing an online catalog
- Building the database for the Pet Catalog
- Designing the Web pages for the Pet Catalog
- Writing the programs for the Pet Catalog

Online catalogs are everywhere on the Web. Every business that has products for sale can use an online catalog. Some businesses use online catalogs to sell their products online, and some use them to show the quality and worth of their products to the world. Many customers have come to expect businesses to be online and provide information about their products. Customers often begin their search for a product online, researching its availability and cost through the Web.

In this chapter, you find out how to build an online catalog. I chose a pet store catalog for no particular reason except that it sounded like more fun than a catalog of socks or light bulbs. And looking at the pictures for a pet catalog was much more fun than looking at pictures of socks. I introduce the Pet Catalog example in Chapter 3 and use it for many of the examples throughout this book.

In general, all catalogs do the same thing: provide product information to potential customers. The general purpose of the catalog is to make it as easy as possible for customers to see information about the products. In addition, you want to make the products look as attractive as possible so that customers will want to purchase them.

Designing the Application

The first step in design is to decide what the application should do. The obvious purpose of the Pet Catalog is to show potential customers information about the pets. A pet store might also want to show information about pet

products, such as pet food, cages, fish tanks, and catnip toys . . . but you decide not to include such items in your catalog. The purpose of your online catalog application is to show just pets.

For the customer, displaying the information is the sole function of the catalog. From your perspective, however, the catalog also needs to be maintained; that is, you need to add items to the catalog. So, you must include the task of adding items to the catalog as part of the catalog application. Thus, the application has two distinct functions:

- Show pets to the customers
- Add pets to the catalog

Showing pets to the customers

The basic purpose of your online catalog is to let customers look at pets. Customers can't purchase pets online, of course. Sending pets through the mail isn't feasible. But a catalog can showcase pets in a way that motivates customers to rush to the store to buy them.

If your catalog contains only three pets, your catalog can be pretty simple — one page showing the three pets. However, most catalogs have many more items than that. Usually, a catalog opens with a list of the types of products, such as cat, dog, horse, and dragon. Customers select the type of pet they want to see, and the catalog then displays the individual pets of that type. For example, if the customer selects dog, the catalog would then show collies, spaniels, and wolves. Some types of products might have more levels of categories before you see individual products. For instance, furniture might have three levels rather than two. The top level might be the room, such as kitchen or bedroom. The second level might be type, such as chairs or tables. The third level would be the individual products.

The purpose of a catalog is to motivate those who look at it to make a purchase immediately. For the Pet Catalog, pictures are a major factor in motivating customers to make a purchase. Pictures of pets make people go ooooh and aaaah and say, "Isn't he cuuuute!" This generates sales. The main purpose of your Pet Catalog is to show pictures of pets. In addition, the catalog also should show descriptions and prices.

To show the pets to customers, the Pet Catalog will do the following:

1. Show a list of the types of pets and allow the customer to select a type.
2. Show information about the pets that match the selected type. The information includes the description, the price, and a picture of the pet.

Adding pets to the catalog

You can add items to your catalog in several ways, but the easiest way is to use an application designed for the purpose. In many cases, you won't be the person who will be adding products to your catalog. One reason for adding maintenance functionality to your catalog application is so someone else can do those boring maintenance tasks. The easier it is to maintain your catalog, the less likely that errors will sneak into it.

An application to add a pet to your catalog should do the following:

1. **Prompt the user to enter a pet type for the pet.** A selection list of possible pet types would eliminate many errors, such as alternate spellings (*dog* and *dogs*) and misspellings. The application also needs to allow the user to add new categories when needed.
2. **Prompt the user to enter a name for the pet,** such as *collie* or *shark.* A selection list of names would help prevent mistakes. The application also needs to allow the user to add new names when needed.
3. **Prompt the user to enter the pet information for the new pet.** The application should clearly specify what information is needed.
4. **Store the information in the catalog.**

The catalog entry application can check the data for mistakes and enter the data into the correct locations. The person entering the new pet doesn't need to know the inner workings of the catalog.

Building the Database

The catalog itself is a database. It doesn't have to be a database; it's possible to store a catalog as a series of HTML files that contain the product information in HTML tags and display the appropriate file when the customer clicks a link. However, it makes my eyes cross to think of maintaining such a catalog. Imagine the tedium of adding and removing catalog items manually — or finding the right location for each item by searching through many files. Ugh. For these reasons, putting your Pet Catalog in a database is better.

The `PetCatalog` database contains all the information about pets. It uses three tables:

- `Pet` table
- `PetType` table
- `Color` table

The first step in building the Pet Catalog is to build the database. It's pretty much impossible to write programs without a working database to test the programs on. First you design your database; then you build it; then you add the data (or at least some sample data to use while developing the programs).

Building the Pet table

In your design for the Pet Catalog, the main table is the `Pet` table. It contains the information about the individual pets that you sell. The following SQL query creates the `Pet` table:

```
CREATE TABLE Pet (
  petID           INT(5)         SERIAL,
  petName         CHAR(25)       NOT NULL,
  petType         CHAR(15)       NOT NULL DEFAULT "Misc",
  petDescription  VARCHAR(255),
  price           DECIMAL(9,2),
  pix             CHAR(15)       NOT NULL DEFAULT "na.gif",
  PRIMARY KEY(petID) );
```

Each row of the `Pet` table represents a pet. The columns are as follows:

- `petID`: A sequence number for the pet. In another catalog, this might be a product number, a serial number, or a number used to order the product. The `petID` column is the primary key, which must be unique. MySQL will not allow two rows to be entered with the same `petID`.

  The `CREATE` query defines the `petID` column as `SERIAL` (added in MySQL 4.1). `SERIAL` is a keyword that defines the column in the following ways:

  - `BIGINT`: The data in the field is expected to be a numeric integer, with a range up to 18446744073709551615. The database won't accept a character string in this field.
  - `UNSIGNED`: The integer in the field can't be a negative number.
  - `NOT NULL`: This definition means that this field can't be empty. It must have a value. The primary key must always be `NOT NULL`.
  - `AUTO-INCREMENT`: This definition means that the field will automatically be filled with a sequential number if you don't provide a specific number. For example, if a row is added with 98 for a `petID`, the next row will be added with 99 for the `petID` unless you specify a different number. This is a useful way of specifying a column with a unique number, such as a product number or an order number. You can always override the automatic sequence number with a number of your own, but if you don't provide a number, a sequential number is stored.

- `petName`: The name of the pet, such as lion, collie, or unicorn. The `CREATE` query defines the `petName` column in the following ways:
  - `CHAR(25)`: The data in this field is expected to be a character string that's 25 characters long. If the stored string is less than 25 characters, the field will be padded so that it always takes up 25 characters of storage.
  - `NOT NULL`: This definition means that this field can't be empty. It must have a value. After all, it wouldn't make much sense to have a pet in the catalog without a name.
  - No default value: If you try to add a new row to the `Pet` table without a `petName`, it won't be added. It doesn't make sense to have a default name for a pet.
- `petType`: The type of pet, such as dog or fish. The `CREATE` query defines the `petType` column in the following ways:
  - `CHAR(15)`: The data in this field is expected to be a character string that's 15 characters long. If the stored string is less than 15 characters, the field will be padded so that it always takes up 15 characters of storage.
  - `NOT NULL`: This definition means that this field can't be empty. It must have a value. The online catalog application will show categories first and then pets within a category, so a pet with no category will never be shown on the Web page.
  - `DEFAULT "Misc"`: The value `"Misc"` is stored if you don't provide a value for `petType`. This ensures that a value is always stored for `petType`.
- `petDescription`: A description of the pet. The `CREATE` query defines `petDescription` in the following way:
  - `VARCHAR(255)`: This data type defines the field as a variable character string that can be up to 255 characters long. The field is stored in its actual length.
- `price`: The price of the pet. The `CREATE` query defines `price` in the following way:
  - `DECIMAL(9,2)`: This data type defines the field as a decimal number that can be up to nine digits and has two decimal places. If you store an integer in this field, it will be returned with two decimal places, such as 9.00 or 2568.00.
- `pix`: The filename of the picture of the pet. Pictures on a Web site are stored in graphic files with names like `dog.jpg`, `dragon.gif`, or `cat.png`. This field stores the filename for the picture that you want to show for this pet. The `CREATE` query defines `pix` in the following ways:
  - `CHAR(15)`: The data in this field is expected to be a character string that's 15 characters long. For some applications, the picture files might be in other directories or on other Web sites requiring a

longer field, but for this application, the pictures are all in a directory on the Web site and have short names. If the stored string is less than 15 characters, the field will be padded so that it always takes up 15 characters of storage.

- `NOT NULL`: This definition means that this field can't be empty. It must have a value. You need a picture for the pet. When a Web site tries to show a picture that can't be found, it displays an ugly error message in the browser window where the graphic would go. You don't want your catalog to do that, so your database should require a value. In this case, you define a default value so that a value will always be placed in this field.
- `DEFAULT "na.gif"`: The value `"na.gif"` is stored if you don't provide a value for `pix`. In this way, a value is always stored for `pix`. The `na.gif` file might be a graphic that reads something like: `"picture not available"`.

Notice the following points about this database table design:

- **Some fields are `CHAR`, and some are `VARCHAR`.** `CHAR` fields are faster, whereas `VARCHAR` fields are more efficient. Your decision on which to use will depend on whether disk space or speed is more important for your application in your environment.

  In general, shorter fields should be `CHAR` because shorter fields don't waste much space. For instance, if your `CHAR` is 5 characters, the most space that you could possibly waste is 4. However, if your `CHAR` is 200, you could waste 199. Therefore, for short fields, use `CHAR` for speed with very little wasted space.

- **The `petID` field means different things for different pets.** The `petID` field assigns a unique number to each pet. However, a unique number is not necessarily meaningful in all cases. For example, a unique number is meaningful for an individual kitten but not for an individual goldfish.

  There are really two kinds of pets. One is the unique pet, such as a puppy or a kitten. After all, the customer buys a specific dog — not just a generic dog. The customer needs to see the picture of the actual animal. On the other hand, some pets are not especially unique, such as a goldfish or a parakeet. When customers purchase a goldfish, they see a tank full of goldfish and point at one. The only real distinguishing characteristic of a goldfish is its color. The customer just needs to see a picture of a generic goldfish, perhaps showing the possible colors — not a picture of the individual fish.

  In your catalog, you have both kinds of pets. The catalog might contain several pets with the name *cat* but with different `petID`s. The picture would show the individual pet. The catalog also contains pets that aren't individuals but that represent generic pets, such as goldfish. In this case, there's only one entry with the name *goldfish,* with a single `petID`.

I've used both kinds of pets in this catalog to demonstrate the different kinds of products that you might want to include in a catalog. The unique item catalog might include such products as artwork or vanity license plates. When the unique item is sold, it's removed from the catalog. Most products are more generic, such as clothing or automobiles. Although a picture shows a particular shirt, many identical shirts are available. You can sell the shirt many times without having to remove it from the catalog.

Building the PetType table

You assign each pet a type, such as dog or dragon. The first Web page of the catalog lists the types for the customer to select from. A description of each type is also helpful. You don't want to put the type description in the main `Pet` table because the description would be the same for all pets with the same category. Repeating information in a table violates good database design.

The `PetCatalog` database includes a table called `PetType` that holds the type descriptions. The following SQL query creates the `PetType` table:

```
CREATE TABLE PetType (
  petType         CHAR(15)      NOT NULL,
  typeDescription VARCHAR(255),
  PRIMARY KEY(petType)  );
```

Each row of this table represents a pet type. These are the columns:

- `petType`: The type name. Notice that the `petType` column is defined the same in the `Pet` table (which I describe in the preceding section) and in this table. This makes table joining possible and makes matching rows in the tables much easier. However, `petType` is the primary key in this table but not in the `Pet` table. The `CREATE` query defines the `petType` column in the following ways:
  - `CHAR(15)`: The data in this field is expected to be a character string that's 15 characters long.
  - `PRIMARY KEY(petType)`: This definition sets the `petType` column as the primary key. This is the field that must be unique. MySQL will not allow two rows to be entered with the same `petType`.
  - `NOT NULL`: This definition means that this field can't be empty. It must have a value. The primary key must always be `NOT NULL`.
- `typeDescription`: A description of the pet type. The `CREATE` query defines the `typeDescription` in the following way:
  - `VARCHAR(255)`: The string in this field is expected to be a variable character string that can be up to 255 characters long. The field is stored in its actual length.

Building the Color table

When I discuss building the `Pet` table (see "Building the Pet table," earlier in this chapter), I discuss the different kinds of pets: pets that are unique (such as puppies) and pets that are not unique (such as goldfish). For unique pets, the customer needs to see a picture of the actual pet. For pets that aren't unique, the customer needs to see only a generic picture.

In some cases, generic pets come in a variety of colors, such as blue parakeets and green parakeets. You might want to show two pictures for parakeets: a picture of a blue parakeet and a picture of a green parakeet. However, because most pets aren't this kind of generic pet, you don't want to add a color column to your main `Pet` table because it would be blank for most of the rows. Instead, you create a separate table containing only pets that come in more than one color. Then when the catalog application is showing pets, it can check the `Color` table to see whether there's more than one color available — and if there is, it can show the pictures from the `Color` table.

The `Color` table points to pictures of pets when the pets come in different colors so that the catalog can show pictures of all the available colors. The following SQL query creates the `Color` table:

```
CREATE TABLE Color (
  petName          CHAR(25)       NOT NULL,
  petColor         CHAR(15)       NOT NULL,
  pix              CHAR(15)       NOT NULL DEFAULT "na.gif",
  PRIMARY KEY(petName,petColor)  );
```

Each row represents a pet type. The columns are as follows:

- `petName`: The name of the pet, such as lion, collie, or Chinese bearded dragon. Notice that the `petName` column is defined the same in the `Pet` table and in this table. This makes table joining possible and makes matching rows in the tables much easier. However, the `petName` is the primary key in this table but not in the `Pet` table. The `CREATE` query defines the `petName` in the following ways:
  - `CHAR(25)`: The data in this field is expected to be a character string that's 25 characters long.
  - `PRIMARY KEY(petName,petColor)`: The primary key must be unique. For this table, two columns together are the primary key — this column and the `petColor` column. MySQL won't allow two rows to be entered with the same `petName` *and* `petColor`.

- NOT NULL: This definition means that this field can't be empty. It must have a value. The primary key must always be NOT NULL.

- petColor: The color of the pet, such as orange or purple. The CREATE query defines the petColor in the following ways:
  - CHAR(15): This data type defines the field as a character string that's 15 characters long.
  - PRIMARY KEY(petName,petColor): The primary key must be unique. For this table, two columns together are the primary key — this column and the petName column. MySQL won't allow two rows to be entered with the same petName *and* petColor.
  - NOT NULL: This definition means that this field can't be empty. It must have a value. The primary key must always be NOT NULL.
- pix: The filename containing the picture of the pet. The CREATE query defines pix in the following ways:
  - CHAR(15): This data type defines the field as a character string that's 15 characters long.
  - NOT NULL: This definition means that this field can't be empty. It must have a value. You need a picture for the pet. When a Web site tries to show a picture that can't be found, it displays an ugly error message in the browser window where the graphic would go. You don't want your catalog to do that, so your database should require a value. In this case, the CREATE query defines a default value so that a value will always be placed in this field.
  - DEFAULT "na.gif": The value "na.gif" is stored if you don't provide a value for pix. In this way, a value is always stored for pix. The file na.gif might contain a graphic that reads something like picture not available.

Adding data to the database

You can add the data to the database in many ways. You can use SQL queries to add pets to the database, or you can use the application that I describe in this chapter. My personal favorite during development is to add a few sample items to the catalog by reading the data from a file. Then, whenever my data becomes totally bizarre during development (as a result of programming errors or my weird sense of humor), I can re-create the data in a moment. Just DROP the table, re-create it with the SQL query, and reread the sample data.

For example, the data file for the `Pet` table might look like this:

```
<TAB>Pekinese<TAB>Dog<TAB>Small, cute, energetic. Good
          alarm system.<TAB>100.00<TAB>peke.jpg
<TAB>House cat<TAB>Cat<TAB>Yellow and white cat. Extremely
          playful. <TAB>20.00<TAB>catyellow.jpg
<TAB>House cat<TAB>Cat<TAB>Black cat. Sleek, shiny. Likes
          children. <TAB>20.00<TAB>catblack.jpg
<TAB>Chinese Bearded Dragon<TAB>Lizard<TAB>Grows up to 2
          feet long. Fascinating to watch. Likes to be
          held.<TAB>100.00<TAB>lizard.jpg
<TAB>Labrador Retriever<TAB>Dog<TAB>Black dog. Large,
          intelligent retriever. Often selected as guide
          dogs for the blind.<TAB>100.00<TAB>lab.jpg
<TAB>Goldfish<TAB>Fish<TAB>Variety of colors. Inexpensive.
          Easy care. Good first pet for small
          children.<TAB>2.00<TAB>goldfish.jpg
<TAB>Shark<TAB>Fish<TAB>Sleek. Powerful. Handle with
          care.<TAB>200.00<TAB>shark.jpg
<TAB>Asian Dragon<TAB>Dragon<TAB>Long and serpentine.
          Often gold or red.<TAB>10000.00<TAB>dragona.jpg
<TAB>Unicorn<TAB>Horse<TAB>Beautiful white steed with
          spiral horn on forehead.<TAB>20000.00<TAB>
          unicorn.jpg
```

These are the data file rules:

- The `<TAB>` tags represent real tabs — the kind that you create by pressing the Tab key.
- Each line represents one pet and must be entered without pressing the Enter or Return key. The lines in the preceding example are shown wrapped to more than one line so that you can see the entire line. However, in the actual file, the data lines are one on each line.
- A tab appears at the beginning of each line because the first field is not being entered. The first field is the `petID`, which is entered automatically; you don't need to enter it. However, you do need to use a tab so that MySQL knows there's a blank field at the beginning.

You can then use an SQL query to read the data file into the `Pet` table:

```
LOAD DATA LOCAL INFILE "pets" INTO TABLE Pet;
```

Any time the data table gets odd, you can re-create it and read in the data again.



The `LOAD DATA LOCAL` query might not be available in your version of MySQL. This query must be enabled before you can use it. If it's not enabled, you will see an error that reads `The used command is not allowed with this MySQL version.` I mention `LOAD DATA LOCAL` also in Chapter 4.

Designing the Look and Feel

After you know what the application is going to do and what information the database contains, you can design the look and feel of the application. The look and feel includes what the user sees and how the user interacts with the application. Your design should be attractive and easy to use. You can plan out this design on paper, indicating what the user sees, perhaps with sketches or with written descriptions. In your design, include the user interaction components, such as buttons or links, and describe their actions. You should include each page of the application in the design. If you're lucky, you know a graphic designer who can develop beautiful Web pages for you. If you're me, you just do your best with a limited amount of graphic know-how.

The Pet Catalog has two look and feel designs: one for the catalog that the customer sees, and another, less fancy one for the part of the application that you or whoever is adding pets to the catalog uses.

Showing pets to the customers

The application includes three pages that customers see:

- **The storefront page:** This is the first page that customers see. It states the name of the business and the purpose of the Web site.
- **The pet type page:** This page lists all the types of pets and allows customers to select which type of pet they want to see.
- **The pets page:** This page shows all the pets of the selected type.

Storefront page

The storefront page is the introductory page for the Pet Store. Because most people already know what a pet store is, this page doesn't need to provide much explanation. Figure 11-1 shows the storefront page. The only customer action available on this page is a link that the customer can click to see the Pet Catalog.

Pet type page

The pet type page lists all the types of pets in the catalog. Each pet type is listed with its description. Figure 11-2 shows the pet type page. Radio buttons appear next to each pet type so that customers can select the type of pet that they want to see.



Figure 11-1: The opening page of the Pet Store Web site.

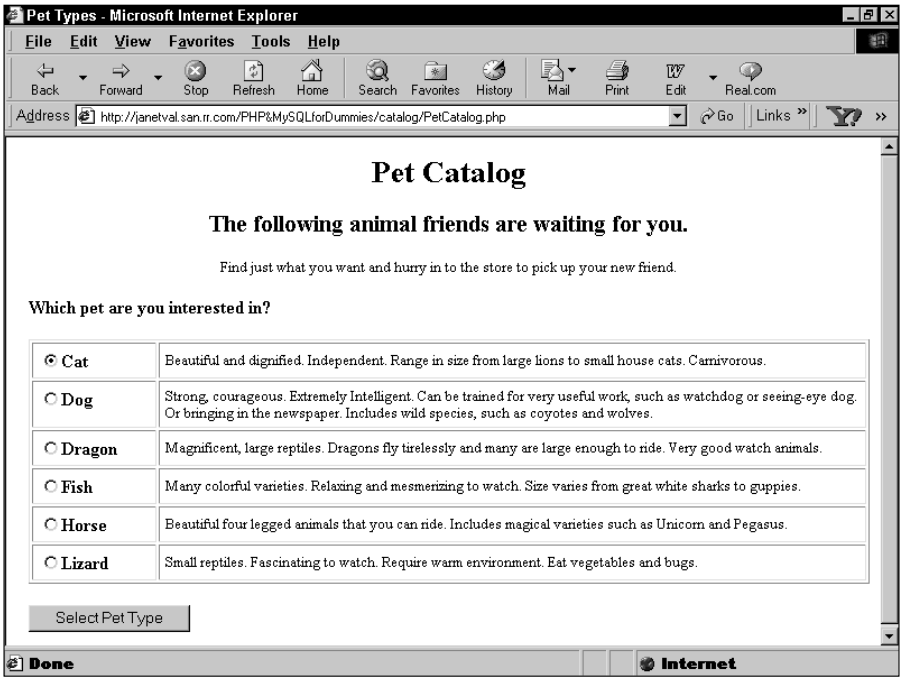


Figure 11-2: The pet type page of the Pet Store Web site.

Pets page

The pets page lists all the pets of the selected type. Each pet is listed with its pet ID, description, price, and picture. The pets page appears in a different format, depending on the information in the catalog database.

Figures 11-3, 11-4, and 11-5 show some possible pets pages.

Figure 11-3 shows a page listing three different dogs from the catalog. Figure 11-4 shows that more than one pet can have the same pet name. Notice that the house cats have different pet ID numbers. Figure 11-5 shows the output when pets are found in the `Color` table, indicating that more than one color is available.

On all these pages, a line at the top reads `Click on any picture to see a larger version.` If the customer clicks the picture, a larger version of the picture is displayed.

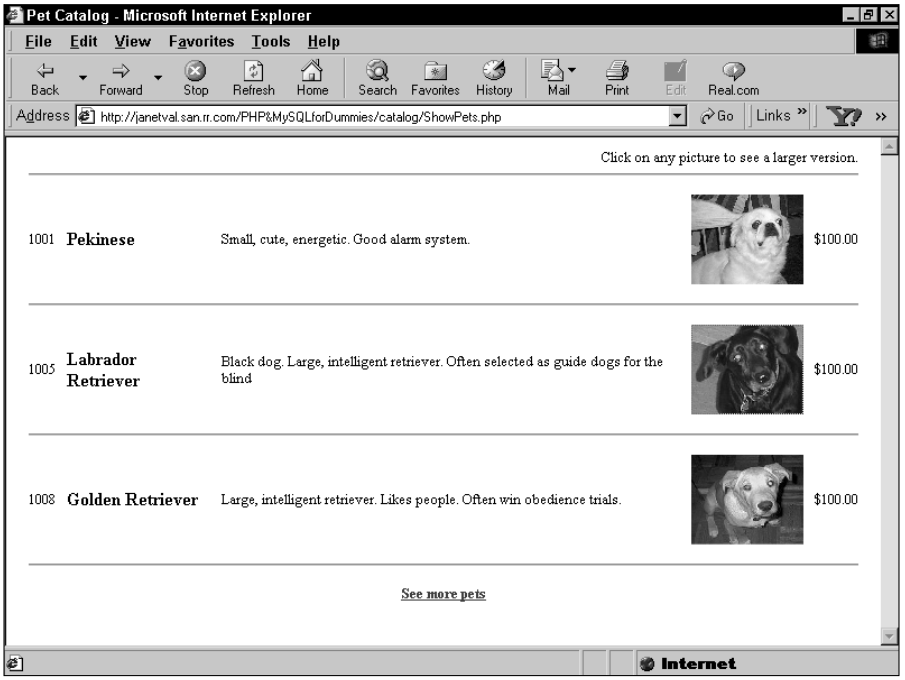


Figure 11-3: This pets page shows three different dogs.

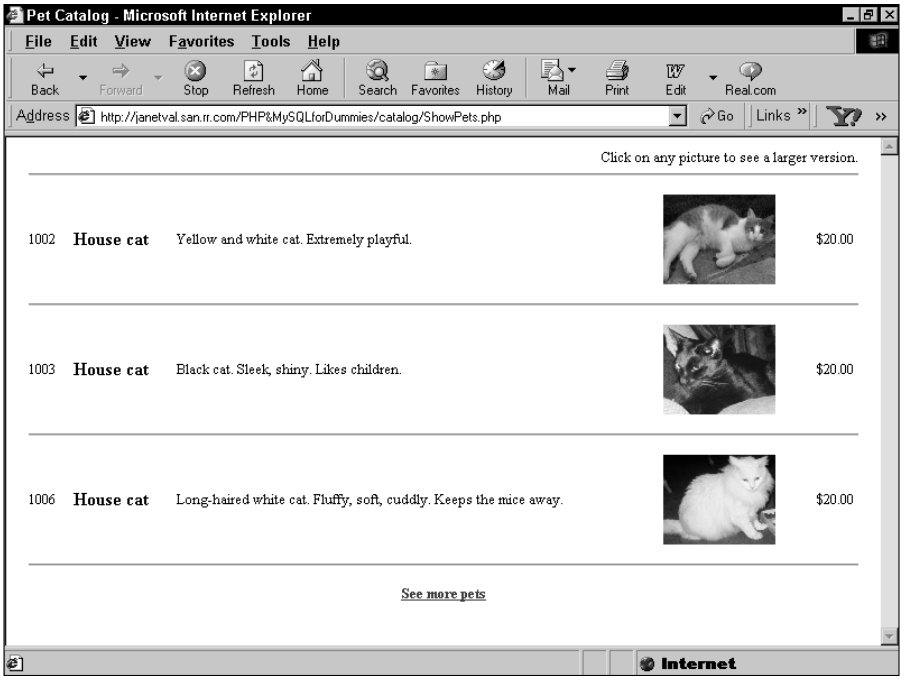


Figure 11-4: This pets page shows three cats with the same pet name.

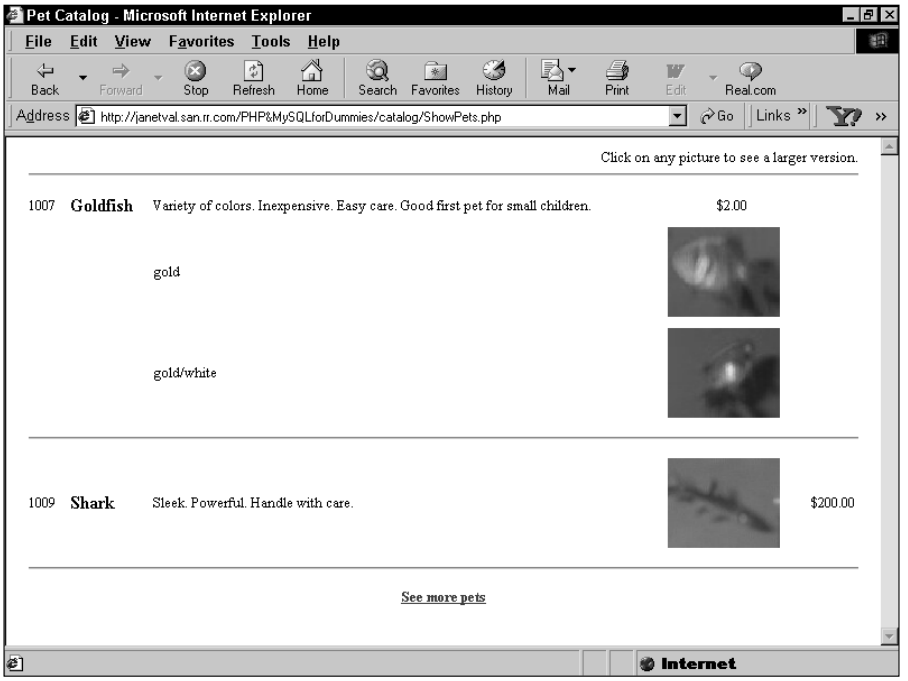


Figure 11-5: This pets page shows goldfish that are available in two colors.

Adding pets to the catalog

The application includes three pages that customers don't see; these are the pages used to add pets to the Pet Catalog. The three pages work in sequential order to add a single pet:

1. **Get pet type page.** The person adding a pet to the catalog selects the radio button for the pet type. The user can also enter a new pet type.
2. **Get pet information page.** The user selects the radio button for the pet being added and fills in the pet description, price, and picture filename. The user can also enter a new pet name.
3. **Feedback page.** A page is displayed showing the pet information that was added to the catalog.

Get pet type page

The first page gets the pet type for the pet that needs to be added to the catalog. Figure 11-6 shows the get pet type page. Notice that all the pet types currently in the catalog are listed, and a section is provided where the user can enter a new pet type if it's needed.

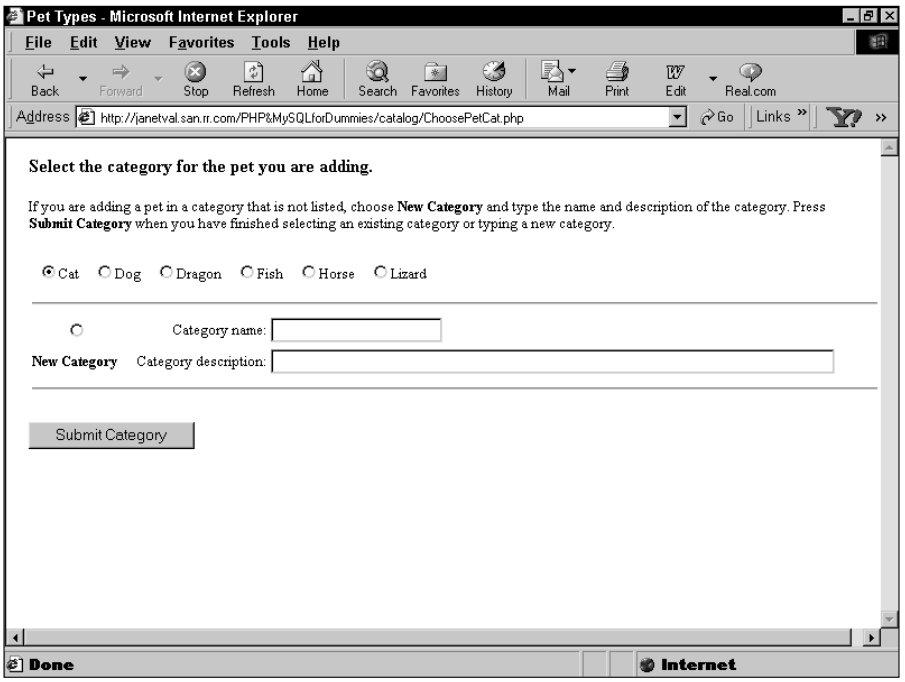


Figure 11-6: The first page for adding a pet to the catalog.

Get pet information page

Figure 11-7 shows the second page. This page lets the user type the information about the pet that goes in the catalog. This page lists all the pet names in the catalog for the selected pet type so that the user can select one. It also provides a section where the user can type a new pet name if needed.

Feedback page

When the user submits the pet information, that information is added to the `PetCatalog` database. Figure 11-8 shows a page that verifies the information that was added to the database. The user can click a link to return to the first page and add another pet.

Get missing information page

The application checks the data to see that the user entered the required information and prompts the user for any information that isn't entered. For instance, if the user selects New Category on the first page, the user must type a category name and description. If the user doesn't type the name or the description, a page is displayed that points out the problem and requests the information. Figure 11-9 shows the page that users see if they forget to type the category name and description.

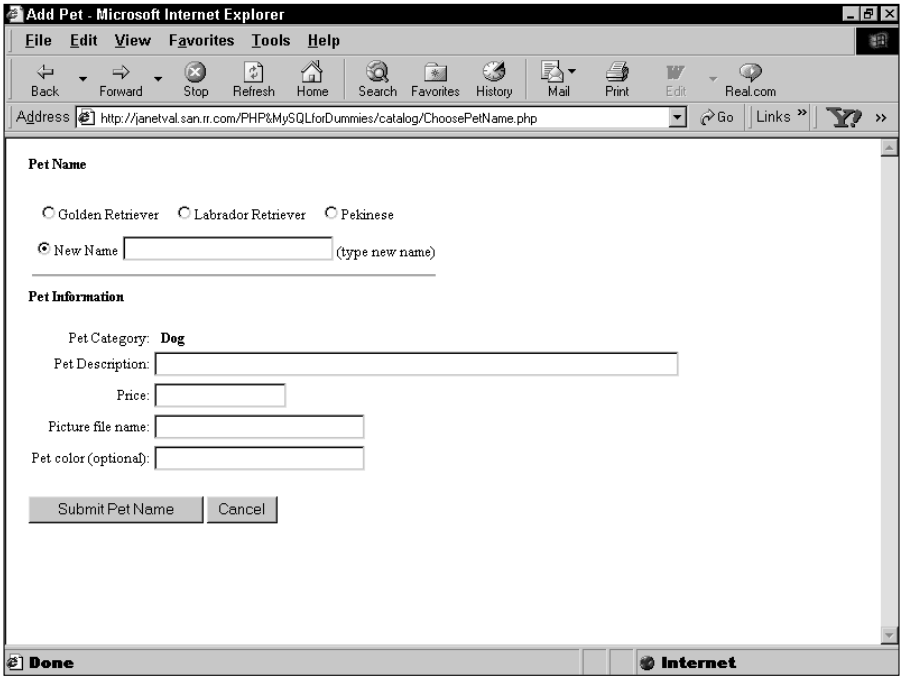


Figure 11-7: The second page asks for the pet name.

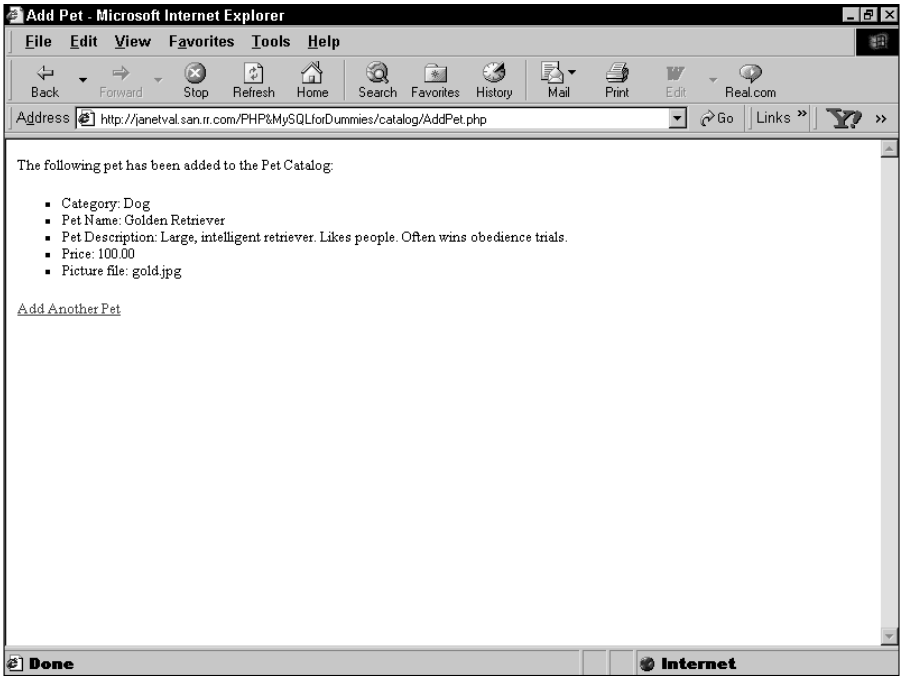


Figure 11-8: The last page provides feedback.

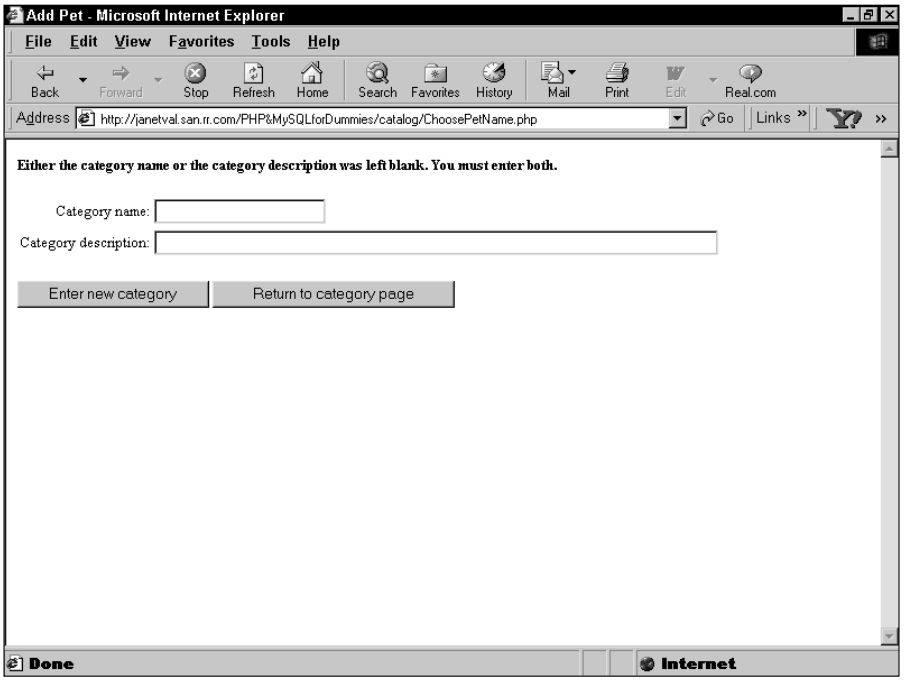


Figure 11-9: This page requests a new category and description.

Writing the Programs

After you know what the pages are going to look like and what they are going to do, you can write the programs. In general, you write a program for each page, although sometimes it makes sense to separate programs into more than one file or to combine programs on a page. (For details on how to organize applications, see Chapter 10.)

As I discuss in Chapter 10, keep the information needed to connect to the database in a separate file and include that file in all the programs that need to access the database. The file should be stored in a secure location and with a misleading name for security reasons. For this application, the following information is stored in a file named `misc.inc`:

```
<?php
  $user="catalog";
  $host="localhost";
  $password="";
  $database="PetCatalog";
?>
```

The Pet Catalog application has two independent sets of programs: one set to show the Pet Catalog to customers and one set to enter new pets into the catalog.

Showing pets to the customers

The application that shows the Pet Catalog to customers has three basic tasks:

- Show the storefront page, with a link to the catalog.
- Show a page where users select the pet type.
- Show a page with pets of the selected pet type.

Showing the storefront

The storefront page doesn't need any PHP statements. It simply displays a Web page with a link. HTML statements are sufficient to do this. Listing 11-1 shows the HTML file that describes the storefront page.

Listing 11-1: HTML File for the Storefront Page

```
<?php
  /* Program: PetShopFront.php
   * Desc:    Displays opening page for Pet Store.
   */
```

```
?>
<html>
<head><title>Pet Store Front Page</title></head>
<body style="margin: 0">
<table width="100%" height="100%" border="0"
       cellspacing="0" cellpadding="0">
  <tr>
    <td style="text-align: center" valign="top">
      <img src="../images/awning-top.gif" alt="awning" />
    </td>
  </tr>
  <tr>
    <td style="text-align: center" valign="top">
      <img src="../images/Name.gif" alt="Pet Store">
      <p style="margin-top: 40pt">
      <img src="../images/lizard-front.jpg"
           alt="animal picture"
        height="186" width="280" />
      <h2>Looking for a new friend?</h2>
      <p>Check out our
          <a href="PetCatalog.php">Pet Catalog.</a>
        <br> We may have just what you're looking for.</p>
    </td>
  </tr>
</table>
</body></html>
```

Notice that the link is to a PHP program called `PetCatalog.php`. When the customer clicks the link, the Pet Catalog program (`PetCatalog.php`) begins.

Showing the pet types

The pet type page (refer to Figure 11-2) shows the customer a list of all the types of pets currently in the catalog. Listing 11-2 shows the program that produces the pet type Web page.

Listing 11-2: Displaying Pet Types

```
<?php
  /* Program: PetCatalog.php
   * Desc:    Displays a list of pet categories from the
              PetType table. Includes descriptions.
   *          Displays radio buttons for user to check.
   */
?>
<html>
<head><title>Pet Types</title></head>
<body>
<?php
```

(continued)

Listing 11-2 *(continued)*

```
   include("misc.inc");                                          #12

   $cxn = mysqli_connect($host,$user,$passwd,$dbname)            #14
          or die ("couldn't connect to server");

   /* Select all categories from PetType table */
   $query = "SELECT * FROM PetType ORDER BY petType";            #18
   $result = mysqli_query($cxn,$query)
             or die ("Couldn't execute query.");                 #20

   /* Display text before form */
   echo "<div style='margin-left: .1in'>\n
   <h1 style='text-align: center'>Pet Catalog</h1>\n
   <h2 style='text-align: center'>The following animal
       friends are waiting for you.</h2>\n
   <p style='text-align: center'>Find just what you want
       and hurry in to the store to pick up your
       new friend.</p>
   <h3>Which pet are you interested in?</h3>\n";

   /* Create form containing selection list */
   echo "<form action='ShowPets.php' method='POST'>\n";          #33
   echo "<table cellpadding='5' border='1'>";
   $counter=1;                                                   #35
   while($row = mysqli_fetch_assoc($result))                     #36
   {
      extract($row);                                             #38
      echo "<tr><td valign='top' width='15%'
                    style='font-weight: bold;
                    font-size: 1.2em'\n";
      echo "<input type='radio' name='interest'
                   value='$petType'\n";                          #43
      if( $counter == 1 )                                        #44
      {
          echo "checked";
      }
      echo ">$petType</td>";                                     #48
      echo "<td>$typeDescription</td></tr>";                     #49
      $counter++;                                                #50
   }
   echo "</table>";
   echo "<p><input type='submit' value='Select Pet Type'>
         </form></p>\n";                                         #54
?>
</div>
</body></html>
```

The program in Listing 11-2 has line numbers at the end of some of the lines. The line numbers are a reference so that I can refer to particular parts of the program. The numbers in the following list correspond to the line numbers in the listing. Here is a brief explanation of what the following lines do:

12 The `include` statement brings in a file that contains the information necessary to connect to the database. I call it `misc.inc` because that seems more secure than calling it `passwords.inc`.

14 Connects to the MySQL server.

18 A query that selects all the information from the `PetType` table and puts it in alphabetical order based on pet type.

20 Executes the query on line 18.

33 The opening tag for a form that will hold all the pet types. The action target is `ShowPets.php`, which is the program that shows the pets of the chosen type.

35 Creates a counter with a starting value of `1`. The counter `keeps-track` of how many pet types are found in the database.

36 Starts a `while` loop that gets the rows containing the pet type and pet description that were selected from the database on lines 19 and 20. The loop executes once for each pet type that was retrieved.

38 Separates the row into two variables: `$petType` and `$petDescription`.

42 Lines 42–43 echo a form field tag for a radio button. The value is the value in `$petType`. This statement executes once in each loop, creating a radio button for each pet type. This statement echoes only part of the form field tag.

44 Starts an `if` block that executes only in the first loop. It echoes the word `"checked"` as part of the form field. This ensures that one of the radio buttons is selected in the form so that the form can't be submitted with no button selected, which would result in unsightly error messages or warnings. The counter was set up solely for this purpose.



Although adding `"checked"` to every radio button works in some browsers, it confuses other browsers. However, the extra programming required to add `"checked"` to only one radio button can prevent potential problems.

48 Echoes the remaining part of the form field tag for the radio button — the part that closes the tag and displays the pet type.

49 Echoes the pet description in a second cell in the table row.

50 Adds 1 to the counter to keep track of the number of times that the loop has executed.

53 Adds the submit button to the form.

54 Closes the form.

When the user selects a radio button and then clicks the submit button, the next program — named `ShowPets.php` in the form tag — runs, showing the pets for the selected pet type.

Showing the pets

The pets page (refer to Figures 11-3, 11-4, and 11-5) shows the customer a list of all the pets of the selected type that are currently in the catalog. Listing 11-3 shows the program that produces the pet Web page.

Listing 11-3: Displaying a List of Pets

```
<?php
  /* Program: ShowPets.php
   * Desc:    Displays all the pets in a category.
   *          Category is passed in a variable from a
   *          form. The information for each pet is
   *          displayed on a single line, unless the pet
   *          comes in more than one color. If the pet
   *          comes in colors, a single line is displayed
   *          without a picture, and a line for each color,
   *          with pictures, is displayed following the
   *          single line. Small pictures are displayed,
   *          which are links to larger pictures.
   */
?>
<html>
<head><title>Pet Catalog</title></head>
<body>
<?php
  include("misc.inc");

  $cxn = mysqli_connect($host,$user,$passwd,$dbname)
         or die ("couldn't connect to server");

  /* Select pets of the given type */
  $query = "SELECT * FROM Pet
              WHERE petType=\"{$_POST['interest']}\"";  #26
  $result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");

  /* Display results in a table */
  echo "<table cellspacing='10' border='0' cellpadding='0'
               width='100%'>";
  echo "<tr><td colspan='5' style='text-align: right'>
              Click on any picture to see a larger
                  version. <hr /></td></tr>\n";
  while($row = mysqli_fetch_assoc($result))              #36
  {
    $f_price = number_format($row['price'],2);
```

```
    /* check whether pet comes in colors */
    $query = "SELECT * FROM Color
                  WHERE petName='{$row['petName']}'"; #42
    $result2 = mysqli_query($cxn,$query)
               or die(mysqli_error($cxn));            #44
    $ncolors = mysqli_num_rows($result2);             #45

    /* display row for each pet */
    echo "<tr>\n";
    echo " <td>{$row['petID']}</td>\n";
    echo " <td style='font-weight: bold;
             font-size: 1.1em'>{$row['petName']}</td>\n";
    echo " <td>{$row['petDescription']}</td>\n";
    /* display picture if pet does not come in colors */
    if( $ncolors <= 1 )                               #54
    {
       echo "<td><a href='../images/{$row['pix']}'
                           border='0'>
                 <img src='../images/{$row['pix']}'
                   border='0' width='100' height='80' />
                </a></td>\n";
    }
    echo "<td align='center'>\$$f_price</td>\n
          </tr>\n";
    /* display row for each color  */
    if($ncolors > 1 )                                 #65
    {
       while($row2 = mysqli_fetch_assoc($result2))    #67
       {
         echo "<tr><td colspan=2> </td>
                   <td>{$row2['petColor']}</td>
                   <td><a href='../images/{$row2['pix']}'
                             border='0'>
                       <img src='../images/{$row2['pix']}'
                          border='0' width='100'
                          height='80' /></a></td>\n";
       }
    }
    echo "<tr><td colspan='5'><hr /></td></tr>\n";
  }
  echo "</table>\n";
  echo "<div style='text-align: center'>
        <a href='PetCatalog.php' />
              <h3>See more pets</h3></a></div>";
?>
</body></html>
```

The following numbers correspond to the line numbers shown as comments at the end of lines in Listing 11-3. I document only some of the lines in this program. Many of the tasks in the listing are also in most of the programs in this application, such as connecting to the database, creating forms, and executing queries. Because I document these common tasks for Listing 11-2, I don't repeat them here. Following is a brief explanation of what some of the other lines do in the program:

25 Lines 25–26 select all the pets in the catalog that match the chosen type, which was passed in a form from the previous page.

36 Sets up a `while` loop that runs once for each pet selected. The loop creates a line of information for each pet found.

42 Lines 42–45 check whether the pet has any entries in the `Color` table. Notice that the query results are put in `$result2`. They couldn't be put in `$result` because this variable name is already in use. `$ncolors` stores the number of rows found in the `Color` table for the pet. Every pet name is checked for colors when it's processed in the loop.

54 Starts an `if` block that is executed only if zero or one row for the pet was found in the `Color` table. The `if` block displays the picture of the pet. If the program found more than one color for the pet in the `Color` table, the pet is available in more than one color, and the picture shouldn't be shown here. Instead, a picture for each color will be shown in later lines. Refer to Figures 11-3 and 11-4 for pet pages that display the pictures and information on a single row, as in this `if` block.

65 Starts an `if` block that's executed if more than one pet color was found. The `if` block echoes a row for each color found in the `Color` table.

67 Sets up a `while` loop within the `if` block that runs once for each color found in the `Color` table. The loop displays a line, including a picture, for each color. Refer to Figure 11-5 for a pet page that displays separate lines with pictures for each color.

The page has a link to more pets at the bottom. The link points to the previous program that displays the pet types.

Adding pets to the catalog

The application that adds a new pet to the catalog should do the following tasks:

1. **Create a form that asks for a pet category.** The person adding the pet can choose one of the existing pet types or create a new one. To create a new type, the user needs to type a category name and description.

2. **If a new type is created, check that the name and description were typed in.**
3. **Create a form that asks for pet information — name, description, price, picture filename, and color.** The person adding the pet can choose one of the existing pet names for the selected category or create a new name. To create a new pet name, the user needs to type a pet name.
4. **If `new` is selected for the pet name, check that the name was typed in.**
5. **Store the new pet in the `PetCatalog` database.**
6. **Send a feedback page that shows what information was just added to the catalog.**

The tasks are performed in three programs:

- `ChoosePetCat.php`: Creates the pet type form (task 1)
- `ChoosePetName.php`: Checks the pet category data and creates the pet information form (tasks 2 and 3)
- `AddPet.php`: Checks the pet name field, stores the new pet in the catalog database, and provides feedback (tasks 4, 5, and 6)

Writing ChoosePetCat

The first program, `ChoosePetCat.php`, produces a Web page with an HTML form in which the person adding a pet can select a pet type for the pet. To make the program easier to read and maintain, as I discuss in Chapter 10, I kept some of the HTML statements used by the program in a separate file that I bring into the program with an `include` statement. `ChoosePetCat.php` is shown in Listing 11-4.

Listing 11-4: Selecting a Pet Type

```
<?php
  /* Program: ChoosePetCat.php
   * Desc:    Allows users to select a pet type. All the
   *          existing pet types from the PetType table
   *          are displayed. A section to enter a new
   *          pet type is provided. Selections are
   *          provided as radio buttons, with text
   *          fields for new category name and
   *          description.
   */
?>
<html>
<head><title>Pet Types</title></head>
```

(continued)

Listing 11-4 *(continued)*

```
<body>
<?php
  include("misc.inc");
  $cxn = mysqli_connect($host,$user,$passwd,$dbname)
         or die ("couldn't connect to server");

  /*gets types from PetType table in alphabetical order*/
  $query="SELECT petType FROM PetType
                          ORDER BY petType";                    #22
$result = mysqli_query($cxn,$query)
            or die ("Couldn't execute query.");

  /* Display text before form */
echo "<div style='margin-left: .1in'>
     <h3>Select a category for the pet you're adding.</h3>
         If you are adding a pet in a category that is not
         listed, choose <b>New Category</b> and type the
         Name and description of the category. Press
         <b>Submit Category</b> when you have finished
         selecting an existing category or typing a new
         category.\n";

  /* Display form for selecting pet type */
  echo "<p><form action='ChoosePetName.php'
                 method='POST'></p>\n";
  $counter=0;
  while($row = mysqli_fetch_assoc($result))                     #40
  {
     extract($row);
     echo "<input type='radio' name='category'
                  value='$petType'";
              if($counter == 0)                                 #45
              {
                 echo "checked";
              }
     echo ">$petType</td>\n";                                   #49
     $counter++;                                                #50
  }

  include("NewCat_table.inc");                                  #53

  echo "<input type='submit' value='Submit Category'>\n";
  echo "</form>\n";
?>
</div>
</body></html>
```

The following numbers correspond to the line numbers shown as comments at the end of lines in Listing 11-4. Only some of the lines are documented in the following list. Many of the tasks in the listing, such as connecting to the database, creating forms, and executing queries, are found in most of the programs in this application; refer to Listing 11-2 for an explanation. The following list provides a brief explanation of what the following lines do.

21 A query (lines 21 and 22) that selects all the pet types from the `PetType` table and sorts them in alphabetical order.

39 Creates a counter with a starting value of `0`. The counter keeps track of how many pet types are found in the database.

40 Starts a `while` loop that executes once for each pet type. The loop creates a list of radio buttons for the pet types, with one button selected. Here are the details of the `while` loop:

- **43** Echoes a form field tag (lines 43 and 44) for a radio button with the value equal to `$petType`. This statement executes once in each loop, creating a radio button for each pet type. This statement echoes only the first part of the form field tag.
- **45** An `if` block that executes only in the first loop. It echoes the word `"checked"` as part of the form field. This ensures that one of the radio buttons is selected when displayed so that the form can't be submitted with no button selected, which would result in unsightly error messages. The counter was set up solely for this purpose.

  Although adding `"checked"` to every radio button works in some browsers, it causes problems in other browsers. The extra programming required to add `"checked"` to only one radio button can prevent problems.



- **49** Echoes the remaining part of the form field tag for the radio button — the part that closes the tag.
- **50** Adds 1 to the counter to keep track of the number of times the loop has executed. This is the last line in the `while` loop.

53 Creates a table that asks for the new pet type name and description. The HTML for the table is read in from another file called `NewCat_table.inc`. As I discuss in Chapter 10, the HTML — especially HTML that describes a form — is often kept in a separate file to make the main program easier to read and to make the form easier to modify when necessary. This file is shown in Listing 11-5.

Listing 11-5: File Containing the New Type Form

```
<?php
  /* Program:  NewCat_table.inc
   * Desc:     HTML code that displays a table for
   *           input of a new category
   */
?>
<table width="100%">
  <tr><td colspan="3"><hr /></td></tr>
  <tr>
    <td style="text-align: center">
      <input type="radio" name="category"
             value="new"> 
    </td>
    <td style="text-align: right">Category name:</td>
    <td><input type="text" name="newCat" size="20"
               maxlength="20"></td>
</tr>
  <tr><td style="text-align: center;
           font-weight: bold">New Category</td>
    <td style="text-align: right">
        Category description:</td>
    <td><input type="text" name="newDesc" size="70%"
               maxlength="255">
    </td>
  </tr>
  <tr><td colspan="3"><hr /></td></tr>
</table>
```

This file is all HTML except for a section of PHP at the top that holds the header as comments. I could have used HTML comments, but I like the PHP comment style better.

Writing ChoosePetName

The second program, `ChoosePetName.php`, accepts the data from the form in the first program. It checks the information and asks for missing information. After the pet type information is received correctly, the program creates a form in which a user can select a pet name for the new pet being added to the catalog and type the information for the pet. This program, as in the preceding program, brings in some of the HTML forms and tables from separate files with `include` statements. This program also calls a function that's in an `include` file. This program brings in two files. Listing 11-6 shows `ChoosePetName.php`.

Listing 11-6: Asking the User for the Pet Name

```
<?php
 /* Program: ChoosePetName.php
  * Desc:    Allows the user to enter the information for
  *          the pet. First, the program checks for and
```

```
  *          enters a new category into the petType
  *          table. Then, all pets in the selected
  *          category are displayed with radio buttons.
  *          The user can enter a new name. Fields are
  *          provided to enter the description, price,
  *          and picture file name.
  */

 if (@$_POST['newbutton'] == "Return to category page"
     or @$_POST['newbutton'] == "Cancel")                    #14
 {
    header("Location: ChoosePetCat.php");
 }

 echo "<html>
       <head><title>Add Pet</title></head>
       <body>";
 include("misc.inc");
 include("functions.inc");

 $cxn = mysqli_connect($host,$user,$passwd,$dbname)
        or die ("Couldn't connect to server");

 $category = $_POST['category'];
 /* If new was selected for pet category, check if text
    fields were filled in. If not, display again for the
    user to enter the category name and category
    description. When the fields are filled in, store
    the new category in the PetType table.*/
 if($category == "new")                                      #34
 {
   if ($_POST['newCat'] == ""
                    or $_POST['newDesc'] == "")              #37
   {
      include("NewCat_form.inc");                            #39
      exit();                                                #40
   }
     /* add new pet type to PetType table */
   else                                                      #43
   {
      addNewType($_POST['newCat'],$_POST['newDesc'],$cxn);
      $category = trim($_POST['newCat']);                    #46
   }
 }                                                           #48

  /* Select pet names from table with given category. If
user entered a new category, it is searched for. */
$query = "SELECT DISTINCT petName FROM Pet
                  WHERE petType='$category'
                  ORDER BY petName";                         #54
 $result = mysqli_query($cxn,$query)
           or die("Couldn't execute select query");
```

(continued)

Listing 11-6 *(continued)*

```
 $nrow = mysqli_num_rows($result);                                    #57

/* create form */
 echo "<div style='margin-left: .1in'>";
 echo "<form action='AddPet.php' method='POST'>\n";
 echo "<h4>Pet Name</h4>\n";
 if($nrow < 1)                                                        #63
 {
   echo "<hr /><b>No pet names are currently in the
         database for the category $category</b><hr />\n";
 }
 else                                                                 #68
 {
   while($row = mysqli_fetch_assoc($result))                          #70
   {
      extract($row);
      echo "<input type='radio' name='petName'
                   value='$petName'";
echo ">$petName\n";
   }
 }
 include ("NewName_table.inc");                                       #78

 $petDescription="";$price = "";$pix = "";$petColor = "";
 include("PetInfo_table.inc");                                        #81

 echo "<input type='hidden' name='category'
              value='$category'>\n";
 echo "<p><input type='submit' value='Submit Pet Name'>
     <input type='submit' name='newbutton' value='Cancel'>
     </form>\n";
?>
</div>
</body></html>
```

The following numbers correspond to the line numbers shown as comments at the end of lines in Listing 11-6. Only some of the lines are documented in the following list because many of the tasks in the listing are found in most of the programs in this application. The common tasks are documented for Listing 11-2 and explained in other parts of the book, so I don't repeat them here. Here's a brief explanation of what the following lines do in the program:

14 Checks whether the user clicked the submit button labeled *Cancel* or *Return to category page*. If so, it returns to the first page.

34 Starts an `if` block that executes only if the user selected the radio button for New Category in the form from the previous program. This block checks whether the new category name and description are filled in. If the user forgot to type them in, he or she is asked for the pet type name and description again. After the name and description are filled in,

the program calls a function that adds the new category to the `PetType` table. The following lines describe this `if` block in more detail:

36 Starts an `if` block that executes only if the category name or the category description is blank. Because this `if` block is inside the `if` block for a new category, this block executes only if the user selected New Category for pet type but did not fill in the new category name *and* description.

39 Creates a form that asks for the category name and description. The HTML for the form is included from a file. This executes only when the `if` statement on line 36 is true — that is, if the category is new and the category name and/or description is blank.

40 Stops the program after displaying the form on line 43. The program can't proceed until the category name and description are typed in. This block repeats until a category name and description are filled in.

43 Starts an `else` block that executes only if both the category name and description are filled in. Because this block is inside the `if` block for the new radio button, this block executes when the user selected `new` and filled in the new category name and description.

45 Calls a function that adds the new category to the `PetType` table.

46 Up to this point, the category is still set to `"new"`. This line sets `$category` to the new category name.

48 This line ends the `if` block. If the user selected one of the existing pet types, the statements between line 36 and this line did not execute.

52 A query (lines 52–54) that selects one of each pet name with the chosen pet type and sorts them alphabetically.

57 Checks whether any pet names were found for the chosen pet type.

63 Starts an `if` block that executes only if *no* pets were found for the pet type. The block echoes a message to the user that no pets were found for the pet type.

68 Starts an `else` block that executes if pets *were* found for the pet type. The `else` block creates a list of radio buttons for the pet names found. The list is created with a `while` loop (starting on line 70) in the same manner that the list of categories was created, as explained in Listing 11-4.

78 Lines 78 and 81 create tables that ask for the new pet name and information, bringing the HTML in from separate files with `include` statements.

This program brings in three files containing HTML using `include` statements. Listings 11-7, 11-8, and 11-9 show the three files that are included: `NewCat_form.inc`, `NewName_table.inc`, and `PetInfo_table.inc`.

Listing 11-7: HTML Code That Creates New Pet Type Form

```
<?php
  /* Program: NewCat_form.inc
   * Desc:    Displays a form to collect a category name
   *          and description.
   */
?>
<h4>Either the category name or the category description
    was left blank. You must enter both.</h4>
<form action="ChoosePetName.php" method="POST">
 <table>
  <tr>
   <td style="text-align: right">Category name:</td>
   <td><input type="text" name="newCat"
              value="<?php echo $_POST['newCat'] ?>"
             size="20" maxlength="20">
   </td></tr>
  <tr>
   <td style="text-align: right">
       Category description:</td>
   <td><input type="text" name="newDesc"
              value="<?php echo $_POST['newDesc'] ?>"
              size="70%" maxlength="255">
   </td></tr>
 </table>
 <input type="hidden" name="category" value="new">
 <p><input type="submit" name="newbutton"
           value="Enter new category">
 <input type="submit" name="newbutton"
        value="Return to category page">
</form>
```

This program is almost all HTML code. Note the following points:

- **This form is created only when the user selects the radio button for New Category on the pet type Web page but does not type the pet type name or description.** This form is displayed to give the user a second chance to type the name or description.
- **Most of the file is HTML, with only two small PHP sections that echo values for the two fields.**
- **The form returns to the program that generated it for processing.** It is processed in the same manner as the form that was sent from the first page. The field names are the same and are checked again to see whether they are blank.
- **A hidden field is included that sends `$category` with a value of `"new"`.** If this form didn't send `$category`, the program that processes it — the same program that generated it — wouldn't know that the pet type was new and wouldn't execute the `if` block that should be executed when a new category is selected.

Listing 11-8: HTML File That Creates Table for New Name

```
<?php
  /* Program: NewName_table.inc
   * Desc:    Displays table to enter new pet name
   */
?>
<p><table border="0">
   <tr><td>
        <input type="radio" name="petName"
               value="new" checked >New Name</td>
      <td><input type="text" name="newName" size="25"
               maxlength="25"> (type new name)</td>
  </tr>
  <tr><td colspan="2"><hr /></td></tr>
</table>
```

This file is all HTML with no PHP. It displays the section of the pet name Web page where the user can enter a new pet name.

Listing 11-9: HTML File That Creates Table for Pet Info

```
<?php
  /* Program: PetInfo_table.inc
   * Desc:    Displays table to collect pet information
   */
?>
<h4>Pet Information</h4>
<p><table>
 <tr><td style="text-align: right">Pet Category:</td>
   <td style="font-weight: bold">
       <?php echo $category ?></td>
 </tr>
 <tr><td style="text-align: right">Pet Description:</td>
   <td><input type="text" name="petDescription"
           value="<?php echo $petDescription ?>"
           size="65" maxlength="255">
   </td></tr>
 <tr><td style="text-align: right">Price:</td>
   <td><input type="text" name="price"
           value="<?php echo $price ?>" size="15"
           maxlength="15">
   </td></tr>
 <tr><td style="text-align: right">Picture file name:</td>
<td><input type="text" name="pix"
           value="<?php echo $pix ?>" size="25"
           maxlength="25">
   </td></tr>
 <tr>
  <td style="text-align: right">Pet color (optional):</td>
   <td><input type="text" name="petColor"
           value="<?php echo $petColor ?>" size="25"
           maxlength="25">
   </td></tr>
</table>
```

This file is all HTML, except for small PHP sections for the variable values.

In addition to HTML for tables and forms, the `ChoosePetName.php` program in Listing 11-6 calls a function. The function is stored in a file named `functions.inc` and is included in the beginning of the program. Listing 11-10 shows the function.

Listing 11-10: Function addNewType()

```
<?php
 /* Function addNewType
  * Desc     Adds a new pet type and description to the
  *          PetType table. Checks for the new pet type
  *          first and does not add it to the table if
  *          it is already there.
  */
function addNewType($petType,$typeDescription,$cxn)
{
  /* Check whether new category is in PetType table.
     If it is not in table, add it to table. */
  $query = "SELECT petType FROM PetType
                           WHERE petType='$petType'";
  $result = mysqli_query($cxn,$query) or
            die("Couldn't execute select query");
  $ntype = mysqli_num_rows($result); //
  if ($ntype < 1)  // if new type is not in table
{
    $petType = ucfirst(strip_tags(trim($petType)));
    $typeDescription =
        ucfirst(strip_tags(trim($typeDescription)));
    $petType = mysqli_real_escape_string($cxn,$petType);
    $typeDescription =
        mysqli_real_escape_string($cxn,$typeDescription);

    $query="INSERT INTO PetType (petType,typeDescription)
                VALUES ('$petType','$typeDescription')";
    $result = mysqli_query($cxn,$query)
              or die("Couldn't execute insert query");
  }
  return;
}
?>
```

The function checks whether the pet type is already in the `PetType` table. If it is not, the function cleans the data and adds it to the table.

Writing AddPet

The last program, `AddPet.php`, accepts the data from the form in the second program. If new was selected for the pet name, the program checks to see that a new name was typed and prompts for it again if it was left blank. After the pet name is filled in, the program stores the pet information from

the previous page. Notice that it does not check the other information because the other information is optional. This program, as in Listing 11-6, brings in some of the HTML forms and tables from two separate files by using an `include` statement. Listing 11-11 shows `AddPet.php`.

Listing 11-11: Adding a New Pet to the Catalog

```
<?php
  /* Program: AddPet.php
   * Desc:    Adds new pet to the database. A confirmation
   *          screen is sent to the user.
   */

  if (@$_POST['newbutton'] == "Cancel")                    #7
  {
     header("Location: ChoosePetCat.php");
  }

  include("misc.inc");                                     #12
  $cxn = mysqli_connect($host,$user,$passwd,$dbname)
         or die ("Couldn't connect to server");

  foreach($_POST as $field => $value)                      #16
  {
     if($field != "newName" and $field != "newbutton"
          and $field != "petColor")                        #19
     {
       if($field == "petName")                             #21
       {
          if($value == "new")                              #23
          {
             if($_POST['newName'] == "")                   #25
             {
                include("NewName_form.inc");
                exit();
             }
             else                                          #30
             {
                $value=$_POST['newName'];
             }
          }
       }
       if($field == "category")                            #36
       {
          $field = "petType";
       }
       if(!empty($value))                                  #40
       {
         $fields_form[$field] =
            ucfirst(strtolower(strip_tags(trim($value))));
         $fields_form[$field] =
            mysqli_real_escape_string($cxn,
```

(continued)

Listing 11-11 ***(continued)***

```
                $fields_form[$field]);
        }
        if(!empty($_POST['petColor']))                    #48
        {
          $petColor = strip_tags(trim($_POST['petColor']));
          $petColor = ucfirst(strtolower($petColor));
          $petColor =
              mysqli_real_escape_string($cxn,$petColor);
        }
    }
  }
?>
<html>
<head><title>Add Pet</title></head>
<body>
<?php
  $field_array = array_keys($fields_form);                #62
  $fields=implode(",",$field_array);
  $values=implode('","',$fields_form);
  $query = "INSERT INTO Pet ($fields)
                            VALUES (\"$values\")";        #66
  $result = mysqli_query($cxn,$query)
       or die ("Couldn't execute query.");

  $petID = mysqli_insert_id($cxn);                        #70
  $query = "SELECT * from Pet WHERE petID='$petID'";      #71
  $result = mysqli_query($cxn,$query)
       or die ("Couldn't execute query.");
  $row = mysqli_fetch_assoc($result);
  extract($row);
  $category=$petType;
  echo "The following pet has been added to the
        Pet Catalog:<br>
        <ul>
         <li>Category: $category
         <li>Pet Name: $petName
         <li>Pet Description: $petDescription
         <li>Price: $price
         <li>Picture file: $pix \n";

  if (@$petColor != "")                                   #86
  {
     $query = "SELECT petName FROM Color
                     WHERE petName='$petName'
                     AND petColor='$petColor'";
     $result = mysqli_query($cxn,$query)
                or die("Couldn't execute query.");
     $num = mysqli_num_rows($result);
     if ($num < 1)
     {
       $query = "INSERT INTO Color (petName,petColor,pix)
                  VALUES ('$petName','$petColor','$pix')";
```

```
        $result = mysqli_query($cxn,$query)
                  or die("Couldn't execute query.");
        echo "<li>Color: $petColor\n";
      }
  }                                                          #102
  echo "</ul>";
  echo "<a href='ChoosePetCat.php'>Add Another Pet</a>\n";
?>
</body></html>
```

Notice the line numbers shown as comments at the end of lines in Listing 11-11. The numbers in the following list correspond to the line numbers in the listing. I document only some of the lines in the following list because many of the most common tasks, such as connecting to the database, have been documented for the previous programs in this chapter.

7 Checks whether the user clicked the Cancel button. If so, returns to the first page.

12 Connects to the database.

16 Starts a `foreach` block that walks through the new pet information submitted on the previous Web page. This block checks and cleans the data. The following line numbers describe the processing in detail.

- **19** Does not process the data in the fields `newName`, `newbutton`, and `petColor`.
- **23** Starts an `if` block that executes only if the user selected `new` for the pet name. If the new name is blank, it displays a form that asks for the new pet name (line 27) repeatedly until the user types one. After the new name is filled in, `$petName` is set to the new name (line 32).
- **36** If the field is category, changes the field name to `petType`.
- **40** Lines 40–54 clean the data.
- **56** End of the `foreach` loop that processes the data in `$_POST`.

62 Lines 62–66 build the query that inserts the data for the new pet into the database.

70 Lines 70–84 retrieve the data that was just entered into the database and display it on a Web page so the user can see what data was entered.

86 Starts an `if` block that executes only if the color was filled in. The `Color` table is checked to see whether the name and color are already there. If not, they are added to the `Color` table. The `if` block ends on line 102.

This program brings in an HTML file that creates the form to prompt the user for the pet name if the user forgot to type it in. Listing 11-12 shows the file that is included: `NewName_form.inc`.

Listing 11-12: HTML That Asks User for a New Pet Name

```
<?php
  /* Program: NewName_form.inc
   * Desc:    Displays form to collect a pet name
   */
  extract($_POST);
?>
<h4>You must type a pet name.</h4>
<form action="AddPet.php" method="POST">
 <table><tr>
   <td style="text-align: right">Pet name:</td>
   <td><input type="text" name="newName"
         value="<?php echo $newName ?>"
         size="25" maxlength="25">
   </td></tr>
 </table>
 <input type="hidden" name="category"
        value="<?php echo $category ?>">
 <input type="hidden" name="petName"
        value="<?php echo $petName ?>">
 <input type="hidden" name="petDescription"
        value="<?php echo $petDescription ?>">
 <input type="hidden" name="price"
        value="<?php echo $price ?>">
 <input type="hidden" name="pix"
        value="<?php echo $pix ?>">
 <input type="hidden" name="petColor"
        value="<?php echo $petColor ?>">
 <p><input type="submit" name="newbutton"
        value="Enter new pet name">
 <input type="submit" name="newbutton"
        value="Cancel">
</form>
```

This file creates the form that's displayed if the user forgets to type the new pet name. It is very similar to the program in Listing 11-7 that's displayed when a user forgets to type a new category. Notice that hidden fields are used to pass on any other information that the user may have entered. When the form is filled in, the values are needed to store the pet information.

At the end, this program provides a link to the first page so that the user can add another new pet to the catalog if desired.

Chapter 12

Building a Members Only Web Site

In This Chapter

- Designing the Members Only Web site
- Building the database for the member directory
- Designing the Web pages for the Members Only section
- Writing the programs for logging in to the Members Only section

Many Web sites require users to log in. Sometimes users can't view any Web pages without entering a password, while sometimes just part of the Web site requires a login. Here are some reasons why you might want to require a user login:

- **The information is secret.** You don't want anyone except a few authorized people to see the information. Or perhaps only your own employees should see the information.
- **The information or service is for sale.** The information or service that your Web site provides is your product, and you want to charge people for it. For instance, you might have a corner on some survey data that researchers are willing to pay for. For example, AAA Automobile Club offers some of its information for free, but you have to be a member to see its hotel ratings.
- **You can provide better service.** If you know who your customers are or have some of their information, you can make their interaction with your Web site easier. For instance, if you have an account with Barnes and Noble or the Gap and log into their site, they use your stored shipping address, and you don't have to type it in again.
- **You can find out more about your customers.** Marketing would like to know who is looking at your Web site. A list of customers with addresses and phone numbers and perhaps some likes and dislikes is useful. If your Web site offers some attractive features, customers may be willing to provide some information to access your site.

Typically, a login requires the user to enter a user ID and a password. Often, users can create their own accounts on the Web site, choosing their own user ID and password. Sometimes users can maintain their accounts — for example, change their password or phone number — online.

In Chapter 11, you find out how to build an online catalog for your Pet Store Web site. Now, you want to add a section to your Web site for Members Only. You plan to offer discounts, a newsletter, a database of pet information, and more in the Members Only section. You hope that customers will see the section as so valuable that they'll be willing to provide their addresses and phone numbers to get a member account that lets them use the services in the restricted section. In this chapter, you build a login section for the Pet Store.

Designing the Application

The first step in design is to decide what the application should do. Its basic function is to gather customer information and store it in a database. It offers customers access to valuable information and services to motivate them to provide information for the database. Because state secrets or credit card numbers aren't at risk, you should make it as easy as possible for customers to set up and access their accounts.

The application that provides access to the Members Only section of the Pet Store should do the following:

- **Provide a means for customers to set up their own accounts with member IDs and passwords.** This includes collecting from the customer the information that's required to become a member.
- **Provide a page where customers type their member ID and password and then check whether they are valid.** If so, the customer enters the Members Only section. If not, the customer can try another login.
- **Show the pages in the Members Only section to anyone who is logged in.**
- **Refuse to show the pages in the Members Only section to anyone who is *not* logged in.**
- **Keep track of member logins,** so you know who logs in and how often.

Building the Database

The database is the core and purpose of this application. It holds the customer information that's the goal of the Members Only section and the Member ID and password that allow the user to log into the Members Only section.

The Members Only application database contains two tables:

- `Member` table
- `Login` table

The first step in building the login application is to build the database. It's pretty much impossible to write programs without a working database to test the programs on. First design your database, then build it, and then add some sample data for use while developing the programs.

Building the Member table

In your design for the login application, the main table is the `Member` table. It holds all the information entered by the customer, including the customer's personal information (name, address, and so on) and the Member ID and password. The following SQL query creates the `Member` table:

```
CREATE TABLE Member (
  loginName     VARCHAR(20)    NOT NULL,
  createDate    DATE           NOT NULL,
  password      VARCHAR(255)   NOT NULL,
  lastName      VARCHAR(50),
  firstName     VARCHAR(40),
  street        VARCHAR(50),
  city          VARCHAR(50),
  state         CHAR(2),
  zip           CHAR(10),
  email         VARCHAR(50),
  phone         CHAR(15),
  fax           CHAR(15),
PRIMARY KEY(loginName) );
```

Each row represents a member. The columns are

- `loginName`: A Member ID for the member to use when logging in. The customer chooses and types in the login name. The `CREATE` query defines the `loginName` in the following ways:
  - `CHAR(20)`: This data type defines the field as a character string that's 20 characters long. If the stored string is less than 20 characters, the field is padded so that it always takes up 20 characters of storage. If a string longer than 20 characters is stored, any characters after 20 are dropped.
  - `PRIMARY KEY(loginName)`: The primary key identifies the row and must be unique. MySQL will not allow two rows to be entered with the same `loginName`.
  - `NOT NULL`: This definition means that this field can't be empty. It must have a value. The primary key must always be `NOT NULL`.

- createDate: The date when the row was added to the database — that is, the date when the customer created the account. The query defines createDate as
  - DATE: This is a string that's treated as a date. Dates are displayed in the format YYYY-MM-DD. They can be entered in that format or a similar format, such as YY/M/D or YYYYMMDD.
  - NOT NULL: This definition means this field can't be empty. It must have a value. Because the program, not the user, creates the date and stores it, this field won't ever be blank.
- password: A password for the member to use when logging in. The customer chooses and types in the password. The CREATE query defines the password in the following ways:
  - VARCHAR(255): This statement defines the field as a variable character string that can be up to 255 characters long. The field is stored in its actual length. You don't expect the password to be 255 characters long. In fact, you expect it to be pretty short. However, you intend to use the MySQL md5 function to encrypt it rather than store it in plain view. After it's encrypted, the string will be longer, so you're allowing room for the longer string.
  - NOT NULL: This statement means that this field can't be empty. It must have a value. You're not going to allow an empty password in this application.
- lastName: The customer's last name, as typed by the customer. The CREATE query defines the field as
  - VARCHAR(50): This data type defines the field as a variable character string that can be up to 50 characters long. The field is stored in its actual length.
- firstName: The customer's first name, as typed by the customer. The CREATE query defines the field as
  - VARCHAR(40): This data type defines the field as a variable character string that can be up to 40 characters long. The field is stored in its actual length.
- street: The customer's street address, as typed by the customer. The CREATE query defines the field as
  - VARCHAR(50): This data type defines the field as a variable character string that can be up to 50 characters long. The field is stored in its actual length.
- city: The city in the customer's address, as typed by the customer. The CREATE query defines the field as
  - VARCHAR(50): This data type defines the field as a variable character string that can be up to 50 characters long. The field is stored in its actual length.

- `state`: The state in the customer's address. The string is the two-letter state code. The customer selects the data from a drop-down list containing all the states. The `CREATE` query defines the field as
  - `CHAR(2)`: This data type defines the field as a character string that's 2 characters long. The field will always take up 2 characters of storage.
- `zip`: The zip code that the customer types in. The `CREATE` query defines the field as
  - `CHAR(10)`: This data type defines the field as a character string that's 10 characters long. The field will always take up 10 characters of storage, with padding if the actual string stored is less than ten characters. The field is long enough to hold a zip+4 code, such as 12345–1234.
- `email`: The e-mail address that the customer types in. The `CREATE` query defines the field as
  - `VARCHAR(50)`: This data type defines the field as a variable character string that can be up to 50 characters long. The field is stored in its actual length.
- `phone`: The phone number that the customer types in. The `CREATE` query defines the field as
  - `CHAR(15)`: This data type defines the field as a character string that's 15 characters long. The field will always take up 15 characters of storage, with padding if the actual string stored is less than 15 characters.
- `fax`: The fax number that the customer types in. The `CREATE` query defines the field as
  - `CHAR(15)`: This data type defines the field as a character string that's 15 characters long. The field will always take up 15 characters of storage, with padding if the actual string stored is less than 15 characters.

Notice that some fields are `CHAR` and some are `VARCHAR`. `CHAR` fields are faster, whereas `VARCHAR` fields are more efficient in using disk space. Your decision on which to use will depend on whether disk space or speed is more important for your application in your environment.



In general, shorter fields should be `CHAR` because they don't waste much space. For instance, if your `CHAR` is 5 characters, the most space that could possibly be wasted is 4 characters. However, if your `CHAR` is 200, you could waste 199 characters. Therefore, for short fields, use `CHAR` for speed with very little wasted space.

Building the Login table

The `Login` table keeps track of member logins by recording the date and time every time a member logs in. Because each member has multiple logins, the login data requires its own table. The `CREATE` query that builds the `Login` table is

```
CREATE TABLE Login (
  loginName     VARCHAR(20) NOT NULL,
  loginTime     DATETIME    NOT NULL,
PRIMARY KEY(loginName,loginTime) );
```

The `Login` table has only two columns, as follows:

- `loginName`: The Member ID that the customer uses to log in with. The `loginName` is the connection between the `Member` table (which I describe in the preceding section) and this table. Notice that the `loginName` column is defined the same in the `Member` table and in this table. This makes table joining possible and makes matching rows in the tables much easier. The `CREATE` query defines the `loginName` in the following ways:
  - `CHAR(20)`: This data type defines the field as a character string that's 20 characters long. The field will always take up 20 characters of storage, with padding if the actual string stored is less than 20 characters. If a string longer than 20 characters is stored, any characters after 20 are dropped.
  - `PRIMARY KEY(loginName,loginTime)`: The primary key identifies the row and must be unique. For this table, two columns together are the primary key. MySQL will not allow two rows to be entered with the same `loginName` *and* `loginDate`.
  - `NOT NULL`: This definition means that this field can't be empty. It must have a value. The primary key must always be `NOT NULL`.
- `loginTime`: The date and time when the member logged in. This field uses both the date and time because the field needs to be unique. It's unlikely that two users would log in at the same second at the Pet Store Web site. However, in some busy Web sites, two users might log in during the same second. At such a site, you might have to create a sequential login number to be the unique primary key for the site. The `CREATE` query defines the `loginTime` in the following ways:
  - `DATETIME`: This is a string that's treated as a date and time. The string is displayed in the format YYYY-MM-DD HH:MM:SS.
  - `PRIMARY KEY(loginName,loginTime)`: The primary key identifies the row and must be unique. For this table, two columns together are the primary key. MySQL will not allow two rows to be entered with the same `loginName` *and* `loginDate`.
  - `NOT NULL`: This definition means that this field can't be empty. It must have a value. The primary key must always be `NOT NULL`.

Adding data to the database

This database is intended to hold data entered by customers — not by you. It will be empty when the application is first made available to customers until customers add data. However, to test the programs while you write them, you need to have at least a few members in the database. You need a few Member IDs and passwords to test the login program. You can add some fake members for testing — by using an `INSERT` SQL query — and remove them when you're ready to go live with your Members Only application.

Designing the Look and Feel

After you know what the application is going to do and what information you want to get from customers and store in the database, you can design the look and feel. The look and feel includes what the user sees and how the user interacts with the application. Your design should be attractive and easy to use. You can create your design on paper, indicating what the user sees, perhaps with sketches or with written descriptions. You should also show the user interaction components, such as buttons or links, and describe their actions. Include each page of the application in the design.

The Pet Store Members Only application has three pages that are part of the login procedures. In addition, the application includes all the pages that are part of the Members Only section, such as the page that shows the special discounts and the pages that provide discussions of pet care. In this chapter, you build only the pages that are part of the login procedure. You don't build the pages that are part of the Members Only section, but I do discuss what needs to be included in them to protect them from viewing by nonmembers.

The login application includes three pages, plus the group of pages that comprise the Members Only section, as follows:

- **Storefront page:** The first page that a customer sees. It provides the name of the business and the purpose of the Web site. In Chapter 11, I introduce a storefront page; in this chapter, you modify the page to provide access to the Members Only section.
- **Login page:** Allows the customer to either log in or create a new member account. It displays a form for the customer to fill in to get a new account.
- **New Member Welcome page:** Welcomes the new users by name, letting them know that their accounts have been created. Provides any information that they need to know. Provides a button so that users can continue to the Members Only section or return to the main page.
- **Members Only section:** A group of Web pages that contain the content of the Members Only section.

Storefront page

The *storefront page* is the introductory page for the Pet Store. Because most people know what a pet store is, the page doesn't need to provide much explanation. Figure 12-1 shows the storefront page. Two customer actions are available on this page: a link that the customer can click to see the Pet Catalog and a link to the Members Only section.

Login page

The login page, shown in Figure 12-2, allows the customer to log in or create a new member account. It includes the form that customers need to fill out to get a member account. This page has two submit buttons: one to log in with an existing member account and one to create a new member account.

If a customer makes a mistake on the login page, either in the login section or the new member section, the form is displayed again with an error message. For instance, suppose that a customer makes an error when typing her e-mail address: She forgot to type the `.com` at the end of the e-mail address. Figure 12-3 shows the screen that she sees after she submits the form with the mistake in it. Notice the error message printed right above the form.

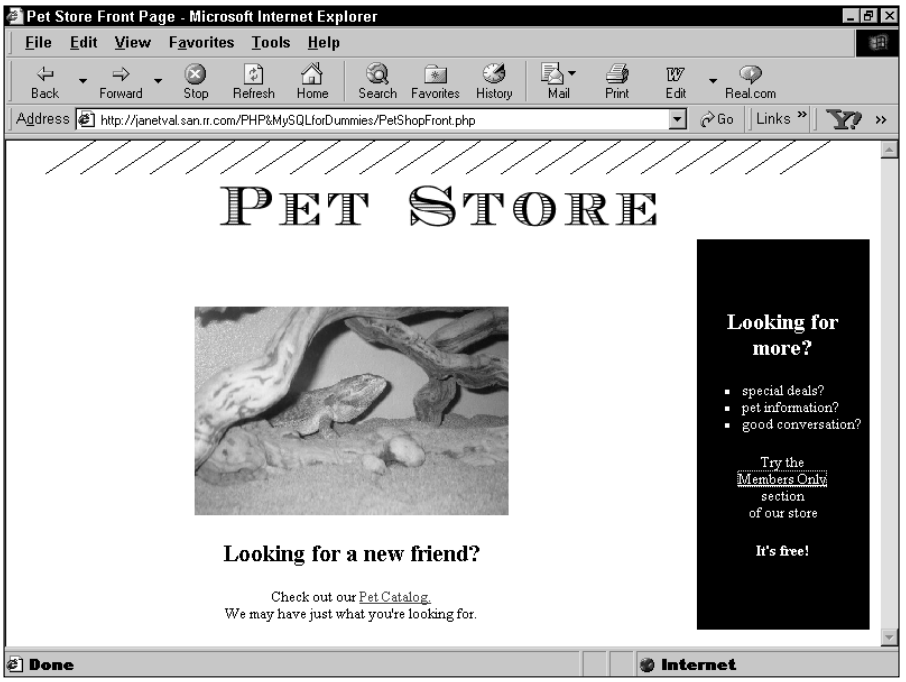


Figure 12-1: The opening page of the Pet Store Web site.

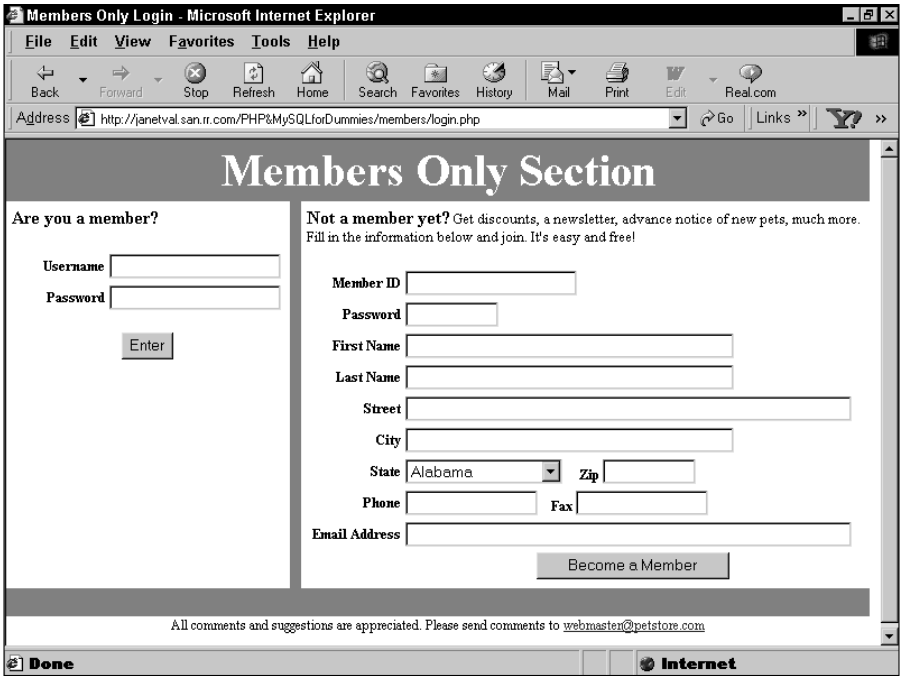


Figure 12-2: The page where customers log in or create a new member account.

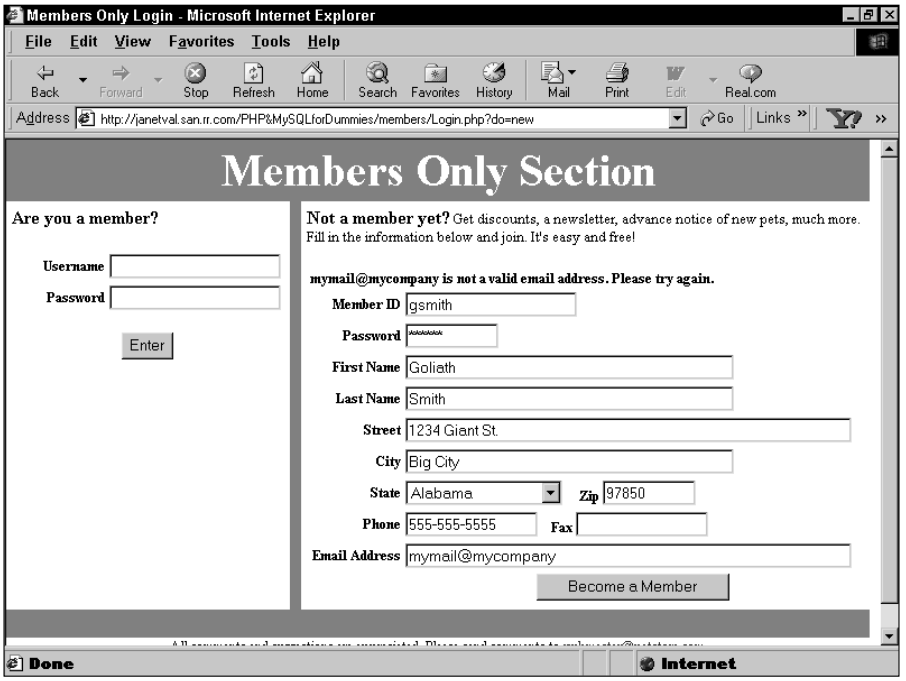


Figure 12-3: Page showing a message resulting from a mistake in the form.

When members successfully log in with a valid Member ID and password, they go to the first page of the Members Only section. When new members successfully submit a form with information that looks reasonable, they go to a New Member Welcome page (see the next section). In addition, an e-mail message is sent to the new member with the following contents:

```
A new Member Account has been set up for you. Your new
Member ID and password are:

gsmith
secret

We appreciate your interest in Pet Store at PetStore.com.

If you have any questions or problems, email
webmaster@petstore.com
```



This e-mail message contains the customer's password. I think that it's helpful to both the customer and the business to provide customers with a hard copy of their password. Customers *will* forget their password. It seems to be one of the rules. An e-mail message with their password might help them when they forget it, saving both them and you some trouble. Of course, e-mail messages aren't necessarily secure, so sending passwords via e-mail isn't a good idea for some accounts, such as an online bank account. But, for this Pet Store application, with only unauthorized discounts and pet care information at risk, sending the password via e-mail is a reasonable risk.

New Member Welcome page

The New Member Welcome page greets the customer and offers useful information. The customer sees that the account has been created and can then enter the Members Only section immediately. Figure 12-4 shows a welcome page.

Members Only section

One or more Web pages make up the contents of the Members Only section. Whatever the content is, the pages are no different than any other Web pages or PHP programs, except for some PHP statements in the beginning of each file that prevent nonmembers from viewing the pages.



Figure 12-4: A page welcoming new members.

Writing the Programs

After you know what the pages are going to look like and what they are going to do, you can write the programs. In general, you create a program for each page, although sometimes it makes sense to separate programs into more than one file or to combine programs on a page. (See Chapter 10 for details on how to organize applications.)



As I discuss in Chapter 10, keep the information needed to connect to the database in a separate file and include it in the programs that need to access the database. Store the file in a secure location, with a misleading name. For this application, the following information is stored in a file named `dogs.inc`:

```
<?php
  $user="catalog";
  $host="localhost";
  $password="";
  $database="MemberDirectory";
?>
```

The member login application has several basic tasks:

1. **Show the storefront page.** This provides a link to the login page.
2. **Show a page where customers can fill in a Member ID and a password to log in.**
3. **Check the Member ID and the password that the customer types against the Member ID and password stored in the database.** If the ID and password are okay, the customer enters the Members Only section. If the ID and/or password are not okay, the customer is returned to the login page.
4. **Show a page where customers can fill in the information needed to obtain a member account.**
5. **Check the information the customer typed for blank fields or incorrect formats.** If bad information is found, show the form again so that the customer can correct the information.
6. **When good information is entered, add the new member to the database.**
7. **Show a welcoming page to the new member.**

The tasks are performed in three programs:

- `PetShopFront.php`: Shows the storefront page (task 1).
- `Login.php`: Performs both the login and create new member account tasks (tasks 2–6).
- `New_member.php`: Shows the page that welcomes the new member (task 7).

Writing PetShopFront

The storefront page doesn't need any PHP statements. It simply displays a Web page with two links — one link to the Pet Catalog and one link to the Members Only section of the Web site. HTML statements are sufficient to do this. Listing 12-1 shows the HTML file that describes the storefront page.

Listing 12-1: HTML File for the Storefront Page

```
<?php
  /* Program: PetShopFrontMembers.php
   * Desc:    Displays opening page for Pet Store.
```

```
   */
?>
<html>
<head><title>Pet Store Front Page</title></head>
<body style="margin: 0">
<table width="100%" height="100%" border="0"
       cellspacing="0" cellpadding="0">
  <tr>
    <td align="center" valign="top" height="30"
        colspan="2">
      <img src="../images/awning-top.gif" alt="awning">
    </td>
  </tr>
  <tr>
    <td align="center" valign="top" colspan="2">
      <img src="../images/Name.gif" alt="Pet Store">
    </td></tr>
  <tr>
    <td width="80%" align="center">
      <p style="margin-top: 40pt">
      <img src="../images/lizard-front.jpg"
           alt="lizard picture"
           height="186" width="280">
      <p><h2>Looking for a new friend?</h2>
      <p>Check out our
            <a href="PetCatalog.php">Pet Catalog.</a>
        <br> We may have just what you're looking for.</p>
    </td>
    <td style="width: 20%; background-color: black">
       <div style="color: white; link: white">
       <p style="text-align: center; font-size: 15pt">
       <b>Looking for <br>more?</b></p>
       <ul>
        <li>special deals?
        <li>pet information?
        <li>good conversation?
       </ul>
       <p style="text-align: center">Try the
          <br><a href="login.php"
                 style="color: white">Members Only</a>
          <br>section <br>of our store
       <p style="text-align: center"><b>It's free!</b></p>
    </td>
  </tr>
</table>
</body></html>
```

Notice the link to the login PHP program. When the customer clicks the link, the login page appears.

Writing Login

The login page (refer to Figure 12-2) is produced by the program `Login.php`, shown in Listing 12-2. The program uses a switch to create two sections: one for the login and one for creating a new account. The program creates a session that's opened in all the Members Only Web pages. The login form itself isn't included in this program; it's in a separate file, `login_form.inc`, and is called into the program, using `include` statements, when the form is needed.

Listing 12-2: Logging into the Members Only Section

```
<?php
/* Program: Login.php
 * Desc:    Login program for the Members Only section of
 *          the pet store. It provides two options:
 *          (1) login using an existing Login Name and
 *          (2) enter a new login name. Login Names and
 *          passwords are stored in a MySQL database.
 */
session_start();                                                    #9
include("dogs.inc");                                                #10
switch (@$_POST['do'])                                              #11
{
  case "login":                                                     #13
    $cxn = mysqli_connect($host, $user,$passwd,$dbname)
          or die ("Couldn't connect to server.");                   #15

    $sql = "SELECT loginName FROM Member
            WHERE loginName='$_POST[fusername]'";                   #18
    $result = mysqli_query($cxn,$sql)
              or die("Couldn't execute query.");                    #20
    $num = mysqli_num_rows($result);                                #21
    if ($num > 0)  // login name was found                          #22
    {
       $sql = "SELECT loginName FROM Member
               WHERE loginName='$_POST[fusername]'
               AND password=md5('$_POST[fpassword]')";
       $result2 = mysqli_query($cxn,$sql)
                  or die("Couldn't execute query 2.");
       $num2 = mysqli_num_rows($result2);
       if ($num2 > 0)  // password is correct                       #30
       {
          $_SESSION['auth']="yes";                                  #32
          $logname=$_POST['fusername'];
          $_SESSION['logname'] = $logname;                          #34
          $today = date("Y-m-d h:i:s");                             #35
          $sql = "INSERT INTO Login (loginName,loginTime)
                  VALUES ('$logname','$today')";
          $result = mysqli_query($cxn,$sql)
                  or die("Can't execute insert query.");
          header("Location: Member_page.php");                      #40
       }
```

```
      else    // password is not correct                    #42
      {
         $message="The Login Name, '$_POST[fusername]'
             exists, but you have not entered the
             correct password! Please try again.<br />";
         include("login_form.inc");                        #47
      }
   }                                                       #49
   elseif ($num == 0)  // login name not found             #50
   {
      $message = "The Login Name you entered does not
                  exist! Please try again.<br>";
      include("login_form.inc");
   }
break;                                                     #56

case "new":
   /* Check for blanks */                                  #59
  foreach($_POST as $field => $value)                      #60
  {
     if ($field != "fax")                                  #62
     {
        if ($value == "")                                  #64
        {
           $blanks[] = $field;
        }
     }
  }
  if(isset($blanks))                                       #70
  {
     $message_new = "The following fields are blank.
         Please enter the required information:  ";
     foreach($blanks as $value)
     {
        $message_new .= "$value, ";
     }
     extract($_POST);
     include("login_form.inc");
     exit();
  }

 /* Validate data */
  foreach($_POST as $field => $value)                      #84
  {
     if(!empty($value))                                    #86
     {
        if(eregi("name",$field) and
           !eregi("login",$field))
        {
           if (!ereg("^[A-Za-z' -]{1,50}$",$value))
           {
              $errors[]="$value is not a valid name.";
```

(continued)

Listing 12-2 ***(continued)***

```
            }
         }
         if(eregi("street",$field) or
           eregi("addr",$field) or eregi("city",$field))
         {
            if(!ereg("^[A-Za-z0-9.,' -]{1,50}$",$value))
            {
               $errors[] = "$value is not a valid
                           address or city.";
            }
         }
         if(eregi("state",$field))
         {
            if(!ereg("[A-Za-z]{2}",$value))
            {
              $errors[]="$value is not a valid state.";
            }
         }
         if(eregi("email",$field))
         {
            if(!ereg("^.+@.+\\..+$",$value))
            {
               $errors[] = "$value is not a valid
                           email address.";
            }
         }
         if(eregi("zip",$field))
         {
            if(!ereg("^[0-9]{5,5}(\-[0-9]{4,4})?$",
                $value))
            {
              $errors[]="$value is not a valid
                          zipcode.";
            }
         }
         if(eregi("phone",$field)
            or eregi("fax",$field))
         {
            if(!ereg("^[0-9)(xX -]{7,20}$",$value))
            {
               $errors[] = "$value is not a valid
                           phone number. ";
            }
         }
      } // end if empty                                    #138
   } // end foreach
   if(@is_array($errors))                                 #140
   {
      $message_new = "";
      foreach($errors as $value)
```

```
        {
          $message_new .= $value." Please try
                                   again<br />";
        }
        extract($_POST);
        include("login_form.inc");
        exit();
     }

      /* clean data */
     $cxn = mysqli_connect($host,$user,$passwd,$dbname);

     foreach($_POST as $field => $value)                    #156
     {
        if($field != "Button" and $field != "do")
        {
           if($field == "password")
           {
              $password = strip_tags(trim($value));
           }
           else
           {
              $fields[]=$field;
              $value = strip_tags(trim($value));
              $values[] =
                   mysqli_real_escape_string($cxn,$value);
              $$field = $value;
           }
        }
     }

      /* check whether user name already exists */
     $sql = "SELECT loginName FROM Member
                    WHERE loginName = '$loginName'";    #177
     $result = mysqli_query($cxn,$sql)
               or die("Couldn't execute select query.");
     $num = mysqli_num_rows($result);                   #180
     if ($num > 0)                                      #181
     {
        $message_new = "$loginName already used.
                         Select another User Name.";
include("login_form.inc");
        exit();
     }
    /* Add new member to database */
     else                                               #189
     {
        $today = date("Y-m-d");
        $fields_str = implode(",",$fields);
        $values_str = implode('","',$values);
        $fields_str .=",createDate";
```

(continued)

Listing 12-2 *(continued)*

```
        $values_str .="'".",".'"'.$today;
        $fields_str .=",password";
        $values_str .= '"'.","."md5"."('".$password."')";
        $sql = "INSERT INTO Member ";
        $sql .= "(".$fields_str.")";

        $sql .= " VALUES ";
        $sql .= "(.'"'.$values_str.")";
        $result = mysqli_query($cxn,$sql)
                or die("Couldn't execute insert query.");
        $_SESSION['auth']="yes";                              #204
        $_SESSION['logname'] = $loginName;                    #205

        /* send email to new member */                        #207
        $emess = "A new Member Account has been set up. ";
        $emess.= "Your new Member ID and password are: ";
        $emess.= "\n\n\t$loginName\n\t$password\n\n";
        $emess.="We appreciate your interest in Pet";
        $emess.= " Store at PetStore.com. \n\n";
        $emess.= "If you have any questions or problems,";
        $emess.= " email webmaster@petstore.com";
        $ehead="From: member-desk@petstore.com\r\n";          #215
        $subj = "Your new Member Account from Pet Store";
        $mailsnd=mail("$email","$subj","$emess","$ehead");
        header("Location: New_member.php");                   #218
      }
    break;                                                    #220

    default:                                                  #222
        include("login_form.inc");
  }
?>
```

The ends of some of the lines in Listing 12-2 have line numbers. The following list refers to the line numbers in the listing to discuss the program and how it works:

9 Starts a session. The session has to be started at the beginning of the program, even though the user hasn't logged in yet.

10 Reads in the file that sets the variables needed to connect to the database. The program is called `dogs.inc`, which is a misleading name that seems more secure than calling the program `mypasswords.inc`.

11 Starts a `switch` statement. The `switch` statement contains three sections, based on the value passed for the hidden variable `$do` in the form, obtained from the built-in array `$_POST`. The first section runs when the value passed for `do` is `login`; the second section runs

when the value passed for `do` is `new`; and the third section is the default that runs if no value is passed for `do`. The third section just displays the login page and runs only when the customer first links to the login page.

13 Starts the `case` block for the login section — the section that runs when the customer logs in. The login section of the form sends the hidden variable `$do` with the value `login`, which causes this section of the `switch` statement to run.

14 Lines 14 and 15 connect to MySQL and select the database.

17 Lines 17–20 look in the database table `Member` for a row with the login name typed by the customer.

21 Checks to see whether a row was found with a `loginName` field containing the Member ID typed by the customer. `$num` equals `0` or `1`, depending on whether the row was found.

22 Starts an `if` block that executes if the Member ID was found. This means the user submitted a Member ID that is in the database. This block then checks to see whether the password submitted by the user is correct for the given Member ID. This block is documented in more detail in the following list:

- **24** Lines 24–26 create a query that looks for a row with both the Member ID and the password submitted by the customer. Notice that the password submitted in the form (`$fpassword`) is encrypted by using the MySQL function `md5()`. Passwords in the database are encrypted, so the password you're trying to match must also be encrypted, or it won't match.
- **27** Lines 27–29 execute the query and check whether a match was found. `$num2` equals `1` or `0`, depending on whether a row with both the Member ID and the password is found.
- **30** Starts an `if` block that executes if the password is correct. This is a successful login. Lines 32–40 are executed, performing the following tasks: 1) The two session variables, `auth` and `logname`, are stored in the `SESSION` array. 2) `$today` is created with today's date and time in the correct format expected by the database table. 3) A row for the login is entered into the Login table. 4) The first page of the Members Only section is sent to the member.
- **42** Starts an `else` block that executes if the password is not correct. This is an unsuccessful login. Lines 44–47 are executed, performing the following tasks: 1) The appropriate error message is set in `$message`. 2) The login page is displayed again. The login page will show the error message.



Notice that the block starting on line 42 lets the user know when he or she has a real login name but the wrong password. If the security of your data is important, you may want to write this loop differently. Providing that information may be helpful to someone who is trying to break in because now the cracker needs to find only the password. For more security, just have one condition that gives the same error message whenever either the login name or the password is incorrect. In this example, I prefer to provide the information because it is helpful to the legitimate member (who may not remember whether he or she installed an account at all), and I'm not protecting any vital information.

49 Ends the block that executes when the Member ID is found in the database.

50 Starts an `if` block that executes when the Member ID is *not* found in the database. This could actually be an `else`, instead of an `elseif`, but I think it's clearer to humans with the `if` condition in the statement. This block creates the appropriate error message and shows the login page again, which includes the error message.

56 Ends the `case` block that executes when the customer submits a Member ID and password to log in. The login block extends from line 13 to this line.

58 Starts the `case` block that executes when the customer fills out the form to get a new member account. The form sends the hidden variable `$do` with the value `new`, causing the program to jump to this section of the `switch` statement.

60 Starts a `foreach` loop that loops through every field in the new member form. The loop checks for empty required fields. The statements in the loop are documented in more detail in the following list:

62 Checks whether the field is the fax field. The fax field is not required, so it isn't checked to see whether it is blank.

64 Checks whether the field is blank. If it is, the `if` block adds the field name to an array named `$blanks`.

70 Starts an `if` statement that executes if any blank fields were found. The `if` block creates an error message and redisplays the login form, including the error message.

84 Starts a `foreach` loop that loops through every field in the new member form. The loop checks the field contents for invalid formats. The program doesn't reach this loop until all the required fields contain some data. The nonrequired fields that are blank are not checked (line 86).

This loop contains a series of `if` blocks that check the fields for the correct format. The `if` block tests the content of the field against a regular expression. If the field content is not valid, an information error message is added to an array named `$errors`.

140 Starts an `if` statement that executes when invalid data was found. That is, it executes if the `$errors` array contains any elements. The `if` block creates an error message and redisplays the form, including the error message, so the user can enter the correct information.

156 Begins a `foreach` loop that loops all the New Member form fields to clean the data. The program does not reach this loop until all the required fields are not blank and all the fields contain valid data. The `if` block creates an array of fields to use in the SQL query that inserts the data. It cleans and escapes the field values and adds them to an array of values for use in the SQL `INSERT` query. The `if` block also creates a variable for each field name that contains the cleaned data. The field password is not added to the array because its value needs to be encrypted.

176 Lines 176–180 create and execute a query that checks whether the user name entered by the user already exists. `loginName` must be unique.

181 Starts an `if` statement that executes if the user name already exists. An error message is created and the new member form is redisplayed, with the error message, so that the user can enter a different user name.

189 Starts an `else` statement that executes if the user name was not found in the database. Lines 192–204 create and execute an SQL query that adds the new member to the database, as follows:

- **191** Sets `$today` to today's date in the correct format for the `createDate` field in the `Member` table.
- **192** Converts the array of field names to a string that contains the field names separated by commas.
- **193** Converts the array of values to a string that contains the values separated by commas and enclosed in quotes.
- **194** Adds `createDate` to the string of field names.
- **195** Adds `$today` to the string of values.
- **196** Adds `password` to the string of field names.
- **197** Adds the function that encrypts the password to the string of values.
- **198** Lines 198–203 create and execute the SQL query.

204 Lines 204 and 205 store variables in the session. These variables are available to all pages in the user session. The session variables can be tested on every session page to determine whether the user is logged in.

207 Lines 207–217 create and send an e-mail message to the new member, letting the user know that his or her new account was successfully created. Notice that the e-mail message is created in the variable `$emess` over several lines — beginning on line 208, adding text (using `.=`) on each line, and ending on line 214. This format is needed to make it easier for humans to read — not because PHP needs it. In an e-mail message, unlike in HTML content, extra spaces and line ends have an effect. For instance, if I created one long message and used extra spaces for indentation, those spaces would appear in the e-mail. So I set the message on several lines that I can indent for readability in the program. Line 215 uses the PHP function `mail` to send the e-mail message.

218 Sends the welcome message page for new members to the user's browser.

220 Ends the `case` statement section for the new member form.

222 Starts the `case` block for the default condition. If `$do` is not set to either `"login"` or `"new"`, the program skips to this block. Because both forms on the login page send `$do`, this block executes only the first time this program runs — when the user links to it from the storefront page and has not yet submitted either form. This section has only one statement: a statement that displays the login page.

This program shows the login page in many places, using `include` statements that call the file `login_form.inc`. This file includes the HTML that produces the login page. The program `Login.php` does *not* produce any output. All the output is produced by `login_form.inc`. This type of application organization is discussed in Chapter 10. This is a good example of the use of `include` files. Just imagine if the statements in `login_form.inc`, shown in Listing 12-3, were included in the Login program at each place where `login_form` is included. Whew, that would be a mess that only a computer could understand.

Listing 12-3: File That Creates the Login Page

```
<?php
 /* File: login_form.inc
  * Desc: Displays login page. Page displays two forms:
  *       one form for entering an existing login name
  *       and password and another form for the
  *       information needed to apply for a new account.
*/
include("function12.inc");                                    #8
?>
<html>
<head>
<title>Members Only Login</title>
<style type="text/css"><!--
  .bold_right {font-weight: bold; text-align: right;}
  .gray_banner { font-weight: bold; color: white;
```

```
                background-color: gray;
                text-align: center; font-size: 3em;}
  .bold_large {font-size: 1.1em; font-weight: bold;}
--></style>
</head>
<body style="margin: 0">
<table border="0" cellpadding="5" cellspacing="0">
   <tr><td colspan="3" class="gray_banner">
        Members Only Section</td></tr>
   <tr><td width="33%" valign="top" class="bold_large">
         Are you a member?
         <!-- form for Member login -->
         <form action="Login.php" method="POST">
         <p><table border="0">
<?php                                                          #30
            if (isset($message))
            {
               echo "<tr><td style='color: red'
                 colspan='2' >$message <br /></td></tr>";
            }
?>
           <tr><td class="bold_right">Username</td>
               <td><input type="text" name="fusername"
                       size="20" maxsize="20"></td></tr>
           <tr><td class="bold_right">Password</td>
               <td><input type="password" name="fpassword"
                       size="20" maxsize="20"></td></tr>
                <input type="hidden" name="do"
                                    value="login">
           <tr><td style="text-align: center" colspan="2">
                <br /><input type="submit" name="log"
                             value="Enter"></td></tr>
         </table>
         </form>
       </td>
       <td style="width: 1; background-color: gray"></td>
       <td style="width: 67%"><p>
         <span class="bold_large">Not a member yet?</span>
           Get discounts, a newsletter, advance notice
           of new pets, much more. Fill in the information
           below and join. It's easy and free!</p>
        <!-- form for new member to fill in -->
        <form action="Login.php" method="POST">
         <table border="0" width="100%">
<?php                                                          #60
            if (isset($message_new))
            {
               echo "<tr><td style='color: red;
                         font-weight: bold' colspan='2'>
                         <p>$message_new</p></td></tr>";
            }
?>
```

(continued)

Listing 12-3 *(continued)*

```
            <tr><td class="bold_right">Member ID</td>
                <td><input type="text" name="loginName"
                       value="<?php echo @$loginName ?>"
                       size="20" maxlength="20"></td></tr>
            <tr><td class="bold_right">Password</td>
                <td><input type="password" name="password"
                       value="<?php echo @$password ?>"
                       size="10" maxlength="8"></td></tr>
            <tr><td class="bold_right">First Name</td>
                <td><input type="text" name="firstName"
                       value="<?php echo @$firstName ?>"
                       size="40" maxlength="40"></td></tr>
            <tr><td class="bold_right">Last Name</td>
                <td><input type="text" name="lastName"
                       value="<?php echo @$lastName ?>"
                       size="40" maxlength="40"></td></tr>
            <tr><td class="bold_right">Street</td>
                <td><input type="text" name="street"
                       value="<?php echo @$street ?>"
                       size="55" maxlength="50"></td></tr>
            <tr><td class="bold_right">City</td>
                <td><input type="text" name="city"
                       value="<?php echo @$city ?>"
                       size="40" maxlength="40"></td></tr>
            <tr><td class="bold_right">State</td>
                <td><select name="state">
<?php
                    $stateName=getStateName();          #95
                    $stateCode=getStateCode();          #96
                    for ($n=1;$n<=50;$n++)
                    {
                       $state=$stateName[$n];
                       $scode=$stateCode[$n];
                       echo "<option value='$scode'";
                       if ($scode== "AL")
                           echo " selected";
                       echo ">$state\n";
                    }
?>
                   </select>
                       <b>Zip</b>
                   <input type="text" name="zip"
                       value="<?php echo @$zip ?>"
                       size="10" maxsize="10">
                </td></tr>
            <tr><td class="bold_right">Phone</td>
                <td><input type="text" name="phone"
```

```
                    value="<?php echo @$phone ?>"
                    size="15" maxlength="20">
                   <b>Fax</b>
                <input type="text" name="fax"
                    value="<?php echo @$fax ?>"
                    size="15" maxlength="20"></td></tr>
          <tr><td class="bold_right">Email Address</td>
              <td><input type="text" name="email"
                    value="<?php echo @$email ?>"
                    size="55" maxlength="67"></td></tr>
             <input type="hidden" name="do" value="new">
          <tr><td> </td>
              <td style="text-align: center">
                <input type="submit"
                    value="Become a Member"></td>
          </tr>
        </table>
       </form>
      </td>
   </tr>
   <tr><td colspan="3"
          style="background-color: gray"> </td></tr>
</table>
<div style="text-align: center; font-size: .8em">
    All comments and suggestions are appreciated. Please
    send comments to
    <a href="mailto:webmaster@petstore.com">
    webmaster@petstore.com</a></div>
</body></html>
```

Notice the following points about `login_form`:

- Most of the statements are HTML, with a few small PHP sections here and there.
- The two forms that start on lines 27 and 57 set `action` to the same program and send the same hidden variable, but send a different value for the hidden variable: `do=login` or `do=new`.
- The error messages are shown on the login page by using small PHP sections. Each form has a section, and the message has different names for the two forms: `$message` and `$message_new`. On line 31, the variable `$message` is tested to determine whether it contains a value. A value (an error message) was assigned to `$message` only if an error was found in the form. If `$message` has a value, the message is displayed; if it has no value, the message is not shown. A similar statement on line 61 shows error messages for the new member form.

- A selection drop-down list (started on line 93) is provided for the customer to select the state, guarding against typing errors by the customer. Note that lines 95 and 96 call functions. These functions — my functions, not PHP — are included in the program on line 8. The functions create arrays from a list of state names and a list of two-letter state codes. The functions eliminate the need to include the two 50-state lists in the program. The functions can be used repeatedly for many programs. The `function12.inc` file contains the two functions, as follows:

```
<?php
function getStateCode()
{
  $stateCode = array(1=> "AL" ,
       "AK" ,
       "AZ" ,
       ...
       "WY" );
  return $stateCode;
}

function getStateName()
{
  $stateName = array(1=> "Alabama",
       "Alaska",
       "Arizona",
       ...
       "Wyoming" );
  return $stateName;
}
?>
```

  A `for` loop then creates 50 options for the select list, using the two state arrays.

After running `Login.php`, if the user is successful with a login, the first page of the Members Only section of the Web site is shown. If the user successfully obtains a new user account, the `New_member.php` program runs.

Writing New_member

The New Member Welcome page greets new members by name and provides information about their accounts. Members then have the choice of entering the Members Only section or returning to the main page. Listing 12-4 shows the program that displays the page that new members see.

Listing 12-4: Welcoming New Members

```
<?php
 /* Program: New_member.php
  * Desc:    Displays the new member welcome page. Greets
             member by name and gives user choice to enter
  *          restricted section or go back to main page.
  */
  session_start();                                          #7

  if (@$_SESSION['auth'] != "yes")                          #9
  {
     header("Location: login.php");
     exit();
  }
  include("dogs.inc");                                      #14
  $cxn = mysqli_connect($host,$user,$passwd,$dbname)
         or die ("Couldn't connect to server.");            #16
  $sql = "SELECT firstName,lastName FROM Member
               WHERE loginName='{$_SESSION['logname']}'";
  $result = mysqli_query($cxn,$sql)
            or die("Couldn't execute query");
  $row = mysqli_fetch_assoc($result);
  extract($row);
  echo "<html>
        <head><title>New Member Welcome</title></head>
        <body>
        <h2 style='margin-top: .7in; text-align: center'>
        Welcome $firstName $lastName</h2>\n";             #27
?>
<p>Your new Member Account lets you enter the Members Only
section of our web site. You'll find special discounts and
bargains, a huge database of animal facts and stories,
advice from experts, advance notification of new pets for
sale, a message board where you can talk to other Members,
and much more.</p>
<p>Your new Member ID and password were emailed to you.
   Store them carefully for future use.</p>
<div style="text-align: center">
<p style="margin-top: .5in; font-weight: bold">
         Glad you could join us!</p>
<form action="member_page.php" method="POST">
   <input type="submit"
          value="Enter the Members Only Section">
</form>
<form action="PetShopFrontMembers.php" method="POST">
   <input type="submit" value="Go to Pet Store Main Page">
</form>
</div>
</body></html>
```

Notice the following points about `New_member.php`:

- A session is started on line 7. This makes the session variables stored in `Login.php` available to this program.
- Beginning on line 9, the program checks whether the customer is logged in. When the customer successfully logs in or creates a new account in `Login.php`, `$auth` is set to `yes` and stored in the `$_SESSION` array. Therefore, if `$auth` doesn't equal `yes`, the customer isn't logged in. If a customer tries to run the `New_member.php` program without running the `Login.php` program first, `$_SESSION[auth]` won't equal `yes`, and the user will be sent to the login page.
- The program gets the customer's first and last names from the database, beginning with the database connection statement on line 15. On lines 17 and 18, the query is created by using `$_SESSION[logname]` to search for the member's information. The session variable `logname` that contains the Member ID was set in the login program.
- The PHP section ends on line 28. The remainder of the program is HTML.
- The program uses two different forms to provide two different submit buttons. The form statements on lines 40 and 44 start different programs.

The customer controls what happens next. If the customer clicks the button to return to the main page, the `PetShopFront.php` program runs. If the customer clicks the Members Only Section submit button, the first page of the Members Only section of your Web site is shown.

Writing the Members Only section

The Web pages in the Members Only section are no different than any other Web pages. You just want to restrict them to members who are logged in. To do this, you start a session and check whether they're logged in at the top of every page. The statements for the top of each program are

```
session_start();
if (@$_SESSION['auth'] != "yes")
{
   header("Location: Login.php");
   exit();
}
```

When `session_start` executes, PHP checks for an existing session. If one exists, it sets up the `session` variables. When a user logs in, `$_SESSION[auth]` is set to `yes`. Therefore, if `$_SESSION[auth]` is not set to `yes`, the user is not logged in, and the program takes the user to the login page.

Planning for Growth

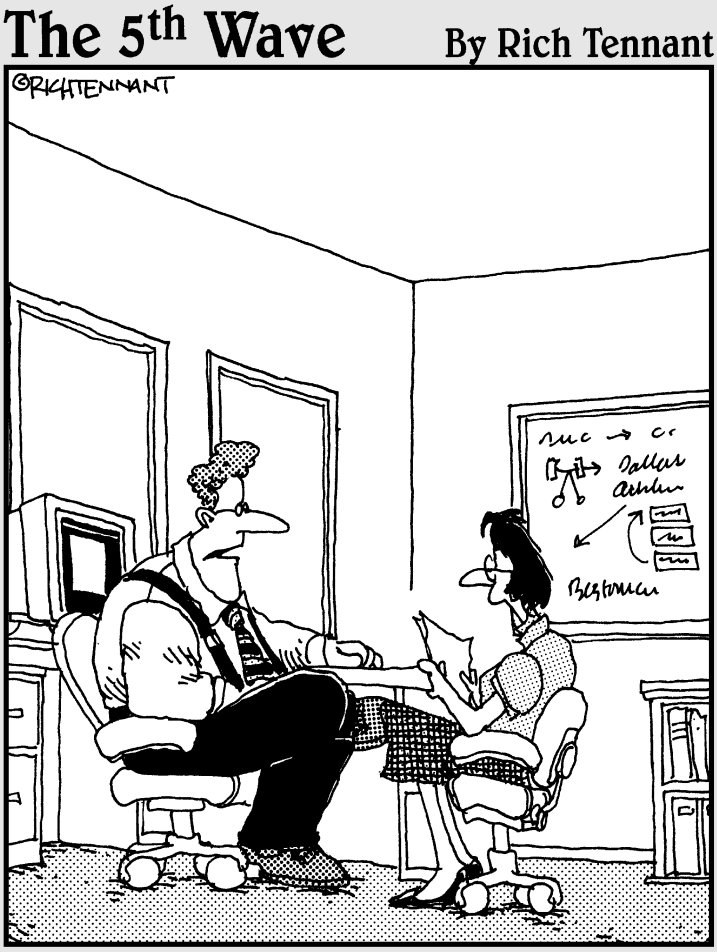
The original plan for an application usually includes every wonderful thing that the user might want it to do. Realistically, it's usually important to make the application available to the users as quickly as possible. Consequently, applications usually go public with a subset of the planned functionality. More functionality is added later. That's why it's important to write your application with growth in mind.

Looking at the login application in this chapter, I'm sure you can see many things that could be added to it. Here are some possibilities:

- **E-mail a forgotten password.** Users often forget their passwords. Many login applications have a link that users can click to have their passwords e-mailed to them.
- **Change the password.** Members might want to change their password. The application could offer a form for password changes.
- **Update information.** Members might move or change their phone number or e-mail address. The application could provide a way for members to change their own information.
- **Create a member list.** You might want to output a nicely formatted list of all members in the database. This probably is something you want to make available only for yourself. In some situations, however, you might want to make the list available to all members.

You can easily add any of these abilities to the application. For instance, you can add to the login form a *Forgot my password* button that, when clicked, e-mails the password to the e-mail address in the database. The button can run the login program with a section for e-mailing the password or run a different program that e-mails the password. In the same manner, you can add buttons for changing the password or updating customer information. You don't need to wait until an application has all its bells and whistles to let your customers use it. You can write it one step at a time.

Part V
The Part of Tens



"Our automated response policy to a large company-wide data crash is to notify management, back up existing data and sell 90% of my shares in the company."

In this part . . .

The chapters in this part contain hints, tips, and warnings based on my experience. Perhaps they can serve as a shortcut for you on your journey to becoming a confident Web developer. I sincerely hope so.

Chapter 13

Ten Things You Might Want to Do Using PHP Functions

In This Chapter

- Finding out about many useful functions
- Understanding what functions can do

One of the strongest aspects of PHP is its many built-in functions. In this chapter, I list the PHP functions that I use most often. I describe some of them elsewhere in this book, some I mention only in passing, and some I don't mention at all. There are many hundreds of functions in the PHP language. For a complete list, see the PHP documentation at `www.php.net/manual/en/funcref.php`.

Communicate with MySQL

PHP has many functions designed specifically for interacting with MySQL. I describe the following MySQL functions thoroughly in this book:

```
mysqli_connect();mysqli_fetch_assoc()
mysqli_num_rows();     mysqli_query()
```

The following functions could be useful, but I either don't discuss them or discuss them only briefly:

- `mysqli_insert_id($cxn)`: For use with an AUTO-INCREMENT MySQL column. This function gets the last number inserted into the column.
- `mysqli_select_db($cxn,$database)`: Selects a database. The currently selected database is changed to the specified database. All succeeding queries are executed on the selected database.

- `mysqli_fetch_row($result)`: Gets one row from the temporary results location. The row is put into an array with numbers as the keys.
- `mysqli_affected_rows($result)`: Returns the number of rows that were affected by a query — for instance, the number of rows deleted or updated.
- `mysqli_num_fields($result)`: Returns the number of fields in a result.
- `mysqli_field_name($result, N)`: Returns the name of the row indicated by *N*. For instance, `mysqli_field_name($result,1)` returns the name of the second column in the result. The first column is 0.

If you use any of the preceding functions with MySQL 4.0 or older, the function's name begins with `mysql_s` rather than `mysqli_`.

Send E-Mail

PHP provides a function that sends e-mail from your PHP program. The format is

```
mail(address,subject,message,headers);
```

These are the values that you need to fill in:

- *address*: The e-mail address that will receive the message.
- *subject*: A string that goes on the subject line of the e-mail message.
- *message*: The content that goes inside the e-mail message.
- *headers*: A string that sets values for headers. For instance, you might have a *headers* string as follows:

  ```
  "From: member-desk@petstore.com\r\nbcc: mom@hercompany.com"
  ```

  The header would set the From header to the given e-mail address, plus send a blind copy of the e-mail message to mom.

The following is an example of PHP statements that you can use in your script to set up and send an e-mail message:

```
$to = "janet@valade.com";
$subj = "Test";
$mess = "This is a test of the mail function";
$headers = bcc:techsupport@mycompany.com\r\n
$mailsend = mail($to,$subj,$mess,$headers);
```

Sometimes you might have a problem with your e-mail. PHP has a configuration setting that must be correct before the mail function can connect to your system e-mail software. The default is usually correct, but if your e-mail doesn't seem to be getting to its destination, check the PHP configuration mail setting by looking for the following in the output of `phpinfo()`:

```
Sendmail_path           (on Unix/Linux)
SMTP                    (on Windows)
```

You can change the setting by editing the `php.ini` file. Look for the following lines:

```
[mail function]
; For Win32 only.
SMTP = localhost

; For Win32 only.
sendmail_from = me@localhost.com

; For Unix only.
;sendmail_path =
```

Windows users need to change the first two settings. The first setting is where you put the name of your outgoing mail server. However you send e-mail — using a LAN at work, a cable modem at home, an ISP via a modem — you send your mail with an SMTP server, which has an address that you need to know.

If you send directly from your computer, you should be able to find the name of the outgoing mail server in your e-mail software. For instance, in Microsoft Outlook Express, choose Tools➪Accounts➪Properties and then click the Servers tab. If you can't find the name of your outgoing mail server, ask your e-mail administrator for the name. If you use an ISP, you can ask the ISP. The name is likely to be in a format similar to the following:

```
mail.ispname.net
```

The second setting is the return address sent with all your e-mail. Change the setting to the e-mail address that you want to use for your return address, as follows:

```
sendmail_from = Janet@Valade.com
```

The third setting is for Unix users. The default is usually correct. If it doesn't work, talk to your system administrator about the correct path to your outgoing mail server. This setting usually refers to Linux as well.

Don't forget to remove the semicolon at the beginning of the lines. The semicolon makes the line into a comment, so the setting isn't active until you remove the semicolon.

Use PHP Sessions

The functions to open or close a session follow. I explain these functions in Chapter 9.

```
session_start();     session_destroy()
```

Stop Your Program

Sometimes you just want your program to stop, cease, and desist. Two functions do this: `exit()` and `die()`. Actually, these are two names for the same function, but sometimes it's just more fun to say "die." Both print a message when they stop if you provide one. The format is

```
exit("message string");
```

When `exit` executes, *`message string`* is output.

Handle Arrays

Arrays are useful in PHP, particularly for getting the results from database functions and for form variables. I explain the following array functions elsewhere in the book, mainly in Chapter 7:

```
array();     extract();     sort();     asort();
rsort();     arsort();      ksort();    krsort();
```

Here are some other useful functions:

- `array_reverse($varname)`: Returns an array with the values in reverse order.
- `array_unique($varname)`: Removes duplicate values from an array.
- `in_array("string",$varname)`: Looks through an array `$varname` for a string `"string"`.
- `range(value1,value2)`: Creates an array containing all the values between *`value1`* and *`value2`*. For instance, `range('a','z')` creates an array containing all the letters between *a* and *z*.

- `explode("sep","string")`: Creates an array of strings in which each item is a substring of *string,* separated by *sep*. For example, `explode(" ",$string)` creates an array in which each word in `$string` is a separate value. This is similar to `split` in Perl.
- `implode("glue",$array)`: Creates a string containing all the values in `$array` with *glue* between them. For instance, `implode(", ",$array)` creates a string: `value1`, `value2`, `value3`, and so on. This is similar to the `join` function in Perl.

There are many more useful array functions. PHP can do almost anything with an array.

Check for Variables

Sometimes you just need to know whether a variable exists. These functions can be used to test whether a variable is currently set:

```
isset($varname);  // true if variable is set
!isset($varname); // true if variable is not set
empty($varname);  // true if value is 0 or is not set
```

Format Values

Sometimes you need to format the values in variables. In Chapter 6, I explain how to format numbers into dollar format by using `number_format()` and `sprintf()`. In Chapter 6, I also discuss `unset()`, which removes the values from a variable. In this section, I describe additional capabilities of `sprintf()`.

The function `sprintf()` allows you to format any string or number, including variable values. The general format is

```
$newvar = sprintf("format",$varname1,$varname2,...);
```

where *format* gives instructions for the format and `$`*varname* contains the value(s) to be formatted. *format* can contain both literals and instructions for formatting the values in `$`*varname*. In addition, a *format* containing only literals is valid, such as the following statement:

```
$newvar = sprintf("I have a pet");
```

This statement outputs the literal string. However, you can also add variables, using the following statements:

```
$ndogs = 5;
$ncats = 2;
$newvar = sprintf("I have %s dogs and %s cats",$ndogs,$ncats);
```

The `%s` is a formatting instruction that tells `sprintf` to insert the value in the variable as a string. Thus, the output is `I have 5 dogs and 2 cats`. The `%` character signals `sprintf` that a formatting instruction starts here. The formatting instruction has the following format:

```
%pad-width.dectype
```

These are the components of the formatting instructions:

- `%`: Signals the start of the formatting instruction.
- *`pad`*: A padding character used to fill out the number when necessary. If you don't specify a character, a space is used. `pad` can be a space, a `0`, or any character preceded by a single quote (`'`). It's common to pad numbers with `0` — for example, `01` or `0001`.
- `-`: A symbol meaning to left-justify the characters. If this isn't included, the characters are right-justified.
- `width`: The number of characters to use for the value. If the value doesn't fill the width, the padding character is used to pad the value. For instance, if `width` is 5, `pad` is 0, and the value is 1, the output is `00001`.
- `.dec`: The number of decimal places to use for a number.
- `type`: The type of value. Use `s` for most values. Use `f` for numbers that you want to format with decimal places.

Some possible `sprintf` statements are

```
sprintf("I have $%03.2f. Does %s have any?",$money,$name);
sprintf("%'.-20s%3.2f",$product,$price);
```

The output of these statements is

```
I have $030.00. Does Tom have any?
Kitten.............. 30.00
```

Compare Strings to Patterns

In earlier chapters in this book, I use regular expressions as patterns to match strings. (I explain regular expressions in Chapter 6.) The following functions use regular expressions to find and sometimes replace patterns in strings:

- `ereg("pattern",$varname)`: Checks whether the *pattern* is found in `$varname`. `eregi` is the same function except it ignores uppercase and lowercase.
- `ereg_replace("pattern","string",$varname)`: Searches for *pattern* in `$varname` and replaces it with *string*. `eregi_replace` is the same function except it ignores uppercase and lowercase.

Find Out about Strings

Sometimes you need to know things about a string, such as its length or whether the first character is an uppercase *O*. PHP offers many functions for checking out your strings:

- `strlen($varname)`: Returns the length of the string.
- `strpos("string","substring")`: Returns the position in *string* where *substring* begins. For instance, `strpos("hello","el")` returns `1`. Remember that the first position for PHP is `0`. `strrpos()` finds the last position in *string* where *substring* begins.
- *substr*`("string",n1,n2)`: Returns the substring from *string* that begins at *n1* and is *n2* characters long. For instance, `substr("hello",2,2)` returns `ll`.
- `strtr($varname,"str1","str2")`: Searches through the string `$varname` for *str1* and replaces it with *str2* every place that it's found.
- `strrev($varname)`: Returns the string with the characters reversed.

Many more string functions exist. See the documentation at `www.php.net`.

Change the Case of Strings

Changing uppercase letters to lowercase and vice versa is not so easy. Bless PHP for providing functions to do this for you:

- `strtolower($varname)`: Changes any uppercase letters in the string to lowercase letters
- `strtoupper($varname)`: Changes any lowercase letters in the string to uppercase letters
- `ucfirst($varname)`: Changes the first letter in the string to uppercase
- `ucwords($varname)`: Changes the first letter of each word in the string to uppercase

Chapter 14

Ten PHP Gotchas

In This Chapter

- Recognizing common PHP errors
- Interpreting error messages

I guarantee that you will do all the things that I mention in this chapter. It's not possible to write programs without making these mistakes. The trick is to find out how to recognize them, roll your eyes, say, "Not again," and then correct your mistakes. One error message that you will see many times is

```
Parse error: parse error in c:\test.php on line 7
```

This is PHP's way of saying, "Huh?" It means it doesn't understand something. This message helpfully points to the file and the line number where PHP got confused. Sometimes it points directly at the error, but sometimes PHP's confusion results from an error earlier in the program.

Missing Semicolons

Every PHP statement ends with a semicolon (`;`). PHP doesn't stop reading a statement until it reaches a semicolon. If you leave out the semicolon at the end of a line, PHP continues reading the statement on the following line. For instance, consider the following statement:

```
$test = 1
echo $test;
```

The statement doesn't make sense to PHP when it reads the two lines as one statement, so it complains with an error message, such as the annoying

```
Parse error: parse error in c:\test.php on line 2
```

Before you know it, you'll be writing your home address with semicolons at the end of each line.

Not Enough Equal Signs

When you ask whether two values are equal in a comparison statement, you need two equal signs (==). Using one equal sign is a common mistake. It's perfectly reasonable because you have been using one equal sign to mean *equal* since the first grade, when you discovered that 2 + 2 = 4. This is a difficult mistake to recognize because it doesn't cause an error message. It just makes your program do odd things, like infinite loops. I'm continually amazed at how long I can stare at the following and not see why it's looping endlessly:

```
$test = 0;
while ( $test = 0 )
{
   $test++;
}
```

Misspelled Variable Names

An incorrectly spelled variable name is another PHP gotcha that doesn't result in an error message, just odd program behavior. If you misspell a variable name, PHP considers it a new variable and does what you ask it to do. Here's another clever way to write an infinite loop:

```
$test = 0;
while ( $test == 0 )
{
   $Test++;
}
```



To PHP, `$test` is not the same variable as `$Test`.

Missing Dollar Signs

A missing dollar sign in a variable name is hard to see, but at least it most likely results in an error message telling you where to look for the problem. It usually results in the old familiar parse error:

```
Parse error: parse error in test.php on line 7
```

Troubling Quotes

You can have too many, too few, or the wrong kind of quotes. You have too many when you put quotes inside of quotes, such as

```
$test = "<table width="100%">";
```

PHP sees the second double quote (") — before `100` — as the ending double quote (") and reads the `1` as an instruction, which makes no sense. Voilà! Another parse error. The line must be either

```
$test = "<table width='100%'>";
```

or

```
$test = "<table width=\"100%\">";
```

You have too few quotes when you forget to end a quoted string, such as

```
$test = "<table width='100%'>;
```

PHP will continue reading the lines as part of the quoted string until it encounters another double quote ("), which might not occur for several lines. This is one occasion when the parse error pointing to where PHP got confused is not pointing to the actual error. The error occurred some lines previously, when you forgot to end the string.

You have the wrong kind of quotes when you use a single quote (') when you meant a double quote (") or vice versa. The difference between single and double quotes is sometimes important, as I explain in Chapter 6.

Invisible Output

Some statements, such as the header statement, must execute before the program produces any output. If you try to use such statements after sending output, they fail. The following statements will fail because the header message isn't the first output:

```
<html>
<?php
   header("Location: http://company.com");
?>
```

<html> is not in a PHP section and is therefore sent as HTML output. The following statements will work:

```
<?php
   header("Location: http://company.com");
?>
<html>
```

The following statements will fail

```
 <?php
   header("Location: http://company.com");
?>
<html>
```

because there's one single blank space before the opening PHP tag. The blank space is output to the browser, although the resulting Web page looks empty. Therefore, the header statement fails because there is output before it. This is a common mistake and difficult to spot.

Numbered Arrays

In PHP, the first value in an array is numbered zero (0). Of course, humans tend to believe that lists start with the number one (1). This fundamentally different way of viewing lists results in us humans believing an array isn't working correctly when it's working just fine. For instance, consider the following statements:

```
$test = 1;
while ( $test <= 3 )
{
   $array[] = $test;
   $test++;
}
echo $array[3];
```

Nothing is displayed by these statements. I leap to the conclusion that there's something wrong with my loop. Actually, it's fine. It just results in the following array:

```
$array[0]=1
$array[1]=2
$array[2]=3
```

and doesn't set anything into $array[3].

Including PHP Statements

When a file is read in using an include statement in a PHP section, it seems reasonable to me that the statements in the file will be treated as PHP statements. After all, PHP adds the statements to the program at the point where I include them. However, PHP doesn't see it my way. If a file named `file1.inc` contains the following statements:

```
if ( $test == 1 )
       echo "Hi";
```

and I read it in with the following statements in my main program:

```
<?php
$test = 1;
include ("file1.inc");
?>
```

I expect the word `Hi` to appear on the Web page. However, the Web page displays this:

```
if ( $test == 1 ) echo "Hi";
```

Clearly, the file that is included is seen as HTML. To send `Hi` to the Web page, `file1.inc` needs to contain the following statements:

```
<?php
if ( $test == 1 )
       echo "Hi";
?>
```

Missing Mates

Parentheses and curly brackets come in pairs and must be used that way. Opening with a `(` that has no closing `)` or a `{` without a `}` will result in an error message. One of my favorites is using one closing parenthesis where two are needed, as in the following statement:

```
if ( isset($test)
```

This statement needs a closing parenthesis at the end. It's much more difficult to spot that one of your blocks didn't get closed when you have blocks inside blocks inside blocks. For instance, consider the following:

```
while ( $test < 3 )
{
if ( $test2 != "yes" )
{
if ( $test3 > 4 )
{
echo "go";
}
}
```

You can see there are three opening curly brackets and only two closing ones. Imagine that 100 lines of code are inside these blocks. It can be difficult to spot the problem — especially if you think the last closing bracket is closing the `while` loop, but PHP sees it as closing the `if` loop for `$test2`. Somewhere later in your program, PHP might be using a closing bracket to close the `while` loop that you aren't even looking at. It can be difficult to trace the problem in a large program.

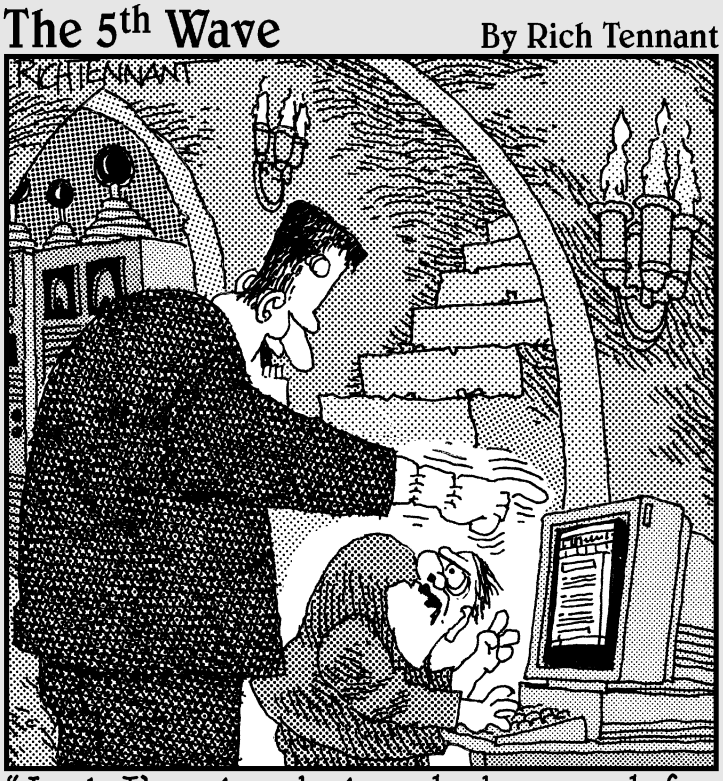
Indenting blocks makes it easier to see where closing brackets belong. Also, I often use comments to keep track of where I am, such as

```
while ( $test < 3 )
{
 if ( $test2 != "yes" )
 {
  if ( $test3 > 4 )
  {
   echo "go";
  } // closing if block for $test3
 } // closing if block for $test2
} // closing while block
```

Confusing Parentheses and Brackets

I'm not sure whether mistaking parentheses for brackets and vice versa is a problem for everyone or just for me because I refuse to admit that I can't see as well as I used to. Although PHP has no trouble distinguishing between parentheses and curly brackets, my eyes are not so reliable. Especially while staring at a computer screen at the end of a ten-hour programming marathon, I can easily confuse `(` and `{`. Using the wrong one gets you a parse error message.

Part VI
Appendixes



"Look, I've already launched a search for 'reanimated babe cadavers' three times and nothing came up!"

In this part . . .

Appendix A provides instructions for installing MySQL, and Appendix B does the same for PHP. Appendix C provides installation and configuration information that could be helpful if you need to install Apache.

Appendix A

Installing MySQL

Although MySQL runs on many platforms, I describe how to install it on Linux, Unix, Windows, and Mac, which together account for the majority of Web sites on the Internet. Be sure to read the instructions all the way through before beginning the installation.

MySQL can be installed most easily from binaries — precompiled, ready-to-install packages. Binaries are available for most operating systems: Linux, Windows, Mac, FreeBSD, many flavors of Unix, and others. If such a package is available for your operating system, use it. Only install MySQL from source if it's necessary, such as when there's no binary for your operating system or you need some functionality that's not compiled into the binaries (for example, a different character set).

If you have trouble starting the MySQL server after installing it, check the error log for useful information. The error log is located in the data directory and has the extension `.err`.

On Windows

In most cases, when you download and install MySQL, the server is started automatically. If it isn't or if you need to stop and start it for another reason, you can start it manually as I describe in the section, "Starting and stopping the MySQL server." You can also set up MySQL so that it starts every time your computer starts.

Downloading and installing MySQL

To install MySQL on Windows, follow these steps:

1. **Point your Web browser to `www.mysql.com`, the MySQL home page.**
2. **Click Downloads in the lower-right corner of the Web page.**
3. **Scroll down the screen until you come to the MySQL Community Edition heading, and click the link for the version you want to download.**

At present, MySQL 5.0 is shown as the recommended version. Most people should download the recommended version. The Downloads page for the selected version opens, such as the MySQL 5.0 Downloads page.

4. **Scroll down the screen until you come to the Windows Downloads heading, and click Pick a Mirror by the version you want to download.**

   For most people, Windows Essentials is the best choice. The Select a Mirror page opens.

5. **Scroll down the page to locate the mirror closest to your location, and click HTTP by the selected mirror.**

   A dialog box opens.

6. **Select the option to save the file.**

   A dialog box opens that lets you select where you want the file saved.

7. **Navigate to where you want to save the file (for example, `c:\downloads`), and then click Save.**

   After the download, you see a file in the download location (for example, `c:\downloads`) containing the MySQL installer. The file is named `mysql-essential-`, followed by the version number, followed by `-win32.msi` — for instance, `mysql-essential-5.0.22-win32.msi`.

8. **Verify the file you just downloaded**.

   See the "Verifying a Downloaded File" section, later in this appendix.

9. **Double-click the installer (.msi) file.**

   The opening screen shown in Figure A-1 is displayed. ***Note:*** If you're installing from a Windows NT/2000/XP system, be sure that you're logged into an account with administrative privileges.

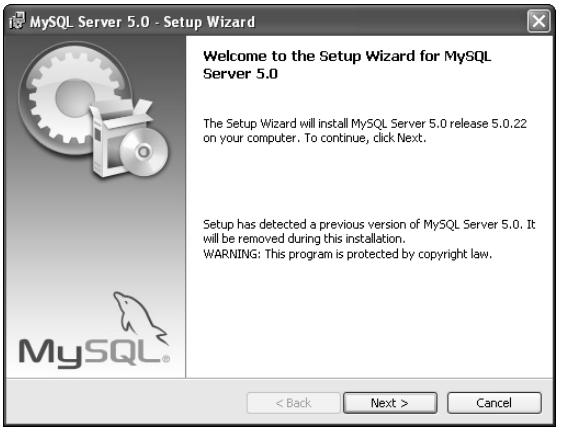


Figure A-1: The opening screen of the MySQL setup wizard.

10. **Click Next.**

    You see a screen for choosing the type of installation.

11. **Select Typical and then click Next.**

    The Ready to Install Program window opens. The current settings are displayed.

12. **Click Install.**

    The installation of MySQL begins. When the installation in complete, a Sign-Up dialog box opens.

13. **Click Skip Signup and then click Next.**

    The Wizard Completed window opens, as shown in Figure A-2.

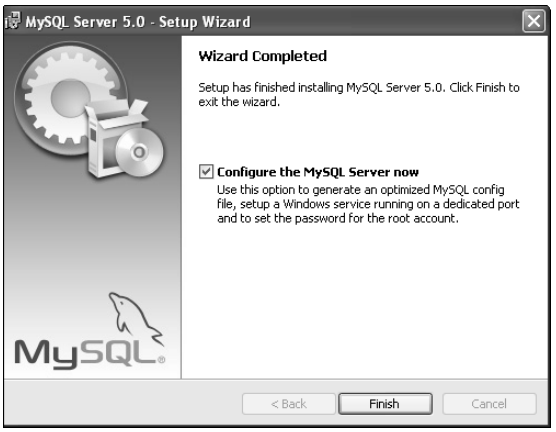


Figure A-2: The Wizard Completed screen of the MySQL setup wizard.

14. **If you are installing this version of the server for the first time, select Configure the MySQL Server Now.**

    If you are upgrading the MySQL server, such as from MySQL 5.0.18 to 5.0.22, you may not need to configure the server. The wizard will give it the same configuration as the existing version. However, if you are upgrading to a new major version, such as from MySQL 5.0 to MySQL 5.1, you will need to run the configuration wizard.

15. **Click Finish.**

If you selected Configure the MySQL Server Now, the configuration wizard starts immediately. Running the MySQL configuration wizard is explained in the next section. If you did not select it, the installation wizard stops running.

Running the MySQL configuration wizard

MySQL must be configured after it is installed. You need to assign a password to the MySQL account, named root, which is installed automatically. You need to start the server and set it up so that it automatically starts when your computer boots.

MySQL provides a configuration wizard. The configuration wizard starts immediately after installation if you selected Configure the MySQL Server Now in the final installation screen. You can also start the configuration wizard at any time using a menu item in the MySQL Start Menu.

1. **Choose Start➪All Programs➪MySQL➪MySQL Server 5.0➪MySQL Server instance config wizard.**

   The configuration wizard starts.

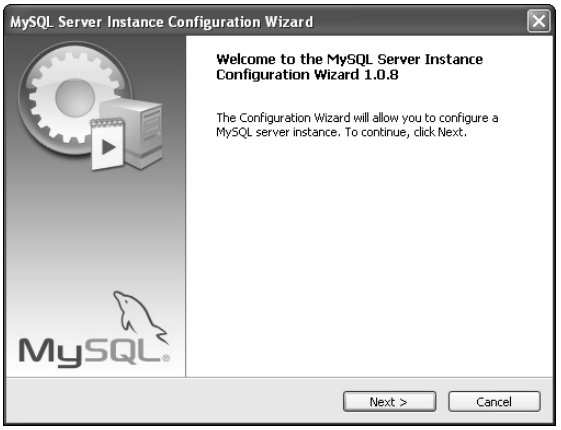


Figure A-3: The first screen in the MySQL configuration wizard.

2. **If you have more than one version of MySQL installed, a screen appears and you can click the version you want to configure. Then, click Next.**

   The MySQL Server Configuration Types window opens.

3. **Click Standard Configuration and then click Next.**

   The Windows Options dialog box opens.

4. **Select Install as a Windows Service.**

   If you are using Windows 98/Me, installing as a Windows service is not possible. Instead, select Add bin Directory to Windows PATH and skip to Step 7.



5. **In the Service Name box, type** mysql50**.**
6. **Select Launch the MySQL Server Automatically.**
7. **Click Next.**

   The security options dialog box opens, as shown in Figure A-4.



Figure A-4: The security options screen in the MySQL configuration wizard.

8. **Select Modify Security Settings.**
9. **In the New Root Password box, type a password. In the Confirm box, retype the same password.**

   You are now setting the password for the root account for your MySQL server. You must use the root account to access your MySQL database. You need to remember the password you type here.



10. **If you are setting up a development environment that no one can access but you, you can select Create an Anonymous Account.**

    An anonymous account is handy. However, if there is any access to your MySQL server from the Internet, do not create an anonymous account. It's a security risk.

11. **Click Next.**

    The Ready to Execute window opens.

12. **Click Execute.**

    A message appears when the configuration is complete.

Starting and stopping the MySQL server

With your MySQL server set up to be running whenever your computer is running, you may sometimes need to stop or start the server. For instance, if you upgrade MySQL, you must shut down the server before starting the upgrade.

Windows NT/2000/XP

To stop or start the MySQL server, do the following:

1. **Choose Start➪Control Panel➪Administrative Tools➪Services.**

   A list of all current services appears.

2. **Scroll down the alphabetical listing, and click the MySQL service you want to stop or start.**

   Stop or Start links appear to the left of the service name.

3. **Click Stop or Start.**

If you do not find the MySQL server in the list, you can set it up as a service using the configuration wizard described in the previous section.

Windows 98/Me

If you are using Windows 98/Me, setting up MySQL as a service is not possible. However, you can start the server manually as follows:

1. **Open a Command Prompt (perhaps called DOS) window.**

   Choose Start➪Programs➪Accessories➪Command Prompt.

2. **Change to the `bin` directory in the directory where MySQL is installed.**

   For instance, you might type `cd c:\Program Files\MySQL\MySQL Server 5.0\bin`.

3. **Type** mysqld**.**

   If this command fails, type **mysqld-nt**. Which program name you type depends on the MySQL version.

   If the server starts, no message is displayed. You must leave this window open while the server is running. If you close the window, the server will shut down, although it sometimes does not shut down immediately. An error message is displayed if the server is unable to start.

4. **Test whether the server is running.**

   To test whether the server is running correctly, open another command prompt window, as described in Step 1. Change to the directory you changed to in Step 2 and type **mysql**. If the server is not running, you will see an error message.

Manual shutdown

Sometimes you may have difficulty shutting down the server. You can shut the server down manually as follows:

1. **Open a Command Prompt (perhaps called DOS) window.**

   Choose Start➪Programs➪Accessories➪Command Prompt.

2. **Change to the `bin` directory in the directory where MySQL is installed.**

   For instance, you might type cd c:\Program Files\MySQL\MySQL Server 5.0\bin.

3. **Type** mysqladmin -u root -p shutdown**.**

   In this command, the account is `root`. The `-p` means password, so you will be prompted to type a password. If the account you specify does not require a password, leave out the `-p`.

On Linux and Unix

Many Linux computers come with MySQL already installed. Many Linux systems install, or give you the option to install, MySQL when Linux is installed. Many Linux systems, such as Fedora, SuSE, and Ubuntu, include built-in utilities that download and install MySQL for you, often the most recent version. In many cases, installing MySQL provided by the Linux distribution is an easier, more efficient choice than downloading and installing MySQL from the MySQL Web site. Check the Web site for your Linux distribution to see whether they offer an easy way to install a current version of MySQL.

If you are installing MySQL on a Linux computer, you can install it from RPM files that you download from the MySQL Web site. Using RPM is the easiest way to install on Linux. See the following section, "Using RPM (Linux only)." For Unix, you can install from a binary file — a file specific to your operating system. The file is formatted for the installation software your operating system uses. As of this writing, MySQL binary files are available for Solaris, HP-UX, AIX, SCO, SGI Irix, OpenBSD, FreeBSD, and other Unix flavors.

If neither an RPM file nor a binary works for you, you can always install MySQL from source files. To do this, follow the instructions in the "From source files" section.

Using RPM (Linux only)

MySQL can be installed on Linux using RPM. Although RPM stands for *Red Hat Package Manager,* RPM is available on most flavors of Linux, not just Red Hat. You can download the RPM file using the following instructions. However, the RPM file might already be on the CD that your Linux operating system came on. Installing the RPM file from a CD saves you the trouble of downloading (you can skip Steps 1–9 in the following list), but if the version of MySQL on your CD is not the most recent, you might want to download an RPM file anyway.

To install MySQL on Linux from an RPM file provided on the MySQL Web site, follow these steps:

1. **Point your Web browser to `www.mysql.com`, the MySQL home page.**
2. **Click Downloads in the lower-right corner of the Web page.**

   The MySQL Downloads page opens.
3. **Scroll down the screen until you come to the MySQL Community Edition heading, and click the link for the version you want to download.**

   At present, MySQL 5.0 is shown as the recommended version. Most people should download the recommended version. The Downloads page for the selected version opens, such as the MySQL 5.0 Downloads page.
4. **Scroll down the screen until you come to the Downloads heading for the appropriate version of Linux, and select Pick a Mirror by the RPM file you want to download.**

   At present, you can find RPMs for Red Hat, SuSE, and Ubuntu, and also generic RPMs for other flavors of Linux. You need to download both the server RPM and the client RPM. The Select a Mirror page opens.
5. **Scroll down the page to locate the mirror closest to your location, and click HTTP by the selected mirror.**

   A dialog box opens.
6. **Select the option to save the file.**

   A box opens that lets you select where you want to save the file.
7. **Navigate to where you want to save the RPM (for example, `/usr/src/mysql`) and then click Save.**
8. **Repeat Steps 4–7 to download the RPM file for Client Programs into the same download location.**
9. **Change to the directory where you saved the downloads.**

For instance, you might type `cd /usr/src/mysql`. You see two files in the directory — one file named `MySQL-server-`, followed by the version number and `.i386.rpm`, and a second file named similarly with `client` embedded in its name. For example, `MySQL-server-5.0.22-0.i386.rpm` and `MySQL-client-5.0.22-0.i386.rpm`.

10. **Verify the downloaded files.**

    See the "Verifying a Downloaded File" section, later in this appendix.

11. **Install the RPM by entering this command:**

    ```
    rpm -i listofpackages
    ```

    For instance, using the example in Step 9, the command would be this:

    ```
    rpm -i MySQL-server-5.0.22-0.i386.rpm MySQL-client-
            5.0.22-0.i386.rpm
    ```

    This command installs the MySQL packages. It sets the MySQL account and group name that you need, and creates the data directory at `/var/lib/mysql`. It also starts the MySQL server and creates the appropriate entries in `/etc/rc.d` so that MySQL starts automatically whenever your computer starts.

    You need to be using an account that has permissions to successfully run the `rpm` command, such as a root account.

12. **To test that MySQL is running okay, type this:**

    ```
    bin/mysqladmin --version
    ```

    You should see the version number of your MySQL server.

From source files

Before you decide to install MySQL from source files, check for RPMs or binary files for your operating system. MySQL RPMs and binary files are precompiled, ready-to-install packages for installing MySQL and are convenient and reliable.

You install MySQL by downloading source files, compiling the source files, and installing the compiled programs. This process sounds technical and daunting, but it's not. However, read all the way through the following steps before you begin the installation procedure. To install MySQL from source code, follow these steps:

1. **Point your Web browser to `www.mysql.com`, the MySQL home page.**
2. **Click Downloads in the lower-right corner of the Web page.**

   The MySQL Downloads page opens.

3. **Scroll down the screen until you come to the MySQL Community Edition heading, and then click the link for the version you want to download.**

   At present, MySQL 5.0 is shown as the recommended version. Most people should download the recommended version. The Downloads page for the selected version opens, such as the MySQL 5.0 Downloads page.

4. **Scroll to the bottom of the screen to the Source Downloads heading, and click Pick a Mirror by the tarball version (currently the first file in the Source Downloads section).**

   The Select a Mirror page opens.

5. **Scroll down the page to locate the mirror closest to your location, and click HTTP by the selected mirror.**

   A dialog box opens.

6. **Select the option to save the file.**

   A box opens that lets you select where the file will be saved.

7. **Navigate to where you want to install MySQL and then click Save.**

   The standard location is `/usr/local`. It's best to use the standard location if possible.

8. **After the download is complete, change to the download directory — for instance, `cd-/usr/local`.**

   You see a file named `mysql-`, followed by the version number and `.tar.gz`. — for instance, `mysql-5.0.22.tar.gz`. This file is a *tarball,* a file that is a container for many files and subdirectories.

9. **Verify the file you just downloaded.**

   See the "Verifying a Downloaded File" section, later in this appendix.

10. **Create a user and group ID for MySQL to run under by using the following commands:**

```
groupadd mysql
useradd -g mysql mysql
```

    The syntax for the commands might differ slightly on different versions of Unix, or they might be called `addgroup` and `adduser`.

    Note: You must be using an account authorized to add users and groups.

11. **Unpack the tarball by typing**

```
gunzip -c filename | tar -xvf -
```

    For example:

```
gunzip -c mysql-5.0.22.tar.gz | tar -xvf -
```

You see a new directory named `mysql-`*`version`* — for instance, `mysql-5.0.22` — which contains many files and subdirectories. You must be using an account that is allowed to create files in `/usr/local`.

12. **Change to the new directory.**

    For instance, you might type `cd mysql-5.0.22`.

13. **Type the following:**

    ```
    ./configure --prefix=/usr/local/mysql
    ```

    You see several lines of output. The output will tell you when `configure` has finished. This might take some time.

14. **Type** make**.**

    You see many lines of output. The output will tell you when `make` has finished. `make` might run for some time.

15. **Type** make install**.**

    `make install` will finish quickly.

    Note: You might need to run this command as `root`.

16. **Type the following:** `scripts/mysql_install_db`**.**

    This command runs a script that initializes your MySQL databases.

17. **Make sure that the ownership and group membership of your MySQL directories are correct. Set the ownership with these commands:**

    ```
    chown -R root  /usr/local/mysql
    chown -R mysql /usr/local/mysql/data
    chgrp -R mysql /usr/local/mysql
    ```

    These commands make `root` the owner of all the MySQL directories except `data` and make `mysql` the owner of `data`. All MySQL directories belong to group `mysql`.

18. **Set up your computer so that MySQL starts automatically when your machine starts by copying the file `mysql.server` from `/usr/local/mysql/support-files` to the location where your system has its startup files.**

19. **To test MySQL, you can start your server manually, without restarting your computer, by typing the following:**

    ```
    bin/safe_mysqld --user=mysql &
    ```

20. **To test that MySQL is running okay, type:**

    ```
    bin/mysqladmin --version
    ```

    You should see the version number of your MySQL server.

On Mac

You can download MySQL using a Mac OS X 10.2 (Jaguar) or later PKG binary package. If your operating system is OS X 10.1 or earlier, you can't use this package. You will need to download a tarball and install MySQL from source code, as described in the previous section.

1. **Point your Web browser to `www.mysql.com`, the MySQL home page.**
2. **Click Downloads in the lower-right corner of the Web page.**

   The MySQL Downloads page opens.
3. **Scroll down the screen until you come to the MySQL Community Edition heading, and click the link for the version you want to download.**

   At present, MySQL 5.0 is shown as the recommended version. Most people should download the recommended version. The Downloads page for the selected version opens, such as the MySQL 5.0 Downloads page.
4. **Scroll down to the Mac OS X Downloads heading. Locate the section for your version of OS X, and click Pick a Mirror by the standard version.**

   The Downloads heading is in the bottom half of the Downloads page. The Select a Mirror page opens.
5. **Scroll down the page and locate the mirror closest to your location, and click HTTP by the selected mirror.**

   A dialog box opens.
6. **Select the option to save the file.**

   A box opens that lets you select where the file will be saved.
7. **Navigate to where you want to install MySQL and then click Save.**

   The standard location is `/usr/local`. It is best to use the standard location if possible.
8. **After the download is complete, change to the download directory — for instance, `/usr/local`.**

   You see a package named `mysql-standard-`, followed by the version number and the OS number and `dmg`, such as `mysql-standard-5.0.22-osx10.3-powerpc.dmg`. If the downloaded file does not have the extension `.dmg`, change the filename to give it the `.dmg` extension.
9. **Verify the file you just downloaded.**

   See the "Verifying a Downloaded File" section, later in this appendix.

10. **Create a user and a group named `mysql` for MySQL to run under.**

    In most newer Mac versions, this user and group already exist.

11. **Mount the disk image by double-clicking its icon in the Finder.**

12. **Double-click the package icon to install the MySQL PKG.**

    The package installer runs and installs the package. It installs MySQL in the directory `/usr/local/mysql-`, followed by the version number. It also installs a symbolic link, `/usr/local/mysql/`, pointing to the directory where MySQL is installed. It also initializes the database by running the script `mysql_install_db`, which creates a MySQL account called root.

13. **If necessary, change the owner of the `mysql` directory.**

    The directory where MySQL is installed (for example, `/usr/local/mysql-5.0.22`) should be owned by root. The data directory (such as `/usr/local/mysql-5.0.22/data`) should be owned by the account `mysql`. Both directories should belong to the group `mysql`. If the user and group are not correct, change them with the following commands:

    ```
    sudo chown -R root /usr/local/mysql-5.0.22
    sudo chown -R mysql /usr/local/mysql-5.0.22/data
    sudo chown -R root /usr/local/mysql-5.0.22/bin
    ```

14. **Start the MySQL server using the following commands:**

    ```
    cd /usr/local/mysql
    sudo ./bin/mysqld_safe
    if necessary, enter your password
    Press Ctrl-Z
    bg
    Press Ctrl-D or type exit
    ```

    This starts the server manually, meaning you must start the MySQL server every time you restart your computer. To have your server start every time the computer starts, you need to install the MySQL Startup Item, which is included in the installation disk image in a separate installation package. To install the Startup Item, double-click the MySQLStartupItem.pkg icon.

To stop the MySQL server, change to the `bin` subdirectory in the directory where MySQL is installed and type

```
mysqladmin –u root –p shutdown
```



The `-p` causes `mysqladmin` to prompt you for a password. If the account doesn't require a password, don't include `-p`.

Verifying a Downloaded File

As a security precaution, the MySQL Web site provides methods to verify the software after you download it, to make sure the file has not been altered by bad guys. You can verify using either the MD5 method or the PGP method. The MD5 method is simpler and is described in this section.

Next to the file you downloaded, on the download Web page, a long string, called a *signature,* is displayed, similar to the following:

```
MD5: 6112f6a730c680a4048dbab40e4107b3
```

The downloaded MySQL file needs to provide the same MD5 signature shown on the download page. You use software on your computer to check the MD5 signature of the downloaded file. Your Linux or Mac system includes software to check the MD5 signature. On Windows, you may need to download and install MD5 software. You can find software that checks MD5 signatures at `www.formilab.ch/md5/`.

To check the MD5 signature of the downloaded file, type the following at a command-line prompt, such as in a command prompt window in Windows, in the directory where the downloaded file resides:

```
md5 filename
```

where *`filename`* is the name of the file that you downloaded, such as `md5 mysql-essential-5.0.22-win32.msi`. In Windows, you may need to copy the downloaded file to the directory where the MD5 software (such as `md5.exe`) is installed, change to this directory using the `cd` command, and then type the preceding command.

A signature appears that should be the same signature displayed by the filename on the download page of the MySQL Web site.

A simple, open source (free) Windows program with a graphical interface that allows you to check MD5 signatures by clicking buttons and dragging filenames, rather than by typing commands in a command prompt window, can be obtained at `www.nullriver.com/index/products/winmd5sum`.

You can verify the downloads for Apache and PHP with a similar procedure.

Configuring MySQL

MySQL reads a configuration file when it starts up. If you use the defaults or an installer, you probably don't need to add anything to the configuration file. However, if you install MySQL in a nonstandard location or want the databases to be stored somewhere other than the default, you might need to edit the configuration file. The configuration file is named `my.ini` or `my.cnf`. It's located in your system directory (such as `Windows` or `Winnt`) if you are using Windows and in `/etc` on Linux, Unix, and Mac. The file contains several sections and commands. The following commands in the `mysqld` section sometimes need to be changed:

```
[mysqld]

# The TCP/IP Port the MySQL Server will listen on
port=3306

#Path to installation directory. All paths are
#     usually resolved relative to this.
basedir="C:/Program Files/MySQL/MySQL Server 5.0/"

#Path to the database root
datadir="C:/Program Files/MySQL/MySQL Server 5.0/Data/"
```

The `#` at the beginning of the line makes the line into a comment. The `basedir` line tells the MySQL server where MySQL is installed. The `datadir` line tells the server where the databases are located. You can change the port number to tell the server to listen for database queries on a different port.

Appendix B

Installing PHP

Although PHP runs on many platforms, I describe installing it on Unix, Linux, Mac, and Windows, which includes the majority of Web sites on the Internet. PHP runs with several Web servers, but these instructions focus mainly on Apache and Internet Information Servers (IIS) because together they power almost 90 percent of the Web sites on the Internet. If you need instructions for other operating systems or Web servers, see the PHP Web site, at `www.php.net`. This appendix provides installation instructions for PHP 5 and 6. If you're installing an earlier version, there are some small differences, so read the `install.txt` file provided with the PHP distribution.

Installing PHP on Unix, Linux, or Mac with Apache

In this section, I provide instructions on installing PHP versions 5 and 6 on Unix, Linux, and Mac.

On Unix and Linux

You can install PHP as an Apache module or as a stand-alone interpreter. If you're using PHP as a scripting language in Web pages to interact with a database, install PHP as an Apache module. PHP is faster and more secure as a module. I don't discuss PHP as a stand-alone interpreter in this book.

You install PHP by downloading source files, compiling these files, and installing the compiled programs. This process isn't as technical and daunting as it sounds. I provide step-by-step instructions in the next few sections. Read all the way through the steps before you begin the installation procedure.



For Linux users only: PHP for Linux is available in an RPM as well as in source files. It might be in RPM format on your distribution CD. However, when you install PHP from an RPM, you can't control the options that PHP is installed with. For instance, you need to install PHP with MySQL support enabled, but the RPM may not have MySQL support enabled. MySQL is popular, so many RPMs enable support for it, but it is out of your control. Also, an RPM usually enables all the most popular options, so an RPM might enable options that you don't need. Consequently, the simplest and most efficient way to install PHP could be from the source. If you're familiar with RPMs, feel free to find an RPM and install it. RPMs are available. However, I am providing steps for source code installation, not RPMs.

Before installing

Before beginning to install PHP, check the following:

- **The Apache module `mod_so` is installed.** It usually is. To display a list of all the modules, type the following:

  ```
  httpd --l
  ```

  You might have to be in the directory where `httpd` is located before the command will work. The output usually shows a long list of modules. All you need to be concerned with for PHP is `mod_so`. If `mod_so` is not loaded, Apache must be reinstalled using the `enable-module=so` option.

- **The `apxs` utility is installed.** `apxs` is installed when Apache is installed. You should be able to find a file called `apxs`. If Apache was installed on Linux from an RPM, `apxs` might not have been installed. Some RPMs for Apache consist of two RPMs: one for the basic Apache server and one for Apache development tools. Possibly the RPM with the development tools, which installs `apxs`, needs to be installed.

- **The Apache version is recent.** See Appendix C for information about Apache versions. To check the version, type the following:

  ```
  httpd --v
  ```

  You might have to be in the directory where `httpd` is located before the command will work.



  As of this writing, the PHP Web site does not recommend using Apache 2 with PHP on Linux/Unix. For use on production Web sites, it might be better to use Apache 1.3 than Apache 2. See Appendix C for a discussion of Apache versions. Keep updated on the status of PHP with Apache 2 by checking `www.php.net/manual/en/install.unix.apache2.php`.

Installing

To install PHP on Unix or Linux with an Apache Web server, follow these steps:

1. **Point your Web browser to `www.php.net`, the PHP home page.**
2. **Click Downloads at the left end of the top menu bar.**
3. **Click the latest version of the PHP source code, such as version 5.2.0 or 6.0.0.**

   A dialog box opens. The file you are about to download contains many files compressed into one file — a *tarball.*
4. **Select the option to save the file.**

   A dialog box opens that lets you select where the file will be saved.
5. **Navigate to where you want to save the source code (for example, `/usr/src`), and then click Save.**
6. **After the download, change to the download directory (for instance, `cd-/usr/src`).**

   You see a file named `php-`, followed by the version name and `tar.gz`.
7. **Verify the file you just downloaded.**

   See the "Verifying a Downloaded File" section in Appendix A.
8. **Unpack the tarball. The command for PHP version 6.0.0 is**

   ```
   gunzip -c php-6.0.0.tar.gz | tar -xf -
   ```

   A new directory called `php-6.0.0` is created with several subdirectories.
9. **Change to the new directory that was created when you unpacked the tarball.**

   For example, type cd php-6.0.0.
10. **Type the `configure` command.**

    Use one of the two following `configure` commands:

    ```
    ./configure --with-mysqli=DIR --with-apxs
    ./configure --with-mysql=DIR --with-apxs
    ```

    When using `with-mysqli`, use the path to the file named `mysql_config`.

    Use `mysql` if you're using MySQL 4.0 or earlier; use `mysqli` if you're using MySQL 4.1 or later. `DIR` is the path to the appropriate MySQL directory. When using `with-mysql`, use the path to the directory where `mysql` is installed, for instance:

    ```
    --with-mysql=/user/local/mysql
    ```

If you're using Apache 2, use the option `with-apxs2`. (See Appendix C for information on using Apache 2.)

You will see many lines of output. Wait until the `configure` command has finished. This might take a few minutes. If the `configure` command fails, it provides an informative message. Usually, the problem is missing software. You see an error message indicating that xyz software can't be found or that xyz version 5.6 is required but xyz version 4.2 is found. You need to install or update the software that PHP needs.



If the `apxs` utility isn't installed in the expected location, you see an error message indicating that `apxs` couldn't be found. If you get this message, check the location where `apxs` is installed (`find / -name apxs`) and include the path in the `with-apxs` option of the `configure` command: `--with-apxs=/usr/sbin/apxs` or `/usr/local/apache/bin/apxs`. If you're using Apache 2, the option is `--with-apxs2=/usr/sbin/apxs`.

11. **Type** make**.**

    You will see many lines of output. Wait until it's finished. This might take a few minutes.

12. **Type** make install**.**

On Mac OS X

With the release of PHP 4.3, you can install PHP on Mac OS X as easily as on Unix and Linux. You install PHP by downloading source files, compiling the source files, and installing the compiled programs. This process isn't as technical and daunting as it might appear. I provide step-by-step instructions in the next few sections. Read all the way through the steps before you begin to be sure that you understand it all clearly and have everything prepared so you don't have to stop in the middle of the installation.

Before installing

If you want to use PHP with Apache for your Web site, Apache must be installed. Most Mac OS X systems come with Apache already installed. For more information on Apache, see Appendix C.

Before beginning to install PHP, check the following:

- **The Apache version is recent:** See Appendix C for a discussion of Apache versions. To check the version, type the following on the command line:

  ```
  httpd --v
  ```

  You might have to be in the directory where `httpd` is located before the command will work.



As of this writing, the PHP Web site does not recommend using Apache 2 with PHP. For use on production Web sites, it might be better to use Apache 1.3 than Apache 2. See Appendix C for a discussion of Apache versions. Keep updated on the status of PHP with Apache 2 by checking the PHP Web site at `www.php.net/manual/en/install.unix.apache2.php`.

- **The Apache module `mod_so` is installed.** It usually is. To display a list of all the modules, type the following:

  ```
  httpd --l
  ```

  You might have to be in the directory where `httpd` is located before the command will work. The output usually shows a long list of modules. All you need to be concerned with for PHP is `mod_so`. If `mod_so` is not loaded, Apache must be reinstalled.

- **The `apxs` utility is installed.** `apxs` is normally installed when Apache is installed. To determine whether it's installed on your computer, you should look for a file called `apxs`, usually in the `/usr/sbin/apxs` directory. If you can find the file, `apxs` is installed; if not, it's not.

- **The files from the Developer's Tools CD are installed.** This CD is supplemental to the main Mac OS X distribution. If you can't find the CD, you can download the tools from the Apple Developer Connection Web site at `developer.apple.com/tools/macosxtools.html`.

Installing

To install PHP on Mac, follow these steps:

1. **Point your Web browser to `www.php.net`, which is the PHP home page.**
2. **Click Downloads at the left end of the top menu bar.**
3. **Click the latest version of the PHP source code, which may be version 6.0.0.**

   A dialog box opens.

4. **Select the option to save the file.**

   A dialog box opens that lets you select where the file is to be saved.

5. **Navigate to where you want to save the source code (for example, `/usr/src`), and then click Save.**
6. **After the download, change to the download directory (for example, `cd-/usr/src`).**

   You see a file named `php-`, followed by the version name and `tar.gz`. This file contains several files compressed into one file. The file might have been unpacked by the StuffIt Expander automatically so that you see the directory `php-6.0.0`. If so, skip to Step 8.

7. **Verify the file you just downloaded.**

   See the "Verifying a Downloaded File" section in Appendix A.

8. **Unpack the tarball.**

   The command to unpack the tarball for PHP version 6.0.0 is

   ```
   tar xvfz php-6.0.0.tar.gz
   ```

   A new directory called `php-6.0.0` is created with several subdirectories.

9. **Change to the new directory that was created when you unpacked the tarball.**

   For example, you can use a command like the following:

   ```
   cd php-6.0.0
   ```

10. **Type the `configure` command:**

    The `configure` command consists of `./configure` followed by all the necessary options. The minimum set of options follows:

    - **Location options:** Because the Mac stores files in different locations than the PHP default locations, you need to tell PHP where files are located. Use the following options:

      ```
      --prefix=/usr
      --sysconfdir=/etc
      --localstatedir=/var
      --mandir=/usr/share/man
      ```

    - **`zlib` option:** `--with-zlib`.
    - **Apache option:** If you are installing PHP for use with Apache, use the following option: `--with-apxs` or `--with-apxs2`.

    The most likely configuration command is

    ```
    ./configure --prefix=/usr --sysconfdir=/etc
        --localstatedir=/var --mandir=/usr/share/man
         --with-apxs --with-zlib
    ```

    You also need to use an option to include MySQL support. Use one of the following options:

    ```
    --with-mysql=DIR

    --with-mysqli=DIR
    ```

    When using `with-mysqli`, use the path to the `mysql_config` file.

Use `mysql` if you're using MySQL 4.0 or earlier; use `mysqli` if you're using MySQL 4.1 or later. `DIR` is the path to the appropriate MySQL directory. When using `with-mysql`, use the path to the directory where `mysql` is installed, as follows:

```
--with-mysql=/user/local/mysql
```

You can type the `configure` command on one line. If you use more than one line, type a `\` at the end of each line.

You will see many lines of output. Wait until the `configure` command has finished. This may take a few minutes.



If the `apxs` utility isn't installed in the expected location, you see an error message, indicating that `apxs` could not be found. If you get this error message, check the location where `apxs` is installed (`find / -name apxs`) and include the path in the `with-apxs` option of the `configure` command: `--with-apxs=/usr/sbin/apxs`.

You might need to use many other options, such as options that change the directories where PHP is installed. These `configure` options are discussed in the "Installation options" section, later in this appendix.

11. **Type** make**.**

    You will see many lines of output. Wait until it is finished. This might take a few minutes.

12. **Type** sudo make install**.**

Alternative installation methods for Mac

Most Mac OS X versions since X.3 come with PHP already installed but it may not be enabled. If PHP is installed but doesn't seem to be working, try following the instructions in "Configuring Apache for PHP," later in this appendix. Editing the `httpd.conf` file may be all that's needed to get your PHP up and running.

A PHP compiled package is available for some versions of OS X. If a package is available for your operating system, it is easier than non-package software to install. However, you have no control over the support that is compiled into PHP. It may not have the support you need or it may have extra support that you will not use. If you are unfamiliar with compiling and installing source code on the Mac, go to `http://www.entropy.ch/software/macosx/php/` and check out what's available. You might make life easier for yourself.

Installation options

The previous sections give you steps to quickly install PHP with the options needed for the applications in this book. However, you might want to install PHP differently. For instance, all the PHP programs and files are installed in their default locations, but you might need to install PHP in different locations. Or you might be planning applications using additional software. You can use additional command-line options if you need to configure PHP for your specific needs. Just add the options to the command shown in Step 10 of the Unix and Mac installation instructions. In general, the order of the options in the command line doesn't matter. Table B-1 shows the most commonly used options for PHP. To see a list of all possible options, type **`./configure --help`**.

Table B-1 PHP Configure Options

Option	Tells PHP To
`prefix=`*`PREFIX`*	Set main PHP directory to *`PREFIX`*. Default *`PREFIX`* is `/usr/local`.
`exec-prefix=`*`EPREFIX`*	Install architecture dependent files in *`EPREFIX`*. Default *`EPREFIX`* is *`PREFIX`*.
`bindir=`*`DIR`*	Install user executables in *`DIR`*. Default is *`EPREFIX`*`/bin`.
`infodir=`*`DIR`*	Install info documentation in *`DIR`*. Default is *`PREFIX`*`/info`.
`mandir=`*`DIR`*	Install `man` files in *`DIR`*. Default is *`PREFIX`*`/man`.
`with-config-file-path=`*`DIR`*	Look for the configuration file (`php.ini`) in *`DIR`*. Without this option, PHP looks for the configuration file in a default location, usually `/usr/local/lib`.
`disable-libxml`	Disable XML support that's included by default.
`enable-ftp`	Enable FTP support.
`enable-magic-`	Enable automatic escaping of quotes with a backslash.
`with-apxs=`*`FILE`*	Build a shared Apache module using the `apxs` utility located at *`FILE`*. Default *`FILE`* is `apxs`.
`with-apxs2=`*`FILE`*	Build a shared Apache 2 module using the `apxs` utility located at *`FILE`*. Default *`FILE`* is `apxs`.

Option	Tells PHP To
`with-mysql=`*DIR*	Enable support for MySQL 4.0 or earlier databases. Default *DIR* where MySQL is located is `/usr/local`.
`with-mysqli=`*DIR*	Enable support for MySQL 4.1 or later databases. *DIR* needs to be the path to the file named `mysql_config` that was installed with 4.1. Available only with PHP 5 or later.
`with-openssl=`*DIR*	Enable OpenSSL support for a secure server. Requires OpenSSL version 0.9.5 or later.
`with-oci8=`*DIR*	Enable support for Oracle 7 or later. Default *DIR* is contained in the environmental variable, `ORACLE_HOME`.
`with-oracle=`*DIR*	Enable support for earlier versions of Oracle. Default *DIR* is contained in the environmental variable, `ORACLE_HOME`.
`with-pgsql=`*DIR*	Enable support for PostgreSQL databases. Default *DIR* where PostgreSQL is located is `/usr/local/pgsql`.
`with-servlet=`*DIR*	Include servlet support. *DIR* is the base install directory for the JSDK. The Java extension must be built as a shared `.dll`.

Configuring Apache for PHP

You must configure Apache to recognize and run PHP files. An Apache configuration file, `httpd.conf`, is on your system, possibly in `/etc` or in `/usr/local/apache/conf`. You must edit this file before PHP can run properly.

Follow these steps to configure your system for PHP:

1. **Open the `httpd.conf` file so you can make changes.**
2. **Configure Apache to load the PHP module.**

   Find the list of `LoadModule` statements. Look for the following line:

   ```
   LoadModule php6_module libexec/libphp6.so
   ```

   If this line isn't there, add it. If a pound sign (#) is at the beginning of the line, remove the pound sign.

For PHP 5, the line would be:

```
LoadModule php5_module libexec/libphp5.so
```

3. **Configure Apache to recognize PHP extensions.**

   You need to tell Apache which files might contain PHP code. Look for a section describing `AddType`. You might see one or more `AddType` lines for other software. Look for the `AddType` line for PHP, as follows:

   ```
   AddType application/x-httpd-php .php
   ```

   If you find it with a pound sign (#) at the beginning of the line, remove the pound sign (#). If you don't find this line, add it to the `AddType` statements. This line tells Apache to look for PHP code in all files with a `.php` extension. You can specify any extension or series of extensions.

4. **Start the Apache httpd server (if it is not running) or restart the Apache httpd server (if it is running).**

   You can start or restart the server by using a script that was installed on your system during installation. This script might be `apachectl` or `httpd.apache`, and might be located in `/bin` or `/usr/local/apache/bin`. For example, you might be able to start the server by typing `apachectl start`, restart it by using `apachectl restart`, or stop it by using `apachectl stop`. Sometimes restarting is not sufficient; you must stop the server first and then start it.

Installing PHP on Windows

PHP runs on Windows 98/Me and Windows NT/2000/XP. Windows 98/Me can be used for development on a local computer but cannot support a public Web site. Windows 95 is no longer supported as of PHP 4.3.0. PHP does not run on Windows 3.1.

To install PHP 5/6 on Windows with MySQL support, you download a Zip file that contains all the necessary files for PHP. The following steps show how to install PHP on Windows:

1. **Point your Web browser at `www.php.net`.**
2. **Click Download on the left end of the top menu bar.**
3. **Go to the Windows Binaries section. Click the download link for the zip package for the most recent version of PHP (as of this writing, 6.0.0).**
4. **Click the link for a mirror Web site from which to download the file and choose the site closest to your location.**

   A dialog box opens.

5. **Select the option to save the file.**

   A dialog box opens that lets you select where the file will be saved.

6. **Navigate to where you want the file to be downloaded. (This should be a temporary location, such as a download directory.) Then click Save.**

   After the download is complete, you see a file in the download location containing all the files needed. The file is named `php-`, followed by the version number and `-Win32.zip`. For example, the file might be named `php-6.0.0-Win32.zip`.

7. **Verify the file you just downloaded.**

   See the "Verifying a Downloaded File" section in Appendix A.

8. **Extract the files from the `.zip` file into the directory where you want PHP to be installed, such as `c:\php`.**

   If you double-click the `.zip` file, it should open in the software on your computer that extracts files from `.zip` files, such as WinZip or PKZIP. Select the menu item for extract and select the directory into which the files are to be extracted. `C:\php` is a good choice for installation because many configuration files assume that's where PHP is installed, so the default settings are more likely to be correct.



   It's best not to install PHP in a directory with a space in the path, such as in `Program Files/PHP`. Doing so can cause problems.

   You now have a directory and several subdirectories that contain all the files from the zip file.

9. **Copy the file required for MySQL to the PHP main directory.**

   The file is located in the `ext` subdirectory in the directory where PHP is installed. Copy one of the following files, depending on which version of MySQL you're using:

```
ext\php_mysqli.dll  (for MySQL 4.1 or later)
ext\php_mysql.dll   (for MySQL 4.0 or earlier)
```

   Copy the file into the main PHP directory, such as `c:\php`.



   Another file, named `libmysql.dll`, is required for MySQL support. This file should already be located in the main PHP directory. If it isn't, you need to find it and copy it there. If it's not in your PHP directory, it's usually installed with MySQL, so find it in the directory where MySQL was installed, perhaps in a `bin` subdirectory, such as `c:\Program Files\MySQL\MySQL Server 5.0\bin`.



   Occasionally PHP needs DLL files that it can't find. When this happens, PHP displays an error message when you run a PHP program, saying that it can't find a particular DLL. You can usually find the DLL in the `ext` subdirectory and copy it into the main PHP directory.

10. **Configure your Web server.**

    The next section provides instructions for configuring your Web server.

11. **Configure PHP.**

    Follow the directions in the "Configuring PHP" section, later in this appendix.

Configuring your Web server for PHP

Your Web server needs to be configured to recognize PHP scripts and run them. You can't have Apache and IIS running at the same time using the same port number. Either shut down one Web server or tell them to listen on different ports.

Follow the steps in the section for your Web server.

Configuring Apache

You must edit an Apache configuration file, called `httpd.conf`, before PHP can run properly. To configure Apache for PHP, follow these steps:

1. **Open `httpd.conf` for editing.**

   You might be able to edit it by choosing Start➪Programs➪Apache HTTPD Server➪Configure Apache Server➪Edit Configuration. If Edit Configuration isn't on your Start menu, find the `httpd.conf` file on your hard drive, usually in the directory where Apache is installed, in a `conf` subdirectory (for example, `c:\program files\Apache group\Apache\conf`). Open this file in an editor, such as Notepad or WordPad.

2. **Activate the PHP module.**

   Look for the module statement section in the file and locate the following line:

   ```
   #LoadModule php6_module "c:/php/php6apache2.dll"
   ```

   Remove the # from the beginning of the line to activate the line. If you are installing PHP 5, you need the following line:

   ```
   LoadModule php5_module "c:/php/php5apache2.dll"
   ```

   If you are using Apache 1.3, rather than Apache 2, the module name is `php6apache.dll` or `php5apache.dll`.

3. **Tell Apache which files are PHP programs.**

   Look for a section describing `AddType`. This section might contain one or more `AddType` lines for other software. The `AddType` line for PHP is

   ```
   AddType application/x-httpd-php .php
   ```

   Look for this line. If you find it with a pound sign (#) at the beginning of the line, remove the pound sign. If you don't find the line, add it to the list of `AddType` statements. You can specify any extension or series of extensions.

This line tells Apache that files with the `.php` extension are files of the type `application/x-httpd-php`. Apache then knows to send files with `.php` extensions to the PHP module.

4. **Start Apache (if it is not running) or restart Apache (if it is running).**

   You can start it as a service on Windows NT/2000/XP by choosing Start➪Programs➪Apache HTTPD Server➪Control Apache Server and then selecting Start or Restart. You can start it on Windows 98/Me by choosing Start➪Programs➪Apache Web Server➪Management.

   Sometimes restarting Apache is not sufficient; you must stop it first and then start it. In addition, your computer is undoubtedly set up so that Apache will start whenever the computer starts. Therefore, you can shut down and then start your computer to restart Apache.

Configuring IIS

To configure IIS to work with PHP, follow these steps:

1. **Enter the IIS Management Console.**

   You should be able to enter by choosing Start➪Programs➪Administrative Tools➪Internet Services Manager or Start➪Control Panel➪Administrative Tools➪Internet Services Manager.

2. **Right-click your Web site (such as Default Web Site).**
3. **Select Properties.**
4. **Click the Home Directory tab.**
5. **Click the Configuration button.**
6. **Click the App Mappings tab.**
7. **Click Add.**
8. **In the Executable box, type the path to the PHP interpreter.**

   For example, type c:\php\php-cgi.exe.

9. **In the Extension box, type** .php**.**

   This will be the extension associated with PHP scripts.

10. **Select the Script Engine check box.**
11. **Click OK.**

    Repeat Steps 6–10 if you want any extensions in addition to `.php` to be processed by PHP, such as `.phtml`.

Configuring PHP

PHP uses settings in a file named `php.ini` to control some of its behavior. PHP looks for `php.ini` when it begins and uses the settings that it finds. If PHP can't find the file, it uses a set of default settings. The default location for the `php.ini` file is one of the following unless you change it during installation:

- **Windows:** The system directory, depending on the Windows version: on Windows 98/Me/XP, `windows`; on Windows NT/2000 (and sometimes XP), `winnt`
- **Unix, Linux, and Mac:** `/usr/local/lib`

If the `php.ini` file isn't installed during installation, you need to install it now. A configuration file with default settings, called `php.ini-dist`, is included in the PHP distribution. Copy this file into the appropriate location, such as the default locations just mentioned, changing its name to `php.ini`.



If you have a previous version of PHP installed (such as PHP 4.3), make a backup copy of the `php.ini` file before you overwrite it with the `php.ini` file for PHP 6. You can then see the settings you are currently using and change the settings in the new `php.ini` file to match the current settings.

To configure PHP, follow these steps:

1. **Open the `php.ini` file for editing.**
2. **Change the settings you want to change.**

   Steps 3, 4, and 5 mention some specific settings that should **always** be changed if you are using the specified environment.

3. **Only if you are using PHP 5 or earlier, turn off magic quotes.**

   Look for the following line:

   ```
   magic_quotes-gpc On
   ```

   Change `On` to `Off`.

4. **Only if you are using PHP 5/6 on Windows, activate `mysqli` or `mysql` support.**

   Look for a list of extensions. Find the line for the `mysqli` extension, as follows:

   ```
   ;extension=php_mysqli.dll
   ```

If you are using a version of PHP earlier than 5.0 or a MySQL version earlier than 4.1, find the following line:

```
;extension=php_mysql.dll
```

Notice the semicolon (`;`) at the beginning of the lines. To activate the extension, remove the semicolon. If the extension line isn't in your `php.ini` file, add it.

5. **Only if you're using PHP on Windows with the IIS Web server, turn off force redirect. Find the line:**

   ```
   ;cgi.force_redirect = 1
   ```

   You need to remove the semicolon so that the setting is active, and also change the `1` to `0`. After the changes, the line looks as follows:

   ```
   cgi.force_redirect = 0
   ```

6. **Save the `php.ini` file.**

7. **Restart the Apache server so that the new settings go into effect.**

In general, the remaining default settings allow PHP to run okay, but you might need to edit some of these settings for specific reasons. I discuss settings in the `php.ini` file throughout the book when I discuss a topic that might require you to change settings. For example, PHP error-handling actions can be changed by settings in the `php.ini` file. The possible settings for error handling and their effects are discussed in Chapter 4.

If you change settings in your `php.ini` file but the changes don't seem to have the expected effect on PHP operations, one of two things is probably the cause. First, you must restart the Web server before the changes go into effect. Second, you may not be editing the `php.ini` file in the location where PHP is reading it. You can check which `php.ini` file PHP is reading. You may have more than one `php.ini` file or you may have it stored in the wrong location. When you test PHP using the `phpinfo()` statement, as shown in Chapter 2 in the "Testing PHP" section, PHP outputs many variable values and settings. One of the settings close to the top is Configuration File Path, which shows the path to the location where PHP is looking for the configuration file. If the path ends in a filename, that's the file PHP is using for its configurations. If the path ends in a directory name, PHP is looking in the directory for the configuration file but can't find it, so PHP is using its default configurations.

Appendix C

Installing and Configuring Apache

Apache is an open source Web server. A *Web server* delivers the files on the Web site to the visitor who wants to see the Web pages. All recent versions of Mac OS X come with Apache already installed. Most Linux distributions include Apache. However, you may want to install Apache yourself to install a newer version or to install with different options. Windows does not come with Apache installed. You must install it yourself.

Selecting a Version of Apache

Apache is currently available in three versions: Apache 1.3, Apache 2.0, and Apache 2.2. All three versions are supported and upgraded. The PHP software runs with all three versions, but some other software related to PHP might have problems with Apache 2.0 or 2.2. On Windows, Apache 2.0 and 2.2 are not supported on Windows 9x installations; they require Windows NT, 2000, or XP.

Apache 2 changed considerably from Apache 1.3; Apache 2.2 changed from Apache 2.0. Some third-party modules may not work correctly on all three versions. Third-party modules that run on 1.3 will not work correctly with Apache 2, and modules that work on Apache 2.0 may not work correctly with Apache 2.2. Therefore, only modules that have been modified for Apache 2 or 2.2 can run on Apache 2 or 2.2.

On the PHP Web site, the recommended setup at present is to use PHP 4.3.0 or later with the most recent version of Apache 2.0. Check the Web page for the current status of PHP with Apache versions at

```
www.php.net/manual/en/install.apache2.php#install.apache2.unix
```

At this time, the current releases are Apache 2.2.2, 2.0.58, and 1.3.36.

Try to install the most current release of the Apache version you choose so that your Apache server includes all the latest security and bug fixes. New features are no longer being added to Apache 1.3, but bugs are still being fixed and security issues are being addressed. New versions of Apache 1.3 continue to be released but on a less frequent basis than for Apache 2.0 or 2.2.

Installing Apache on Linux and Unix

Apache can be downloaded and installed on your Web server for free. It's available for almost every operating system, including Windows, Linux, many flavors of Unix, and Mac.

To install Apache on Linux or Unix, you download the source code, compile it, and install it. This is much easier than it sounds.

Before installing

Before installing Apache, check the following requirements:

- **Disk space:** You may need as much as 50MB of disk space while installing. Apache will probably use 10MB after installation, although the amount varies depending on the options used and modules installed.
- **C compiler:** Your computer has an ANSI-compliant C compiler installed. GNU C (`gcc`) is a good choice.

Installing

To install Apache from source files, follow these steps:

1. **Point your Web browser to `httpd.apache.org`, the Apache server home page.**
2. **On the left side of the page, under Downloads, click the `From a Mirror` link.**
3. **Scroll down to the Mirror section.**

   A specific mirror is selected for you. If you don't want to use this mirror, select another. Or if you have problems downloading from this mirror, return to this page and select another.
4. **Scroll farther down the same page to the section for Apache 2, Apache 1.3, or Apache 2.2, whichever you want to install. Locate and highlight the file you want to download.**

   For instance, at this time, the most recent version of Apache 2.0 for Linux is `apache-2.0.58.tar.gz`.
5. **Click the latest version to download it.**
6. **Select the option to save the file.**

7. **Navigate to where you want to save the source code (for example, `/usr/src`), and click Save.**

8. **After the download, change to the download directory (for example, `cd-/usr/src`).**

   You see a file named `apache-`, followed by the version name and `tar.gz`. This file is called a *tarball* because it contains many files compressed by a program called `tar`.



   Be sure you're using an account that has permission to write into `/usr/src`, such as `root`.

9. **Verify the downloaded file to be sure it hasn't been tampered with.**

   See the "Verifying a downloaded file" section in Appendix A. Click MD5 to see the MD5 signature referred to in the section in Appendix A.

10. **Unpack the tarball.**

    The command to unpack the tarball for version 2.0.58 is the following:

```
gunzip -c apache-2.0.58.tar.gz | tar -xf –
```

    A new directory called `apache-2.0.58` is created with several subdirectories containing all the files that you unpacked from the tarball.

11. **Change to the new directory that was created when you unpacked the tarball.**

    For example, you can use a command like the following:

```
cd apache-2.0.58
```

12. **Type the `configure` command.**

    The `configure` command consists of `./configure` followed by all the necessary options. If you can use all the default options, you can use `configure` without any options. However, to use Apache with PHP as a module, use the `configure` command as follows:

```
./configure --enable-module=so
```

    One of the more important installation options you may want to use is `prefix`, which sets a different location where you want Apache to be installed. By default, Apache is installed at `/usr/local/apache` or `usr/local/apache2`. You can change the installation location with the following line:

```
./configure --prefix=/software/apache
```

    You can see a list of all available options by typing the following line:

```
./configure --help
```

    This script may take a while to finish running. As it runs, it displays output. When the script is finished, the system prompt is displayed. If `configure` encounters a problem, it displays a descriptive error message.

13. **Type the following command:** make**.**

    This command builds the Apache server. It may take several minutes to finish running. As it runs, it displays messages telling you what it's doing. There may be occasional, longer pauses as it completes some action. When it's finished, it returns to the system prompt. If it has a problem, it displays a descriptive error message.

14. **Type the following command:** make install**.**

    This command installs the Apache software in the proper locations, based on the `configure` command you used in Step 12.

15. **Start the Apache Web server.**

    See the following section, "Starting and stopping Apache," for details.

16. **Type the URL for your Web site (for example, `www.mysite.com` or `localhost`) into a browser to test Apache.**

    If all goes well, you see a Web page telling you that Apache is working.

Starting and stopping Apache

A script named `apachectl` is available to control the server. By default, the script is stored in a subdirectory called `bin` in the directory where Apache is installed. Some Linux distributions may put it in another directory.

The script requires a keyword. The most common keywords are `start`, `stop`, and `restart`. The general syntax is as follows:

```
path/apachectl keyword
```

For example, if Apache was installed in the default directory, type the following line to start Apache:

```
/usr/local/apache/bin/apachectl start
```

Starting Apache

The `apachectl` script starts the Apache server, which then runs in the background, listening for HTTP requests. By default, the compiled Apache server is named `httpd` and is stored in the same directory as the `apachectl` script, unless you changed the name or location during installation. The `apachectl` script serves as an interface to the compiled server, called `httpd`.

You can run the `httpd` server directly, but it's better to use `apachectl` as an interface. The `apachect1` script manages and checks data that `httpd` commands require. Use the `apachectl` script to start Apache with the following command:

```
/usr/local/apache/bin/apachectl start
```

The `apachectl` script contains a line that runs `httpd`. By default, `apachectl` looks for `httpd` in the default location — `/usr/local/apache/bin` or `/usr/local/apache2/bin`. If you installed Apache in a nonstandard location, you may need to edit `apachectl` to use the correct path. Open `apachectl` and then search for the following line:

```
HTTPD='/usr/local/apache2/bin/httpd'
```

Change the path to the location where you installed `httpd`. For example, the new line might be this:

```
HTTPD='/usr/mystuff/bin/httpd'
```

After you start Apache, you can check whether Apache is running by looking at the processes on your computer. Type the following command to display a list of the processes that are running:

```
ps -A
```

If Apache is running, the list of processes includes some `httpd` processes.

Restarting Apache

Whenever you change the configuration file, the new directives take effect the next time Apache starts. If Apache is shut down when you make the changes, you can start Apache as described earlier in "Starting Apache." However, if Apache is running, you can't use `start` to restart it. Using `start` results in an error message saying that Apache is already running. You can use the following command to restart Apache when it's currently running:

```
/usr/local/apache2/bin/apachectl restart
```

Although the `restart` command usually works, sometimes it doesn't. If you restart Apache and the new settings don't seem to be in effect, try stopping Apache and starting it again. Sometimes this solves the problem.

Stopping Apache

To stop Apache, use the following command:

```
/usr/local/apache/bin/apachectl stop
```

You can check to see whether Apache is stopped by checking the processes running on your computer using the following command:

```
ps -A
```

The output from `ps` should not include any `httpd` processes.

Getting information from Apache

You can use options with the `httpd` server to obtain information about Apache. For instance, you can find out what version of Apache is installed by changing to the directory with `httpd` and typing one of the following:

```
httpd -v
./httpd -v
```

You can find out what modules are installed with Apache by typing

```
httpd -l
```

To see all the options that are available, type

```
httpd -h
```

Installing Apache on Windows

You can install Apache on almost any version of Windows, although Windows NT, 2000, and XP are preferred.

You cannot install Apache with the following directions if IIS (Internet Information Services) is already running on port 80. If IIS is running, you will find the IIS console at Start➪Control Panel➪Administrative Tools➪Internet Services Manager. If you do not find this menu item, IIS is not installed. If IIS is already running, you must shut it down before installing Apache or install Apache on a different port.

Installing

To install Apache, follow these steps:

1. **Point your Web browser to `httpd.apache.org`, the Apache server home page.**
2. **On the left side of the page, under Download, click the From a Mirror link.**
3. **Scroll down to the Mirror section.**

   A specific mirror is selected for you. If you don't want to use this mirror, select another. Or if you have problems downloading from this mirror, return to this page and select another.

4. **Scroll further down the same page to the section for Apache 2, Apache 1.3, or Apache 2.2, whichever you want to install. Locate and highlight the line for Win 32 Binary (MSI installer).**

   For instance, at this time, the most recent version of Apache 2.0 for Windows is `apache_2.0.58`.

5. **Click the filename to download it.**

6. **Select the option to save the file.**

7. **Navigate to where you want to save the installer. (This should be a temporary directory, such as a download directory.) Then click Save.**

   After the download is complete, you see a file in the download location containing all the files needed. The file is named `apache_`, followed by the version number and `win32-x86-no_ssl.msi`. For the current version, the file is named `apache_2.0.58-win32-x86-no_ssl.msi`.

8. **Verify the downloaded file to be sure it hasn't been tampered with.**

   See the "Verifying a Downloaded File" section in Appendix A. Click MD5 to see the MD5 signature referred to in the section in Appendix A.

9. **Double-click the downloaded file.**

   The Apache installation wizard begins, and a welcome screen appears.

10. **Click Next.**

    The license agreement is displayed.

11. **Select I Accept the Terms in the License Agreement, and then click Next.**

    If you don't accept the terms, you can't install the software. A screen of information about Apache is displayed.

12. **Click Next.**

    A screen is displayed asking for information.

13. **Enter the requested information, and then click Next.**

    The information requested is

    - **Domain Name:** Type your domain name, such as MyFineCompany.com. If you're installing Apache for testing and plan to access it only from the machine where it's installed, you can enter **localhost**.
    - **Server Name:** Type the name of the server where you're installing Apache, such as www.MyFineCompany.com or s1.mycompany.com. If you're installing Apache for testing and plan to access it only from the machine where it's installed, you can enter **localhost**.
    - **E-mail Address:** Type the e-mail address where you want to receive e-mail messages about the Web server, such as WebServer@MyFineCompany.com.

- **Run Mode:** Select whether you want Apache to run as a service (starting automatically when the computer boots up) or whether you want to start Apache manually when you want to use it. In most cases, you want to run Apache as a service.

The Installation Type screen is displayed.

14. **Select an installation type, and then click Next.**

    In most cases, you should select Complete. Only advanced users who understand Apache well should select Custom. A screen showing where Apache will be installed is displayed.

15. **Select the directory where you want Apache installed, and click Next.**

    You see the default installation directory for Apache, usually `C:\Program Files\Apache Group`. If this is okay, click Next. If you want Apache installed in a different directory, click Change and select a different directory, click OK, and click Next. A screen is displayed that says the wizard is ready to install Apache.

16. **Click Install.**

    If you need to, you can go back and change any of the information you entered before proceeding with the installation. A screen displays the progress. When the installation is complete, a screen appears saying that the wizard has successfully completed the installation.

17. **Click Finish to exit the installation wizard.**

Apache is installed on your computer based on your operating system. If you install it on Windows NT/2000/XP, it is automatically installed as a service that automatically starts when your computer starts. If you install it on Windows 95/98/Me, you need to start it manually or set it up so that it starts automatically when your computer boots. See the next section, "Starting and stopping Apache," for more information.

Starting and stopping Apache

When you install Apache on Windows NT, 2000, or XP, it's automatically installed as a service and started. It's ready to use. You can test it by typing your Web site name (or **localhost**) into your browser window. You see a welcome Web page that reads, "If you can see this, it means that the installation of the Apache Web server software on this system was successful." On Windows 95, 98, and Me, you have to start Apache manually, using the menu.

Apache installs menu items for stopping and starting Apache during installation. To find this menu, choose Start➪Programs➪Apache HTTP Server➪ Control Apache Server. The menu has the following items:

- **Start:** Used to start Apache when it is not running. If you click this item when Apache is running, you see an error message saying that Apache has already been started.
- **Stop:** Used to stop Apache when it is running. If you click this item when Apache is not running, you see an error message saying that Apache is not running.
- **Restart:** Used to restart Apache when it is running. If you make changes to Apache's configuration, you need to restart Apache before the changes become effective.

Getting information from Apache

Sometimes you want to know information about your Apache installation, such as the installed version. You can get this information from Apache by opening a command prompt window (Start➪Programs➪Accessories➪ Command Prompt), changing to the directory where Apache is installed (such as, `cd C:\Apache`), and accessing Apache with options. For example, to find out which version of Apache is installed, type the following in the command prompt window:

```
Apache -v
```

To find out what modules are compiled into Apache, type

```
Apache -l
```

You can also start and stop Apache directly, as follows:

```
Apache -k start
Apache -k stop
```

You can see all the options available by typing the following:

```
Apache -h
```

Installing Apache on Mac

Installing Apache on the Mac is similar to installing Apache on Unix and Linux. You download the source code and compile it. To install Apache on the Mac, follow these steps:

1. **Download the source code, save it in a directory, and change to the directory where the downloaded file is saved.**

   Follow Steps 1–8 of the directions for Unix and Linux. You will see a file named `httpd-`, followed by the version name and `tar.gz`, such as `httpd-2.0.58.tar.gz`. This file is the *tarball* — a single file that contains all the files needed, compressed into one file.

2. **Verify the downloaded file to be sure it hasn't been tampered with.**

   See the "Verifying a Downloaded File" section in Appendix A. Click MD5 to see the MD5 signature referred to in the section in Appendix A.

3. **Unpack the tarball by using a command similar to the following:**

   ```
   gnutar -xzf /httpd_2.0.58.tar.gz
   ```

   After unpacking the tarball, you see a directory called `httpd_2.0.58`. This directory contains several subdirectories and many files.

4. **Use a `cd` command to change to the new directory created when you unpacked the tarball (for example, `cd httpd_2.0.58`).**

5. **Type the following command:**

   ```
   ./configure --enable-module=most --enable-shared=max
   ```

   This command may take some time to run.

6. **Type the following command to build the Apache server:** make**.**

   This command may take a few minutes to run. It displays messages while it is running, with occasional pauses for a process to finish running.

7. **Type the following command to install Apache:** sudo make install**.**

8. **Start the Apache Web server.**

   See the "Starting and stopping Apache" section in the "Installing Apache on Linux and Unix" section for details.

9. **Type the URL for your Web site (for example, `www.mysite.com` or `localhost`) into a browser to test Apache.**

   If all goes well, you see a Web page telling you that Apache is working.

Configuring Apache

When Apache starts, it reads information from a configuration file. If Apache can't read the configuration file, it can't start. Unless you tell Apache to use a different configuration file, it looks for the file `conf/httpd.conf` in the directory where Apache is installed.

Changing settings

Apache behaves according to commands, called *directives,* in the configuration file. You can change some of Apache's behavior by editing the configuration file and restarting Apache so that it reads the new directives.

The configuration file is a text file containing commands called *directives.* Apache behaves according to the directives in this file. In most cases, the default settings allow Apache to start and run on your system. However, you may need to change the settings in some cases, such as the following:

- **Installing PHP:** If you install PHP, you need to configure Apache to recognize PHP programs. How to change the Apache configuration for PHP is described in Appendix B.
- **Changing your Web space:** Apache looks for Web page files in a specific directory and its subdirectories, often called your Web space. You can change the location of your Web space.
- **Changing the port where Apache listens:** By default, Apache listens for file requests on port 80. You can configure Apache to listen on a different port.

To change any settings, edit the `httpd.conf` file. On Windows, you can access this file through the menu at Start➪Programs➪Apache HTTPD Server➪Configure Apache Server➪Edit the Apache httpd.conf File. When you click this menu item, the `httpd.conf` file opens in Notepad.

The `httpd.conf` file has comments (beginning with `#`) that describe the directives, but make sure you understand their functions before changing any. All directives are documented on the Apache Web site.

When adding or changing filenames and paths, use forward slashes, even when the directory is on Windows. Apache can figure it out. Also, path names don't need to be in quotes unless they include special characters. A colon (`:`)

is a special character; the underscore (_) and hyphen (-) are not. For instance, to indicate a Windows directory, you would use something like the following:

```
"c:/temp/mydir"
```



The settings don't go into effect until Apache is restarted. Sometimes using the `restart` command doesn't work to change the settings. If the new settings don't seem to be in effect, try stopping the server with `stop` and then starting it with `start`.

Changing the location of your Web space

By default, Apache looks for your Web page files in the subdirectory `htdocs` in the directory where Apache is installed. You can change this with the `DocumentRoot` directive. Look for the line that begins with `DocumentRoot`, such as the following:

```
DocumentRoot "C:/Program Files/Apache Group/Apache/htdocs"
```

Change the filename and path to the location where you want to store your Web page files. Don't include a forward slash (/) on the end of the directory path. For example, the following might be your new directive:

```
DocumentRoot /usr/mysrver/Apache2/webpages
```

Changing the port number

By default, Apache listens on port 80. You might want to change this, for instance, if you are setting up a second Apache server for testing. The port is set by using the `Listen` directive as follows:

```
Listen 80
```

With Apache 2.0 and 2.2, the `Listen` directive is required. If no `Listen` directive is included, Apache 2 won't start.

You can change the port number as follows:

```
Listen 8080
```

Remember to restart Apache after you change any directives.

Index

• Symbols and Numerics •

&& (ampersands, double), in joined comparisons, 141
< > (angle brackets)
  <?php ... ?> tag, 17
  stripping or cleaning from form data, 240
  <? ... ?> tag, 114
* (asterisk)
  arithmetic operator in PHP, 123
  in patterns, 137
*= (asterisk, equal sign), in increment statements, 150
@ (at sign), in PHP statements, 121, 157
\ (backslash)
  escaping form data using, 241–242, 250
  \n in PHP strings, 126, 146–148
  in patterns, 137
  in PHP strings, 125
  \t in PHP strings, 126
{} (braces)
  in echo statement output strings, 146
  in if statement sections, 164, 166
  mismatched, 371–372
  in patterns, 137
^ (caret), in patterns, 136
$ (dollar sign)
  in patterns, 136
  preceding variable names, 119, 368
. (dot)
  concatenating strings in PHP, 127
  in patterns, 136
  separating date format symbols, 129
.= (dot, equal sign), append operator in PHP, 127
“” (double quotes)
  in SQL, 67, 195
  around strings in PHP, 125–127, 146, 369
... (ellipses), in examples, 2
= (equal sign)
  in PHP statements, 119–120
  in WHERE clause, 85
== (equal signs, double), comparison operator in PHP, 134, 368
!= (exclamation point, equal sign), comparison operator in PHP, 134
>= (greater than or equal sign)
  comparison operator in PHP, 134
  in WHERE clause, 85
> (greater than sign)
  comparison operator in PHP, 134
  in WHERE clause, 85
- (hyphen)
  in patterns, 136
  separating date format symbols, 129
<> (less than, greater than sign), comparison operator in PHP, 134
<= (less than or equal sign)
  comparison operator in PHP, 134
  in WHERE clause, 86
< (less than sign)
  comparison operator in PHP, 134
  in WHERE clause, 86
- (minus sign)
  arithmetic operator in PHP, 123
  in strtotime function, 130
-= (minus, equal sign), in increment statements, 150
-- (minus signs, double), in increment statements, 150
() (parentheses)
  in arithmetic expressions in PHP, 124
  in function call, 151, 178
  in joined comparisons, 140–141
  mismatched, 371–372
  in patterns, 136
  in WHERE clause, 86–87

% (percent sign)
  arithmetic operator in PHP, 123
  in hostname, 95
  in sprintf function, 364
+= (plus, equal sign), in increment statements, 150
+ (plus sign)
  arithmetic operator in PHP, 123
  in patterns, 137
  in strtotime function, 130
++ (plus signs, double), in increment statements, 149–150
# (pound sign)
  preceding comments in PHP, 142
  in URLs, 257
? (question mark)
  in patterns, 136
  in URLs, 257, 260–261
; (semicolon), ending PHP statements, 116, 144, 367
‘ ’ (single quotes)
  in SQL, 195
  around strings in PHP, 125–127, 146, 369
/ (slash)
  arithmetic operator in PHP, 123
  separating date format symbols, 129
  /* ... */, surrounding comments in PHP, 141–142
/= (slash, equal sign), in increment statements, 150
// (slashes, double), preceding comments in PHP, 142
[] (square brackets)
  creating arrays using, 151–152
  in patterns, 136
| (vertical bar), in patterns, 137
|| (vertical bars, double), in joined comparisons, 141

• A •

accessibility, usability affected by, 41
accounts, MySQL
  creating, 100–101, 102–103
  deleting, 103
  hostname for, 94–95
  IT providing access to, 23
  listing existing accounts, 100
  name of, 94–95
  passwords for, 96–97, 101–102
  permissions for, 97–98, 102–104
  root account, 70, 96
  root@% account, 96
  security features for, 93–94
ADD keyword, ALTER TABLE query, 77
ago keyword, in strtotime function, 130
akst keyword, in strtotime function, 131
ALL permission, 98
ALTER keyword, ALTER TABLE query, 77
ALTER permission, 98
ALTER TABLE query, 76–77
am keyword, in strtotime function, 130
ampersands, double (&&), in joined comparisons, 141
and keyword, in comparisons, 139–141
AND keyword, WHERE clause, 86–87
angle brackets (< >)
  <?php ... ?> tag, 17
  stripping or cleaning from form data, 240
  <? ... ?> tag, 114
announcement lists, 20
Apache Web server
  configuring, 399–400, 402–403, 417–418
  information about, getting, 412, 415
  installing
    on Linux and Unix, 408–410
    on Mac, 416
    reasons for, 29
    on Windows, 412–414
  location of Web space, changing, 418
  port number, changing, 418
  starting and stopping
    on Linux and Unix, 410–411
    on Windows, 414–415
  using PHP with, 17
  version of, choosing, 407
appending to strings in PHP, 127
application, 10, 11. *See also* programs; Web database application
arithmetic operators, PHP, 123–124
array_reverse function, PHP, 362
arrays, PHP
  creating, 151–152
  definition of, 143
  deleting values from, 154

displaying as output, 152–153
duplicate values in, removing, 362
exploding (splitting), 363
fetching data into
functions for, built-in, 196
functions for, writing, 202–206
retrieving all rows of data, 198–202
retrieving one row of data, 196–198
first index value of, 152, 370
imploding (joining), 363
long arrays, 20
multidimensional, 160–163
passing to functions, 182–183
range in, 362
retrieving values from, 156–157
reversing order of, 362
searching for strings in, 362
sorting, 154–156
superglobal arrays, 20, 208–209
walking through (iterating, traversing), 158–160
words as keys of, 152
array_unique function, PHP, 362
arsort statement, PHP, 156
asort statement, PHP, 155–156
assignment statements, PHP
for date values, 130–131
for numeric values, 119–121, 148–149
for strings, 119–121, 125, 148–149
asterisk (*)
arithmetic operator in PHP, 123
in patterns, 137
asterisk, equal sign (*=), in increment statements, 150
at sign (@), in PHP statements, 121, 157
attributes in database, 47. *See also* columns (attributes) in tables
AUTO_INCREMENT definition, CREATE TABLE query, 75
AVG function, SELECT query, 83

• B •

backslash (\)
escaping form data using, 241–242, 250
\n in PHP strings, 126, 146–148
in patterns, 137
in PHP strings, 125
\t in PHP strings, 126
backups
of database
creating, 104–106
restoring data from, 107–110
provided by Web hosting company, 25
BETWEEN keyword, WHERE clause, 86
BIGINT data type, MySQL, 58
block of statements, PHP
definition of, 118
executing conditionally, 132–133, 143, 163–168, 170–173
functions
built-in functions, 185
calling, 150–151
creating, 178–179
definition of, 178
in include files, 282
naming, 284
passing values to, 181–184
reasons to use, 280, 284
for retrieving data from database, 202–206
returning values from, 184–185
variables in, local and global, 180–181
loops
breaking out of, 176–177
definition of, 144, 168
do..while statement, 172–173
fetching data into arrays using, 198
for statement, 169–170
infinite loops, 172, 174–177
while statement, 170–172
bold text used in this book, 2
books. *See* publications
braces ({})
in echo statement output strings, 146
in if statement sections, 164, 166
mismatched, 371–372
in patterns, 137
break statement, PHP, 168, 176–177
browsers, differences between, usability affected by, 41
built-in functions, PHP, 185

• C •

caret (^), in patterns, 136
case of strings, changing, 366
case-sensitivity
  of PHP constants, 122
  of PHP statements, 118
  of PHP variables, 119
  of SQL statements, 66
catalog. *See* Pet Catalog example
CHANGE keyword, ALTER TABLE query, 77
CHAR data type, MySQL, 58, 239, 294, 331
character data types, MySQL, 56, 58
character strings, PHP
  appending to, 127
  assigning to variables, 119–121, 125, 148–149
  case of, changing, 366
  comparisons between, 134–139
  definition of, 125
  escaping characters in, 125
  formatting, 363–364
  joining (concatenating), 127
  length of, 365
  new line in, 126
  pattern matching using, 135–139, 365
  quotes around, types of, 125–127, 146, 369
  reversing order of characters in, 365
  searching for substrings in, 365
  substrings of, 365
  tabs in, 126
check boxes in HTML forms, 223–224
Color table, Pet Catalog example, 296–297
columns (attributes) in tables
  adding to existing table, 77
  changing default value of, 77
  changing definition of, 77
  default values for, 49
  definition of, 47
  deleting from existing table, 77, 92
  primary key as, 48
  renaming, 77
  requiring data for, 49
comma-delimited files, 80
comments, in PHP, 141–142, 280, 288
commercial license, for MySQL, 13
company Web site, 22–24
comparisons, PHP
  definition of, 133
  incorrect operator in, 368
  joining, 139–141
  matching strings to patterns, 135–139
  simple (between two values), 133–135
complex statements, PHP, 118
concatenating (joining) strings, PHP, 127
conditional block, PHP, 118
conditional statements, PHP
  comparisons for
    definition of, 133
    incorrect operator in, 368
    joining, 139–141
    matching strings to patterns, 135–139
    simple (between two values), 133–135
  definition of, 132–133, 143, 163
  do..while statement, 172–173
  if statement, 164–167
  switch statement, PHP, 167–168
  while statement, 170–172
configuration file, PHP. *See* php.ini file
connection verification, 94
constants, PHP, 122, 280
continue statement, PHP, 176–177
conventions used in this book, 2, 5
cookies
  disabling, 270
  sessions without, 271–273
  sharing information between pages using, 260, 264–266
$_COOKIES built-in array, 208, 266
COUNT function, SELECT query, 83
cracking passwords, 96
CREATE DATABASE query, 73–74
CREATE permission, 98
CREATE TABLE query, 74–76
CREATE USER query, 101
curly braces ({})
  in echo statement output strings, 146
  in if statement sections, 164, 166
  mismatched, 371–372
  in patterns, 137
currency, formatting in PHP, 124–125
current function, PHP, 158
$cxn variable, 191

• D •

data types, MySQL
  character, 56, 58
  date and time, 57–58, 131–132
  enumeration, 57–58
  list of, 58–59
  numbers, 56–57, 58
data types, PHP. *See also* arrays, PHP
  formatting to prepare for database, 238–239
  numbers
    arithmetic operators for, 123–125
    comparisons between, 134
    formatting, 124–125, 363–364
  strings
    appending to, 127
    assigning to variables, 119–121, 125, 148–149
    case of, changing, 366
    comparisons between, 134–139
    definition of, 125
    escaping characters in, 125
    formatting, 363–364
    joining (concatenating), 127
    length of, 365
    newline in, 126
    pattern matching using, 135–139, 365
    quotes around, types of, 125–127, 146, 369
    reversing order of characters in, 365
    searching for substrings in, 365
    substrings of, 365
    tabs in, 126
  timestamp format, 128–131
  for variables, 119–120
database. *See also* MySQL; Web database application
  adding data to, 78–81, 297–298, 303–305, 333
  backups of
    creating, 104–106
    restoring, 107–110
  building (creating), 62, 73–77, 291–297, 328–332
  connecting to, 189–194
  connecting to, failure of, 36, 192–193
  data file for, reading in, 297–298
  data types for, 56–59
  definition of, 10, 11
  deleting, 74, 92
  deleting data from, 92
  designing, 44–50, 288
  disconnecting from, 191
  displaying existing databases, 74
  querying, 194–195
  retrieving data from
    functions for, 202–206
    process for, 195
    retrieving all rows of data, 198–202
    retrieving one row of data, 196–198
    `SELECT` query for, 82–87, 194, 196
  security features for, 93–94
  selecting in program, after connecting, 191–194
  storing form data in
    inserting new data, 242–247
    preparing data for, 238–242
    updating existing data, 247–250
  tables in
    changing structure of, 76–77
    corrupted, 107
    creating, 74–76
    deleting, 76, 92
    displaying existing tables, 76
    displaying structure of, 76
    organizing data in, 46–49
    relationships between, 49–50
    renaming, 77
    retrieving data from, 82–87
  updating data in, 92
database application. *See* Web database application
Database Management System (DBMS), 11
date and time data types, MySQL
  definition of, 57
  list of, 58
  storing values of PHP variables in, 131–132
date and time values in PHP
  assigning to variables, 130–131
  formatting, 129–130
  separator characters for, 129
  storing in database, 131–132

date and time values in PHP *(continued)*
time zone, default, 128
timestamp format for, 128–131
DATE data type, MySQL, 58, 131, 239
date function, PHP, 129–130, 132
date_default_timezone_get function, PHP, 128
date_default_timezone_set function, PHP, 128
DATEDIFF function, MySQL, 132
DATETIME data type, MySQL, 58, 131
date.timezone configuration setting, PHP, 128
day keyword, in strtotime function, 130
day names, in strtotime function, 130
DAYNAME function, SELECT query, 84
DBMS (Database Management System), 11
DECIMAL data type, 58, 239
DEFAULT definition, CREATE TABLE query, 75
define statement, PHP, 122
DELETE permission, 98
DELETE query, 92
DESCRIBE query, 76
die statement, PHP, 150, 192, 362
discussion lists, 12, 13
DISTINCT keyword, SELECT query, 85
document root, Apache, 33
documentation for Web database application, 288
dollar sign ($)
in patterns, 136
preceding variable names, 119, 368
domain names, 25, 27
dot (.)
concatenating strings in PHP, 127
in patterns, 136
separating date format symbols, 129
dot, equal sign (.=), append operator in PHP, 127
double quotes (“”)
in SQL, 67, 195
around strings in PHP, 125–127, 146, 369
do..while statement, PHP, 172–173
DROP DATABASE query, 74
DROP keyword, ALTER TABLE query, 77
DROP permission, 98
DROP TABLE query, 76
DROP USER query, 103
“dynamic duo” of MySQL and PHP, 18
dynamic Web site, 9–10. *See also* Web database application

• E •

echo statement, PHP, 116, 145–148, 152–153
ellipses (...), in examples, 2
else section, if statement, 164
elseif section, if statement, 164
e-mail discussion lists, 12, 13
e-mail, sending, 360–361
empty form fields, checking for, 227–231
empty function, PHP, 363
encryption, 287
end function, PHP, 158
entities (rows) in tables, 47
ENUM data type, 58, 239
enumeration data types, MySQL, 57–58
equal sign (=)
in PHP statements, 119–120
in WHERE clause, 85
equal signs, double (==), comparison operator in PHP, 134, 368
ereg function, PHP, 138–139, 232–233, 365
ereg_replace function, PHP, 239–240, 365
Error messages, 117
error_reporting configuration setting, PHP, 117
errors. *See* troubleshooting
examples used in this book. *See* Members Only example; Pet Catalog example
exclamation point, equal sign (!=), comparison operator in PHP, 134
exit statement, PHP, 150, 227, 362
explode function, PHP, 363
external files. *See* include files
extract statement, PHP, 157

• F •

Feedback page, Pet Catalog example, 304, 322–326
FILE permission, 98
files
include files
definition of, 281
including in program, 282–283

location of, 283–284
naming, 286
PHP statements in, 371
security of, 283, 286
uses of, 281–282
loading into database, 80–81
uploading using HTML forms, 250–254
$_FILES built-in array, PHP, 251–253
fonts used in this book, 2
for statement, PHP, 169–170
foreach statement, PHP
nesting for multidimensional arrays, 162
walking through array using, 159–160
format of form fields, checking, 232–236
forms. *See* HTML forms
fortnight keyword, in strtotime function, 130
functions in SELECT query, 83–84
functions, PHP
built-in functions, 185
calling, 150–151
creating, 178–179
definition of, 178
in include files, 282
naming, 284
passing values to, 181–184
reasons to use, 280, 284
for retrieving data from database, 202–206
returning values from, 184–185
variables in, local and global, 180–181

• G •

General Public License (GNU GPL), for MySQL, 12–13
$_GET built-in array, 208–209, 225
get method, 208, 228
Get missing information page, Pet Catalog example, 304–305
Get pet information page, Pet Catalog example, 304, 316–322
Get pet type page, Pet Catalog example, 303, 313–316
global variables, in PHP functions, 180
gmt keyword, in strtotime function, 131
GNU GPL (General Public License), for MySQL, 12–13
GRANT permission, 98
GRANT query, 97–98
grants (permissions) for MySQL accounts
changing, 102
definition of, 97
list of, 98
listing current permissions for account, 102
removing, 103–104
stored in mysql database, 99
graphics, usability affected by, 41
greater than or equal sign (>=)
comparison operator in PHP, 134
in WHERE clause, 85
greater than sign (>)
comparison operator in PHP, 134
in WHERE clause, 85
GROUP BY clause, SELECT query, 84

• H •

header statement, PHP, 256–259, 369–370
hidden fields in HTML forms, 267
hostname, MySQL account, 94–95
hour keyword, in strtotime function, 130
HTML 4 For Dummies, 4th Edition (Tittel and Pitts), 3
HTML 4 For Dummies Quick Reference (Ray and Ray), 3, 256
HTML forms
cleaning data from, 240–241, 287
displaying all field contents of, 209–211
displaying with PHP, 207–208
dynamic check box lists in, 223–224
dynamic information in fields of, 212–215
dynamic radio button lists in, 221–222
dynamic selection lists in, 215–221
escaping data from, 241–242
hidden fields in, 267
retrieving data from, 208–211, 224–226
sharing information between pages using, 260, 267
storing data in database
inserting new data, 242–247
preparing data for, 238–242
updating existing data, 247–250
submitting information from
methods for, 208, 224–225, 228
multiple buttons for, 236–238
uploading files using, 250–254

HTML forms *(continued)*
uses of, 206–207
validating data
checking for empty fields, 227–231
checking format of fields, 232–236
validating data retrieved from, 287
HTML (HyperText Markup Language). *See also* Web sites
capabilities of, 113
generating as output with `echo` statement, 145–148
learning, 3
PHP embedded in, 16–17, 113–116
removing tags from form data, 240
`htmlspecialchars` function, PHP, 240
`HTTP`, in URLs, 257
`$HTTP_COOKIE_VARS` built-in array, 266
`httpd.conf` file, 417
`$HTTP_GET_VARS` built-in array, 209
`$HTTP_POST_VARS` built-in array, 20, 209
`HTTPS`, in URLs, 287
hyperlinks, navigation between Web pages using, 256
HyperText Markup Language. *See* HTML
hyphen (-)
in patterns, 136
separating date format symbols, 129

• I •

icons used in this book, 5
`if` statement, PHP, 133, 164–167, 192–193
IIS (Internet Information Server)
configuring for PHP, 403
definition of, 30
`implode` function, PHP, 363
`IN` keyword, `WHERE` clause, 86
`in_array` function, PHP, 362
`include` files
definition of, 281
including in program, 282–283
location of, 283–284
naming, 286
PHP statements in, 371
security of, 283, 286
uses of, 281–282
`include` statement, PHP, 281–284
`include_once` statement, PHP, 282
`include_path` configuration setting, PHP, 283
increment statements
`for` statement, PHP, 169–170
operators for, 149–150
infinite loops, 172, 174–177
`ini_set` statement, PHP, 283
inner joins, in `SELECT` query, 89
`INSERT` permission, 98
`INSERT` query, 79–80, 242–247
`INT` data type, MySQL, 58, 239
`INT UNSIGNED` data type, MySQL, 58
interactive Web site, 9. *See also* Web database application
Internet Information Server (IIS)
configuring for PHP, 403
definition of, 30
IP addresses
definition of, 27
hostname given as, 95
`isset` function, PHP, 363
italic text used in this book, 2

• J •

JavaScript, 10
joining comparisons, 139–141
joining (concatenating) strings, PHP, 127
joins, in `SELECT` query, 87, 89–91

• K •

`krsort` statement, PHP, 156
`ksort` statement, PHP, 156

• L •

`last` keyword, in `strtotime` function, 130
`LEFT JOIN` keyword, `SELECT` query, 89–90
Lerdorf, Rasmus (developer of PHP), 15
less than, greater than sign (<>), comparison operator in PHP, 134
less than or equal sign (<=)
comparison operator in PHP, 134
in `WHERE` clause, 86
less than sign (<)
comparison operator in PHP, 134
in `WHERE` clause, 86

LIKE keyword, WHERE clause, 86
LIMIT keyword, SELECT query, 85, 87
links, navigation between Web pages using, 256
Linux
  installing Apache on, 408–410
  installing MySQL on, 381–385
  installing PHP on, 391–394
  for local Web site, 28
  starting and stopping Apache on, 410–411
list manager, 12
list statement, PHP, 157
lists
  announcement lists, 20
  e-mail discussion lists, 12–13
LOAD query, 79, 80–81, 298
local time zone configuration setting, PHP, 20
local variables, in PHP functions, 180
Login page, Members Only example
  file containing HTML for, 348–352
  look and feel for, designing, 333–336
  program for, 337–338, 340–348
Login table, Members Only example, 332
logins for Web sites
  definition of, 328
  file containing HTML for, 348–352
  look and feel for, designing, 333–336
  programs for, 337–338, 340–348
  reasons to use, 327
long arrays, PHP, 20, 209, 214
loops, PHP
  breaking out of, 176–177
  definition of, 144, 168
  do..while statement, 172–173
  fetching data into arrays using, 198
  for statement, 169–170
  infinite loops, 172, 174–176, 177
  while statement, 170–172

• M •

Mac computers
  installing Apache on, 416
  installing MySQL on, 386–387
  installing PHP on, 394–397
  for local Web site, 29
  starting and stopping MySQL on, 387
magic quotes configuration setting, PHP, 20, 241–242, 250
mail function, PHP, 360–361
MAX function, SELECT query, 83
MD5 method of verification, 388
Member table, Members Only example, 329–331
Members Only example
  application for, designing, 328
  database for
    adding data to, 333
    building, 328–332
    designing, 45, 53–55, 60–61
  description of, 43–44
  enhancements for, planning, 355
  login page
    file containing HTML for, 348–352
    look and feel for, designing, 333–336
    program for, 337–338, 340–348
  look and feel for, designing, 333–337
  programs for
    login, 337–338, 340–348
    Members Only section, 354–355
    New Member Welcome page, 352–354
    storefront, 338–339
Members Only section, Members Only example, 336, 354–355
MIN function, SELECT query, 83
minus, equal sign (-=), in increment statements, 150
minus sign (-)
  arithmetic operator in PHP, 123
  in strtotime function, 130
minus signs, double (–), in increment statements, 150
minute keyword, in strtotime function, 130
MODIFY keyword, ALTER TABLE query, 77
monitor program. *See* mysql program
month keyword, in strtotime function, 130
month names, in strtotime function, 130
move_uploaded_file statement, PHP, 251–252
multidimensional arrays, PHP, 160–163
my.cnf file, 389
my.ini file, 389

MySQL
  advantages of, 13–14
  configuring, 389
  configuring on Windows, 378–379
  data types, 57–59
  definition of, 12, 14
  determining if running or installed, 30
  e-mail discussion lists for, 12–13
  installing
    checking for existence of, 30–31
    on Linux and Unix, 381–385
    on Mac, 386–387
    on Windows, 375–377
  licensing for, 12–13
  location of databases for, 23
  operating systems supported by, 14
  requirements for
    on company Web site, 23
    from Web hosting company, 24, 26
  security of, 14, 93–94
  starting
    on all systems, 31
    on Linux and Unix, 385
    on Mac, 387
    on Windows, 380
  stopping
    on Mac, 387
    on Windows, 380–381
  technical support for, 14
  testing, 32–33, 35–36
  updates for, 19–20
  upgrading, 110
  used with PHP, 18–19, 32–33
  verification of, after downloading, 388
  versions of, 188
MySQL accounts
  creating, 100–103
  deleting, 103
  hostname for, 94–95
  IT providing access to, 23
  listing existing accounts, 100
  name of, 94–95
  passwords for, 96–97, 101, 102
  permissions for, 97–98, 102–104
  root account, 70, 96
  root@% account, 96
  security features for, 93–94
MySQL Community Edition, 13
mysql database, 98, 99. *See also* database
mysql functions, PHP, 32–33, 188–189
MySQL Network, 13
mysql program
  sending queries in a file using, 109
  sending queries on command line using, 72–73
MySQL server
  connecting to, 190–191
  definition of, 14
  sending messages to, 15
mysql_connect function, PHP, 189
mysqldump program, 105–106
mysql_errno function, PHP, 189
mysql_error function, PHP, 189
mysql_fetch_array function, PHP, 189
mysql_fetch_assoc function, PHP, 189
mysql_fetch_row function, PHP, 189
mysql_fix_privileges_tables script, 110
mysqli functions, PHP, 32–33, 188–189, 359–360
mysqli_affected_rows function, PHP, 360
mysqli_close function, PHP, 191
mysqli_connect function, PHP, 189, 190–191, 359
mysqli_errno function, PHP, 189
mysqli_error function, PHP, 189
mysqli_fetch_array function, PHP, 189, 196–198
mysqli_fetch_assoc function, PHP, 189, 196–198, 359
mysqli_fetch_row function, PHP, 189, 196–198, 360
mysqli_field_name function, PHP, 360
mysqli_insert_id function, PHP, 189, 359
mysql_insert_id function, PHP, 189
mysqli_num_fields function, PHP, 360
mysqli_num_rows function, PHP, 189, 359
mysqli_query function, PHP, 189, 194–195, 196, 359
mysqli_real_escape_string function, PHP, 189, 241
mysqli_select_db function, PHP, 189, 191–194, 359
mysql_num_rows function, PHP, 189

mysql_query function, PHP, 189
mysql_real_escape_string function, PHP, 189, 241
mysql_select_db function, PHP, 189
mysql_upgrade script, 110

• N •

\n characters in PHP strings, 126, 146–148
navigation
  usability affected by, 41
  between Web pages, 256–259
new line character (\n) in PHP strings, 126, 146–148
New Member Welcome page, Members Only example, 333, 336, 352–354
next function, PHP, 158
next keyword, in strtotime function, 130
NOT IN keyword, WHERE clause, 86
NOT LIKE keyword, WHERE clause, 86
NOT NULL definition, CREATE TABLE query, 75
Notices
  for array errors, 156–157
  definition of, 117, 121
  preventing for current statement, 121, 157
  suppressing, 121
NOW function, MySQL, 132
now keyword, in strtotime function, 130
number_format function, PHP, 124–125
numbers, MySQL, 56–58
numbers, PHP
  arithmetic operators for, 123–125
  comparisons between, 134
  formatting, 124–125, 363–364

• O •

objects in database, 46–47. *See also* tables in database
open source software
  MySQL as, 12–13, 19
  PHP as, 19
operating systems
  Linux
    installing Apache on, 408–410
    installing MySQL on, 381–385
    installing PHP on, 391–394
    for local Web site, 28
    starting and stopping Apache on, 410–411
  for local Web site, 28–29
  Mac
    installing Apache on, 416
    installing MySQL on, 386–387
    installing PHP on, 394–397
    for local Web site, 29
    starting and stopping MySQL on, 387
  Unix
    installing Apache on, 408–410
    installing MySQL on, 381–385
    installing PHP on, 391–394
    for local Web site, 28
    starting and stopping Apache on, 410–411
  Windows
    configuring MySQL on, 378–379
    installing Apache on, 412–414
    installing MySQL on, 375–377
    installing PHP on, 400–403
    for local Web site, 28
    starting and stopping Apache on, 414–415
    starting MySQL on, 380
    stopping MySQL on, 380–381
Or keyword, in comparisons, 139–141
OR keyword, WHERE clause, 86–87
ORDER BY clause, SELECT query, 84
outer joins, in SELECT query, 89–91

• P •

parentheses (())
  in arithmetic expressions in PHP, 124
  in function call, 151, 178
  in joined comparisons, 140–141
  mismatched, 371–372
  in patterns, 136
  in WHERE clause, 86–87
Parse errors, 116, 367
passwords for MySQL accounts
  changing, 102
  choosing, 96–97
  definition of, 96
  removing, 101
  setting, 101, 102
pattern matching in PHP, 135–139, 232–233

pdt keyword, in strtotime function, 131
percent sign (%)
  arithmetic operator in PHP, 123
  in hostname, 95
  in sprintf function, 364
permissions for MySQL accounts
  changing, 102
  definition of, 97
  list of, 98
  listing current permissions for account, 102
  removing, 103–104
  stored in mysql database, 99
Pet Catalog example
  application for, designing, 289–291
  database for
    adding data to, 297–298, 303–305
    building, 291–297
    designing, 45, 51–53, 59–60
  description of, 42–43
  look and feel for, designing, 299–305
  program to add pets to catalog
    list of tasks for, 312–313
    selecting pet name, 316–322
    selecting pet type, 313–316
    storing new pet, 322–326
  program to show pets to customers, 306–312
  retrieving data from database, 198–202, 203–206
Pet table, Pet Catalog example, 292–295
Pet type page, Pet Catalog example, 299–300, 307–309
Pets page, Pet Catalog example, 301–302, 310–312
PetType table, Pet Catalog example, 295
PGP method of verification, 388
PHP. *See also* statements, PHP
  advantages of, 16
  case-sensitivity of, 118
  comments in, 141–142, 280
  comparisons, 133–141, 368
  configuring, 398–399, 404–405
  configuring Apache for, 399–400
  constants, 122
  databases supported by, 15
  date and time values in, 128–132
  definition of, 10, 15–17
  editors for, 118
  e-mail discussion lists for, 12
  embedding in HTML, 113–116
  file extensions processed by, 23
  format of statements, 115–118, 280
  installing
    checking for existence of, 31–32
    on Linux and Unix, 391–394
    on Mac, 394–397
    options for, 398–399
    on Windows, 400–403
  numeric operations, 123–125
  operating systems supported by, 16
  processing of, by Web server, 114
  requirements for
    on company Web site, 23
    from Web hosting company, 24, 26
  security of, 16
  string operations, 125–127
  technical support for, 16
  testing, 32–34
  updates for, 19–20
  used with MySQL, 18–19, 32–33
  variables, 119–121
  versions of, changes in, 20
  Web servers supported by, 17
.php file, 20, 23, 114, 143
PHP interpreter, file name for, 20
PHP sessions
  closing, 274, 362
  without cookies, 271–273
  private, 273–274
  sharing information between pages using, 260, 267–274, 362
  variables in, 269–271
<?php ... ?> tag, 17, 114–116
.php-cgi file, 20
php.ini file
  changing settings in, 34
  configuring PHP with, 404–405
  date.timezone configuration setting, 128
  error_reporting configuration setting, 117
  include_path configuration setting, 283
  local time zone configuration setting, 20

magic quotes configuration setting, 20, 241–242, 250
trans-sid configuration setting, 271–273
$PHPSESSID system variable, 271–273
$PHPSESSION system variable, 268
.phtml file, 23, 143
Pitts, Natanya (*HTML 4 For Dummies,* 4th Edition), 3
plus, equal sign (+=), in increment statements, 150
plus sign (+)
arithmetic operator in PHP, 123
in patterns, 137
in strtotime function, 130
plus signs, double (++), in increment statements, 149–150
pm keyword, in strtotime function, 130
port number
for database connection, 190
in URLs, 257
$_POST built-in array, 208–209, 225–226
post method, 208, 228
pound sign (#)
preceding comments in PHP, 142
in URLs, 257
previous function, PHP, 158
primary key, 48
PRIMARY KEY keyword, CREATE TABLE query, 75
print_r statement, PHP, 153
problems. *See* troubleshooting
programs. *See also* Web database application
designing, 288
ending, 150, 227, 362
including external files in, 281–284
Members Only example
for login, 337–338, 340–348
for Members Only section, 354–355
for New Member Welcome page, 352–354
for storefront, 338–339
naming, 278
number of, 278
organizing, 279–284
Pet Catalog example
to add pets to catalog, 312–326
to show pets to customers, 306–312
subdirectories for, 279
writing (creating), 62
publications
HTML 4 For Dummies, 4th Edition (Tittel and Pitts), 3
HTML 4 For Dummies Quick Reference (Ray and Ray), 3, 256

• Q •

queries. *See* SQL (Structured Query Language)
question mark (?)
in patterns, 136
in URLs, 257, 260–261
quotes
magic quotes configuration setting, PHP, 20, 241–242, 250
around strings in PHP, 125–127, 146, 369
in SQL, 67, 195

• R •

radio buttons in HTML forms, 221–222
range function, PHP, 362
Ray, Deborah S. (*HTML 4 For Dummies Quick Reference*), 3, 256
Ray, Eric J. (*HTML 4 For Dummies Quick Reference*), 3, 256
RDBMS (Relational Database Management System), 11
Red Hat Package Manager (RPM), 382–383
regex (regular expressions), 135, 232–233
register_globals configuration setting, PHP, 20
Relational Database Management System (RDBMS), 11
RENAME keyword, ALTER TABLE query, 77
repeating blocks of statements. *See* loops, PHP
$_REQUEST built-in array, 208

request verification, 94
reset statement, PHP, 158
resources. *See* publications; Web site resources
return statement, PHP, 179, 184–185
REVOKE query, 97–98, 103–104
RIGHT JOIN keyword, SELECT query, 91
root account, MySQL, 70, 96
root@% account, MySQL, 96
rows (entities) in tables, 47
RPM (Red Hat Package Manager), 382–383
rsort statement, PHP, 156

• S •

second keyword, in strtotime function, 130
Secure Sockets Layer (SSL), 287
security. *See also* MySQL accounts
  backups of database
    creating, 104–106
    restoring, 107–110
  of computer, 285
  of database connection information, 191
  of database name, 193–194
  of filenames on server, 286
  of HTML form data, 287
  of include files, 283, 286
  of MySQL, 14, 93–94, 99
  of passing information in URLs, 261
  of PHP, 16
  of PHP Version 4.3.0 or earlier, 20
  of Web database application, 285–287
  of Web server, 287
SELECT permission, 98
SELECT query
  combining tables in, 87–91
  functions in, 83–84
  limiting rows retrieved by, 85, 87
  ordering data retrieved from, 84
  retrieving all data, 82
  retrieving specific columns, 82–84
  retrieving specific rows, 84–87
  sending to database, 196
selection lists in HTML forms, 215–221
semicolon (;), ending PHP statements, 116, 144, 367
SERIAL data type, MySQL, 58
$_SESSION built-in array, 268, 269
session function, PHP, 258
session variables, PHP, 269–271
session_destroy statement, PHP, 274, 362
sessions, PHP
  closing, 274, 362
  without cookies, 271–273
  private, 273–274
  sharing information between pages using, 260, 267–274, 362
  variables in, 269–271
session_start statement, PHP, 268–269, 362
SET PASSWORD query, 101
setcookie statement, PHP, 258, 265–266
short tags, 114
SHOW DATABASES query, 74
SHOW TABLES query, 76
SHUTDOWN permission, 98
SID constant, 271, 272
single quotes (' ')
  in SQL, 195
  around strings in PHP, 125–127, 146, 369
sizeof function, PHP, 170
slash (/)
  arithmetic operator in PHP, 123
  separating date format symbols, 129
  /* ... */, surrounding comments in PHP, 141–142
slash, equal sign (/=), in increment statements, 150
slashes, double (//), preceding comments in PHP, 142
sort statement, PHP, 154–156
sorting arrays, 154–156
sprintf function, PHP, 124, 363–364
SQL (Structured Query Language)
  ALTER TABLE query, 76–77
  case-sensitivity of, 66
  CREATE DATABASE query, 73–74
  CREATE TABLE query, 74–76
  CREATE USER query, 101
  definition of, 15, 66
  DELETE query, 92
  DESCRIBE query, 76
  DROP DATABASE query, 74
  DROP TABLE query, 76
  DROP USER query, 103

fetching data after querying
functions for, built-in, 196
functions for, writing, 202–206
retrieving all rows of data, 198–202
retrieving one row of data, 196–198
GRANT query, 97–98
INSERT query, 79–80, 242–247
LOAD query, 79–81, 298
quotes in, 67, 195
REVOKE query, 97–98, 103–104
SELECT query
combining tables in, 87–91
functions in, 83–84
limiting rows retrieved by, 85, 87
ordering data retrieved from, 84
retrieving all data, 82
retrieving specific columns, 82–84
retrieving specific rows, 84–87
sending to database, 196
sending queries to MySQL, 67–73, 194–195
SET PASSWORD query, 101
SHOW DATABASES query, 74
SHOW TABLES query, 76
spaces in, 67
UPDATE query, 92, 247–250
SQRT function, SELECT query, 84
square brackets ([])
creating arrays using, 151–152
in patterns, 136
SSL (Secure Sockets Layer), 287
statements, PHP. *See also* block of statements, PHP; *specific statements*
assignment statements, 119–121, 125, 130–131, 148–149
complex, 118
conditional statements, 132–133, 143, 163–168, 170–173
ending program, 150, 227, 362
increment statements, 149–150
simple statements, 144–151
static Web pages, 9, 10
Storefront page
Members Only example, 333–334, 338–339
Pet Catalog example, 299, 306–307
Strict messages, 117
strings, PHP
appending to, 127
assigning to variables, 119–121, 125, 148–149
case of, changing, 366
comparisons between, 134–139
definition of, 125
escaping characters in, 125
formatting, 363–364
joining (concatenating), 127
length of, 365
newline in, 126
pattern matching using, 135–139, 365
quotes around, types of, 125–127, 146, 369
reversing order of characters in, 365
searching for substrings in, 365
substrings of, 365
tabs in, 126
strip_tags function, PHP, 240
strlen function, PHP, 365
strpos function, PHP, 365
strrev function, PHP, 365
strtolower function, PHP, 366
strtotime function, PHP, 130–131
strtoupper function, PHP, 366
strtr function, PHP, 365
Structured Query Language. *See* SQL
submitting form information
methods for, 208, 224–225, 228
multiple buttons for, 236–238
navigating to another Web page after, 256
substr function, PHP, 365
SUM function, SELECT query, 83
superglobal arrays, PHP, 20, 208–209
switch statement, PHP, 167–168

• T •

tab character (\t) in PHP strings, 126
tab-delimited files, 80
tables in database
changing structure of, 76–77
corrupted, 107
creating, 74–76

tables in database *(continued)*
deleting, 76, 92
displaying existing tables, 76
displaying structure of, 76
organizing data in, 46–49
relationships between, 49–50
renaming, 77
retrieving data from, 82–87
technical support
by company Web site, 24
by Web hosting company, 25
`TEXT` data type, MySQL, 58
text files. *See* files
`this` keyword, in `strtotime` function, 130
time and date data types, MySQL
definition of, 57
list of, 58
storing values of PHP variables in, 131–132
time and date values in PHP
assigning to variables, 130–131
formatting, 129–130
separator characters for, 129
storing in database, 131–132
time zone, default, 128
timestamp format for, 128–131
`TIME` data type, MySQL, 58
time zone, PHP, 128
time zones, in `strtotime` function, 130
timestamp format, 128–131
Tittel, Ed (*HTML 4 For Dummies,* 4th Edition), 3
`today` keyword, in `strtotime` function, 130
`tomorrow` keyword, in `strtotime` function, 130
`trans-sid` configuration setting, PHP, 271, 272–273
troubleshooting
array numbering incorrect, 370
corrupted tables, 107
equality operator incorrect, 368
Error messages, 117
header output not first, 369–370
including PHP statements incorrectly, 371
infinite loops, 174–177
level of error messages to report, 117
MySQL connection failed, 36, 192–193
MySQL errors, handling, 192–193
Notices, 117
parentheses and brackets mismatched, 371–372
Parse errors, 116, 367
quotes used incorrectly, 369
semicolons missing, 367
Strict messages, 117
undefined `mysql` functions, 33
values passed to functions, incorrect number of, 183
variable names misspelled, 368
Warning messages, 117
typefaces used in this book, 2
types of data. *See* data types, MySQL

• U •

`ucfirst` function, PHP, 366
`ucwords` function, PHP, 366
Uniform Resource Locator (URL)
definition of, 257
HTTP in, 257
HTTPS in, 287
sharing information between pages using, 260–264
`UNION` keyword, `SELECT` query, 87–89
Unix
installing Apache on, 408–410
installing MySQL on, 381–385
installing PHP on, 391–394
for local Web site, 28
starting and stopping Apache on, 410–411
`unset` statement, PHP, 120, 151, 154, 274
`UNSIGNED` definition, `CREATE TABLE` query, 75
`UPDATE` permission, 98
`UPDATE` query, 92, 247–250
uploading files using HTML forms, 250–254
URL (Uniform Resource Locator)
definition of, 257
HTTP in, 257
HTTPS in, 287
sharing information between pages using, 260–264
usability engineering, 41

USAGE permission, 98
user logins. *See* logins for Web sites
users, getting information from. *See* HTML forms
users of MySQL. *See* MySQL accounts
usort statement, PHP, 156

• V •

validation of form data
  checking for empty fields, 227–231
  checking format of fields, 232–236
VARCHAR data type, MySQL, 58, 239, 294, 331
var_dump statement, PHP, 153
variables, in cookies, 264–266
variables, in PHP sessions, 269–271
variables, in URLs, 260–261
variables, PHP
  assigning dates to, 130–131
  assigning numbers to, 119–121, 148–149
  assigning strings to, 119–121, 125, 148–149
  checking for existence of, 363
  definition of, 119
  deleting, 120
  dollar sign missing from, 368
  enclosing in quotes, 126
  in functions, local and global, 180–181
  misspelling name of, 368
  naming, 119
  Notices regarding, 121
  storing form data in, 238
vertical bar (|), in patterns, 137
vertical bars, double (||), in joined comparisons, 141

• W •

Warning messages, 117
Web application, 10
Web database application. *See also* Members Only example; Pet Catalog example
  company Web site for, 22–24
  definition of, 10–11
  developing, tasks for, 61–62
  documenting, 288
  expansion of, planning for, 41–42
  local Web site for, 26–32
  logins for
    definition of, 328
    file containing HTML for, 348–352
    look and feel for, designing, 333–336
    programs for, 337–338, 340–348
  look and feel for, designing, 299–305, 333–337
  MySQL functions, 132
  organizing, 278–279
  planning, 37–42, 277–284, 288–291, 328
  programs in
    designing, 288
    ending, 150, 227, 362
    including external files in, 281–284
    naming, 278
    number of, 278
    organizing, 279–284
    subdirectories for, 279
    writing (creating), 62
  purpose of, identifying, 38
  security of, 285–287
  software required for, 21
  tasks performed by, defining, 38–40
  usability engineering for, 41
  user's requirements for, defining, 40–41
  Web hosting company for, 24–26
  Web site options for, 22
Web hosting company, 24–26
Web pages. *See* Web sites
Web servers
  installing, 29–30
  PHP processing by, 114
  PHP support for, 17
  preventing from displaying filenames, 286
  security of, 287
  in URLs, 257
Web site resources
  Apache Web server, 29
  MySQL, 13, 31, 375
  MySQL announcement lists, 20
  MySQL data types, 59
  MySQL functions, 84

Web site resources *(continued)*
MySQL upgrades, 110
PHP announcement lists, 20
PHP built-in functions, 185
PHP date format symbols, 129
PHP editors, 118
SSL implementations, 287
Web sites. *See also* Web database application
company Web site, 22–24
interactive, 9
on local computer, setting up, 26–32
navigation between pages, 256–259
operating systems for, 28–29
sharing information between pages
with cookies, 260, 264–266
with HTML forms, 260, 267
with PHP sessions, 260, 267–274
with URL, 260–264
static, 9, 10
Web space, 33
`week` keyword, in `strtotime` function, 130
`WHERE` clause, `SELECT` query, 84–87
`while` statement, PHP, 170–172
Windows
configuring MySQL on, 378–379
installing Apache on, 412–414
installing MySQL on, 375–377
installing PHP on, 400–403
for local Web site, 28
starting and stopping Apache on, 414–415
starting MySQL on, 380
stopping MySQL on, 380–381

• X •

`xor` keyword, in comparisons, 139–140

Y

`year` keyword, in `strtotime` function, 130
`yesterday` keyword, in `strtotime` function, 130

Notes

Notes