PHP and MySQL

24-Hour Trainer

Andrea Tarr

PHP AND MYSQL® 24-HOUR TRAINER

Continues

PHP and MySQL®

24-HOUR TRAINER

Andrea Tarr

WILEY

John Wiley & Sons, Inc.

PHP and MySQL® 24-Hour Trainer

For my parents, who gave me the feeling that it was perfectly natural for a girl to have a passion for math.

CREDITS

EXECUTIVE EDITOR
Carol Long

PROJECT EDITOR
Charlotte Kughen, The Wordsmithery LLC

TECHNICAL EDITOR
Wim Mostrey

PRODUCTION EDITOR
Kathleen Wisor

COPY EDITOR
Kim Cofer

EDITORIAL MANAGER
Mary Beth Wakefield

FREELANCER EDITORIAL MANAGER
Rosemarie Graham

ASSOCIATE DIRECTOR OF MARKETING
David Mayhew

MARKETING MANAGER
Ashley Zurcher

BUSINESS MANAGER
Amy Knies

PRODUCTION MANAGER
Tim Tate

VICE PRESIDENT AND EXECUTIVE GROUP PUBLISHER
Richard Swadley

VICE PRESIDENT AND EXECUTIVE PUBLISHER
Neil Edde

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Katie Crocker

PROOFREADER
Corina Copp, Word One

INDEXER
Robert Swanson

COVER DESIGNER
Ryan Sneed

COVER IMAGE
© Clayton Hansen / iStockPhoto

VERTICAL WEBSITES PROJECT MANAGER
Laura Moss-Hollister

VERTICAL WEBSITES ASSISTANT PROJECT MANAGER
Jenny Swisher

VERTICAL WEBSITES ASSOCIATE PRODUCER
Shawn Patrick

DVD TECHNICAL PRODUCER
Focal Point Studios LLC

ABOUT THE AUTHOR

ANDREA TARR has been a programmer and IT manager for 30 years and now works for Tarr Consulting and 4Web Inc. writing custom extensions, templates, and websites with the open source content management system Joomla! She is currently a member of the Joomla Production Leadership Team and is active in the Joomla Bug Squad. Andrea was involved in the development of Joomla 1.6 and created the accessible administrator template Hathor. She wrote the first computerized library circulation system in the state of New Hampshire and holds a Master of Science in Information Technology from Marlboro College Graduate School.

ABOUT THE TECHNICAL EDITOR

WIM MOSTREY has 10 years' experience in PHP development and is a long-time Drupal developer. He's passionate about enabling corporate, non-profit, and governmental organizations to switch to free and open-source software.

ACKNOWLEDGMENTS

CONTENTS

SECTION III: OBJECTS AND CLASSES

www.it-ebooks.info

INTRODUCTION

PHP IS A POPULAR PROGRAMMING LANGUAGE that powers many websites. It originally started out as a way to make dynamic websites by generating HTML. Today it stands on its own as a general-purpose programming language and is available on most web hosting sites. Because of its roots, it is very easy to insert bits and pieces of PHP inside of standard HTML/XHTML code.

MySQL is a popular relational database management system. It is the standard database system available on web hosting sites. Although it works with many different programming languages, it is frequently paired with PHP.

WHO THIS BOOK IS FOR

This book is for beginners who have never programmed before or who have never worked with databases. It's also for those who have copied a few lines of PHP into their HTML pages and want to know more. General programming concepts are explained while you learn to program PHP and manipulate data with MySQL. If you already program other languages, this book may be too basic for you.

To get the most out of this book, you need to understand HTML and the basic concept of CSS. Much of the PHP that you do is aimed at creating HTML, so you need to know what you are trying to create.

WHAT THIS BOOK COVERS

This book teaches you to take a static website and turn it into a dynamic website run from a database using PHP and MySQL. You start by preparing your computer to run PHP and MySQL by downloading and installing free software. Next, you write your first PHP by including some PHP code on an HTML page. Then you dive into PHP, learning what variables are, how to work with them, and how to debug your programs. You learn how to have your programs make decisions and loop through code.

The modern PHP is object oriented. You learn what that means and how to use it to make your programs less buggy and error prone, and easier to maintain. Along with that you learn best practices and how to write secure code.

You learn how databases work and how to design one, as well as how to use phpMyAdmin to work with MySQL. You learn different ways of connecting to MySQL through PHP, and how to create tables, enter data, select data, change data, and delete data. Finally, you learn how to combine all of these things into creating a mini content management system with a dynamic menu.

PHP is a general-purpose language that isn't limited to running websites. However, this beginning book is concentrated on programming websites because that is a natural extension for those who have been coding in HTML and CSS. By the same token, database programming can be quite complex. This book teaches the fundamentals needed to work with databases and how to do it safely.

HOW THIS BOOK IS STRUCTURED

This book consists of short lessons, each focusing on a particular aspect of PHP and/or MySQL. The lessons are arranged in a logical order of study. Although you can study the lessons in any order, you often need to know what is taught in the early lessons before the later ones make sense. It is not meant as an exhaustive resource or as an in-depth look at technical aspects of the language. The goal is to teach you what you need to know in order to start using PHP and MySQL in your web pages and applications.

This book consists of 31 lessons, broken into six sections:

➤ **Section I, "Getting Started with PHP":** In this section you set up your computer to run PHP and MySQL. You learn the fundamentals of programming as you learn the fundamentals of programming in PHP.

➤ **Section II, "Working with PHP Controls, Functions, and Forms":** In this section you learn how to control what lines your programs will process and how to loop through repeating program steps. You learn about creating your own functions and how to process HTML forms.

➤ **Section III, "Objects and Classes":** In this section you learn what object-oriented programming is and why you want to use it. Then you learn how to use it.

➤ **Section IV, "Preventing Problems":** In this section you learn how to handle errors and how to write secure code.

➤ **Section V, "Using a Database":** In this section you are introduced to databases and how to design a database. You learn the basics of how MySQL works and then how to integrate it with PHP. You learn how to take static information on your HTML page and put it in a database and retrieve it.

➤ **Section VI, "Putting It All Together":** In this last section you take what you have learned and create user logins, a mini content management system, and a menu based on database information.

The lessons end with a tutorial called "Try It." Each tutorial applies concepts from the lesson.

> *A Case Study is used in most of the Try It sections. This Case Study starts as a static website created from HTML and CSS. As the lessons progress, you replace parts of it with PHP and MySQL until at the end you have a website that takes its information from a database that you maintain through pages on the website.*
>
> *As you work through the Case Study, you start most Try It sections with the Case Study that you finished in the previous Try It section. However, at any time you can download the Case Study code as it should be at the beginning of each Try It section and also download the code as it should be at the end of the Try It section.*

You can watch the DVD to see the Try It sections from the lesson done by the author. After you've finished reading the book and watching the DVD, you can visit Wrox's P2P forums, where your author offers support.

WHAT YOU NEED TO USE THIS BOOK

To get the best results from this book, you should perform the examples and do the Try It sections. In order to do that, you need the following resources:

➤ PHP and MySQL need to run on a web server. You have two options: You can turn your computer into a local web server or you can use an online web host that runs PHP 5.3 and MySQL 5. This book assumes that you will be running a local web server and the first lesson steps you through the process of downloading free software and configuring your computer.

➤ You need a text editor that can produce plain-text files. The first lesson shows you how to download and install the Eclipse PDT, which is a very helpful editor for writing PHP. However, other text editors such as Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans also work. A word processing program such as Microsoft Word does not work.

INSTRUCTIONAL VIDEOS ON DVD

Some people learn better with a visual and audio aid. That is why a DVD that includes a video tutorial for each lesson accompanies this book. So if seeing something done and hearing it explained help you understand a subject better than just reading about it, this book-and-DVD combination is just the thing for you.

CONVENTIONS

To help you get the most from the text and keep track of what's happening, this book uses a number of conventions.

> *Boxes like this one hold important, not-to-be forgotten information that is directly relevant to the surrounding text.*

> *Notes, tips, hints, tricks, and asides to the current discussion are offset and placed in italic like this.*

> *References like this one point you to watch the instructional video on the DVD with the print book or watch online at* `www.wrox.com/go/24phpmysql`*.*

As for styles in the text:

➤ New terms and important words are *italicized* when introduced.

➤ Code appearing in text looks like this: `document.body`.

➤ URLs look like the following when inside text: `www.wrox.com`.

➤ Code blocks are presented in the following way:

```
A monofont type on its own line(s)
denotes code examples.
```

➤ Important or changed parts of code blocks are highlighted in the following way:

```
Some code examples have
sections that are highlighted which
illustrate different or key parts.
```

SUPPORTING PACKAGES AND CODE

As you work through the lessons in this book, you can choose to type the code and create all the files manually or you can use the supporting code files that accompany the book. All the code and other support files used in this book are available for download at `www.wrox.com`. On the site, simply locate the book's title (either by using the Search box or by using one of the title lists), and then click the Download Code link on the book's detail page to obtain all the source code for the book.

Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-1-118-06688-1.

After you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at `www.wrox.com/dynamic/books/download .aspx` to see the code available for this book and all other Wrox books.

ERRATA

Every effort is made to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in this book or any Wrox book for that matter, such as a spelling mistake or faulty piece of code, your feedback is appreciated. By sending in errata, you can save a reader hours of frustration, and at the same time, you can help your author and Wrox provide even higher-quality information.

To find the errata page for this book, go to `www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the Book Search Results page, click the Errata link. On this page, you can view all errata that have been submitted for this book and posted by Wrox editors.

A complete book list, including links to errata, is also available at
`www.wrox.com/misc-pages/booklist.shtml.`

If you don't spot "your" error on the Errata page, click the Errata Form link and complete the form to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's Errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at `p2p.wrox.com`. The forums are a Web-based system you can use to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to email you topics of interest of your choosing when new posts are made to the forums. Wrox authors and editors, other industry experts, and your fellow readers are present on these forums.

At `http://p2p.wrox.com` you can find a number of different forums that help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `p2p.wrox.com` and click the Register link.

2. Read the terms of use and click Agree.

3. Complete the required information for joining as well as any optional information you want to provide and click Submit.

4. You receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.

After you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum emailed to you, click the Subscribe to This Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

SECTION I
Getting Started with PHP

In this section, you learn the basics of working with PHP. In the first lesson, you learn what PHP requires on your computer before PHP will run. If your computer does not have the necessary software, you can use the instructions provided to download the free software, install it, and configure it to work. In the next lesson, you learn how HTML and PHP work together as you add your first PHP code to a web page. You are also introduced to the Case Study website you use throughout the book.

You learn in the third lesson about the syntax of PHP and how to write PHP statements. In the fourth lesson, you learn what variables are and how to use them. At this point, you will have learned enough to start making mistakes, so in the next lesson you learn about how to find your errors and debug your code. You need to know about debugging as you work with more complex data in the final lesson of this section.

1

Setting Up Your Workspace

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You learn how to download and install XAMPP in this lesson.

If you already have a web server with PHP and MySQL running on your computer, you do not need XAMPP. Other packages that fulfill the same need are WAMPServer and MAMP.

You also need a text editor that can produce plain-text files. You learn how to download and install Eclipse PDT in this lesson. Some other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

INSTALLING XAMPP

XAMPP stands for whatever operating system you have: (X), Apache (A), MySQL (M), PHP (P), and Perl (P). Separate packages are available for each of the different operating systems such as Windows, Mac OS X, or Linux. This lesson covers installing the Windows and Mac versions.

> *Perl is another programming language. It's popular for housekeeping tasks and for communications between different programs and programming languages. You won't need to use it for the lessons in this book.*

XAMPP is intended for local development work. It is not set up for running production websites.

> ⊗ *Do not use XAMPP to host websites on the Internet. Although it uses the same building blocks as production hosts, it is not set up to be secure. You will get hacked if you try it.*

Installing XAMPP on a Windows PC

This section walks you through downloading the proper XAMPP package and installing it on your Windows PC. If you have a Mac, skip forward to the section "Configuring XAMPP on Mac OS X."

1. Go to the Apache Friends website at `www.apachefriends.org/en/xampp.html`.

2. Locate the section labeled XAMPP for Windows and click the title. Scroll down to the Download section that lists the versions available for download. See Figure 1-1.

FIGURE 1-1

3. You have a choice of three ways to install this package: via the installer, via a ZIP file, or via a 7zip file. The easiest way to change options is to use the installer, but you are more likely to encounter problems. Because you are using the defaults, use the ZIP version. Click the ZIP link and save the ZIP file.

4. Unzip all the files to `c:\`. The ZIP file contains a folder called xampp that holds all the folders and files so unzipping to the `c:` drive creates the `c:\xampp` folder.

5. The program you use is `c:\xampp\xampp-control.exe`. In Windows Explorer, right-click the file and select Create Shortcut. Drag that shortcut to your desktop.

6. Double-click the XAMPP Control desktop icon you just created. The Control Panel is displayed. See Figure 1-2.
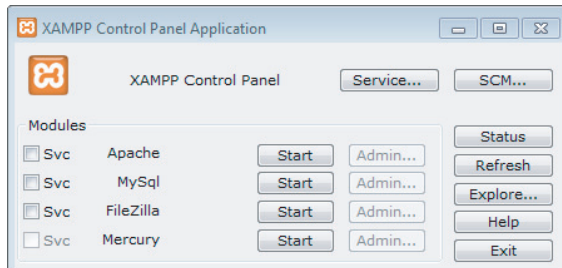
7. To start XAMPP, first start the Apache web service by clicking the Start button next to Apache. Then start MySQL by clicking the Start button next to MySQL. You do not need to start FileZilla or Mercury. When you click the Start buttons, they change to Stop buttons to indicate that the processes are running.

8. To stop XAMPP, click the Stop button next to MySQL and then click the Stop icon next to Apache.

9. To test that XAMPP is properly working, go to your browser and enter `http://localhost/xampp`. You should see a screen similar to Figure 1-3.
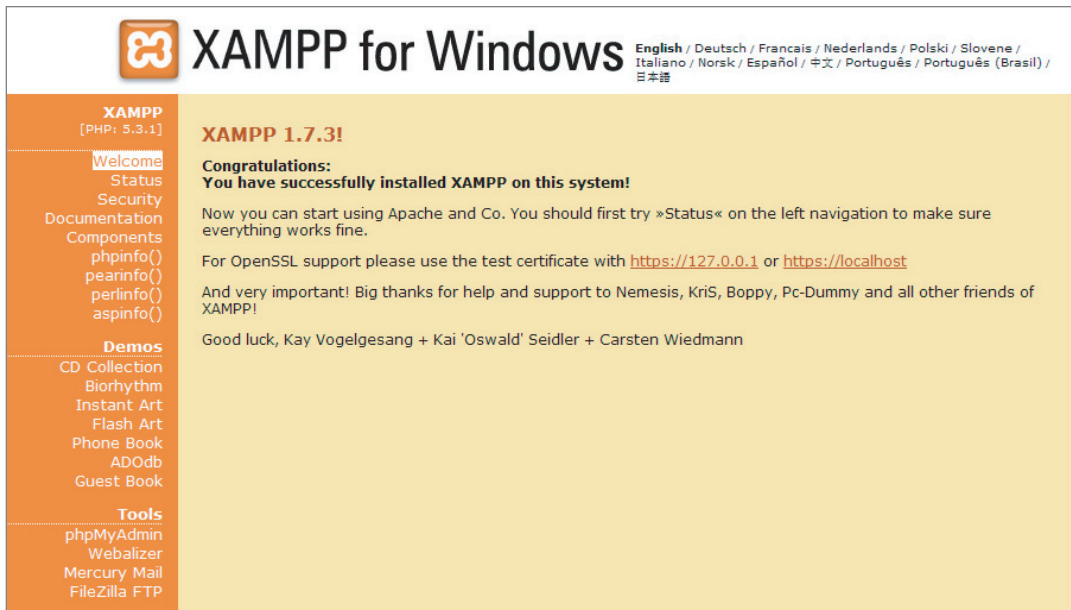
FIGURE 1-3

If the installation is successful, skip to the "Configuring XAMPP" section later in this lesson. Otherwise, check out the "Troubleshooting Your XAMPP Installation" section that follows the "Installing XAMPP on Mac OS X" section.

Installing XAMPP on Mac OS X

This section walks you through downloading the proper XAMPP package and installing it on your Mac OS X system. If you are using a Windows PC, you used the prior section to install XAMPP so you can jump forward to the "Configuring XAMPP" section.

1. Go to the Apache Friends website at `www.apachefriends.org/en/xampp.html`.

2. Locate the section labeled XAMPP for Mac OS X and click the title. Scroll down to find the section labeled Installation in 4 Steps. See Figure 1-4.

Installation in 4 Steps

Step 1: Download

Simply click on the link below. It's a good idea to get the latest version. :)

A complete list of downloads (older versions) is available at ⧉ SourceForge. There are none yet, but there will be.

XAMPP for Mac OS X 1.7.3, 2010/03/04		
Version	Size	Notes
⧉ XAMPP Mac OS X 1.7.3 Universal Binary	86 MB	Apache 2.2.14, MySQL 5.1.44, PHP 5.3.1, Perl 5.10.1, ProFTPD 1.3.3, phpMyAdmin 3.2.4, OpenSSL 0.9.8k, GD 2.0.35, Freetype 2.3.5, libjpeg 6b, libpng 1.2.32, libungif-4.1.4, zlib 1.2.3, expat 2.0.1, Ming 0.4.2, Webalizer 2.01-10, pdf class 009e, mod_perl 2.0.4, SQLite 3.6.3, gdbm-1.8.3, libxml-2.7.2, libxslt-1.1.24, openldap-2.3.43, imap-2004g, gettext-0.16.1, libmcrypt-2.5.8, mhash-0.9.9, zziplib-0.13.48, bzip2-1.0.5, freetds-0.64 MD5 checksum: fcbd4b14461a5b9e7a817f99defd0be2
⧉ Developer package	32 MB	Developer package MD5 checksum: f31a0619a35507a0e4305b674ae1159b

FIGURE 1-4

3. Click XAMPP Mac OS X. You want the Universal Binary, not the Developer Package. Click OK to save the file when asked.

4. Open the `.dmg` file you just saved. Drag the XAMPP icon over to the Applications icon as shown in Figure 1-5.

5. Find the XAMPP `Control.app` in `/Applications/XAMPP/Xamppfiles`. This is the application file that you use to start and stop XAMPP and you will find it convenient to add it to your dock. The first time you open it you receive the standard warning about using files from the Internet. Click the Open button to start the Control Panel. The Control Panel looks like Figure 1-6.

6. To start XAMPP, first start the Apache web service by clicking the Start button next to Apache. Then start MySQL by clicking the Start button next to MySQL. You do not need to start FTP. When you click the Start buttons, they change to Stop buttons to indicate that the processes are running.
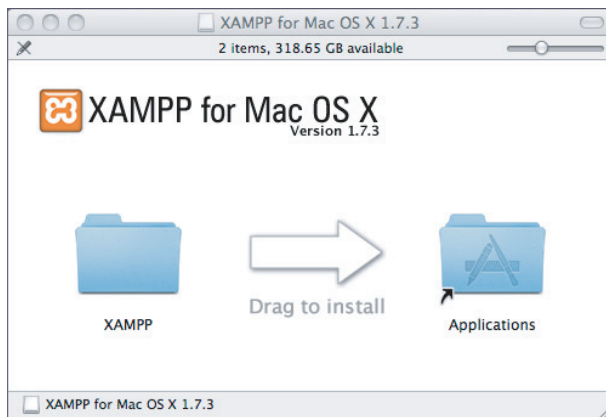
FIGURE 1-5
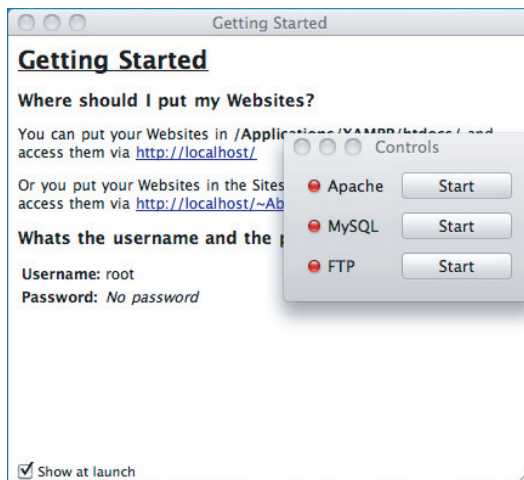
FIGURE 1-6

7. To stop XAMPP, click the Stop button next to MySQL and then click the Stop button next to Apache.

Apache needs to be running for `http://localhost` *and PHP to work. If you get an error that the server cannot be found, check that you've started Apache.*

8. To test that XAMPP is properly working, go to your browser and enter **http://localhost/xampp**. You should see a screen similar to Figure 1-7.

FIGURE 1-7

Troubleshooting Your XAMPP Installation

Usually, XAMPP installs easily. Sometimes, however, you can run into issues. The Apache Friends have a forum where you can find answers to many problems at `www.apachefriends.org/f/viewforum.php?f=34`.

The Mac OS X ships with Apache. Apache works by listening on a specific port. If you run two copies of Apache, both listening to the same port, you will have problems. The default port is 80 and the common alternate port to use is 8080. If you need both, change the port on one of them and restart Apache.

If you want to change the port in XAMPP, go to `/Applications/XAMPP/xamppfiles/etc/httpd.conf` and change `Listen 80` to **`Listen 8080`**. Stop and restart Apache for the change to take effect. If you cannot get into the XAMPP control to stop and start Apache, shut down your Mac and restart it.

If you want to change the port in the pre-installed Apache, go to `etc/Apache2/http.conf` and change `Listen 80` to **`Listen 8080`**. To get to this hidden file, go to Finder and press Shift+Command+G and then enter **`\etc`**. You need to restart the pre-installed Apache. The easiest way to do that is to shut down your Mac and restart it.

> *If you changed the port that Apache listens to, you need to enter it as part of the address. If you changed the port to 8080, the address is* `http://localhost:8080/xampp`.

Skype is another program that might conflict with port 80. If you have problems, look in the Skype Advanced section of Tools/Options (on the PC) or Preferences (on the Mac) and be sure it isn't using port 80 for incoming or alternative ports.

Configuring XAMPP

Now that you have successfully installed XAMPP on your Windows PC or Mac, make sure XAMPP is running and then call up XAMPP in your browser. The address to call up XAMPP is `http://localhost/xampp`. A screen similar to Figure 1-8 displays.

FIGURE 1-8

You need to create a password on MySQL. Some programs do not allow you to use MySQL unless MySQL has a password, for security reasons. Click the phpMyAdmin link on the left-side navigation under Tools to open the page shown in Figure 1-9.

Click Privileges on the top menu. You see a table of the users. Click the Edit icons next to the users. Scroll down to find the Change Password box as shown in Figure 1-10.

Enter a password and click Go. Do this for each of the users with All Privileges.

Now that you've added a password to MySQL, you need to change the configuration in XAMPP for phpMyAdmin so that it can access the database. The configuration file is in `c:\xampp\phpMyAdmin\config.inc.php` on the Windows PC or in `/Applications/XAMPP/xamppfiles/phpmyadmin/config.inc.php` on the Mac. Find the following code:

```
/* Authentication type */
$cfg['Servers'][$i]['auth_type'] = 'config';
/* Server parameters */
$cfg['Servers'][$i]['host'] = 'localhost';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = '';
```
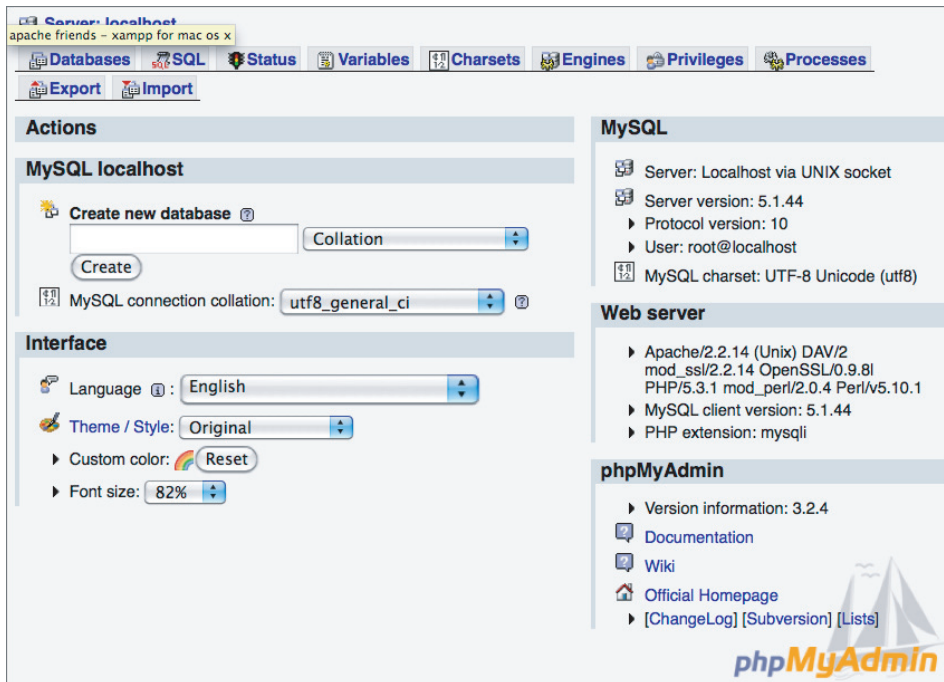
FIGURE 1-9
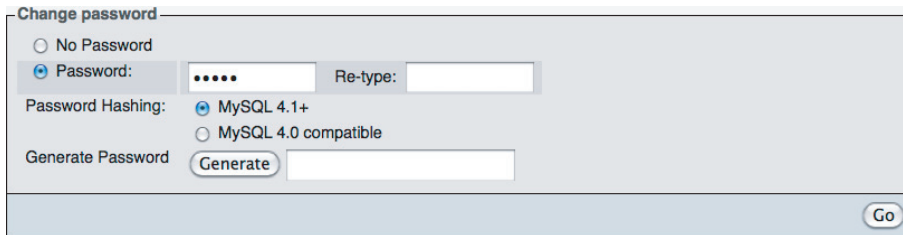
FIGURE 1-10

Change the password to your new password. For instance, if your new password is !xYz72g, the change looks like the following:

```
$cfg['Servers'][$i]['password'] = '!xYz72g';
```

Restart the XAMPP server by going into the Control Panel and stopping first MySQL and then Apache. Restart by starting Apache and then restarting MySQL.

Call up XAMPP in your browser and see that you can open phpMyAdmin.

INSTALLING YOUR EDITOR

You need a text editor for programming. Word processing editors such as Word change your code and add extraneous codes and characters that invalidate your program, so you should not use them. Possible text editors are Notepad, TextWrangler, Dreamweaver in the code mode, NetBeans, or Eclipse.

A good text editor for PHP is Eclipse PDT. It has syntax checking, auto-completion for commands, color syntax coding, debugging, and other features that become important as you do more complex PHP programming.

To install Eclipse PDT, go to `http://www.eclipse.org/pdt/downloads/` to download the program. You see a screen similar to Figure 1-11.
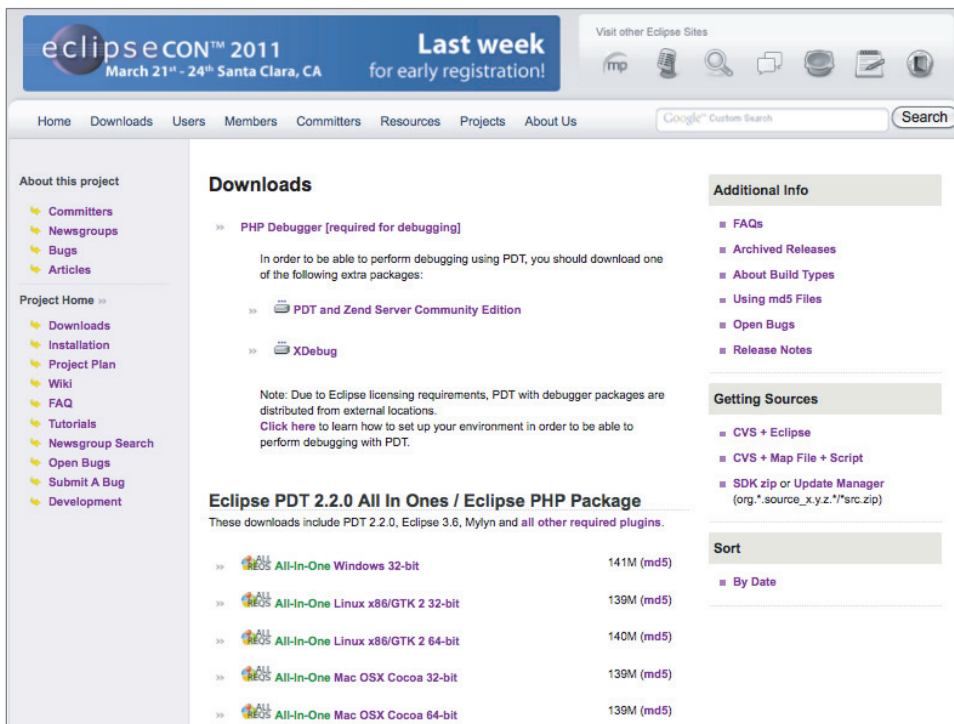
FIGURE 1-11

Find the All-in-One package for your operating system and click it. You are given a choice of mirrors from which you can download the package, as shown in Figure 1-11. Click the mirror displayed (which in this figure is Georgia Tech) and save the file when requested.

Unzip the file in an appropriate folder. In Windows a good folder is `c:\eclipse`. It does not need to be installed. On the Mac, put it in the Applications directory.

The program file is `eclipse.exe` in the `eclipse` folder. Make a shortcut on your desktop or add it to your dock (on the Mac) so you can find it easily.

CONFIGURING YOUR WORKSPACE

Now that you've installed the programs, you need to do some configuring.

Preparing a Place to Put Your Files

The first thing you need to do is decide where you are going to put your files. By default the Apache web server looks for web files in the htdocs folder. On a Windows PC, this is directly off where you installed XAMPP. If you installed XAMPP in `c:/xampp`, then your htdocs file is in the `c:/xampp/htdocs` folder. On a Mac the path is `/Applications/XAMPP/xamppfiles/htdocs`.

If you are going to be doing a lot of development work, you should change this default and set up virtual hosts so that you can put your files in more convenient places. However, setting up virtual hosts is beyond the scope of this book, so use the default htdocs folder.

If you are on a Windows PC, skip forward to the next section, "Using Eclipse for the First Time."

On the Mac OS X you need to change the permissions to the htdocs folder in order to add folders and files to it. Open Finder, browse to `/Applications/XAMPP/xamppfiles`, and select htdocs as shown in Figure 1-12.
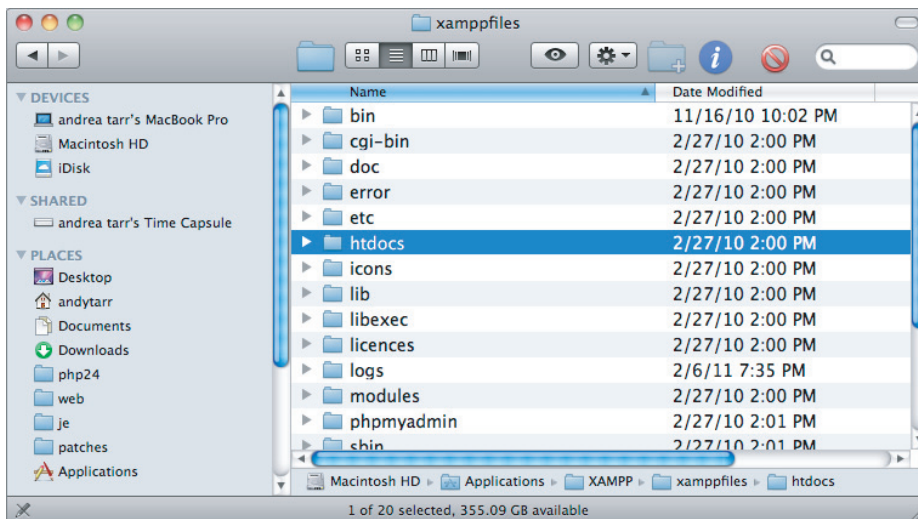
FIGURE 1-12

Press Command+I to display the htdocs Info as shown in Figure 1-13.
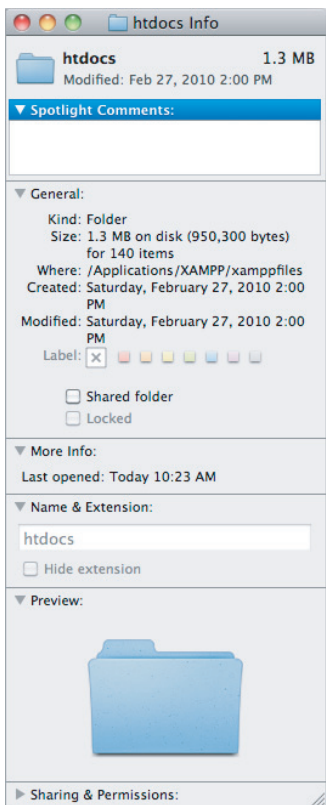
FIGURE 1-13

Click the arrow next to Sharing & Permissions to expand the section as shown in Figure 1-14.
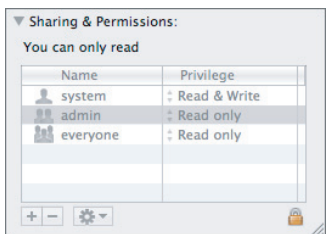
FIGURE 1-14

You need to unlock the padlock before you are allowed to make changes to the permissions. Click the padlock in the lower-right corner. Enter your administrator password for the Mac when asked.

Now you can click the Privilege drop-down for the admin. Change from Read Only to Read & Write. Your permissions should look similar to Figure 1-15.
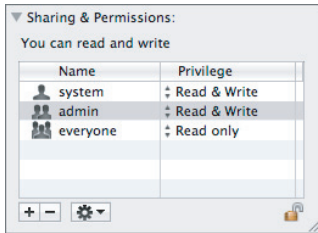
FIGURE 1-15

Click the padlock again to relock the permissions.

Using Eclipse for the First Time

The first time you go into Eclipse, you have to identify an Eclipse workspace. See Figure 1-16. This workspace is the place that you put your files. You use the htdocs folder.
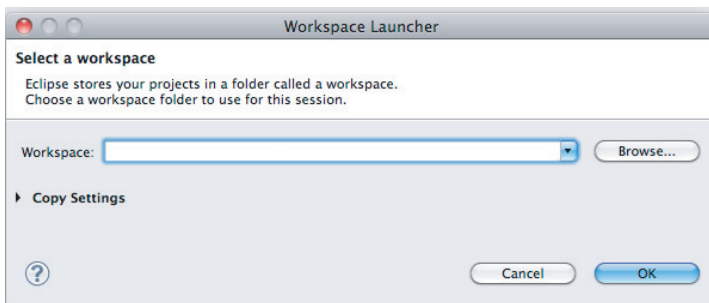
FIGURE 1-16

Use the Browse button to locate and accept the htdocs folder. On the PC, if you installed XAMPP in `c:/xampp`, your htdocs file is in the `c:/xampp/htdocs` folder. On a Mac, the path is `/Applications/XAMPP/xamppfiles/htdocs`. See Figure 1-17. Click OK.
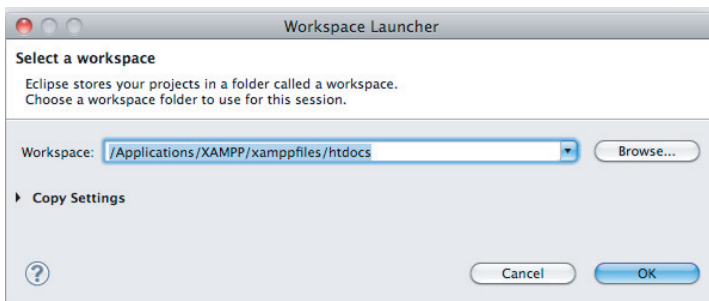
FIGURE 1-17

Eclipse displays a splash screen along with a request for permission for Eclipse to collect and send usage information, as shown in Figure 1-18.
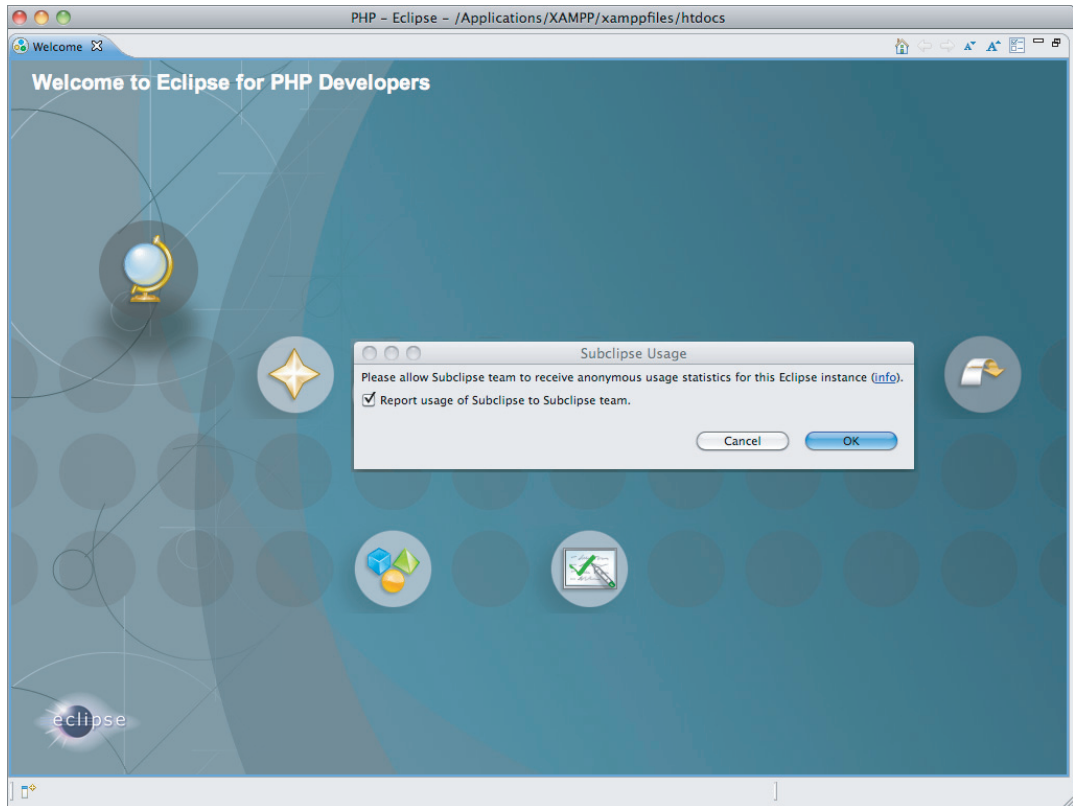
FIGURE 1-18

Uncheck the box if you do not want to have this information collected and sent. Click OK and then close the tab for the splash screen. You see the main workspace for Eclipse, as shown in Figure 1-19.

Across the top of the window, Eclipse lists the path to the workspace you are in. If you use virtual hosts you can create multiple workspaces, although you can have only one open at a time. To switch between workspaces, or to add a new workspace, use File ⇨ Switch Workspaces. You use only one workspace in this book.

Within the workspace, you have projects. All your folders and files are created inside these projects. To create a project for this book, click File ⇨ New ⇨ PHP Project. You see a screen similar to Figure 1-20.
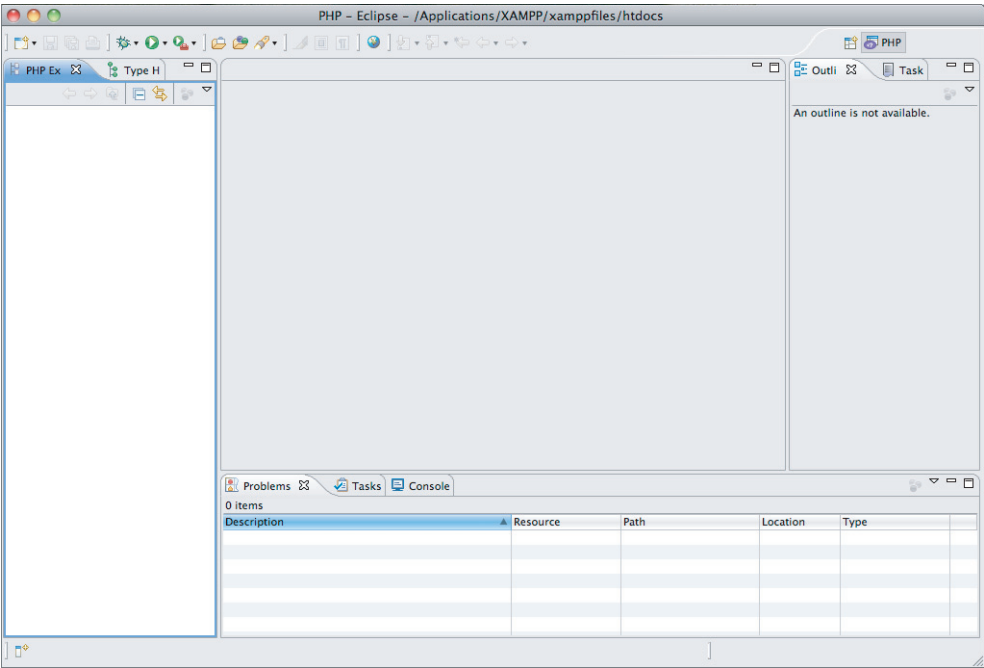
FIGURE 1-19
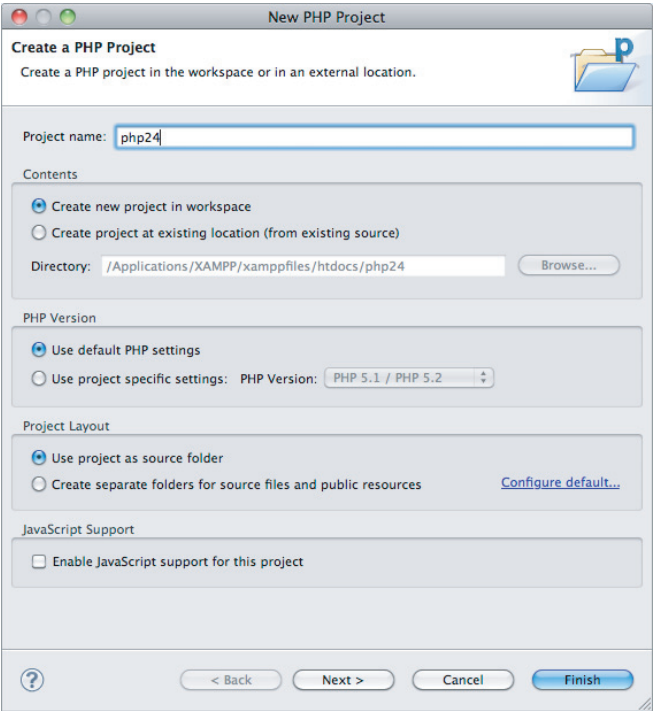
FIGURE 1-20

Type in a project name of `php24`. You can use what you want here, but this is part of the address you use to call your programs, so short and uncomplicated is best. Leave all the rest as the defaults and click Finish. You should see a screen similar to Figure 1-21.
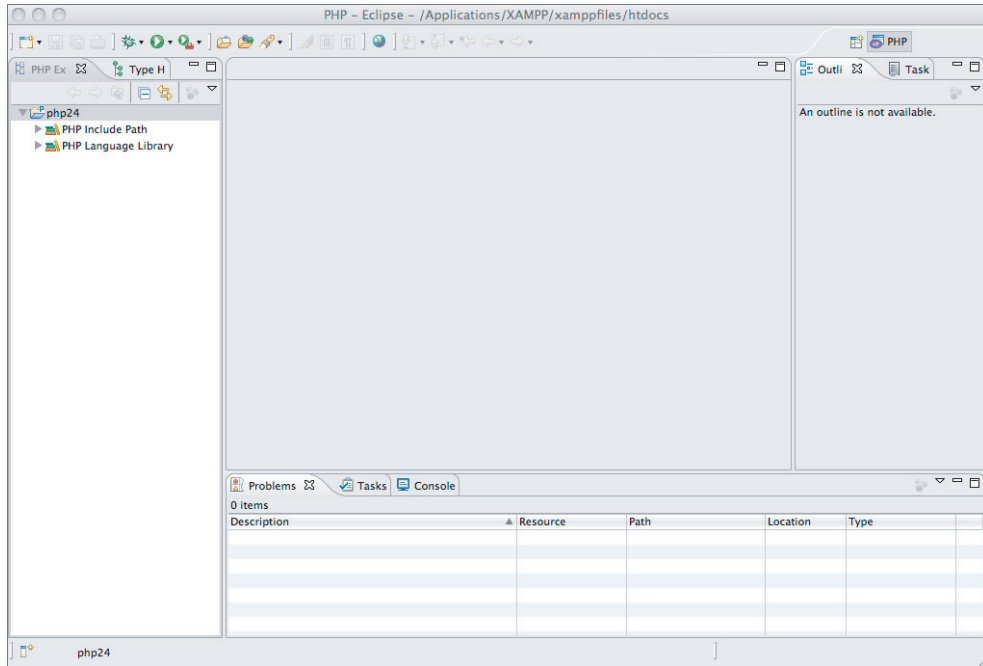
FIGURE 1-21

Notice that you now have the project php24 listed in the left window. This is the PHP Explorer. It works like the Windows Explorer, showing a hierarchical view of the folders and files within the different projects. Just as in Windows Explorer, you use this to open and select files and perform other actions on them. Right-clicking php24 displays a menu as shown in Figure 1-22.

Here's what some of these options do:

➤ **New:** Use this to create new folders and files.

➤ **Copy/Paste/Delete:** These work as you would expect.

➤ **Refactor:** This is where Rename and Move are hidden.

➤ **Refresh:** Use this if you've added, renamed, or moved files outside of Eclipse.

➤ **Debug As:** You learn more about this in Lesson 5.

➤ **Validate:** This is for validating files without needing to open them.

➤ **Compare With:** This compares files either with another file or with an earlier version of the same file. Files are automatically posted to Local History so you can roll back to a prior version. See Options (in Windows) or Preferences (Mac) if you want to change how many copies and how long they are kept.

➤ **Properties:** This is for quite a bit of Meta information.

```
New                              ▶
Go Into

Open in New Window

📋 Copy                          ⌘C
📋 Paste                         ⌘V
✖ Delete                         ⌦

🔧 Remove from Context    ⌥⇧⌘↓
Build Path                       ▶
Refactor                         ▶
Include Path                     ▶

📥 Import...
📤 Export...

🔄 Refresh                       F5
Close Project

🔍 Search...                     ^H
Run As                           ▶
Debug As                         ▶
Profile As                       ▶
Validate
Team                             ▶
Compare With                     ▶
Restore from Local History...
Source                           ▶
Configure                        ▶

Properties                       ⌘I
```
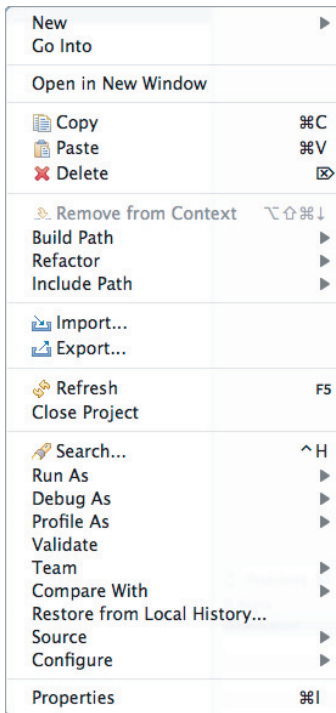
FIGURE 1-22

In the center of the screen is a blank window. This is where you edit your files. Over to the right is another window with tabs in it. The Outline tab shows you the outline of whatever file you have open and selected and can be used to navigate to various sections in the file. The bottom of the screen has another window with tabs. The Problems tab shows validation errors and the Tasks tab lists to-do's you have put inside your files and enables you to navigate directly to them.

This full screen that you see is called by Eclipse a "perspective." This particular perspective is the PHP perspective. Later you use a Debug perspective. Perspectives, which can be customized in Windows ➪ Customize Perspectives, enable you tailor your work area to best suit the project you are working on.

🔘 TRY IT

Available for
download on
Wrox.com

In this Try It, you create a `.php` file and run it to test your PHP setup. This program runs a very handy function to see how you have your PHP configured. This is even handier for hackers so it is good practice not to have a file running it on your system. For a local system it is not a problem however, because you delete it when you are done.

Lesson Requirements

To run PHP files locally, you need to set up your computer so it can process PHP and run MySQL. This lesson shows you how to do that using XAMPP.

You also need to be able to create the PHP files themselves. You can do this with any editor that can produce plain-text files, including Notepad, TextWrangler, Dreamweaver in the code mode, NetBeans, or Eclipse. You learned in this lesson how to install the Eclipse PDT.

You should have a folder called php24 in the htdocs folder where you put your code files. If you created the Eclipse workspace, the folder was created for you automatically.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson01 folder in the download. You will find code for both before and after completing the exercises.*

Hints

This is the code you need in the .php file:

```
<?php phpinfo();
```

Be sure that the Apache server is running when you run the program.

Step-by-Step

> *In this lesson I am giving detailed instructions using Eclipse PDT for those new to Eclipse. You can use a different text editor if you chose not to install Eclipse PDT.*

1. Create a blank .php file called test.php. In Eclipse, create it with the following steps:

   a. Right-click your project in the PHP Explorer.

   b. Select New ⇨ PHP File. Note that this is a PHP file, not a PHP project.

   c. Leave Source Folder as /php24 (your project name).

   d. Type **test.php** as the name of the file in the File Name field.

   e. Click Finish. You see a screen similar to Figure 1-23.

   f. The center window contains a file called test.php. This file is also showing on the left in the PHP Explorer window. If you don't see it below your project, click the triangle next to your project to expand the list.
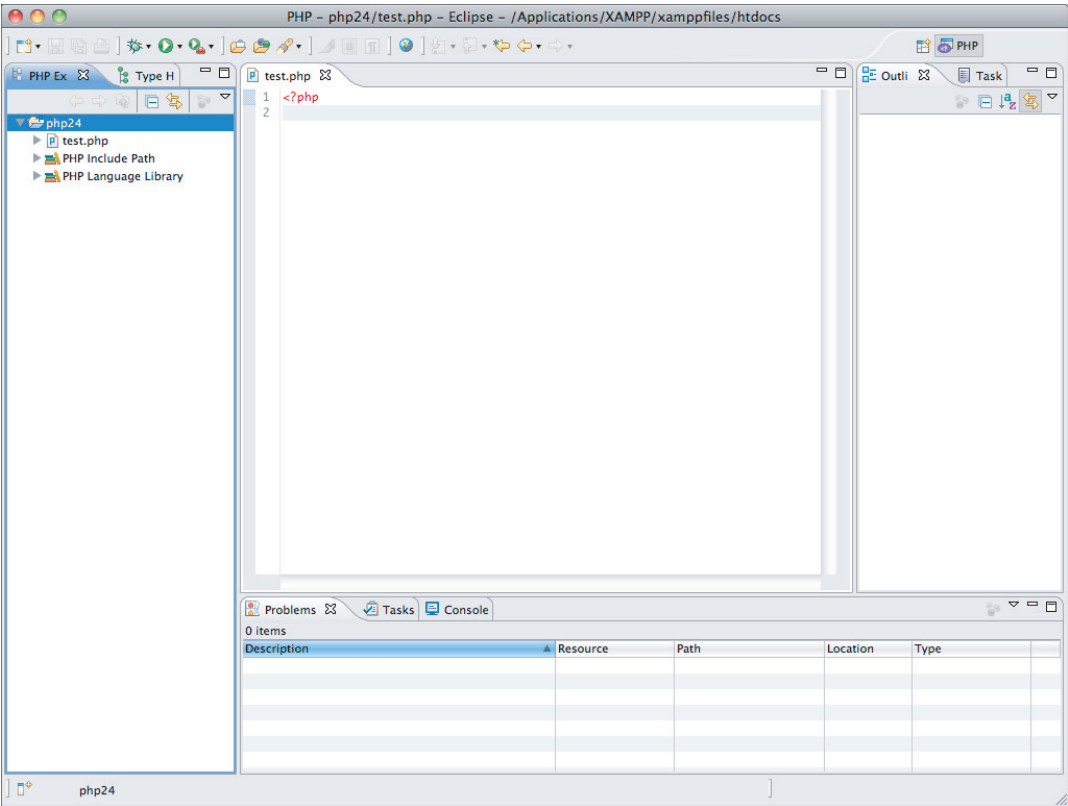
FIGURE 1-23

2.   Your text editor may have already typed in `<?php` for you. If it did, do not retype it in step 3.

3.   Type in **`<?php phpinfo();`**.

4.   Your screen should look like Figure 1-24.
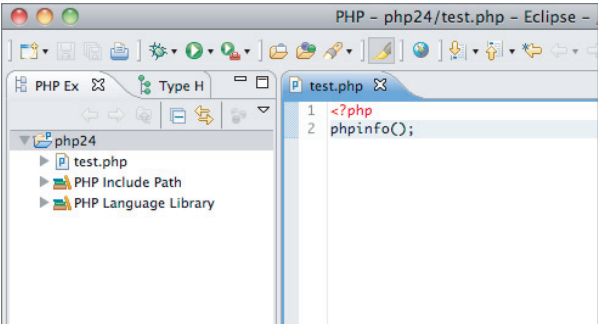
FIGURE 1-24

5. Save the file. In Eclipse you save by pressing Ctrl+S on the PC or Command+S on the Mac. You can also save the file by selecting File ⇨ Save.

6. To run the program, go to your browser and enter `localhost/php24/test.php`.

    a. **localhost:** This identifies the web server, which in our case points to the htdocs folder.

    b. **php24:** This is the folder that was automatically created to hold your project if you use Eclipse, or, if you are not using Eclipse, it is a folder you created to contain your code files.

    c. **test.php:** This is the PHP file to run.

7. Your screen should look similar to Figure 1-25.

PHP Version 5.3.1	
System	Darwin mbpro.local 10.6.0 Darwin Kernel Version 10.6.0: Wed Nov 10 18:13:17 PST 2010; root:xnu-1504.9.26~3/RELEASE_I386 i386
Build Date	Feb 27 2010 12:28:52
Configure Command	'./configure' '--prefix=/Applications/XAMPP/xampfiles' '--program-suffix=-5.3.1' '--libdir=/Applications/XAMPP/xampfiles/lib/php/php-5.3.1' '--includedir=/Applications/XAMPP/xampfiles/include/php/php-5.3.1' '--with-apxs2=/Applications/XAMPP/xampfiles/bin/apxs' '--with-config-file-path=/Applications/XAMPP/xampfiles/etc' '--with-mysql=/Applications/XAMPP/xampfiles' '--disable-debug' '--enable-cli' '--enable-cgi' '--enable-bcmath' '--enable-calendar' '--enable-ctype' '--enable-discard-path' '--enable-filepro' '--enable-filter' '--enable-force-cgi-redirect' '--enable-fastcgi' '--enable-ftp' '--enable-hash' '--enable-ipv6' '--enable-json' '--enable-odbc' '--enable-path-info-check' '--enable-gd-imgstrttf' '--enable-gd-native-ttf' '--with-ttf' '--enable-magic-quotes' '--enable-memory-limit' '--enable-safe-mode' '--enable-shmop' '--enable-sysvsem' '--enable-sysvshm' '--enable-track-vars' '--enable-trans-sid' '--enable-reflection' '--enable-session' '--enable-spl' '--enable-tokenizer' '--enable-wddx' '--enable-yp' '--enable-xmlreader' '--enable-xmlwriter' '--enable-zlib' '--enable-zts' '--with-simplexml' '--with-iconv' '--with-libxml' '--with-wddx' '--with-xml' '--with-ftp' '--with-ncurses=/Applications/XAMPP/xampfiles' '--with-gdbm=/Applications/XAMPP/xampfiles' '--with-jpeg-dir=/Applications/XAMPP/xampfiles' '--with-png-dir=/Applications/XAMPP/xampfiles' '--with-freetype-dir=/Applications/XAMPP/xampfiles' '--without-xpm' '--with-zlib=shared' '--with-zlib-dir=/Applications/XAMPP/xampfiles' '--with-openssl=/Applications/XAMPP/xampfiles' '--with-expat-dir=/Applications/XAMPP/xampfiles' '--enable-xslt=shared,/Applications/XAMPP/xampfiles' '--with-xsl=shared,/Applications/XAMPP/xampfiles' '--with-dom=shared,/Applications/XAMPP/xampfiles' '--with-ldap=shared,/Applications/XAMPP/xampfiles' '--with-gd=shared' '--with-mysql-sock=/Applications/XAMPP/xampfiles/var/mysql/mysql.sock' '--with-mcrypt=/Applications/XAMPP/xampfiles' '--with-mhash=/Applications/XAMPP/xampfiles' '--enable-sockets' '--enable-zend-multibyte' '--with-libxml-dir=/Applications/XAMPP/xampfiles' '--enable-pcntl' '--enable-dbx=shared' '--with-mysqli=shared,/Applications/XAMPP/xampfiles/bin/mysql_config' '--with-pear=/Applications/XAMPP/xampfiles/lib/php/pear' '--with-mssql=/Applications/XAMPP/xampfiles' '--with-imap-dir=/Applications/XAMPP/xampfiles' '--with-imap=shared,/Applications/XAMPP/xampfiles' '--enable-mbstring=shared,all' '--with-pgsql=shared,/Applications/XAMPP/xampfiles' '--with-gettext=/Applications/XAMPP/xampfiles' '--enable-apache2-2filter=shared' '--enable-apache2-2handler=shared' '--with-bz2=shared' '--with-curl=shared' '--with-dba=shared' '--enable-dbase=shared' '--with-fdf=shared' '--enable-mbregex' '--enable-mbregex-backtrack' '--with-mime-magic=shared' '--with-mysql=shared,/Applications/XAMPP/xampfiles' '--enable-shmop=shared' '--with-snmp=shared' '--enable-sockets=shared' '--enable-pdo' '--with-sqlite=shared' '--enable-zip=shared,/Applications/XAMPP/xampfiles' '--enable-exif=shared' '--with-pdo-mssql=shared,/Applications/XAMPP/xampfiles' '--with-pdo-mysql=shared,/Applications/XAMPP/xampfiles/bin/mysql_config' '--with-pdo-pgsql=shared,/Applications/XAMPP/xampfiles' '--with-pdo-sqlite=shared' '--with-pdo-sqlite-external=shared' '--enable-soap=shared' '--with-xmlrpc=shared' '--with-oracle=shared' '--with-pdf=shared' '--with-sqlite3=shared,/Applications/XAMPP/xampfiles'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/Applications/XAMPP/xampfiles/etc
Loaded Configuration	/Applications/XAMPP/xampfiles/etc/php.ini

FIGURE 1-25

TROUBLESHOOTING

If you're having trouble, verify the following things:

➤ There should be a space between `<?php` and `phpinfo()`.

➤ There should be no space between `phpinfo` and the `()`.

➤ Don't forget the final semicolon.

➤ Don't forget to use `localhost:8080` if you changed your XAMPP port to 8080.

8. Go back to your text editor.

9. Close the file. In Eclipse, click the X on the tab with the `test.php` filename to close the file.

10. Delete the file. In Eclipse, delete your file by doing the following:

a. Expand your project by clicking the triangle next to the project name if it is not already expanded.

b. Right-click `test.php`.

c. Click Delete.

d. Confirm that you want to delete the file.

Watch the video for Lesson 1 on the DVD or watch online at `www.wrox.com/go/24phpmysql.`

2

Adding PHP to a Web Page

PHP is a programming scripting language that was first developed to generate HTML statements. Even programs written totally in PHP are ultimately displayed as ordinary HTML.

You can also write programs that are mostly HTML with just the occasional PHP statement. In this lesson you start with an HTML page and learn how to add PHP statements to it.

You are also introduced to the Case Study website. By the end of this book, you will have changed it from a static HTML/CSS site to a dynamic website. In this lesson you add the first bit of dynamic code.

WRITING YOUR FIRST PHP PAGE

You start with a simple HTML page that looks like Figure 2-1. The file is called `lesson2a.html`.

Welcome

Today is Sep 27, 2011

FIGURE 2-1

Here's the HTML for that page:

```
<html>
<head>
    <title>Lesson 2a</title>
</head>

<body>

<h1>Welcome</h1>
<p>Today is Sep 27, 2011</p>

</body>
</html>
```

Now you turn this into a PHP page. All you do is change the file extension from `.html` to `.php` and it becomes a PHP file. When you call that in your browser it still looks just like Figure 2-1.

> *You can display standard HTML in your browser by entering the file path. An example of a file path is* `c:/xampfiles/htdocs/lesson2a.html`. *With* `.php` *files, you enter the address that starts with the web root. An example of an address is* `http://localhost/lesson2a.php`.

The difference is that, because of the `.php` extension, the PHP processor reads through the file to see if there is any PHP to process. Any HTML that it finds it prints to the screen. Because all it found was HTML, the results were just the same as the `.html` file.

So the first rule of writing PHP code is to use the file extension `.php`.

> *Other extensions run PHP code, but for this book,* `.php` *is the only one you need.*

Today isn't really Sep 27, 2011, so you can add PHP code to dynamically display today's date. Make a copy of the previous file, call it `lesson2b.php`, change the title to **Lesson 2b** and replace `Sep 27, 2011` with the following code:

```
<?php echo date('M j, Y'); ?>
```

The file now looks like the following:

```
<html>
<head>
    <title>Lesson 2b</title>
</head>

<body>

<h1>Welcome</h1>
<p>Today is <?php echo date('M j, Y'); ?></p>

</body>
</html>
```

When you run this code in your browser it looks similar to Figure 2-2. Of course your screen displays the current date.

Welcome

Today is Feb 8, 2011

FIGURE 2-2

The PHP processor looks for blocks of code that start with `<?php`. It interprets everything after that as PHP code until it gets to a `?>`. The `echo date('M j, Y')` tells the processor to find today's date;

format it with a three-character month, the day without leading zeros followed by a comma, and a four-digit year; and then output it. The program continues to output the HTML that it finds, but also processes the new PHP code.

You learn more about `echo` later in this lesson and about processing dates in Lesson 6.

INTRODUCING THE CASE STUDY

To give you a feel for how PHP code can make a website dynamic, you take a static HTML/CSS website and replace sections of it with PHP code as you work through the lessons. The site is for a fictitious company called Smithside Auctions. This company holds auctions of historic clothing throughout the year. Figure 2-3 shows the Home page.
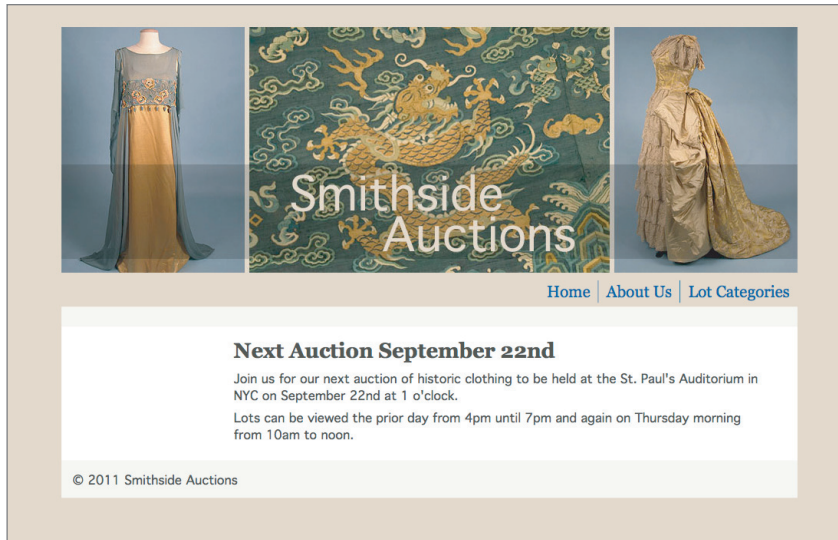
FIGURE 2-3

The About Us page lists the employees, as shown in Figure 2-4.

Smithside Auctions assigns the clothing into lots for the auctions and then puts those lots into appropriate categories. The categories are shown in Figure 2-5 and a sample of the detail of one of the categories is shown in Figure 2-6.
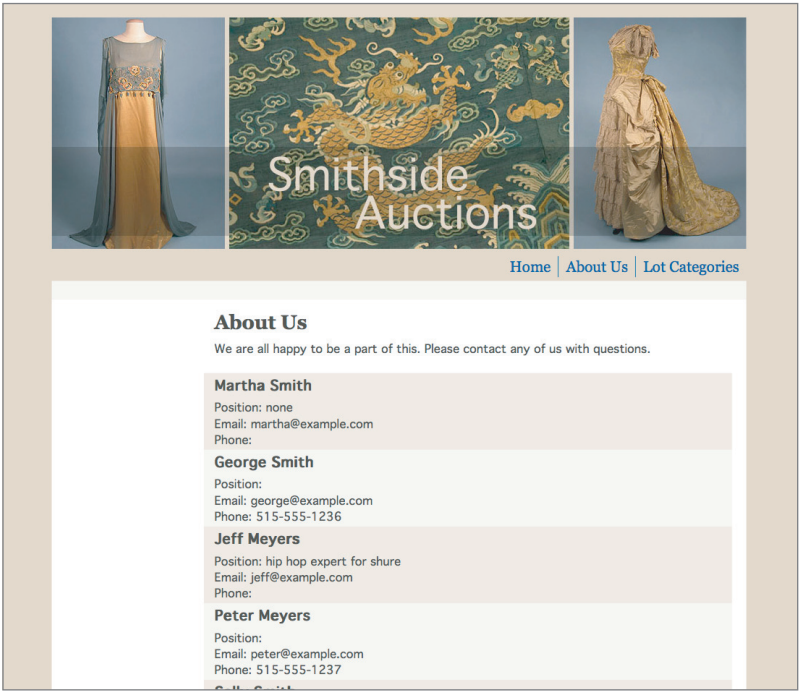
FIGURE 2-4
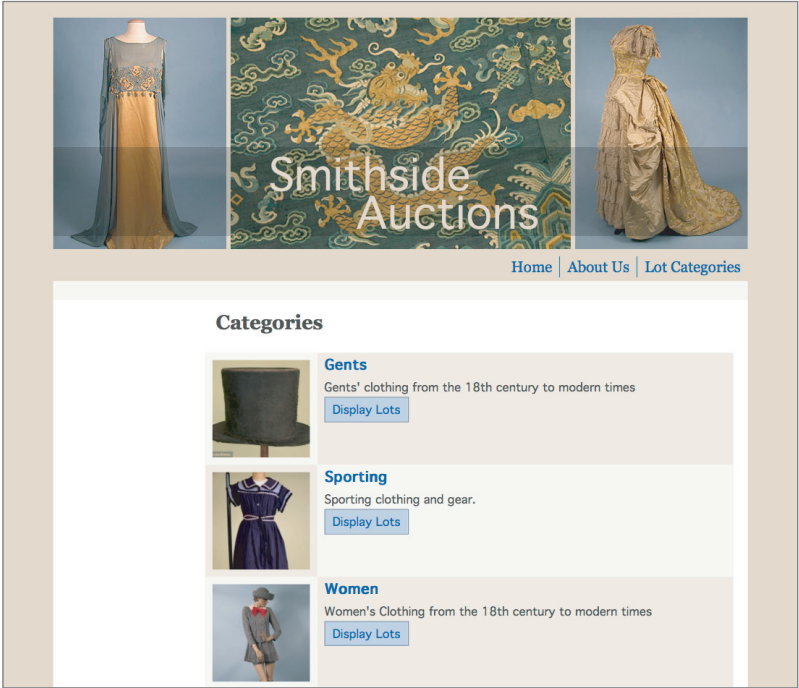
FIGURE 2-5
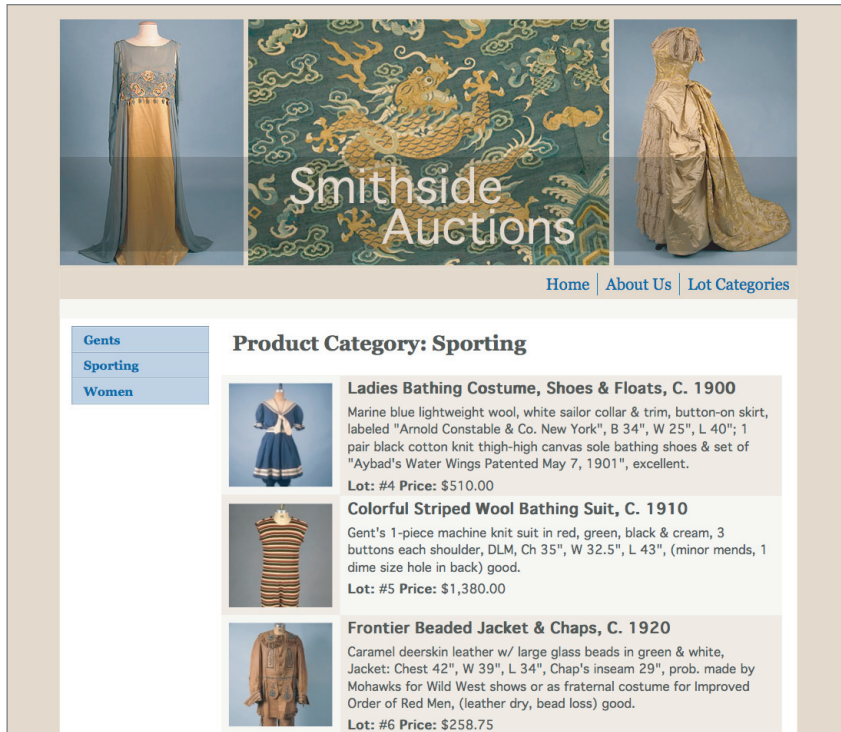
FIGURE 2-6

You can download the code for this site from the book's website at www.wrox.com. There is a version available for the start of each lesson in which you use the case study. Thus you can follow along, making the changes as you do the lessons, or you can simply download appropriate code at any point in the process if you want.

> *The photographs are from Augusta Auctions, an actual company that auctions consigned historic clothing and textiles. You can use the photographs for the exercises in this book. If you want to use them for any other purpose, contact Augusta Auctions through its website at www.augusta-auction.com. While there, you can browse through the rest of the 30,000 photos on the site.*

USING ECHO AND INCLUDE

The echo function tells the processor to output data. The PHP version

```
<?php echo '<p>Where is the dog?</p>'; ?>
```

is the same as the HTML version

```
<p>Where is the dog?</p>
```

You can use either a single quote (') or a double quote ("). If the text you are printing already has single or double quotes in it, use the other type of quote to surround it. For example:

```
<?php echo "<p>Where's my dog?</p>"; ?>
```

is the same as

```
<p>Where's my dog?</p>
```

You learn about some differences between the two quote types in Lesson 4.

The `include` function tells the processor to take the file that you specify and insert it in place of the `include` statement.

Create the file `lesson02d.php` with this code:

```
<h1>Welcome</h1>
<p>Today is <?php echo date('M j, Y'); ?>.</p>
```

> *Some text editors automatically put a `<?php` at the beginning of any PHP file you create. Just type over it if you are not starting the file with PHP code, as in this example.*

Create the file `lesson02e.php` with the following code. Your output should look similar to Figure 2-7.

```
<html>
<head>
  <title>Lesson 2e</title>
</head>

<body>

<?php include('lesson02d.php'); ?>

</body>
</html>
```


Welcome

Today is Feb 15, 2011

FIGURE 2-7

This is what the processor sees:

```
<html>
<head>
  <title>Lesson 2e</title>
</head>

<body>

<h1>Welcome</h1>
<p>Today is <?php echo date('M j, Y'); ?></p>

</body>
</html>
```

If you view the source in your browser, you see:

```
<html>
<head>
  <title>Lesson 2e</title>
</head>

<body>

<h1>Welcome</h1>
<p>Today is Feb 15, 2011.</p>

</body>
</html>
```

The `include` makes it possible to create a single file that can be called multiple times or to create a single file where you can swap different parts in and out.

TRY IT

In this Try It, you take the content out of the Home page (`index.html`), put it in a separate file, and include it into the renamed `index.php` file. This content file goes in a new folder called content. Then you change `about.html` so that the content section is called as an include in `index.php`.

Like most websites, the header information and the footer information in the Case Study site stay the same while the content on the page changes. Using the `include` enables you to create several web pages that share a single file for the header and footer information, making changes easier.

> *You can download the code and resources for this Try It from the book's web page at* `www.wrox.com`*. You can find them in the Lesson02 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of the previous lesson. Alternatively, you can download the files from the book's website at `www.wrox.com`.

Hints

Remember to rename the files to .php.

Remember to use the localhost address to call the website locally.

Step-by-Step

1. Start with the HTML Case Study code. You can download it from the Lesson02/lesson02cs folder on the book's web page at www.wrox.com. This exercise could also be done on any simple HTML site.

2. Rename the index.html file to **index.php**.

3. Create a folder called content.

4. Create a blank file **content/home.php**.

5. Cut the following code from index.php and put it in contents/home.php:

```
<h1>Next Auction September 22nd</h1>
<p>Join us for our next auction of historic clothing
to be held at the St. Paul's Auditorium in NYC on September 22nd at
1 o'clock.</p>

<p>Lots can be viewed the prior day from 4pm until 7pm and again on Thursday
morning from 10am to noon.</p>
```

6. Replace that code in index.php with the following:

```
<?php include 'content/home.php';?>
```

7. Change the menu to look for the .php file instead of the .html file. The code should look like this:

```
<li><a href="index.php">Home</a></li>
```

8. Call up the Home page in your browser. It should still look like Figure 2-3.

9. Rename about.html to **about.php**.

10. Move about.php to the content folder.

11. Remove all the code except the code in the <div class="content"> div.

12. Check that the remaining code in content/about.php is

```
<h1>About Us</h1>
<p>We are all happy to be a part of this. Please contact any of us
with questions.</p>

<ul class="ulfancy">
<li class="row0">
```

```
      <h2>Martha Smith</h2>
      <p>Position: none<br />
      Email: martha@example.com<br />
      Phone: <br /></p>
    </li>

    <li class="row1">
      <h2>George Smith</h2>
      <p>Position: <br />
      Email: george@example.com<br />
      Phone: 515-555-1236<br /></p>
    </li>

    <li class="row0">
      <h2>Jeff Meyers</h2>
      <p>Position: hip hop expert for shure<br />
      Email: jeff@example.com<br />
      Phone: <br /></p>
    </li>

    <li class="row1">
      <h2>Peter Meyers</h2>
      <p>Position: <br />
      Email: peter@example.com<br />
      Phone: 515-555-1237<br /></p>
    </li>

    <li class="row0">
      <h2>Sally Smith</h2>
      <p>Position: <br />
      Email: sally@example.com<br />
      Phone: 515-555-1235<br /></p>
    </li>

    <li class="row1">
      <h2>Sarah Finder</h2>
      <p>Position: Lost Soul<br />
      Email: finder@a.com<br />
      Phone: 555-123-5555<br /></p>
    </li>

  </ul>
```

13. Go to the `index.php` file and change the `include` statement from `<?php include 'content/home.php';?>` to **`<?php include 'content/about.php';?>`**.

14. Change the menu to look for the `index.php` file instead of the `about.html` file. The code should look like this:

```
<li><a href="index.php">About Us</a></li>
```

15. Call up the About Us page in your browser. It should still look like Figure 2-4.

> *Both the Home page and the About Us page are called by the* index.php *file. At the moment, you need to change the* include *statement in the* index.php *file to switch between the pages. In Lesson 7 you learn how to make them change automatically.*

16. Go back to the index.php file and change the include statement from `<?php include 'content/about.php';?>` to **`<?php include 'content/home.php';?>`**.

> *Watch the video for Lesson 2 on the DVD or watch online at* www.wrox.com/ go/24phpmysql.

3

Learning PHP Syntax

You've seen how PHP can work on a web page. Now it's time to learn some basics of coding in PHP before you get into the detail of the language.

In this lesson you find out about formatting styles for PHP. You learn the general rules of PHP syntax and how to create comments. You learn the specific syntax of different PHP elements as you go over them in subsequent chapters.

Finally, you learn some best practices to make life easier and your code better.

PICKING A FORMATTING STYLE

When you read a book or type a letter you are used to certain conventions. For instance, each paragraph might have the first line indented and a space before the next paragraph. Each chapter might start with the first letter enlarged.

Styling makes the text easier to understand because it organizes the information and tells you what to expect. If there are no paragraphs, if all the sentences continue one after the other without a break, you can read it but it is more difficult.

Programming uses formatting styles in the same way. The program runs fine without using any formatting but it is more difficult for a human being to read. It is harder to see what is happening and harder to find errors.

A computer has no trouble reading this code:

```
<?php $messages='';$task=filter_input(INPUT_POST,'task',FILTER_SANITIZE_
STRING);if ($task=='product.maint'):$results=maintProduct();$a==true;$mess
ages .=$results;endif;if ('contact.maint'):$results=maintContact();$messa
ges .=$results;endif;if ('category.maint'):$results=maintCategory();$messages
.=$results;endif;?>
```

However, even if you don't know PHP, the following is much easier to understand:

```php
<?php
$messages = '';
$task = filter_input(INPUT_POST,'task', FILTER_SANITIZE_STRING);

if ($task == 'product.maint') :
    $results = maintProduct();
    $a == true;
    $messages .= $results;
endif;
if ('contact.maint')  :
$results = maintContact();
    $messages .= $results;
endif;

if ('category.maint') :
    $results = maintCategory();
    $messages .= $results;
endif;

?>
```

Formatting styles address the following issues:

➤ **Indention:** What should be indented? How big is the indentation? Do you use spaces or a tab?

➤ **Line length:** Do you restrict how long a single line can be before you use more than one line? How many characters is it?

➤ **Whitespace:** What additional whitespace (that is, spaces, new or blank rows) do you add for readability?

➤ **Use of {} (curly braces):** If these are optional in the syntax, do you use them anyway or leave them off?

Just as one book might have a different style than another, there are different PHP formatting styles. It's more important that you are consistent in your coding style than the particular style that you use.

Two popular styles are the Zend Framework (`http://framework.zend.com/manual/en/coding-standard.html`) and the Pear Coding Standards (`http://pear.php.net/manual/en/standards.php`).

LEARNING PHP SYNTAX

As you saw in the preceding lesson, the first step to using PHP is to put your PHP code in a file with the extension of `.php`. This warns the server to be on the lookout for a PHP block.

A block of PHP code starts with `<?php` and ends with `?>`. So an example code block looks like this:

```
<?php
some php code here
?>
```

You can have one PHP block that encompasses an entire file or a number of PHP blocks interspersed with HTML. If your file contains only PHP, or if it ends with a PHP block, leave off the final `?>`.

> *The final `?>` is omitted because if there is any information, including blanks or an extra line, after that final `?>`, it is interpreted as HTML and the system sends that data to the browser. This results in extraneous whitespace and possible header errors.*

> *Some servers permit you to use the short tag version of `<?` to start a PHP block. Don't do this. Your code will not be accepted on all servers and is open to misinterpretation.*

Each statement ends with a semicolon:

```
<?php
$first_name = 'Andrea';
?>
The end of a statement is often at the end of a line, but not always. This is
 particularly noticeable when mixing PHP with HTML.<p>Hello, <?php echo
$first_name; ?></p>
```

The official manual for PHP located on the php.net website is a great resource, although it can be confusing. As an example, look up information on the function `echo` that you used in the previous lesson.

Go to `www.php.net/manual/en/index.php`. You see a table of contents as shown in Figure 3-1.
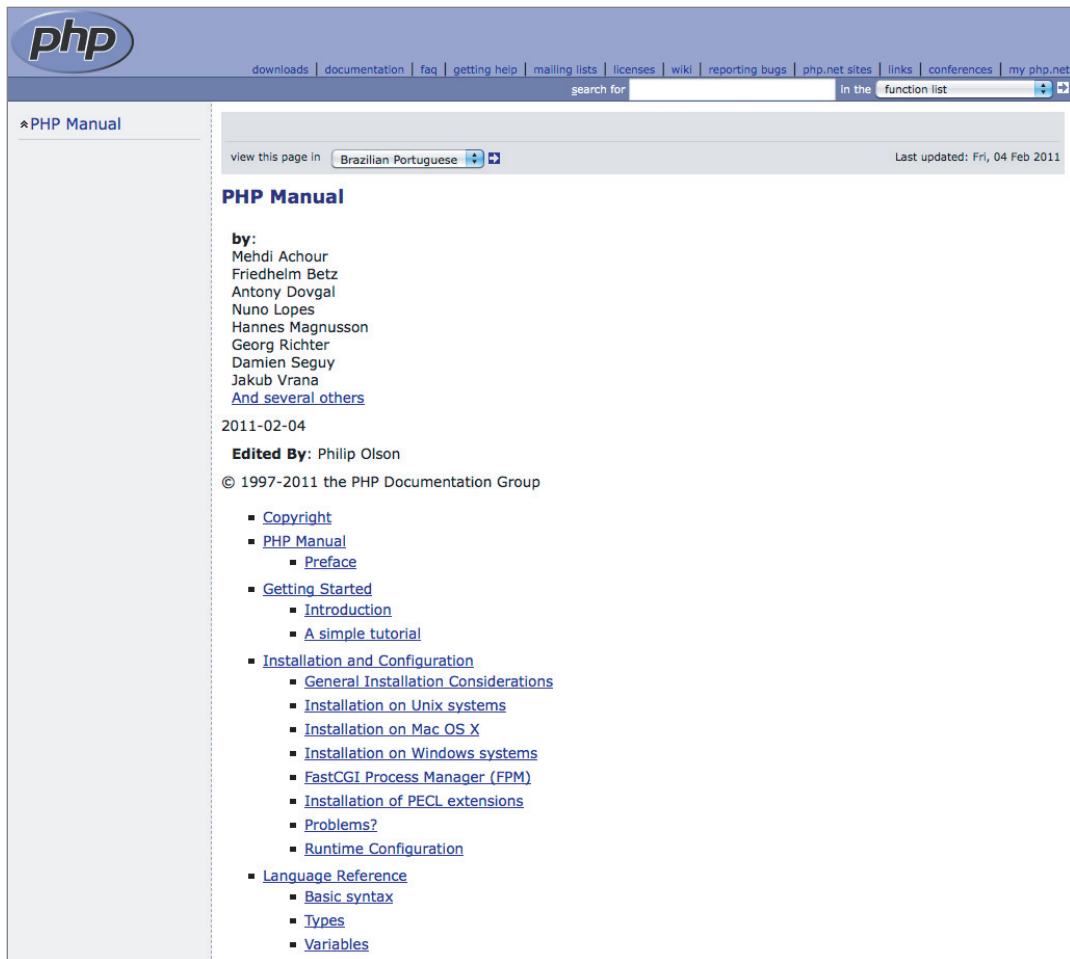
FIGURE 3-1

You could use the table of contents to locate what you are looking for, but there's a quicker way. In the upper-right corner, type `echo` in the Search For box and press Enter. Your window should look like Figure 3-2.

FIGURE 3-2

You may be thinking it's a good thing that you know what `echo` does because the explanation here doesn't help much. The following explanation should help you make more sense of it.

The first section contains a description that starts with the syntax of the element. See Figure 3-3.

```
void echo ( string $arg1 [, string $... ] )
```

FIGURE 3-3

The element is in bold text.

The keywords `void` and `string` tell you what the type is. You learn about types in Lesson 4. This is just informational and you don't type those words.

The parameters are shown in italic. Parameters are the changeable data that the function uses. Detail about the parameters is contained in the second section. You learn more about parameters in Lesson 6.

The square brackets around the second parameter indicate that the parameter is optional. The ellipsis (...) means that you can have more than one additional parameter.

The third section gives the return values. You learn more about return values in Lesson 10.

The most helpful section is often the next one: Examples. Here you can see how the function is actually used, as shown in Figure 3-4.

⊟ Examples   Report a bug

  Example #1 echo() examples

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

echo "This spans
multiple lines. The newlines will be
output as well";

echo "This spans\nmultiple lines. The newlines will be\noutput as well.";

echo "Escaping characters is done \"Like this\".";

// You can use variables inside of an echo statement
$foo = "foobar";
$bar = "barbaz";

echo "foo is $foo"; // foo is foobar

// You can also use arrays
$baz = array("value" => "foo");

echo "this is {$baz['value']} !"; // this is foo !

// Using single quotes will print the variable name, not the value
echo 'foo is $foo'; // foo is $foo

// If you are not using any other characters, you can just echo variables
echo $foo;          // foobar
echo $foo,$bar;     // foobarbarbaz

// Some people prefer passing multiple parameters to echo over concatenation.
echo 'This ', 'string ', 'was ', 'made ', 'with multiple parameters.', chr(10);
echo 'This ' . 'string ' . 'was ' . 'made ' . 'with concatenation.' . "\n";
```

FIGURE 3-4

Programmers are allowed to add notes directly to the bottom of the page. These often consist of tips and techniques in using the function.

ENTERING COMMENTS

Comments are an important part of PHP coding. Although you might remember tomorrow what you were trying to do with a certain bit of code, you probably won't remember in six months. Document what you are trying to do as you do it.

Comments are not sent on to the browser so they don't slow down your system. If you look at the source code of a web page, the only comments you see are HTML comments, not PHP comments.

There are two main types of PHP comments. The first type is for commenting a single line or partial line. The comment starts with a `//` and ends at the end of the line:

```php
<?php

// This is a single line comment
$temperature = 65;
$celsius = ($temperature - 32)* (5/9); // this is also a valid comment
echo '<p>The answer is ' . $celsius . '</p>';

// You can, of course, have multiple lines
// of single line comments.

?>
```

The second type of comment is a block comment. The block comment starts with a `/*` and ends with a `*/`:

```php
<?php

/* This is an example of using a multiple line comment. With
this type of comment you don't need to keep repeating
the comment code. */
$temperature = 65;
$celsius = ($temperature - 32)* (5/9); /* this is also a valid comment */
echo '<p>The answer is ' . $celsius . '</p>';

/* You can also use different techniques
 * to make your comments stand out
 * because everything is ignored until the */

?>
```

Note that, unlike the single-line comments, you need to indicate when the block comment ends.

There is a special type of multiple-line comment called a PHPDoc block comment. Automated tools, including Eclipse PDT, pick up comments that use this specific style.

The PHPDoc comment starts with `/**`, has a `*` at the beginning of each line, and ends with `*/`. This is an example that documents the file itself:

```
/**
 * Short description for file
 *
 * Long description for file (if any)...
 *
 * @version    1.2 2011-02-03
 * @package    Your Project Name
 * @copyright  Copyright (c) 2011 Your company name
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
```

There are specific setups for PHPDoc comments depending on what you are documenting. You learn the common blocks as you learn about the different elements in later lessons. You can find details about the PHPDoc comment at the phpDocumentor site at `http://manual` `.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_phpDocumentor` `.howto.pkg.html`.

USING BEST PRACTICES

Following best practices makes your code less error prone, easier to maintain, and more secure.

You've already learned some of the best practices earlier in this lesson:

➤ Don't use the `<? ?>` short tag.

➤ Comment your code.

➤ Be consistent within your own code but recognize that other developers may have their own styles.

➤ If you are working on a project with other developers, use the style adopted by the project rather than your personal preference.

Additionally, you should

➤ Create the ending tag, bracket, or parenthesis when you create the beginning tag, bracket, or parenthesis. You save a lot of headaches this way.

➤ Use extra lines to separate related blocks of code for improved readability.

➤ Indent nested elements.

➤ Be consistent with naming conventions and use meaningful names.

➤ Leave error reporting on while developing and turn it off when your code goes into production. This enables you to locate errors, but protects the user from strange error messages. If you are using XAMPP, error reporting is turned on by default.

TRY IT

Available for
download on
Wrox.com

In this Try It you add comments to `.php` files. You use all three types of comments.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson03 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of the previous lesson. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Remember that the comments need to be within `<?php ?>` tags.

Step-by-Step

Add comments to the given code.

1. Open Eclipse or your text editor.

2. Create a blank `.php` file called **exercise03a.php**.

3. Type in the following code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
 <title>Lesson 3</title>
</head>

<body>
 <?php
 echo '<h1>Lesson 3</h1>';
 echo '<p>This is the first paragraph</p>' ;
 echo '<p>This is the second paragraph</p>';
 ?>
</body>
</html>
```

4. To enter the PHPDoc block at the beginning, you need to go into PHP, so go to the start of the file and press Enter to get a new line.

5. Type the beginning and ending PHP tags of `<?php ?>`.Your editor may auto-complete the commands once you start typing.

6. Put your cursor between the tags and press Enter twice to make space to enter the PHPDoc block.

7. Type in the following code:

```
/**
 * Try it Lesson 3
 *
 * This program creates a php file demonstrating PHPDoc blocks
 * and basic PHP syntax
 *
 * @version    1.0 2011-02-03
 * @package    PHP & MySQL 24-hr Trainer
 * @subpackage Lesson 3
 * @copyright  Copyright (c) 2011 Your company name
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
```

8. On a new line above `echo '<h1>Lesson 3</h1>';` add the following code:

```
/* This is a long comment about
 * the h1 command
 */
```

9. On a new line above `echo '<p>This is the first paragraph</p>' ;` add the following code:

```
// This is a comment about the first paragraph
```

10. Add the following comment on the same line after `echo '<p>This is the second paragraph</p>';`:

```
// Comment about paragraph 2
```

11. Save the file.

12. In your browser enter **http://localhost/exercise03a.php**.

You should see the results shown in Figure 3-5. None of your comments show.

Add the PHPDoc blocks to the beginning of the `.php` files in the Case Study.

1. Copy the program you created in Lesson 2.

2. Open `index.php`.

3. At the beginning of the file, add `<?php ?>` tags.

4. Enter the following code between those tags. Leave the beginning and ending tags on their own lines.

```
/**
 * index.php
 *
 * Main file
```


Lesson 3

This is the first paragraph

This is the second paragraph

FIGURE 3-5

```
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
```

5. Open `content/home.php`.

6. At the beginning of the file, add **`<?php ?>`** tags.

7. Enter the following code between those tags. Leave the beginning and ending tags on their own lines.

```
/**
 * home.php
 *
 * Content for the home page
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
```

8. Open the Case Study in your browser. You should not be able to see the comments on the Home page. See Figure 3-6.

9. Open `content/about.php`.

10. At the beginning of the file, add **`<?php ?>`** tags.

11. Enter the following code between those tags. Leave the beginning and ending tags on their own lines.

```
/**
 * about.php
 *
 * Content for About Us page
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
```

12. In order to see the About Us page, you need to change the content page from `home.php` to `about.php`.

    a. Open `index.php`.

    b. Find this code: `<?php include 'content/home.php';?>`.

    c. Change `home` to **`about`**.

    d. Save the file.
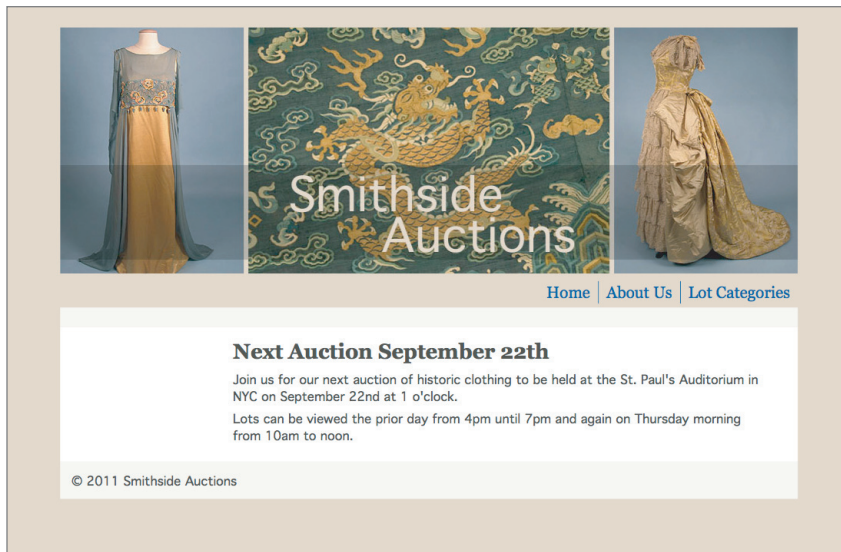
FIGURE 3-6

13. Open the Case Study in your browser. The About Us page displays. You should not be able to see the comments. See Figure 3-7.
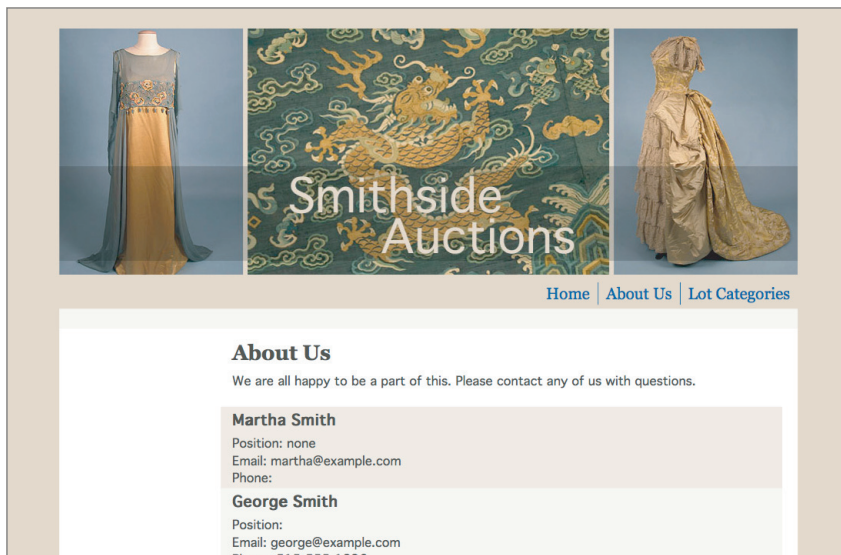
FIGURE 3-7

Watch the video for Lesson 3 on the DVD or watch online at `www.wrox.com/go/24phpmysql`.

4

Working with Variables

In this lesson you learn what variables are, how to define them, and how to use them in several ways. You learn how PHP treats text and numbers differently.

INTRODUCTION TO VARIABLES

Variables are used to store information that can change. By creating variables, you have a way to write a program that can be used again and again with different data.

Say you want to calculate the tip at a restaurant. You normally give 20 percent. You know that to calculate a 20 percent tip you multiply the cost of the dinner by .20. You then add the tip to the cost of the dinner to get the total you pay. Your formula stays the same every time you go out to eat (well, unless the service is exceptionally bad or good, of course), but the cost of the dinner is different each time. You can think of the cost of the dinner as a variable.

In PHP, variables start with a dollar sign and you assign the variable a value using the = sign:

```
$costOfDinner = 15.95;
```

The following list includes rules for naming variables (after the $):

➤ You must start variables with a letter or an underscore. By convention, underscores are used only at certain times. You learn about when to use underscores in Lesson 15. For now, always start with a letter.

➤ You can use only alphanumeric characters and underscores (a–z, A–Z, 0–9 and _).

➤ You cannot use dashes or spaces.

➤ If the variable is more than one word, you should separate the words with capitalization or underscores.

PHP is case sensitive, which means that it sees lowercase and uppercase letters as totally different characters. So $myVar and $myvar are not the same. They are two separate variables. Remember this when you are trying to troubleshoot problems.

In some programming languages you have to declare the variable ahead of time and say what type of information you are putting in the variable, such as whether it is text or numeric. In PHP you don't need to do this for simple variables. The variable takes on the type of the information you assign to it. You do need to assign a value to the variable before you can use it or you get an undefined variable error, however.

Complex variables, such as arrays and objects, should be declared. You learn about the arrays in Lesson 6 and objects in Lesson 13.

WORKING WITH TEXT

In programming, another term for text is *string*. Variables that contain text are called string variables. "Hello, world!" is a string. To assign that string to a variable, use the equal sign (=). This is actually called the *assignment operator*. Try not to think of it as an equal sign because that might cause you trouble later on. The = takes what is on the right side and uses that to set the value on the left side. This piece of code assigns a value to $myVar and then displays it.

```php
<?php
$myVar = 'Hi, my name is Andy';
echo $myVar;
?>
```

You can use either the single quote or the double quote when assigning a string value.

You can create a file and run that piece of code in your browser. You don't actually need any HTML at all. However, if you write a real program, you should wrap code that you are displaying to the browser in HTML such as this:

```php
<?php
$myVar = 'Hi, my name is Andy';
?>
<html>
<head>
    <title>Lesson 4b</title>
</head>

<body>

<h1>Welcome</h1>
<p><?php echo $myVar; ?></p>

</body>
</html>
```

> *Technically, if you want your HTML to validate, you also need a Doctype declaration. To add a Doctype, replace* <html> *with*
>
> ```
> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
> "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
> <html xmlns="http://www.w3.org/1999/xhtml">
> ```
>
> *To keep the examples simple, I am not using a Doctype declaration in most of them.*

Up to now, single and double quotes have been used interchangeably. There is a difference in how the two act. If you use double quotes, the PHP parser converts variables into their value within the string. The following code will interpret $myVar as "Hi, my name is Andy":

```php
<?php
$myName = 'Andy';
$myVar = "Hi, my name is $myName";
echo $myVar;
?>
```

If, however, you use single quotes, $myVar is interpreted as "Hi, my name is $myName." The PHP parser does not *expand* the variable; that is, it does not replace the variable with the value inside the variable.

```php
<?php
$myName = 'Andy';
$myVar = 'Hi, my name is $myName';
echo $myVar;
?>
```

Sometimes with double quotes there is some ambiguity about what exactly the parser should interpret as a variable to be translated. This can happen if text starts immediately after the variable. You can put curly braces ({}) around the variable to delineate it. The following code interprets $myVar as "There are 5 cats":

```php
<?php
$myAnimal = 'cat';
$myVar = "There are 5 {$myAnimal}s";
echo $myVar;
?>
```

It can happen that you have both single quotes and double quotes in the data that you need to quote. At that point you need to *escape* the quote. To escape something is to tell the PHP parser that the character is a data character and is not to be used as a *control character*, such as the ending quote. To escape, type a backslash (\) before the character that needs to be escaped:

```php
<?php
$saying1 = 'She said, "I didn\'t hear what you said."';
$saying2 = "She said, \"I didn't hear what you said.\"";
echo $saying1;
echo '<br />';
echo $saying2;
?>
```

Working with the Concatenation Operator

You can attach two string values together. They can either be actual strings or variables that have strings in them. This is called *concatenation*. You use a period (.) to concatenate the different strings:

```php
<?php
$myName = 'Andy';
$myVar = 'Hi, my name is ' . $myName;
echo $myVar;
?>
```

Notice that there is a space after the word `is`. All the spaces are ignored after the single quote until you get to the variable. So if you do not have the space after `is` and before the single quote, `$myVar` would be "Hi, my name isAndy." The spaces are optional around the concatenation operator and are there to make the code easier to read.

If you are concatenating two variables and need a space between them, you concatenate a string that consists of just a space:

```php
<?php
$firstName = 'Andy';
$lastName = 'Tarr';
$myVar = 'Hi, my name is ' . $firstName. ' ' . $lastName . '.';
echo $myVar;
?>
```

The last`.` is not a concatenation period, but the period at the end of the sentence. So `$myVar` is equal to "Hi, my name is Andy Tarr."

In PHP you often have different ways of doing the same thing. You could accomplish the same thing using double quotes and no concatenation:

```php
<?php
$firstName = 'Andy';
$lastName = 'Tarr';
$myVar = "Hi, my name is $firstName $lastName.";
echo $myVar;
?>
```

Notice that you don't need curly braces around `$lastName`. Even though it is immediately followed by text, a period cannot be part of a variable name so the parser knows to stop with `$lastName`. It would not hurt anything to use curly braces, however. Notice also that the space between the two variables shows in the final result.

Working with String Functions

You can think of *functions* as little programs that perform tasks for you. PHP has a number of functions for manipulating strings. Often you pass the function a variable by putting the variable in the parentheses that are suffixed to the function. In addition to performing tasks, a function can *return* a value; that is, the value of the function is the value that is returned by the function.

Functions do not start with a dollar sign and they are immediately followed by parentheses.

> For a complete list of string functions, see the PHP manual at `http://php.net/manual/en/ref.strings.php`. This also contains more detail and examples of the functions you are about to learn.

strlen()

String Length returns the length of the string. The following function returns a result of 4:

```php
<?php
$myName = 'Andy';
echo strlen($myName);
?>
```

htmlspecialchars()

HTML Special Characters takes a string and converts &, <, >, and double quotes to proper HTML entities.

```php
<?php
$myName = 'Andy & Amos';
echo htmlspecialchars($myName);
?>
```

This code displays "Andy & Amos," but if you look at the source for the browser page, you see "Andy & Amos."

ucfirst()

Upper Case First changes the first character to uppercase. The following function returns a result of "The book of days":

```php
<?php
$myVar = 'the book of days';
echo ucfirst($myVar);
?>
```

ucwords()

Upper Case Words changes the first character of each word to uppercase. The following function returns "The Book Of Days":

```php
<?php
$myVar = 'the book of days';
echo ucwords($myVar);
?>
```

trim()

Trim removes any blank characters from the beginning and end of the string. The following function returns "the book of days":

```php
<?php
$myVar = '  the book of days  ';
echo trim($myVar);
?>
```

strtolower()

String to Lower converts any uppercase letters to lowercase. The following function returns "the book of days":

```php
<?php
$myVar = 'THE BOOK OF DAYS';
echo strtolower($myVar);
?>
```

strtoupper()

String to Upper converts any lowercase letters to uppercase. The following function returns "THE BOOK OF DAYS":

```php
<?php
$myVar = 'the book of days';
echo strtoupper($myVar);
?>
```

You can nest the functions as well. The following code converts the text to lowercase and then capitalizes the first letter of each word, resulting in "The Book of Days":

```php
<?php
$myVar = 'THE BOOK OF daYs';
echo ucwords(strtolower($myVar));
?>
```

UNDERSTANDING DIFFERENT TYPES OF NUMBERS

PHP works with two different types of numbers: *integer* and *floating-point* numbers.

Integers are whole numbers. They have no decimal points and can be positive, negative, or zero.

Floating-point numbers are numbers that have decimals. Arguably one of the strangest concepts of working with numbers in computers is that after you introduce decimals you introduce rounding errors. You may not always notice it because the error may be small enough that it does not affect what you are doing. The most likely place you will get tripped up is if you try to determine if two numbers are equal and at least one of them is a floating point. The two numbers might print out looking identical, but one could be 6.599999991234 while the other is 6.599999992236.

Integers do not have the rounding issue.

When you perform arithmetic with integers, they could turn into floating-point numbers if the results require decimals.

WORKING WITH NUMBERS

You can assign values to numeric variables with the = sign. This is called the *assignment operator.* Try not to think of it as an equal sign because that might cause you trouble later on. The = takes what is on the right side and uses that to set the value on the left side. The following code sets `$result` to 2. Notice that, unlike strings, there are no quotes around the number.

```php
<?php
$result = 2;
echo $result;
?>
```

The following code sets $result to 1.45:

```php
<?php
$result = 1.45;
echo $result;
?>
```

Besides assigning a specific value in the same way that you assign a value to a text variable, numeric variables have additional ways to assign values, as shown in Table 4-1.

TABLE 4-1: Assignment Operators

OPERATOR	EXAMPLE	EQUALS
=	$a = 3;	$a = 3;
+=	$a += 3;	$a = $a + 3;
-=	$a -= 3;	$a = $a - 3;
*=	$a *= 3;	$a = $a * 3;
/=	$a /= 3;	$a = $a / 3;
%=	$a %= 3;	$a = $a % 3;

The *numeric operators* should look very familiar to you. $a is equal to 3 in Table 4-2.

TABLE 4-2: Numeric Operators

OPERATOR	DESCRIPTION	EXAMPLE	RESULT
+	Addition	$a + 2;	5
-	Subtraction	$a - 2;	3
*	Multiply	$a * 2;	6
/	Divide	$a / 2;	1.5
%	Remainder	$a%2;	1
++	Increment	$a++;	$a equals 4
--	Decrement	$a--;	$a equals 2

CHANGING BETWEEN TEXT AND NUMBERS

Various functions and operations require that the variables be of a particular type. For instance, you can multiply only numeric variables. PHP quietly converts variables to the right data type if it can. In the following code, `$stringNumber` is automatically converted to a number and 15 is displayed:

```
<?php
$stringNumber = '3';
$number = 5;
echo $stringNumber * $number;
?>
```

Here, the `$stringNumber` is converted to 15, giving 75 as the result:

```
<?php
$stringNumber = '15a4';
$number = 5;
echo $stringNumber * $number;
?>
```

And here, the `$stringNumber` is converted to 0, giving the result as 0 as well:

```
<?php
$stringNumber = 'aaa';
$number = 5;
echo $stringNumber * $number;
?>
```

The following code uses concatenation, which expects string variables. The variables are converted to strings and the result displayed is 35.

```
<?php
$stringNumber = 3;
$number = 5;
echo $stringNumber . $number;
?>
```

Here's another string function that takes a float number and displays it as a formatted string. Because it expects `$number` to be a floating-point number, it converts it from a string to a float:

```
<?php
$number = '15';
echo number_format($number, 2);
?>
```

TRY IT

In this Try It, you start changing static information in the Case Study website to variables. You work with the page that displays the gents category of lots.

You start by moving the information in the content `<div>` from the `gents.html` file into a PHP file in the content folder and then deleting the HTML page. You also change the link references to `gent.html` in `index.php`.

You create and assign values to variables for the data used on the page. You use those variables to display the data. Use the string functions to automatically change the `&` to `&` for you in the text.

Finally, you replace the odd/even row class with a calculated formula.

> 🖉 *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson04 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 3. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Remember that you moved the `home.php` and `about.php` into the content folder in Lesson 2.

The string function `htmlspecialchars()` encodes `&` characters into the proper HTML entities.

Use `number_format()` to display the price with two decimals.

Step-by-Step

Move the content in `gents.html` to the new file `content/gents.php`.

1. Move `gents.html` to the content folder. If you are using Eclipse, right-click the `gents.html` file in the list on the left, select Refactor ➪ Move, and select the content folder.

2. Rename `gents.html` to **gents.php**. If you are using Eclipse, right-click and then select Refactor ➪ Rename and change the `.html` to `.php`.

3. Open `content/gents.php`.

4. Delete all the code up to and including `<div class="content">`. Your first line is now the h1 tags.

5. Delete `</div><!—end content -->` and all code below it. Your last line is an ending `</ul>` tag.

6. Open `index.php` and change the line `<?php include 'content/about.php';?>` to **`<?php include 'content/gents.php';?>`**.

7. Open the Case Study. The first page that shows should be the Gents lots page similar to Figure 4-1.
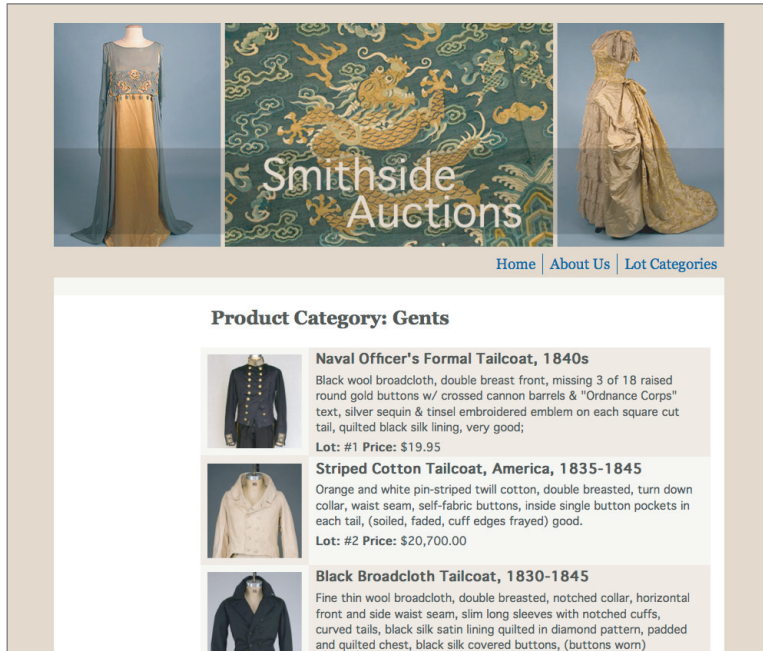
FIGURE 4-1

Change the image, name, description, price, and lot (`lot_number`) to variables and assign values to the variables.

1. Add the following variable assignments to the top of the `gents.php` file:

```php
<?php
// Get the lot information
$lot_number  = '1';
$image       = "naval-19-173.jpg";
$name        = "Naval Officer's Formal Tailcoat, 1840s";
$description = 'Black wool broadcloth, double breast front, missing
 3 of 18 raised round gold buttons w/crossed cannon barrels &
"Ordnance Corps" text, silver sequin & tinsel embroidered emblem
on each square cut tail, quilted black silk lining, very good; ';
$price       = 5700.00;
$lot_number2 = '2';
$image2      = "gents-striped-8-26.jpg";
```

```
$name2        = "Striped Cotton Tailcoat, America, 1835-1845";
$description2 = 'Orange and white pin-striped twill cotton, double
breasted, turn down collar, waist seam, self-fabric buttons, inside
single button pockets in each tail, (soiled, faded, cuff edges
frayed) good. ';
$price2       = 20700.00;
$lot_number3  = '3';
$image3       = "gents-black-8-27.jpg";
$name3        = "Black Broadcloth Tailcoat, 1830-1845";
$description3 = 'Fine thin wool broadcloth, double breasted, notched collar,
horizontal front and side waist seam,
slim long sleeves with notched cuffs, curved tails, black silk satin lining
quilted in diamond pattern, padded and quilted chest, black silk covered
buttons, (buttons worn) excellent. ';
$price3       = 3450.00;
?>
```

   a. Notice that the descriptions with single quotes have double quotes around them and descriptions with double quotes have single quotes around them. Also notice that the HTML entities & are changed to a simple &. You automatically convert these before they are output, so you don't need to manually translate them ahead of time.

   b. The commas in the prices have been removed.

2. Replace the hardcoded data with the variables.

   a. **Images:** Use the same variable for the thumbnail and the full size image with this code:

```
<div class="list-photo"><a href="images/<?php echo $image; ?>">
  <img src="images/thumbnails/<?php echo $image; ?>"  alt="" /></a>
</div>
```

   b. **Name:** Add the ucwords() string function to be sure that the first letter of each word is capitalized:

```
<h2><?php echo ucwords($name); ?></h2>
```

   c. **Description:** Because the data now might have &'s in it, use the string function htmlspecialchars() to automatically turn them into &:

```
<p><?php echo htmlspecialchars($description); ?></p>
```

   d. **Lot:** Use the variable $lot_number for the lot:

```
<p><strong>Lot:</strong> #<?php echo $lot_number; ?>
```

   e. **Price:** Use number_format() to format the price:

```
<strong>Price:</strong> $<?php echo number_format($price,2); ?></p>
```

3. Repeat step 2 for the second and third lots.

4. Save the file.

5. Open the Case Study. You should see the Gents list of lots still looking like Figure 4-1.

> *When you learn about databases you will be able to load the variables from the database. You will also be able to loop around and fill all three lots from one set of variables.*

Replace the hardcoded `row0`, `row1` classes with a calculated class in `gents.php`.

1. Initialize a variable for counting the number of rows. Put this at the end of the list of variables at the top of the file:

```
$i = 0; // for counting line number
```

2. Take the row number and divide it by 2. The remainder, either 0 or 1, is then appended to the class row. To do this, replace `<li class="row0">` with

```
<li class="row<?php echo $i % 2; ?>">
```

3. You need to add 1 to the number of rows after printing each row. Use the increment operator. The following code is added just after displaying the price:

```
<?php $i++; ?>
```

4. Repeat steps 2 and 3 for the next two rows.

5. Save the file.

6. Open the Case Study. You should see the Gents list of lots still looking like Figure 4-1.

> *Watch the video for Lesson 4 on the DVD or watch online at* `www.wrox.com/go/24phpmysql`.

5

Debugging Code

From Grace Hopper and her moth in the 1940s to the present day, programmers have been debugging to locate errors. No one writes perfect code the first time, and the better you are at finding your errors, the less frustrated you will be and the better your code will be.

Your first line of defense is using a good editor that validates syntax for you. That catches many possible errors.

In this lesson you learn techniques for locating problems and identifying common issues. You are also introduced to a debugging program you can use to facilitate your debugging.

TROUBLESHOOTING TECHNIQUES

In this section you learn when and how to display PHP errors in order to get automatic feed-back on PHP problems. You also learn what many of the common problems are and how to avoid them. Finally, you are introduced to various ways to see what is happening inside your program as it processes.

Display Errors while Developing

You want errors to be displayed for you while you are developing your code. To do this, make sure that `display_errors` is on in your `php.ini` file. You can check this by running `phpinfo()`. If you are using XAMPP, `display_errors` is on by default because it is set up for development, not production. This is the code for running `phpinfo()`:

```php
<?php
phpinfo();
```

> *In the earlier lessons, I included the ending PHP tag of `?>` to make it easier and clearer. However, it is good practice not to use the ending tag at the end of a file, so from now on I leave off the ending tag where it is not needed.*

Find the section called Core. It should look similar to Figure 5-1.

Core

PHP Version		5.3.1

Directive	Local Value	Master Value
allow_call_time_pass_reference	On	On
allow_url_fopen	On	On
allow_url_include	Off	Off
always_populate_raw_post_data	Off	Off
arg_separator.input	&	&
arg_separator.output	&	&
asp_tags	Off	Off
auto_append_file	no value	no value
auto_globals_jit	On	On
auto_prepend_file	no value	no value
browscap	no value	no value
default_charset	no value	no value
default_mimetype	text/html	text/html
define_syslog_variables	Off	Off
detect_unicode	On	On
disable_classes	no value	no value
disable_functions	no value	no value
display_errors	On	On
display_startup_errors	Off	Off
doc_root	no value	no value
docref_ext	no value	no value
docref_root	no value	no value
enable_dl	On	On
error_append_string	no value	no value
error_log	no value	no value
error_prepend_string	no value	no value
error_reporting	30711	30711
exit_on_timeout	Off	Off

FIGURE 5-1

If your `display_errors` is not on, edit the `php.ini` file so `display_errors` is on. If you are using XAMPP, the `php.ini` file is located at `c:\xampp\php` on a Windows PC and at `/Applications/XAMPP/Xamppfiles/etc` on a Mac OS X.

```
display_errors = on
```

`display_errors` determines if reported errors are displayed. `error_reporting` determines which errors should be reported. Errors have different levels, from minor notices to warning to severe. Your error reporting level is also listed on the `phpinfo()` report, but it is in machine-readable code, so it is easier to look in your `php.ini` file. A good level for developing is to show all errors except for notices. The tilde (~) prefixed to `E_NOTICE` provides direction to not show those errors.

```
error_reporting = E_ALL & ~E_NOTICE
```

If you want to make sure you get all notices, including if you have a defined variable, do not exclude the notices. This is helpful if you have typos or problems remembering the right cases.

```
error_reporting = E_ALL
```

If you want to go one step further, you can also display errors that don't meet very strict PHP standards. The pipe symbol (|) means to display if the error violates `E_ALL` or `E_STRICT`.

```
error_reporting  =  E_ALL | E_STRICT
```

If you make changes to `php.ini`, be sure to stop your Apache web server and start it again.

Take a look at what the errors look like when they are reported. Can you find the error in the following code?

```php
<?php
$firstName = 'Andy';
$nameLength = str_len($firstName);
$myVar = 'Hi, my name is ' . $nameLength. ' letters long.';
echo $myVar;
```

If you run the code, you see an error message similar to Figure 5-2. The error message starts out with the error level, which in this case is "Fatal error." It then says what the error is, which is a "Call to undefined function." Then it gives the name of the function, which is `str_len()`. Then it says what file contains the error, which is `lesson05b.php`, and finally what line the system was on when it realized there was an error, which is line 3.

You know the problem is that `str_len()` in line 3 doesn't exist. Oops. That should be `strlen()`. So you remove the underscore and successfully rerun.

(!) Fatal error: Call to undefined function str_len() in /Users/andytarr/Documents/php24/wphp24/php24/lesson05code/lesson05b.php on line 3				
Call Stack				
#	Time	Memory	Function	Location
1	0.0003	324236	{main}( )	../lesson05b.php:0

FIGURE 5-2

> *Note that the line number is when the system realizes the problem. Your actual error could be on another line. For instance, a common error is a missing semicolon, which often results in a parse error on the following line. A parse error means that the PHP parser just threw up its hands and said, "I have no idea what you are trying to tell me."*

Common Issues

Here are things to check when your code isn't working:

➤ **Typos:** Did you type what you meant to?

➤ **Missing echo:** One of the most common errors, even for experienced programmers, is forgetting to echo a variable when you need to display it. `<p>$myVar</p>` displays nothing.

➤ **Case:** PHP is case sensitive, so `$firstname` is not the same as `$firstName`.

➤ **Semicolons:** Check that you have not missed ending a statement with a semicolon.

➤ **Misplaced or missing closing braces:** You'll use parentheses, curly braces, and even square brackets. Make sure they all begin and end when you want them to and make sure you are using the right type of brace.

➤ **= versus ==:** This is assignment versus comparison. You learn about comparisons in Lesson 7. For now, realize that the question "Is x equal to y?" uses a double equal sign (==) or, sometimes, a triple equal sign (===). If you use a single equal sign for comparison, you won't get any errors but the code does not work the way you think it does. Instead of comparing, it assigns. So, in this example, x is made equal to y instead of checking to see if it is equal to y.

➤ **$ on variables:** Don't forget that variables have to start with a $ sign.

➤ **Quotes:** Check your single quotes, double quotes, nested quotes, and escaped quotes, and also your use of MySQL backticks (`) with regard to the following.

  ➤ Double quotes expand enclosed variables and single quotes do not.

  ➤ Use curly braces ({}) around variables if there is any ambiguity.

  ➤ If you have one type of quotes in your text, enclose with the other type.

  ➤ If you have both types or need to enclose with the same type of quote, escape the quote by prefixing with a backslash (\).

  ➤ When concatenating a mix of PHP and HTML, it can be helpful to replace the PHP with the values and then replace each block of PHP code one at a time.

  ➤ MySQL uses backticks (`) around database, table, and column names. You learn about this in Lesson 19.

➤ **Array number:** Arrays in PHP start counting with 0, not 1. So the third element is number 2. You learn about arrays in Lesson 6.

Seeing What's What

So how do you see what's going on inside a program before it displays to your browser?

If your web page does not look right, first check the source for the browser page. Is the HTML showing you what you expect? If it is, you have an HTML issue to fix. If it is not, you can see what PHP is actually outputting.

The `echo` command that you have been using throughout these lessons can be used to display the value of a variable. You can use it to trace where you are in a program by echoing a variable or even just echoing `here`:

```php
<?php
$number = 42;
if ($number > 2) {
  // some code here
} else {
  // some other code here
}
```

This code does something if `$number` is greater than 2 and something else if `$number` is not greater than 2. In this instance you are setting `$number` to 42 just before this statement, but pretend that that was several lines up and not so obvious. You learn about the `if` statement in Lesson 7.

You want to know what the number is just before the `if` statement and which of the two paths it follows. So you add an `echo` statement before the `if` statement to display `$number` and add an `echo` statement in each of the branches. Now when you run the program you know what is in the variable

and which path it took. Obviously you remove these after you don't need them anymore. Running the following code looks similar to Figure 5-3:

```php
<?php
$number = 42;
echo $number.'<br />';
if ($number > 2) {
  echo 'here 1';
} else {
  echo 'here 2';
}
```

FIGURE 5-3

Arrays are lists of variables that you learn about in Lesson 6. The `print_r()` function works like `echo` but is used for arrays because arrays are too complex for the `echo` statement. The syntax for print_r() is different but you can use it in the same way as the `echo` statement. The following code displays output similar to Figure 5-4:

FIGURE 5-4

```php
<?php
$myArray = array("Mon", "Tue", "Wed", "Thu", "Fri");
print_r($myArray);
```

`var_dump()` is another function that displays a variable. It also displays the complex data types such as the arrays you learn about in Lesson 6 and objects that you learn about in Section III. The following code displays output similar to Figure 5-5:

```php
<?php
$myArray = array("Mon", "Tue", "Wed", "Thu", "Fri");
var_dump($myArray);
```

If you want processing to stop when you reach a certain point, you can use the `die()` function. Notice that the following code never gets to the `echo` statement, as shown in Figure 5-6:

```php
<?php
$myArray = array("Mon", "Tue", "Wed", "Thu", "Fri");
var_dump($myArray);
die('Stop here');
echo 'We never get here.';
```

FIGURE 5-5

FIGURE 5-6

USING XDEBUG

If you are working with complex PHP programs, you will find that `echo`s and `var_dump`s can be cumbersome. At that point, you should use a debugger program. A debugger program does not tell you what is wrong with your program, but it does let you step through your code line by line so you can see where it goes. It enables you to see the value of all the variables whenever you want.

Eclipse PDT comes with two debugger programs. They can be tricky to configure but it is well worth the time to set one up. If you don't use Eclipse, your text editor may have its own debugger or you can use the methods you learned earlier in this lesson.

Configuring Xdebug

In Eclipse, go to Tools ⇨ Option (Windows) or Eclipse ⇨ Preferences (Mac). You see a window similar to Figure 5-7. You take several steps in this window.
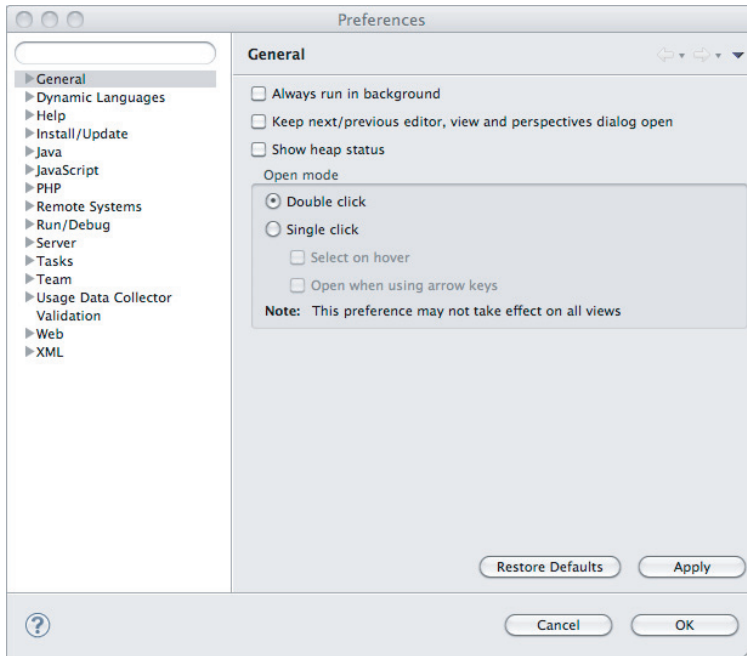
FIGURE 5-7

Go to PHP ⇨ Debug and change PHP Debugger to Xdebug as shown in Figure 5-8. Click Apply to save the change.

Click PHP ⇨ Debug ⇨ Installed Debuggers. Select each of the debuggers in turn and click the Configure link to change the ports so they match Figure 5-9. Zend Debugger is Port 10001 and Xdebug is Port 10000. Click Apply to save the change.

Go to General ⇨ Web Browser. Select the Use External Web Browser function. The window looks similar to Figure 5-10.

If you don't use `http://localhost` as your root, you need to change your PHP server. So if you didn't change your port, and you aren't using virtual hosts (if you don't know what they are, you are not using them), you can skip this next step.

To change your PHP server, go to PHP ⇨ PHP Servers. The window looks similar to Figure 5-11.

Select the Default PHP Web Server, click Edit, and change the URL to your root. If your root is `http://localhost:8080`, your resulting window looks similar to Figure 5-12. Click Apply to save the change.
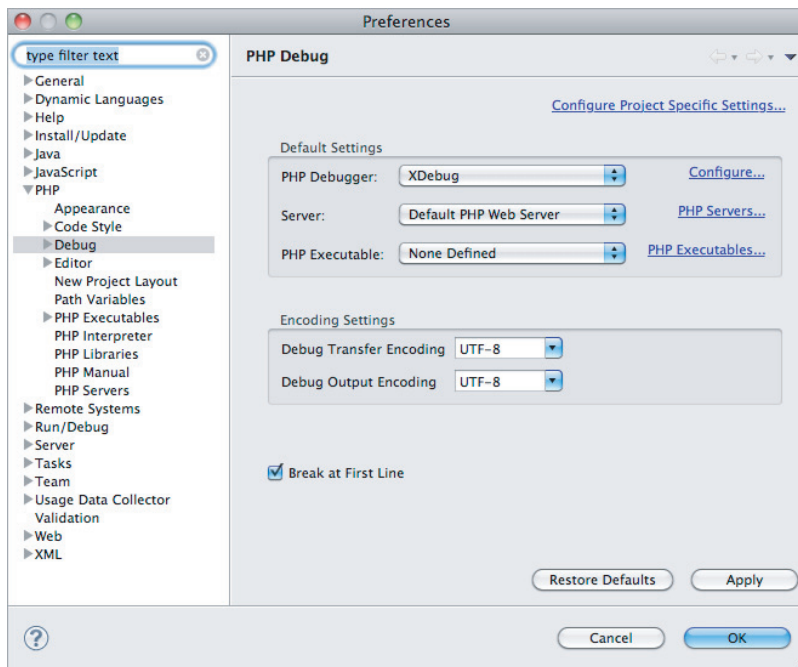
FIGURE 5-8
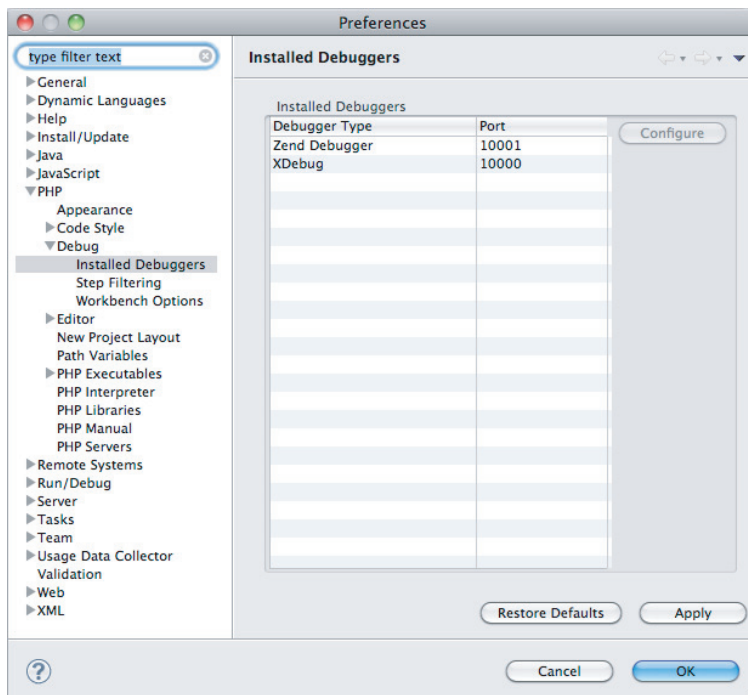
FIGURE 5-9
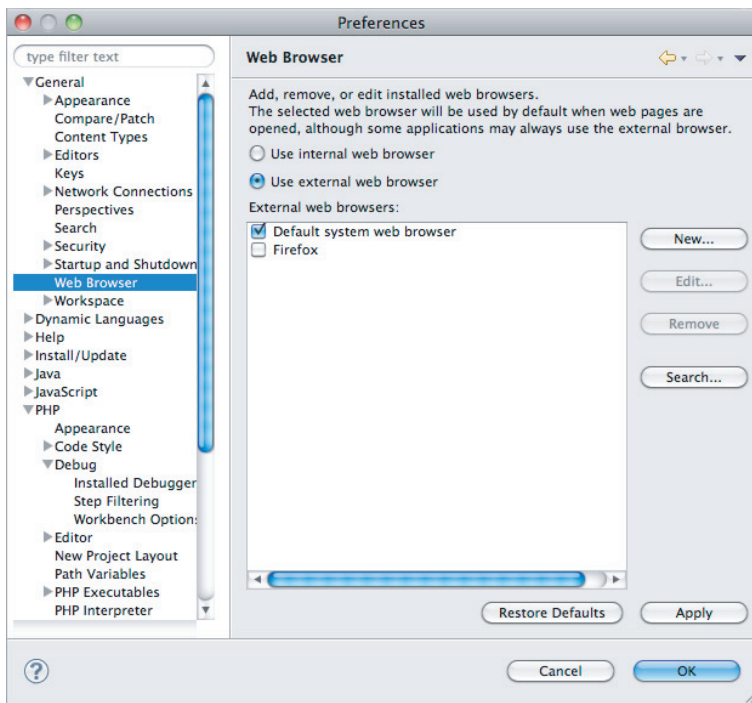
FIGURE 5-10
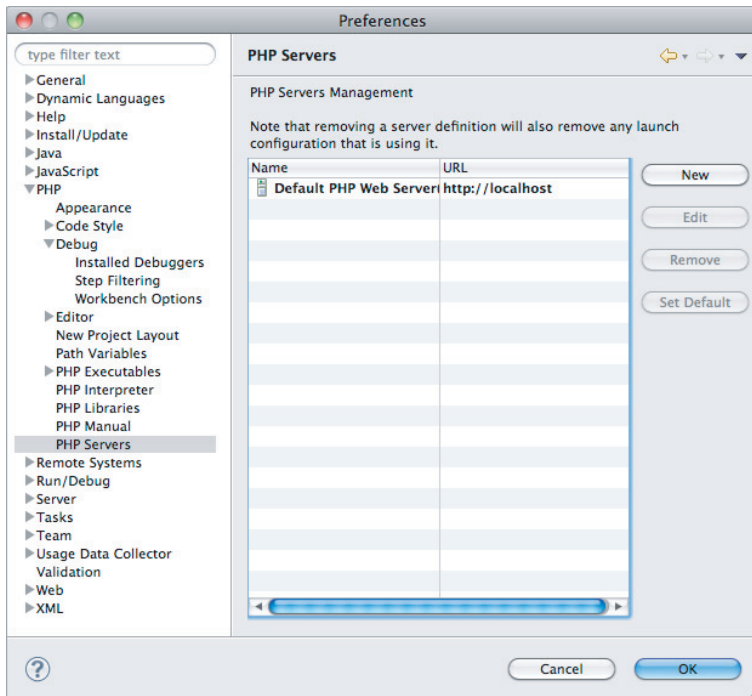
FIGURE 5-11
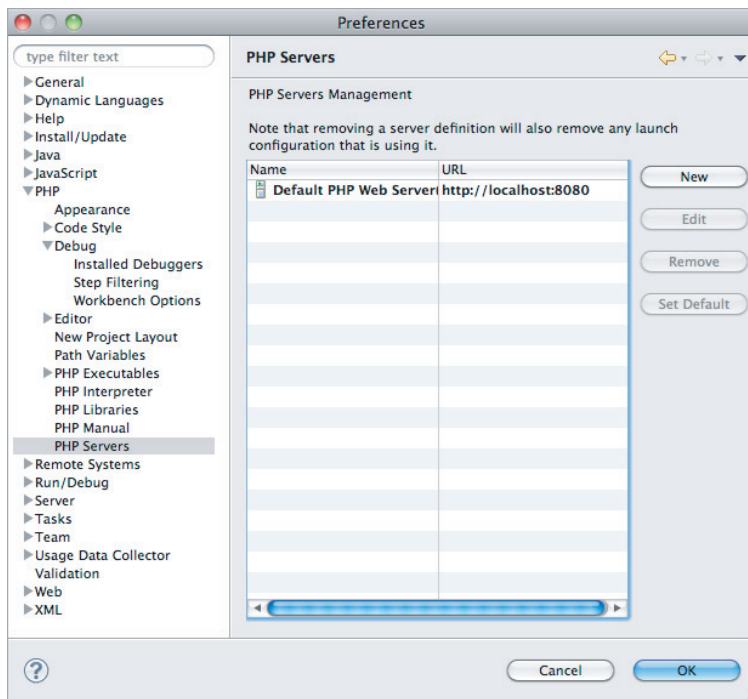
FIGURE 5-12

Click OK to exit.

The last task to configuring is to change the `php.ini` file. XAMPP sets up the debug configuration needed, though depending on the exact install you may need to change the `xdebug.remote_port`. The following code is an example of the Xdebug configuration code on the Mac:

```
;xdebug Configuration starts

zend_extension="/Applications/XAMPP/xamppfiles/lib/php/php-5.3.1/extensions/
no-debug-non-zts-20090626/xdebug.so"

xdebug.profiler_output_dir = "/tmp/xdebug/"
xdebug.profiler_enable = On
xdebug.remote_enable=On
xdebug.remote_host="localhost"
xdebug.remote_port=10000
xdebug.remote_handler="dbgp"

;xdebug Configuration ends
```

You need to exit Eclipse, stop and restart Apache, then restart Eclipse.

Using Xdebug

When you start the debugger, Xdebug stops on the first line of your program by default. From there you can choose to go to the next line or, if the line is an `include` or calls another file, you can choose to step into that file or just to the next line. You can also put breakpoints on PHP lines, which pause the program on that line. By using breakpoints you do not need to pause on every line.

To use Xdebug, you right-click the program you want to debug. If this is a program that is dependent on other programs, you start with the top-most program. Select Debug ⇨ Debug as Web Page. If you get a dialog box, confirm the starting URL. The perspective switches to the PHP Debug perspective. You see a window similar to Figure 5-13 that shows your program.
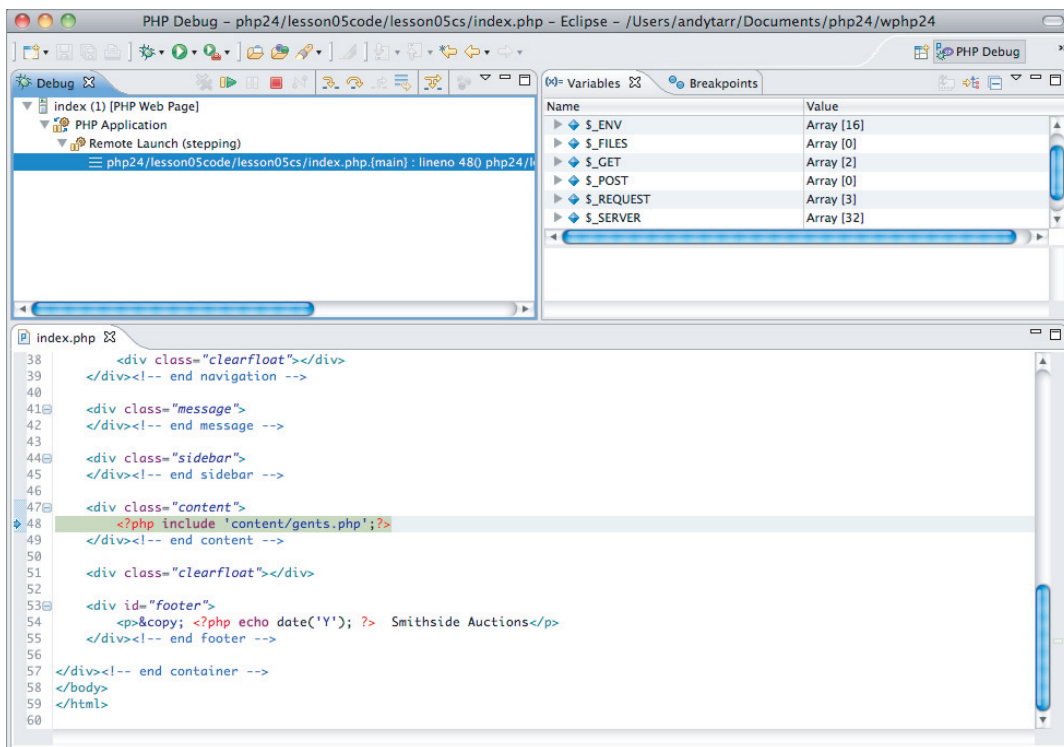
FIGURE 5-13

> *Perspectives are the different prearranged groupings of screens that show for the editor. If the PHP Debug perspective does not automatically appear, look at the upper-right corner where the name of the current perspective shows. Click the >> symbol and choose PHP Debug from the list to switch to that perspective. If the perspective does not appear in the list, go to Window ⇨ Open Perspective ⇨ Other. Click PHP Debug and click Open. It now appears in your list of perspectives.*

The main screen shows the source code for the file you are debugging with the current line highlighted.

In the upper right are the tabs that show you the variables and a list of the breakpoints. The Variable tab shows all the current variables and their values.

In the upper left you see the Debug tab. See Figure 5-14. This shows you which file you are in along with the files you went through to get there. Along the top of this window are icons that you use to control the debugger.
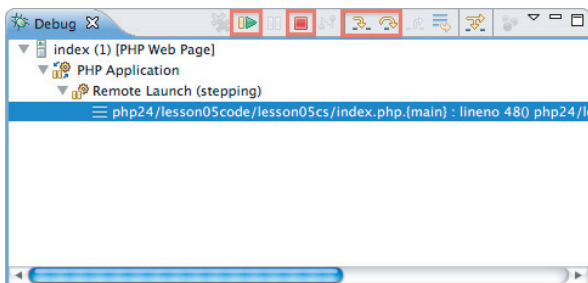
FIGURE 5-14

➤ **Resume:** This arrow tells the program to continue processing. It pauses at the next breakpoint or the end of the program.

➤ **Stop:** This square ends the debugging.

➤ **Step into:** This arrow pointing between two dashes steps into the current line. For instance, if the line you are on is an `include` statement, clicking this icon jumps you into the included file.

➤ **Step over:** The arrow pointing past the dash takes you to the next statement in the same file.

It is easy to create breakpoints. All you need to do is double-click in the column to the left of the line numbers in the file. Breakpoints are remembered even after you close a file.

TRY IT

In this Try It, you use the techniques learned in this lesson to debug a code sample. You have numerous other opportunities to debug as you complete the rest of the lessons.

You also use Xdebug to explore the Case Study files.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson05 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study you need your files from the end of Lesson 4. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Remember to go through the list of common errors.

Step-by-Step

Find the error in the following code:

```php
<?php
$firstName = 'Andy';
$nameLength = str_len($firstName);
$myVar = 'Hi, my name is ' . $namelength. ' letters long.';
echo $myVar;
```

1. If you have Notices printing, you see a message saying that `$namelength` is uninitialized. If you are not displaying Notices, you still see that you are missing the number of letters. See Figure 5-15.

#	Time	Memory	Function	Location
1	0.0006	323940	{main}( )	../exercise05a.php:0

Notice: Undefined variable: namelength in /Users/andytarr/Documents/php24/wphp24/php24/lesson05code/exercise05a.php on line *4*

Call Stack

Hi, my name is letters long.
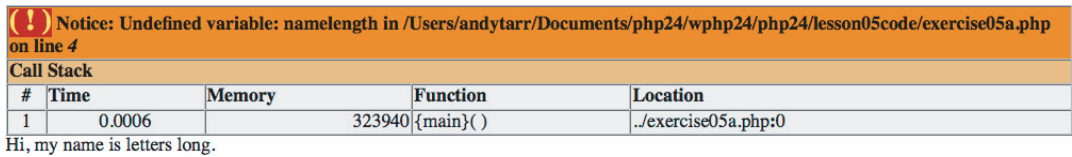
FIGURE 5-15

2. The variable that you initialized was `$nameLength` not `$namelength`. Change `$namelength` to **$nameLength** to fix.

    The following code should print out the name and number of characters. They are missing, as shown in Figure 5-16. The correct result is shown in Figure 5-17. Find the error.

```php
<?php
$firstName = 'Andy';
$nameLength = strlen($firstName);
?>
<html>
<head>
```

Welcome

My name is and it is characters long.
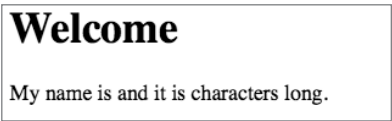
FIGURE 5-16

```
  <title>Exercise 5b</title>
</head>
<body>

<h1>Welcome</h1>
<p>My name is <?php $firstName; ?>
and it is <?php $nameLength; ?> characters long.</p>

</body>
</html>
```

3. Go through the list of the common errors. This is the corrected code:

> # Welcome
>
> My name is Andy and it is 4 characters long.

FIGURE 5-17

```
<?php
$firstName = 'Andy';
$nameLength = strlen($firstName);
?>
<html>
<head>
  <title>Exercise 5bfinal</title>
</head>
<body>

<h1>Welcome</h1>
<p>My name is <?php echo $firstName; ?>
and it is <?php echo $nameLength; ?> characters long.</p>

</body>
</html>
```

Use Xdebug to explore the Case Study files.

1. Right-click the index.php file from the Case Study.

2. Select Debug ⇨ Debug as Web Page. The program stops on the first PHP line, which happens to be the include line to the gents.php file.

3. Click the Step Into icon twice to go into that file.

4. Step Into or Step Over several times and then scroll through the Variable tab. You see the variables for those that have been given a value. Variables not assigned a value are uninitialized. See Figure 5-18.

5. Try creating a breakpoint by double-clicking the column to the left of the numbers. Remember that only lines with PHP code can be debugged.

6. Click the Resume icon to jump to the breakpoint.

7. When you are done exploring, click the red square Stop icon to terminate the debugger.

8. If the perspective does not change automatically, you can return to the PHP perspective by clicking the >> in the upper-right corner and choosing PHP.
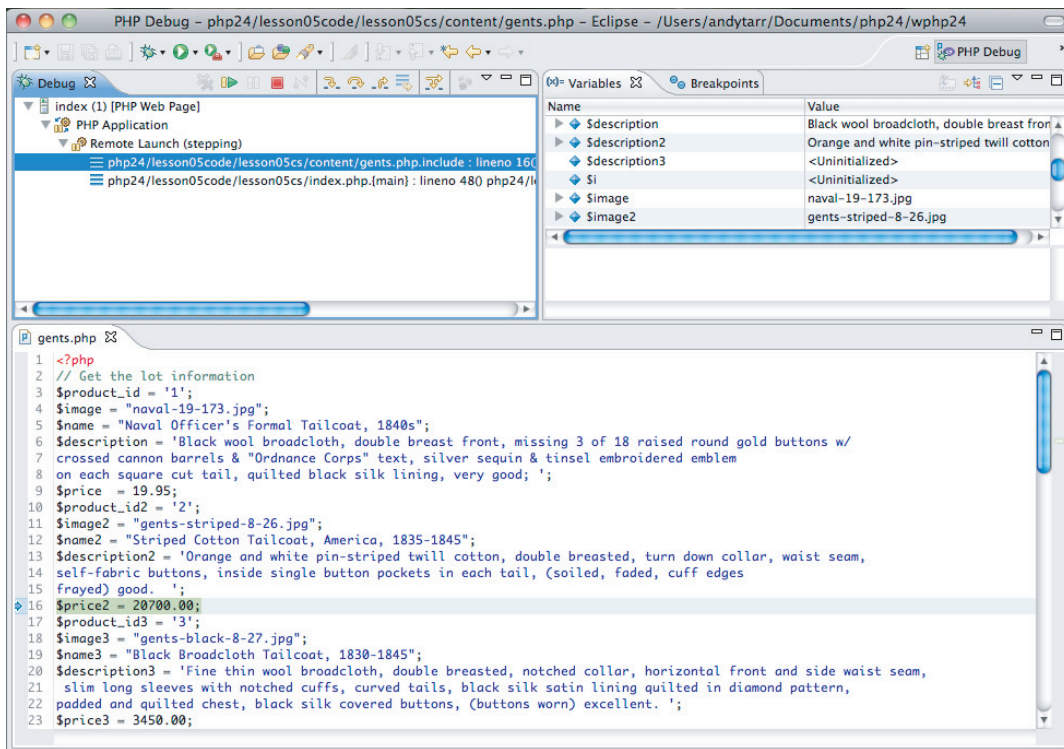
FIGURE 5-18

Watch the video for Lesson 5 on the DVD or watch online at `www.wrox.com/go/24phpmysql`.

6

Working with Complex Data

You have learned how to work with variables that store text and numbers. This works well for simple data, but what if you have a list you need to manipulate or you want to know how many days there are between January 15, 2011, and March 2, 2011?

In this lesson you learn how to use special variables that are designed for specific types of data.

WORKING WITH ARRAYS

An *array* holds multiple values in a single variable. Within one variable you have an entire list of values. You refer to and access the entire array just by the array name as you would a regular variable, or you can use indexes to access the individual values. You can even nest arrays within arrays. These nested arrays are called *multi-dimensional* arrays.

Let's display a list of employees on a web page. You could assign each employee to a regular variable and then display each of those variables. Your results look similar to Figure 6-1.

```php
<?php
// Assign Value
$employee1 = 'Sally Meyers';
$employee2 = 'George Smith';
$employee3 = 'Peter Hengel';
?>
<html>
<head>
  <title>Lesson 6a</title>
</head>

<body>

<h1>Employee List</h1>
  <p><?php echo $employee1; ?></p>
  <p><?php echo $employee2; ?></p>
  <p><?php echo $employee3; ?></p>

</body>
</html>
```


FIGURE 6-1

Now try it using an array. There are two types of arrays in PHP. The first type is a numeric array where the index is the position of the value in an array. You can assign the values in the array just by telling PHP this variable is an array and then listing the values:

```
$employee = array('Sally Meyers', 'George Smith', 'Peter Hengel');
```

Just like regular variables, you need quotes around text and not around numbers. To reference a single value, add the index in square brackets to the array variable name. Your results look like Figure 6-2.

```
<?php
$employee = array('Sally Meyers', 'George Smith', 'Peter Hengel' );
echo $employee[1];
```

Not quite what you were expecting? Unlike normal counting where you start with 1, PHP uses standard computer geek counting and starts with 0. Therefore index 1 displays the second value, which happens to be George Smith.

George Smith

FIGURE 6-2

You can also assign values using the same syntax you used to display the array element:

```
$employee[0] = 'Sally Meyers';
$employee[1] = 'George Smith';
$employee[2] = 'Peter Hengel';
```

To see what is in an array use the `print_r()` function instead of the `echo` statement you are familiar with. If you echo an array it displays the word "array" instead of the values in the array. Add the following line of code to display the values in the array `$employee`. See Figure 6-3.

```
print_r($employee);
```

Array ([0] => Sally Meyers [1] => George Smith [2] => Peter Hengel)

FIGURE 6-3

The second type of array is the associative array. In addition to being able to access the element values by their position, you can assign a name (key) to each value, which you can use to reference the element. Instead of a list of employees, make a list of information about a particular employee:

```
$employee = array('name'=>'Sally Meyers', 'position'=>'President',
                  'yearEmployed'=>2001 );
```

Alternatively, you can assign the values with the same syntax you use to display the elements:

```
$employee['name'] = 'Sally Meyers';
$employee['position'] = 'President';
$employee['yearEmployed'] = 2001;
```

Either way, if you `print_r()` the employee array you see a result similar to Figure 6-4.

```
print_r($employee);
```

Array ([name] => Sally Meyers [position] => President [yearEmployed] => 2001)

FIGURE 6-4

You can combine these two arrays into a multi-dimensional array that holds information on all the employees. Use the HTML `<pre>` tag around the `print_r()` so that the display is easier to read. You should see a window similar to Figure 6-5.

```php
<?php
$employees = array(
array('name'=>'Sally Meyers', 'position'=>'President', 'yearEmployed'=>2001 ),
array('name'=>'George Smith', 'position'=>'Treasurer', 'yearEmployed'=>2006 ),
array('name'=>'Peter Hengel', 'position'=>'Clerk', 'yearEmployed'=>1992 ),
);
?>
<pre>
<?php print_r($employees);  ?>
</pre>
```

If you want to reference a specific element, use both of the indexes. To reference Sally's position, for example, use the following code:

```php
echo $employees[0]['position'];
```

WORKING WITH LOGICAL VARIABLES

PHP has special types of variables to show simple true/false conditions and to indicate a variable with no value.

A Boolean variable value is either true or false. You expressly set a variable to true or false using TRUE or FALSE. The results of the following code are shown in Figure 6-6:

```php
<?php
$myVar1 = TRUE; // No quotes and case-insensitive
$myVar2 = FALSE;
?>
<p>True: <?php echo $myVar1; ?></p>
<p>False: <?php echo $myVar2; ?></p>
```

```
Array
(
    [0] => Array
        (
            [name] => Sally Meyers
            [position] => President
            [yearEmployed] => 2001
        )

    [1] => Array
        (
            [name] => George Smith
            [position] => Treasurer
            [yearEmployed] => 2006
        )

    [2] => Array
        (
            [name] => Peter Hengel
            [position] => Clerk
            [yearEmployed] => 1992
        )

)
```

FIGURE 6-5

```
True:1

False:
```

FIGURE 6-6

As you see from the results, a TRUE resolves to 1 and FALSE is nothing. When PHP converts a different type of variable to Boolean, the following are false:

➤ Numeric 0 or string '0'

➤ An empty string or an array with no elements

➤ A variable with no value

Everything else evaluates to true.

> If you put quotes around the TRUE or FALSE, the variable becomes a string variable and evaluates to true. So `$myVar = 'FALSE';` is the same as `$myVar = TRUE`.

PHP has a special value to represent a variable with no value. This is the null type. A variable is null if you have not assigned it to a value yet or if you assigned it to NULL. The following code displays nothing:

```php
<?php
$myVar = NULL; // No quotes and case-insensitive
echo $myVar;
```

WORKING WITH CONSTANTS

Variables are variable because they can change throughout the program. Sometimes you have a value that does not alter during the running of the program. Rather than directly using that value, you can assign it to a *constant* and use the constant instead. Constants are frequently used for configurations where different values may be assigned for different times you run the program.

In Lesson 20, you use constants to define your database name, username, and password. You can then use the constant throughout your program without having to change it if you change any of those values.

Constants use the same rules as variables for naming. They are not prefixed with a $. They are case-sensitive, but by convention constants are all uppercase. Rather than using the assignment operator (=), you use the define() function:

```php
<?php
define('DATABASE', 'mydatabase');
define('USERNAME', 'andyt');
define('PASSWORD', 's0mePassw0rd')
?>
<p>This program uses the <?php echo DATABASE;?> database with the user name
<?php echo USERNAME;?> and password <?php echo PASSWORD?>.</p>
```

As its name implies, you cannot change a constant after you have defined it. If you try to do so, you get an error message. You use the function defined() to see if the constant is already defined. defined('DATABASE') is true if the constant is already defined, and false if it is not.

WORKING WITH DATES

PHP uses *Unix timestamps* to represent dates. Unix timestamps represent a given date/time by the number of seconds since January 1, 1970. By translating dates into a number rather than actual dates, you can use ordinary math to manipulate the dates. Negative numbers show dates before January 1, 1970, but they do not work in pre-5.1.0 versions of PHP on Windows.

Time Zone Functions

It is important to be aware of time zones when dealing with dates. Your server has a time zone, which may be different from the local time zone. The following code displays your current default time zone:

```php
<?php
echo 'Current timezone: ' . date_default_timezone_get() . '<br />';
```

You can set your default time zone if it should be different. The following code changes the default time zone to that of New York City:

```php
<?php
date_default_timezone_set('America/New_York');
echo 'Current timezone: ' . date_default_timezone_get() . '<br />';
```

> *To find your time zone, check out the list of supported time zones at* `http://us2.php.net/manual/en/timezones.php`.

Your time zone may also be set in the `php.ini` file. This is a best practice. By default, XAMPP adds the UTC time zone as shown in the following code:

```
date.timezone = 'UTC'
```

You change it by supplying a different supported time zone. The following code changes the time zone to that in New York City:

```
date.timezone = 'America/New_York'
```

If you start receiving unexpected results from your dates, check what your time zone is. That could be the problem. Depending on your error reporting and your PHP version, you could receive error messages if the time zone is not set before you use the date/time functions.

Date/Time Functions

Dealing with dates and times is complex because you are dealing with arbitrary measurements. PHP has a series of functions for getting the current time, dealing with time zones, and doing calculations with dates. The following is a list of some of the common functions that you will come across.

time()

You use the `time()` function to get the current time as a timestamp. Your results for the following code are an integer such as that in Figure 6-7:

```php
<?php
echo time();
```

```
1299442037
```

FIGURE 6-7

date()

You use the `date()` function to take a timestamp and format it so it is easier to read. By default, it uses the current time. The following code displays the current date in various formats as shown in Figure 6-8. Table 6-1 has a partial listing of the format codes.

```php
<?php
echo date('c') . '<br />';
echo date('m/d/Y') . '<br />';
echo date('l, F n, Y') . '<br />';
echo date('l ga') . '<br />';
echo date('h:i a') . '<br />';
```

```
2011-03-06T15:08:58-05:00
03/06/2011
Sunday, March 3, 2011
Sunday 3pm
03:08 pm
```

FIGURE 6-8

TABLE 6-1: Date Formats for date()

FORMAT CHARACTER	DESCRIPTION
Day	
d	01 to 31
D	Mon through Sun
j	1 to 31
l (lowercase L)	Sunday through Saturday
Week	
W	Week number in the year
Month	
F	January through December
m	01 through 12
M	Jan through Dec
n	1 through 12
t	Number of days in the month
Year	
L	1 if leap year, otherwise 0
Y	Four-digit year
y	Two-digit year
Time	
a	am or pm
A	AM or PM
g	1 through 12 (hours)
G	0 through 23 (hours)
h	01 through 12 (hours)
H	00 through 23 (hours)
i	00 to 59 (minutes)
s	00 to 59 (seconds)

FORMAT CHARACTER	DESCRIPTION
Timezone	
e	Timezone Identifier (e.g., UTC, GMT)
I (Capital i)	1 for Daylight Savings Time, else 0
O	Offset from GMT in hours (e.g., +0200)
P	Offset from GMT in hours (e.g., +02:00)
T	Timezone abbreviation (e.g., EST, MDT)
Z	Timezone offset in seconds
Full Date/Time	
c	2004-02-12T15:19:21+00:00

`php.net/manual/en/function.date.php`

You can also specify the date you want to format. It needs to be in the timestamp format. This example takes the current time, adds seven days in seconds to it and then displays it formatted. See Figure 6-9 for sample output.

```php
<?php
echo '<p>Original date/time: ' . date('l, F j, Y g:ia T') . '</p>';
$myTime = time() + (60 * 60 * 24 * 7);
echo '<p>New date/time in different formats: </p>';
echo date('c', $myTime) . '<br />';
echo date('m/d/Y', $myTime) . '<br />';
echo date('l, F j, Y', $myTime) . '<br />';
echo date('l g:ia T', $myTime) . '<br />';
echo date('h:i a', $myTime) . '<br />';
```

Notice the automatic change from standard to daylight savings time. Programmers frequently leave the calculations for the seconds as shown for clarity.

Original date/time: Sunday, March 6, 2011 4:26pm EST

New date/time in different formats:

2011-03-13T17:26:01-04:00
03/13/2011
Sunday, March 13, 2011
Sunday 5:26pm EDT
05:26 pm

FIGURE 6-9

strftime()

The `strftime()` function also formats dates. It has the advantage of begin able to convert based on the locale settings. However, it does not have some of the formatting options of `date()`. The syntax is the same, but, just to make things confusing, the formatting options are different. The following code performs the same function as the previous example for `date()`. See Figure 6-10 for example results. Table 6-2 has a partial listing of format codes.

```php
<?php
echo '<p>Original date/time: ' . strftime('%A, %B %e, %Y %I:%M%p %Z') . '</p>';
$myTime = time() + (60 * 60 * 24 * 7);
```

```
echo '<p>New date/time in different formats: </p>';
echo strftime('%c', $myTime) . '<br />';
echo strftime('%m/%e/%Y', $myTime) . '<br />';
echo strftime('%A, %B %e, %Y', $myTime) . '<br />';
echo strftime('%A %I:%M%p %Z', $myTime) . '<br />';
echo strftime('%I:%M %p', $myTime) . '<br />';
```

Original date/time: Sunday, March 6, 2011 04:31PM EST

New date/time in different formats:

Sun Mar 13 17:31:56 2011
03/13/2011
Sunday, March 13, 2011
Sunday 05:31PM EDT
05:31 PM

FIGURE 6-10

TABLE 6-2: Date Formats for strftime()

FORMAT CHARACTER	DESCRIPTION
Day	
%d	01 to 31
%a	Sun through Sat
%e	1 to 31
%A	Sunday through Saturday
Week	
%U	Week number in the year
Month	
%B	January through December
%m	01 through 12
%b	Jan through Dec
Year	
%Y	Four-digit year
%y	Two-digit year
Time	
%P	am or pm

FORMAT CHARACTER	DESCRIPTION
%p	AM or PM
%I (upper case i)	01 through 12 (hours)
%H	00 through 23 (hours)
%M	00 to 59 (minutes)
%S	00 to 59 (seconds)
Timezone	
%z or %Z (depending on operating system)	Offset from GMT in hours (e.g., +0200)
%z or %Z (depending on operating system)	Timezone abbreviation (e.g., EST, MDT)
Full Date/Time	
%c	2004-02-12T15:19:21+00:00

```
php.net/manual/en/function.strftime.php
```

Depending on your technical specifications, your system may not support all of the formatting codes.

mktime()

You use `mktime()` to put a date into a timestamp so that you can use it in other date/time functions. The syntax is as follows:

```
mktime(hour, minute, second, month, day, is_dst)
```

The last parameter, `is_dst`, is depreciated and not to be used. It was for specifying daylight savings time. The following code displays the date 12/5/2011:

```php
<?php
$myDate =  mktime(0,0,0,12,5,2011);
echo date('n/j/Y', $myDate);
```

A helpful feature of `mktime()` is that it converts out-of-bounds dates to valid dates. In other words, if you specify arithmetic that, for instance, gives you 14 months, it adds one to the year and changes the months to 2. The following code displays the date 2/5/2012:

```php
<?php
$offset = 2;
echo date('n/j/Y', mktime(0,0,0,12+$offset,5,2011));
```

strtotime()

The `strtotime()` function is another way to get a Unix timestamp. With this function you translate a string into a timestamp. This is the syntax:

```
strtotime(time, now)
```

The first parameter is the text string of the date and/or time. It can be a simple date such as 12/5/2011 or a relative term such as yesterday. See Figure 6-11 for an example of results from the following code:

```php
<?php
echo date('l, F j, Y', strtotime('12/5/2011')) . '<br />';
echo date('l, F j, Y', strtotime('yesterday', strtotime('12/5/2011')))
 . '<br />';
echo date('l, F j, Y', strtotime('yesterday')) . '<br />';
echo date('l, F j, Y', strtotime('now')) . '<br />';
echo date('l, F j, Y', strtotime('Dec 5 2011')) . '<br />';
echo date('l, F j, Y', strtotime('+4 hours')) . '<br />';
echo date('l, F j, Y', strtotime('+1 week')) . '<br />';
echo date('l, F j, Y', strtotime('+2 weeks 1 day 4 hours 10 seconds'))
 . '<br />';
echo date('l, F j, Y', strtotime('next Tuesday')) . '<br />';
echo date('l, F j, Y', strtotime('last Monday'));
```

For a complete list of the terms that can be used, see www.php.net/manual/en/datetime.formats.php.

getdate()

The getdate() function takes a Unix timestamp and puts the date and time information in an array. If there is no timestamp it uses the current time. See Figure 6-12 as an example of the following code:

```php
<pre><?php print_r(getdate()); ?></pre>
```

To see full documentation on all the date/time functions, see http://www.php.net/manual/en/ref.datetime.php.


```
Monday, December 5, 2011
Sunday, December 4, 2011
Saturday, March 5, 2011
Sunday, March 6, 2011
Monday, December 5, 2011
Sunday, March 6, 2011
Sunday, March 13, 2011
Monday, March 21, 2011
Tuesday, March 8, 2011
Monday, February 28, 2011
```

FIGURE 6-11

WORKING WITH BUILT-IN FUNCTIONS

PHP has built-in functions for passing data and communicating.

$_GET

The $_GET function stores values from a form sent with the method="get" or added to the URL. You learn more about using forms in PHP in Lesson 11. This simple form displays like Figure 6-13:

```html
<form action="lesson06w.php" method="get">
  <label for="username">User Name:</label><br />
  <input type="text" id="username" name="username" /><br />
  <label for="password">Password</label><br />
  <input type="text" name="password" /><br />
  <button type="submit">Submit</button>
</form>
```



```
Array
(
    [seconds] => 5
    [minutes] => 45
    [hours] => 17
    [mday] => 6
    [wday] => 0
    [mon] => 3
    [year] => 2011
    [yday] => 64
    [weekday] => Sunday
    [month] => March
    [0] => 1299451505
)
```

FIGURE 6-12

FIGURE 6-13

When you type a username and password and click Submit, the screen looks the same, but the username and password are added to the URL address. If you type in "Andy" as the username and "12345" as the password you see `?username=andy&password=12345` added to the end of your address.

The parameter section starts with a `?` and each subsequent parameter starts with an `&`. The parameter names are taken from the `name` attribute of the input tag.

PHP reads those parameters with the `$_GET` function. `$_GET` is an associative array of the GET variables. To select the appropriate parameter, put that name surrounded by quotes in square brackets, just as you would an associative array. The following code lets you enter a username and password and then displays them. Results look similar to Figure 6-14.

```
<form action="lesson06x.php" method="get">
  <label for="username">User Name:</label><br />
  <input type="text" id="username" name="username" /><br />
  <label for="password">Password</label><br />
  <input type="text" name="password" /><br />
  <button type="submit">Submit</button>
</form>
<p>You entered <?php echo $_GET["username"] ?> as the User Name
 and <?php echo $_GET["password"] ?> as the Password.</p>
```


FIGURE 6-14

Now, obviously, you never actually use a GET for a password because everyone would be able to see it. By the same token, you should always filter the input that you receive. You learn how to filter later in this lesson.

Use the GET method when you are doing inquiries that can be repeated, if the variables are less than 2,000 characters, and the variables do not need to be private.

$_POST

The `$_POST` function stores values from a form sent with the `method="post"`. This simple form displays just like the `$_GET` did in Figure 6-13:

```
<form action="lesson06y.php" method="post">
  <label for="username">User Name:</label><br />
  <input type="text" id="username" name="username" /><br />
  <label for="password">Password</label><br />
```

```
<input type="text" name="password" /><br />
<button type="submit">Submit</button>
</form>
```

When you type a username and password and click Submit, the screen looks the same though PHP has saved them. The following code uses $_POST to retrieve the values and display them. Again, the display is the same as using the GET method shown in Figure 6-14.

```
<form action="lesson06z.php" method="post">
  <label for="username">User Name:</label><br />
  <input type="text" id="username" name="username" /><br />
  <label for="password">Password</label><br />
  <input type="text" name="password" /><br />
  <button type="submit">Submit</button>
</form>
<p>You entered <?php echo $_POST["username"] ?> as the User Name and <?php echo $_
POST["password"] ?> as the Password.</p>
```

Use the $_POST method if you are adding or updating data, doing something that should not be repeated, if the variables are more than 2,000 characters, or if the variables need to be private. As with GET, you should always filter the input that you receive. You learn how to filter later in this lesson.

Cookies

Cookies are little files of data that the server puts on the user's computer. They are often used to identify a user and retain data needed in multiple screens.

setcookie()

You create a cookie with the setcookie() command and retrieve it with the $_COOKIE function that works just like $_GET and $_POST. The syntax of the setcookie() looks like:

```
setcookie(name, value, expire, path, domain);
```

The following code creates two cookies called "username" and "password" and assigns the values "andyt" and "12345" to them. It expires in one day. The setcookie() function must be before any HTML including the <html> tag.

```
<?php
setcookie("username", "andy", time()+(60*60*24));
setcookie("password", "12345", time()+(60*60*24));
```

You can view cookies in your browser. Each browser has a different way of viewing cookies. In Firefox on the PC, go to Tools ➪ Options, select Privacy, and then select Show Cookies. On the Mac, go to Preferences, select the Privacy tab, and then click Remove Individual Cookies. The cookies are displayed by the domain name. If you have used the setup in this book, your domain is "localhost." Figures 6-15 and 6-16 show what the display looks like on a Mac with the domain of wphp24.
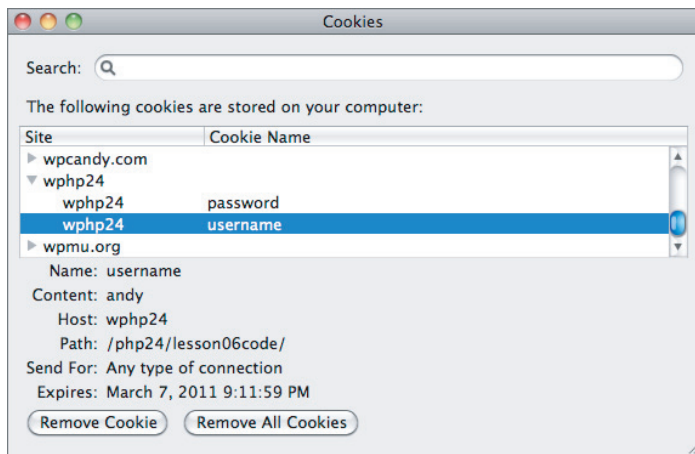
FIGURE 6-15
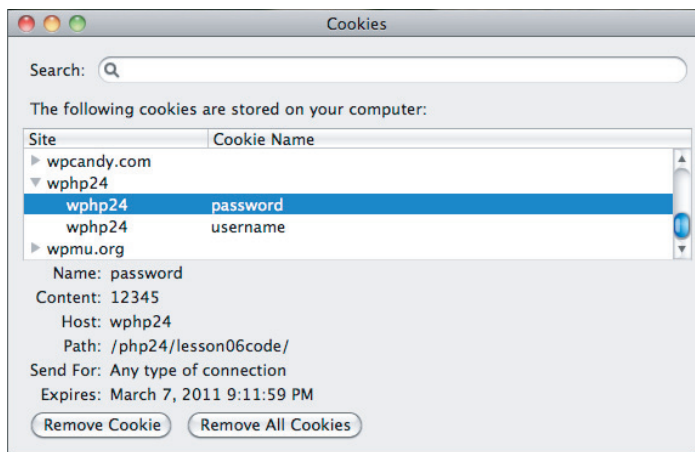
FIGURE 6-16

$_COOKIE

The following code displays the information in the two cookies with the $_COOKIE function:

```
<p>You entered <?php echo $_COOKIE["username"] ?> as the User Name
   and <?php echo $_COOKIE["password"] ?> as the Password.</p>
```

You should remember how easy it was to display the cookies in the browser when you decide what you want to save in plain text in a cookie.

filter_var()

Any time you accept input from a user or an unknown source, you need to be sure that the data is in an appropriate format both to be sure you are not using garbage and to prevent against hacking attacks. PHP has a number of filters you can use to cleanse your data. You work with the `filter_var()` function to filter the built-in functions you have just learned.

The filters can be used to either verify that you have good data (returns true or false) or to sanitize the data of particular issues (returns safe or usable data). The syntax of the `filter_var()` function is

```
filter_var(variable, filter, options)
```

Table 6-3 contains some of the most useful filters that sanitize your data by removing or changing specific characters. The function returns the sanitized data. Your original variable remains unchanged.

TABLE 6-3: Common Sanitation Filters

ID	DESCRIPTION
FILTER_SANITIZE_STRING	Strip tags, optionally strip or encode special characters
FILTER_SANITIZE_ENCODED	URL-encode string, optionally strip or encode special characters
FILTER_SANITIZE_EMAIL	Remove all characters, except letters, digits and !#$%&'*+-/=?^_`{l}~@.[]
FILTER_SANITIZE_URL	Remove all characters, except letters, digits and $-_.+!*'(),{}l\\^~[]<>#%";/?:@&=
FILTER_SANITIZE_NUMBER_INT	Remove all characters, except digits and +-
FILTER_SANITIZE_NUMBER_FLOAT	Remove all characters, except digits, +- and optionally .,eE
FILTER_SANITIZE_SPECIAL_CHARS	HTML-escape '"<>& and characters with ASCII value less than 32

Table 6-4 contains some of the most useful filters that tell you if your data is valid. The function does not make changes to your data. The function returns true or false, unless otherwise indicated.

The following code takes the $_POST data and sanitizes it before displaying the data:

```
<form action="lesson06zc.php" method="post">
  <label for="username">User Name:</label><br />
  <input type="text" id="username" name="username" /><br />
  <label for="password">Password</label><br />
```

```
    <input type="text" name="password" /><br />
    <button type="submit">Submit</button>
</form>
<p>You entered
<?php echo filter_var($_POST["username"],
FILTER_SANITIZE_STRING) ?>
 as the User Name and
<?php echo filter_var($_POST["password"],
FILTER_SANITIZE_STRING) ?>
 as the Password.</p>
```

TABLE 6-4: Common Validation Filters

ID	DESCRIPTION
FILTER_VALIDATE_INT	Validate value as integer, optionally from the specified range
FILTER_VALIDATE_BOOLEAN	Return TRUE for "1", "true", "on" and "yes", FALSE for "0", "false", "off", "no", and "", NULL otherwise
FILTER_VALIDATE_FLOAT	Validate value as float
FILTER_VALIDATE_URL	Validate value as URL, optionally with required components
FILTER_VALIDATE_EMAIL	Validate value as e-mail

Enter some invalid data, such as that shown in Figure 6-17.

FIGURE 6-17

After you click Submit, you should see that the invalid data has been stripped out, as shown in Figure 6-18.

FIGURE 6-18

WORKING WITH OBJECTS

An object is a complex type that combines variables and functions in a single unit. You learn about objects starting with Lesson 12.

⏺ TRY IT

Available for
download on
Wrox.com

In this Try It, you change the Case Study to use arrays instead of numerous individual variables in the gents.php file.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson06 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study you need your files from the end of Lesson 4. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Use a numbered array for each row. Make the variables on each row the indexes for a nested associative array.

If you have trouble with the array, you can use print_r() or var_dump to see what you have.

Step-by-Step

1. Define the array at the beginning of the assignments. This clears the array if anything is in it.

```php
<?php
// Get the lot information
$lots = array();
```

2. Change the first row of variables to an associative array within the array row [0]:

```php
$lots[0]['lot_number'] = '1';
$lots[0]['image'] = "naval-19-173.jpg";
```

```
    $lots[0]['name'] = "Naval Officer's Formal Tailcoat, 1840s";
    $lots[0]['description'] = 'Black wool broadcloth, double breast front,
    missing 3 of 18 raised round gold buttons w/crossed cannon barrels &
    "Ordnance Corps" text, silver sequin & tinsel embroidered emblem
    on each square cut tail, quilted black silk lining, very good; ';
  $lots[0]['price'] = 5700.00;
```

3. Change the second row of variables to an associative array within the array row [1]. Notice that the associative array indexes are exactly the same in this row as the first row.

```
    $lots[1]['lot_number'] = '2';
    $lots[1]['image'] = "gents-striped-8-26.jpg";
    $lots[1]['name'] = "Striped Cotton Tailcoat, America, 1835-1845";
    $lots[1]['description'] = 'Orange and white pin-striped twill cotton,
    double breasted, turn down collar, waist seam, self-fabric buttons,
    inside single button pockets in each tail, (soiled, faded, cuff edges
    frayed) good.  ';
    $lots[1]['price'] = 20700.00;
```

4. Change the third row of variables to an associative array within the array row [2]:

```
    $lots[2]['lot_number'] = '3';
    $lots[2]['image'] = "gents-black-8-27.jpg";
    $lots[2]['name'] = "Black Broadcloth Tailcoat, 1830-1845";
    $lots[2]['description'] = 'Fine thin wool broadcloth, double breasted,
      notched collar, horizontal front and side waist seam, slim long sleeves
      with notched cuffs, curved tails, black silk satin lining quilted
      in diamond pattern, padded and quilted chest, black silk covered buttons,
     (buttons worn) excellent. ';
    $lots[2]['price'] = 3450.00;
```

5. Change the first display row to use the array. Use `$i` for the array row index.

```
    <div class="list-photo"><a href="images/<?php echo $lots[$i]['image']; ?>">
      <img src="images/thumbnails/<?php echo $lots[$i]['image']; ?>"  alt="" /></a>
    </div>
    <div class="list-description">
      <h2><?php echo ucwords($lots[$i]['name']); ?></h2>
      <p><?php echo htmlspecialchars($lots[$i]['description']); ?></p>
      <p><strong>Lot:</strong> #<?php echo $lots[$i]['lot_number']; ?>
      <strong>Price:</strong> $<?php echo number_format($lots[$i]['price'],2); ?></p>
      <?php $i++; ?>
    </div>
```

6. Copy and paste the first row over rows two and three. Because you are using a numbered array, you can let `$i` do all the work for you.

Watch the video for Lesson 6 on the DVD or watch online at `www.wrox.com/go/24phpmysql`.

SECTION II
Working with PHP Controls, Functions, and Forms

▶ **LESSON 7:** Making Decisions

▶ **LESSON 8:** Repeating Program Steps

▶ **LESSON 9:** Learning about Scope

▶ **LESSON 10:** Reusing Code with Functions

▶ **LESSON 11:** Creating Forms

In this section you learn how to make a program perform different actions based on different criteria. If variables are nouns, control structures are verbs.

In Lesson 7, you learn how to set up conditional statements so that code is performed only if the conditions are met. In Lesson 8, you learn how to make the program loop, performing the same action multiple times.

You discover how local and global scope work in Lesson 9. Scope refers to where a specific variable, function, or object can be seen. You create user-defined functions in Lesson 10, and in Lesson 11 you use what you have learned so far as you process forms.

7

Making Decisions

One of the most useful aspects of a programming language is the ability to do different actions in different situations. If it rains today I use an umbrella, but if it is sunny I wear my sunglasses. When the light is red I stop; when it is green I go; and when it is yellow I go fast.

In this lesson you learn how to use various conditional statements to make the program perform differently depending on the situation.

IF/ELSE

The most recognizable conditional statement is the `if` statement. You use the `if` statement to tell the program to execute some code if a given condition is true. Optionally, you can add an `else` statement to tell the program what to do if the condition is not true.

Basic If Statements

The most basic `if` statement consists of the condition that is being evaluated along with the code to be executed enclosed in curly braces This is how you write a basic `if` statement:

```
if (some condition) {
    some lines of PHP code here;
    that are performed;
    if the condition evaluated to true;
}
```

> ⊗ *If you have only a single line of code, you can leave off the curly braces. However, this can cause errors if you later add another line and do not add the curly braces.*

Following is an example of how you could report on a rainy day. The results look similar to Figure 7-1.

```html
<html>
<head>
  <title>Lesson 7a</title>
</head>
<body>
  <h1>Weather Report</h1>
  <?php
  $weather = 'rainy';
  if ($weather == 'rainy') {
    echo "<p>It will be rainy today. Use your umbrella.</p>";
  }
  ?>
</body>
</html>
```

Weather Report

It will be rainy today. Use your umbrella.

FIGURE 7-1

Notice that when you check to see if the weather is rainy, you use a double equal sign (==) and that all the PHP statements still end with a semicolon, although the `if` statement itself does not have a semicolon.

> *It is a common error to use a single equal sign in a conditional statement. The single equal sign (=) is the assignment operator. It takes whatever is on the right side and makes the left side equal to it. That is usually not what you want to do. The double equal sign (==) is the equal comparison operator, which determines if the left and right sides are equal. There is also the triple equal sign (===), which checks that the two sides are even more equal. You learn about that later in this lesson.*
>
> *In a few instances you could use the single equal sign in a conditional statement, but the best practice is to avoid it completely because of the ambiguity of whether or not it is there in error.*

Now, of course, it is always rainy with that code. If you make the day sunny, nothing displays because that code is never executed. If you want to have code execute if the condition is not met, you add an `else` statement. `else` statements never live on their own. They must always follow an `if` statement. The following code shows the syntax of an `if` statement with an `else` statement.

```
if (some condition) {
  some lines of PHP code here;
  that are performed;
```

```
    if the condition evaluated to true;
  } else {
    some lines of PHP code here;
    that are performed;
    if the condition did not evaluate to true;
  }
```

So now the code performs some action whatever the weather. If the weather is rainy then the output tells you to use your umbrella. Otherwise it tells you to wear your sunglasses. Because the variable weather is set to sunny, you are told to wear your sunglasses. See Figure 7-2.

```
<html>
<head>
  <title>Lesson 7b</title>
</head>
<body>
<h1>Weather Report</h1>
  <?php
  $weather = 'sunny';
  if ($weather == 'rainy') {
    echo "<p>It will be rainy today. Use your umbrella.</p>";
  } else {
    echo "<p>It will be sunny today. Wear your sunglasses.</p>";
  }
  ?>
</body>
</html>
```


Weather Report

It will be sunny today. Wear your sunglasses.

FIGURE 7-2

You can add a condition on the else statement by turning it into an elseif statement. In the following code, if it is not rainy, you check to see if it is actually sunny before you decide to wear your sunglasses. After you know it is sunny, you perform your actions and jump to the bottom of the if statement. See Figure 7-3.

```
<html>
<head>
  <title>Lesson 7c</title>
</head>
<body>
<h1>Weather Report</h1>
  <?php
  $weather = 'sunny';
  if ($weather == 'rainy') {
    echo "<p>It will be rainy today. Use your umbrella.</p>";
  } elseif ($weather == 'sunny') {
    echo "<p>It will be sunny today. Wear your sunglasses.</p>";
  } elseif ($weather == 'snowy') {
```

```
        echo "<p>It will be snowy today. Bring your shovel.</p>";
    } else {
        echo "<p>I don't know what the weather is doing today.</p>";
    }
    ?>
</body>
</html>
```

Weather Report

It will be sunny today. Wear your sunglasses.

FIGURE 7-3

Comparison Operators for If/Else Statements

So far you have just been using the equal *comparison operator*, but conditional statements are really checking to see if a statement evaluates to true, not if something is equal. You can also check to see if something is not equal to something, less than something else, more than something, and so on. Strings that consist of numbers are converted to numeric before the test except for the identical ===. Table 7-1 has a list of comparison operators.

TABLE 7-1: Comparison Operators

OPERATOR	DESCRIPTION	EXAMPLE
==	Is equal to	6=='6' returns true
===	Is identical to (including the type cast)	6==='6' returns false
!=	Is not equal to	6!=5 returns true
<>	Is not equal to	6<>5 returns true
<	Is less than	6<5 returns false
>	Is greater than	6>5 returns true
<=	Is less than or equal to	6<=5 returns false
>=	Is greater than or equal to	6>=5 returns true

Some if statements can become complex as you perform actions within the conditional statement itself. Because the conditional statement is testing for "true" and anything that is not false is true, you have a wide range of conditions that you can test for. Refer to Lesson 6 for a discussion on what is true and false.

TRUE AND FALSE

As a reminder, the following are false:

➤ Numeric 0 or string '0'.

➤ An empty string or an array with no elements.

➤ A variable with no value. Undeclared variables or variables set to NULL have no value.

Everything else is true.

The following code gives some examples of the different ways that you use if statements. See Figure 7-4 for the results.

```
<html>
<head>
  <title>Lesson 7d</title>
</head>
<body>
  <h1>If Statements</h1>
  <?php
  $a = 5;
  $b = 8;
  $c = 58;
  $d = 40;
  $date = date('D, M d, Y');
  if ($d < 50) {

    if ($a >= strlen($data)) {
      echo "<p>Position 1: $a</p>";
    }

    echo "<p>Position 2: $d</p>";

    if (($d + $b) < ($c - $a)) {
      echo '<p>Position 3: ' . ($d + $b) .'</p>';
    } else {
      echo '<p>Position 4: ' . ($c - $a) . '</p>';
    }

  }

  ?>
  <div>
    <?php if ($date) { ?>
      <p>Today is <?php echo $date; ?>.</p>
      <p>You can check to see if there's something
```

```
       in a variable before you print it.<p>
       <p>You can also jump in and out of PHP in the
       middle of an if statement.</p>
     <?php } ?>
   </div>
 </body>
 </html>
```

If Statements

Position 1: 5

Position 2: 40

Position 3: 48

Today is Tue, Mar 08, 2011.

You can check to see if there's something in a variable before you print it.

You can also jump in and out of PHP in the middle of an if statement.

FIGURE 7-4

If/Else with Ternary Operator

You can use a shortcut syntax with simple if-then-else statements that can make your code easier to read, especially if you are interspersing it in HTML code. The following code does a simple test for gender:

```
<?php
$gender = 'F';
if ($gender == 'M') {
  echo 'Man';
} else {
  echo 'Woman';
}
```

This same code can be written using the *ternary operator* (?) as:

```
<?php
$gender = 'F';
echo ($gender == 'M') ? 'Man' : 'Woman';
```

Ternary stands for three parts. This statement consists of the condition in parentheses followed by the ternary operator (the question mark), then the result-if-the-condition-is-true separated with a colon from the result-if-the-condition-is-false. Notice that there is no `if` in the ternary `if` statement at all as can be seen in the previous statement and in the following syntax statement:

```
(condition) ? whentrue : whenfalse;
```

This shorthand version of if/else is useful as well when assigning defaults. In the following code, if a variable is empty, you give it a value; otherwise you use whatever is in the variable:
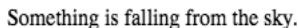
```php
<?php
echo (empty($_GET['task'])) ? 'home' : $_GET['task'];
```

LOGICAL OPERATORS

Sometimes you need to look at multiple conditions before you decide to execute a block of code. The following example displays similar to Figure 7-5:

```php
$weather = 'sleeting';

if ($weather == 'snowing') {
  echo '<p>Something is falling from the sky.</p>';
} elseif ($weather == 'sleeting') {
    echo '<p>Something is falling from the sky.</p>';
} elseif ($weather == 'raining') {
    echo '<p>Something is falling from the sky.</p>';
} elseif ($weather == 'sunny') {
    echo '<p>I need my sunglasses.</p>';
} elseif ($weather == 'partly sunny') {
    echo '<p>I need my sunglasses.</p>';
} else {
    echo '<p>The weather is ' . $weather .'.</p>';
}
```

Something is falling from the sky.

FIGURE 7-5

The same actions are performed whether it is snowing, sleeting, or raining. Imagine that that code block was 20 lines long. That is a lot of extra code, not to mention there is a greater chance for errors and more work to maintain it.

Instead of the multiple `if` statements, you can link together multiple conditions. In this case, you want to say that something is falling from the sky if it is snowing, sleeting, or raining. Here is how that works. The results are shown in Figure 7-6.

```php
<?php
$weather = 'sleeting';

if ($weather == 'snowing' OR $weather == 'sleeting' OR $weather == 'raining') {
  echo '<p>Something is falling from the sky.</p>';
} elseif ($weather == 'sunny' OR $weather == 'partly sunny') {
    echo '<p>I need my sunglasses.</p>';
```

```
} else {
    echo '<p>The weather is ' . $weather .'.</p>';
}
```

Something is falling from the sky.

FIGURE 7-6

When using the OR operator, the condition is true if any of the conditions are true. The OR operator is case insensitive and can also be written as a ||. This vertical bar symbol is called a *double pipe* symbol. This code is identical to the preceding code as you can see in Figure 7-7:

```
<?php
$weather = 'sleeting';

if ($weather == 'snowing' || $weather == 'sleeting' || $weather == 'raining') {
  echo '<p>Something is falling from the sky.</p>';
} elseif ($weather == 'sunny' || $weather == 'partly sunny') {
    echo '<p>I need my sunglasses.</p>';
} else {
    echo '<p>The weather is ' . $weather .'.</p>';
}
```

Something is falling from the sky.

FIGURE 7-7

In the same way, you use the AND operator when you need all conditions to be true before executing a block of code. This code produces output as shown in Figure 7-8:

```
<?php
$author = 'Dodie Smith';
$title = 'I Capture the Castle';

if ($author == 'Dodie Smith' AND $title == 'I Capture the Castle') {
  echo "<p>Found the book.</p>";
} else {
  echo "<p>$title by $author is the wrong book.</p>";
}
```

Found the book.

FIGURE 7-8

Instead of using AND you can use the symbol &&, which gives you the same result as the previous example, as shown in Figure 7-9.

```php
<?php
$author = 'Dodie Smith';
$title = 'I Capture the Castle';

if ($author == 'Dodie Smith' && $title == 'I Capture the Castle') {
  echo "<p>Found the book.</p>";
} else {
  echo "<p>$title by $author is the wrong book.</p>";
}
```
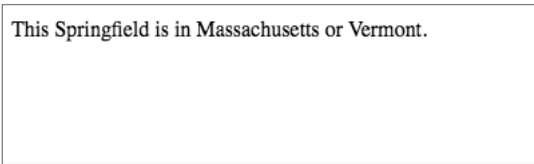
Found the book.

FIGURE 7-9

You can mix AND and OR as needed as shown here. See Figure 7-10.

```php
<?php
$city = 'Springfield';
$state = 'MA';

if ($city == 'Springfield' AND ($state == 'MA' OR $state =='VT')) {
  echo "<p>This Springfield is in Massachusetts or Vermont.</p>";
} else {
  echo "<p>$city, $state is not Springfield in MA or VT.</p>";
}
```
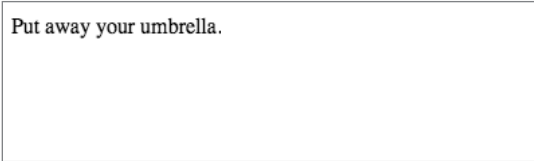
This Springfield is in Massachusetts or Vermont.

FIGURE 7-10

There is a slight difference between AND/OR and &&/||. The words AND/OR have a lower *operator pre-cedence* than the symbols &&/||, which for practical purposes means that you do not want to mix the word version and the symbol version in the same statement or you could get unexpected results. Just as in regular mathematics, you can use parentheses to change the precedence order, to improve readability, or when there is any ambiguity.

The final logical operator is the not (!) operator. Use this to negate. The following code tells you to put away your umbrella if it's not rainy, otherwise it tells you it is raining. See Figure 7-11.

```php
<?php
$weather = 'sunny';
if (!($weather == 'rainy')) {
  echo "<p>Put away your umbrella.</p>";
} else {
  echo "<p>It's raining!</p>";
}
```


FIGURE 7-11

SWITCH STATEMENTS

switch statements are also used to perform specific blocks of code depending on conditions. They are used when you want to compare the same variable or expression to several different values. Take, for example, the following series of if statements:

```php
<html>
<head>
  <title>Lesson 7m</title>
</head>
<body>
  <h1>Weather Report</h1>
  <?php
  $weather = 'sunny';
  if ($weather == 'rainy') {
    echo "<p>It will be rainy today. Use your umbrella.</p>";
  } elseif ($weather == 'sunny') {
    echo "<p>It will be sunny today. Wear your sunglasses.</p>";
  } elseif ($weather == 'snowy') {
    echo "<p>It will be snowy today. Bring your shovel.</p>";
  } else {
    echo "<p>I don't know what the weather is doing today.</p>";
  }
  ?>
</body>
</html>
```

Each if statement is comparing to the variable $weather. With a switch statement you write the same code in a clearer fashion. The following code performs the same functions:

```
<html>
<head>
  <title>Lesson 7n</title>
</head>
<body>
  <h1>Weather Report</h1>
  <?php
  $weather = 'sunny';

  switch ($weather) {
  case 'rainy':
    echo "<p>It will be rainy today. Use your umbrella.</p>";
    break;
  case 'sunny':
    echo "<p>It will be sunny today. Wear your sunglasses.</p>";
    break;
  case 'snowy':
    echo "<p>It will be snowy today. Bring your shovel.</p>";
    break;
  default:
    echo "<p>I don't know what the weather is doing today.</p>";
  }
  ?>
</body>
</html>
```

The syntax of the switch statement is as follows:

```
switch ($variable) {
case value1:
  code to be executed if $variable is equal to value1;
  break;
case value2:
  code to be executed if $variable is equal to value2;
  break;
default:
  code to be executed if $variable is different from both value1 and value2;
}
```

The switch ($variable) establishes what is compared. Each of the case statements lists a value. Notice that the case statements end with a colon. This is the normal practice, though a semicolon works as well. If the variable in the switch statement matches the value in the case statement, the associated block of code is performed. The break; tells the program to jump to the end of the switch block, ignoring all the rest of the case statements. If you want to perform some action when the value matches and then get out, use the break;. The default statement at the end is always performed unless an earlier matching case contained a break.

If you want to have two or more values perform the same actions, skip the break on the first value. It then drops through and executes the lines (regardless of matching any subsequent case

statements) until it finds a `break`. In the following code the state is `MA`, so after it gets a match, it displays "Southern New England" because it continues until it finds a `break;`.

```php
<?php
$state = 'MA';

switch ($state) {
case 'ME':
case 'VT':
case 'NH':
  echo "<p>Northern New England</p>";
  break;
case 'CT':
case 'MA':
case 'RI':
  echo "<p>Southern New England</p>";
  break;
default:
  echo "<p>$state is not in New England.</p>";
}
```

ALTERNATIVE SYNTAX

There is an alternative syntax for the control structures: the `if` and `switch` statements that you already know as well as the control statements you learn in the next lesson.

Curly braces have a tendency to get lost in long stretches of code, especially if you are hopping in and out of PHP and HTML. You often end up with lines that look like this:

```php
<?php } ?>
```

This alternative syntax replaces the curly braces with a colon and an end word. The syntax for an `if` statement is as follows:

```php
if (condition) :
  some lines of PHP code here;
  that are performed;
  if the condition evaluated to true;
elseif (condition) :
  some lines of PHP code here;
else :
  some lines of PHP code here;
endif;
```

The `switch` statement uses this syntax:

```php
switch ($variable) :
case value1:
  code to be executed if $variable is equal to value1;
  break;
case value2:
  code to be executed if $variable is equal to value2;
```

```
      break;
  default:
      code to be executed if $variable is different from both value1 and value2;
  endswitch;
```

The recommendation is to use this syntax if you are mixing HTML and PHP on a page and to use the curly brace syntax if you are doing straight PHP.

TRY IT

In this Try It, you finally put the case study back together so that the menus display the right content. You use the GET parameters in the URL that you learned to manipulate in Lesson 6 along with the conditional statements that you learned in this lesson to create a dynamic menuing system that tells the program what content to display.

> You can download the code and resources for this Try It from the book's web page at www.wrox.com. You can find them in the Lesson07 folder in the download. You will find code for both before and after completing the exercises.

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the case study, you need your files from the end of Lesson 6. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Lesson 6 teaches how to work with GET values.

Remember the shortcut if/else statement for setting up a default:

```
$variable = (empty($variable)) ? "default" : $variable;
```

Step-by-Step

The first task is to create a .php file in the content folder that contains the sidebar menu in the sporting.html file.

1. Create a blank file called **catnav.php** in the content folder. If `<?php` is automatically entered in the file, remove it.

2. Copy the code inside the sidebar div from the `sporting.html` file, paste it into the new `catnav.php` file, and save the file:

```
<h3 class="element-invisible">Lot Categories</h3>
  <ul class="catnav">
    <li><a href="gents.html">Gents</a></li>
    <li><a href="sporting.html">Sporting</a></li>
    <li><a href="women.html">Women</a></li>
  </ul>
```

The next task you need to do is to take the remaining HTML pages (`categories.html`, `sporting .html`, and `women.html`) and turn them into PHP content files. This process is described in more detail in the Try It sections of Lessons 2 and 4. If you want to skip this step, you can download the lesson07cs01 folder for the completed files.

1. Move `categories.html`, `sporting.html`, and `women.html` to the `content` folder.

2. Change the extensions to **.php**.

3. Delete everything in the files except the code inside the div `class="content"`.

4. Save the files.

Add a parameter called `content` to each of the menu options in the `index.php` mainnav menu.

1. Open `index.php`.

2. Find the `ul` with the `class="mainnav"`.

3. Change the `href` for each of the items to **index.php?content=*contentname*** where ***contentname*** is the name of the file in the `content` folder without the extension. This passes a GET parameter that you use in the next step. This is standard HTML, not PHP. The code should look like this:

```
<ul class="mainnav">
  <li><a href="index.php?content=categories">Lot Categories</a></li>
  <li><a href="index.php?content=about">About Us</a></li>
  <li><a href="index.php?content=home">Home</a></li>
</ul>
```

Change the include statement in the content div to `include` based on the GET parameter.

1. Find the div with `class="content"` in the `index.php` file.

2. Between the `<?php` and the `include` statement, get the content from the URL if there is one. The `isset()` function checks to see if the variable exists.

```
<?php
$content = '';
// Get the content from the url
if (isset($_GET['content'])) :
$content = $_GET['content'];
```

3. Someone could enter malicious data into a GET parameter, so sanitize the data before using it:

```
// Sanitize it for security reasons
$content = filter_var($content, FILTER_SANITIZE_STRING);
endif;
```

4. Set up the home page as a default if there is no GET parameter:

```
// Set up the home page as the default
$content = (empty($content)) ? "home" : $content;
```

5. Change the include to use the $content variable:

```
// Include the chosen page
include 'content/' . $content . '.php';
?>
```

6. Your links in the main menu should now work to take you to the Home page, the About Us page, and the Lot Categories page.

Now you change the categories.php file to also use the GET parameters on its links to the lots pages.

1. Open contents/categories.php.

2. Find the h2 tag containing the link to the Gents page and change it as you did the mainnav menu:

```
<h2><a href="index.php?content=gents">Gents</a></h2>
```

3. Find the a tag with class="button display" containing the link to the Gents page and change it the same way:

```
<a class="button display" href="index.php?content=gents">Display Lots</a>
```

4. Find the h2 tag containing the link to the Sporting page and change it the same way:

```
<h2><a href="index.php?content=sporting">Sporting</a></h2>
```

5. Find the a tag with class="button display" containing the link to the Sporting page and change it the same way:

```
<a class="button display" href="index.php?content=sporting">Display Lots</a>
```

6. Find the h2 tag containing the link to the Women page and change it the same way:

```
<h2><a href="index.php?content=women">Women</a></h2>
```

7. Find the a tag with class="button display" containing the link to the Women page and change it the same way:

```
<a class="button display" href="index.php?content=women">Display Lots</a>
```

8. Save the file and you should now be able to link to the Gents, Sporting, and Women's pages from the Lot Categories page either by clicking the titles or by clicking the Display Lots buttons.

Now you add these changes to the catnav menu and add the menu to show on the lots pages.

1. Open the `content/catnav.php` file.

2. Change the `href` link to match the links from the previous steps and then save the file:

```
<li><a href="index.php?content=gents">Gents</a></li>
<li><a href="index.php?content=sporting">Sporting</a></li>
<li><a href="index.php?content=women">Women</a></li>
```

3. Open the `index.php` file.

4. Find the div where `class="sidebar"`.

5. Include the content for the catnav if the page is Gents, Sporting, or Women and save the file. This example uses a `switch` statement, but you could use multiple `if` statements here instead.

```
<div class="sidebar">
  <?php
  switch (isset($_GET['content'])) :
  case 'gents' :
  case 'sporting' :
  case 'women' :
    include 'content/catnav.php';
  endswitch;
  ?>
</div><!-- end sidebar -->
```

6. You can now use a menu on the left to move between the different categories of lots without returning to the Lot Categories page.

Watch the video for Lesson 7 on the DVD or watch online at `www.wrox.com/ go/24phpmysql.`

8

Repeating Program Steps

Loops let you repeat blocks of code multiple times. Depending on the type of loop, you can repeat the code a given amount of times or while a certain condition is true.

In this lesson you learn how to use the different types of loops:

➤ `While` loops repeat the code while a given condition is true.

➤ `Do/While` loops do the code and then repeat the code while a given condition is true.

➤ `For` loops repeat the code a given number of times.

➤ `Foreach` loops repeat the code separately for each element in an array or object.

You also learn how to break out of a loop or jump to the next iteration using `break` and `continue`.

Finally, you learn what array pointers are and how to manipulate them. This enables you to use loops more effectively with arrays.

WHILE LOOPS

`While` loops execute a block of while a given condition is true. This is the syntax of a `while` loop:

```
while (condition) {
  lines of code here;
  as many as you need;
}
```

As with the other control structures, the `while` loop also has the alternative syntax that is preferred if you are mixing PHP and HTML:

```
while (condition) :
  lines of code here;
  as many as you need;
endwhile;
```

A standard use of this type of loop is to set a counter and loop through until it meets the condition. Each of the loops is called an *iteration* and you often find that programmers use the variable $i as the counter. The following program sets $i to 1, then loops through printing $i and adding 1 to it until $i reaches 5 and makes the `while` condition false. See Figure 8-1.

```
<h1>Print 1 through 4</h1>
<?php
$i = 1;
while ($i < 5) {
  echo "<p>The counter is $i.";
  $i++;
}
```

Print 1 through 4

The counter is 1.

The counter is 2.

The counter is 3.

The counter is 4.
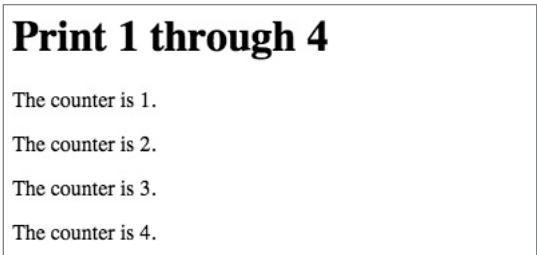
FIGURE 8-1

The `++` attached to $i is the *increment operator.* It is used to increase $i by 1. The `while` loop does not automatically change the variable in the condition as it loops, so you need to remember to do that.

> ⊗ *You need to make sure that all this looping eventually ceases or you get an infamous infinite loop. Make sure your condition eventually goes false. A common problem is to forget to increment a counter variable.*

Increment and decrement operators are convenient ways to count up and down. You can attach the operator before or after the variable depending on whether you want to increment/decrement before or after you use the original variable. Assume that $i is equal to 3 in Table 8-1.

TABLE 8-1: Increment/Decrement Operators

EXAMPLE	NAME	RESULT
`echo ++$i;`	Pre-increment	Adds 1 and then prints 4
`echo $i++;`	Post-increment	Prints 3 and then adds 1
`echo --$i;`	Pre-decrement	Subtracts 1 and then prints 2
`echo $i--;`	Post-decrement	Prints 3 and then subtracts 1

You do not need to use the increment operator. You can use any calculation that is appropriate to change the counter.

You do not even need to use a counter to run a `while` loop. Any statement that can evaluate to true or false can be used as the condition. Just be sure that it does not end up in an infinite loop!

DO/WHILE LOOPS

The `do/while` loops are the same as `while` loops except that the condition is checked after running the block of code instead of before. `While` loops that start with a false condition never run; `do/while` loops always run at least once.

Following is the syntax for a `do/while` loop. The `do/while` loop does not have an alternate syntax.

```
do {
   lines of code here;
   as many as you need;
} while (condition);
```

Here is the same example as the `while` loop, *refactored* as a `do/while` loop. See Figure 8-2.

```
<h1>Print 1 through 4</h1>
<?php
$i = 1;
do {
   echo "<p>The counter is $i.";
   $i++;
} while ($i < 5);
```

Print 1 through 4

The counter is 1.

The counter is 2.

The counter is 3.

The counter is 4.

FIGURE 8-2

> *Refactoring is rewriting the same code in a different, presumably better, way. It can include restructuring to take advantage of new techniques or reworking an old program to more elegantly integrate added features. You are refactoring the Case Study as you add more advanced PHP.*

FOR LOOPS

The `for` loops execute a block of code a given number of times. Unlike `while` loops, `for` loops handle the incrementing/decrementing of the counter for you. This is the syntax of the `for` loop:

```
for (init; condition; increment) {
  lines of code here;
  as many as you need;
}
```

➤ `init` is executed at the beginning of the first loop. It is usually used to initialize the counter. It can contain more than one expression (separated by commas) or be empty. It always ends with a semicolon.

➤ `condition` is evaluated at the beginning of each loop. If it is false, the looping stops. It can contain more than one expression (separated by commas) or be empty. If there is more than one expression, they are all processed, but the last one determines if the loop should continue. If this element is empty, you need to exit the loop using the `break` statement, which you learn about later in this lesson. It always ends with a semicolon.

➤ `increment` is executed at the end of each pass of the loop. It is usually used to increment the counter. It can contain more than one expression (separated by commas) or be empty. Notice that it does not end with a semicolon.

The `for` loop has alternative syntax as well:

```
for (init; condition; increment) :
  lines of code here;
  as many as you need;
endfor;
```

Following is the same function used for the `while` loop, but refactored as a `for` loop. See Figure 8-3.

```
<h1>Print 1 through 4</h1>
<?php
for ($i=1; $i < 5; $i++) {
  echo "<p>The counter is $i.";
}
```

Print 1 through 4

The counter is 1.

The counter is 2.

The counter is 3.

The counter is 4.

FIGURE 8-3

As you can see, the `for` loop takes elements that you commonly used for iteration in the `while` loop and puts them tidily in one spot. `For` loops are handy for looping through arrays. The following code lists out the contacts in the array. See the display in Figure 8-4.

```
<h1>Contacts</h1>
<?php
$contacts = array(
        array('name' => 'George Smith', 'email' => 'george@example.com'),
        array('name' => 'Sally Carpenter', 'email' => 'sally@example.com'),
        array('name' => 'Peter Jason', 'email' => 'peter@example.com'),
        array('name' => 'Lila Carhausen', 'email' => 'lila@example.com')
        );
?>
<ul>
<?php for ($i=0, $size=count($contacts); $i < $size; $i++) : ?>

  <li><?php echo $contacts[$i]['name']; ?><br />
  <?php echo $contacts[$i]['email']; ?></li>

<?php endfor; ?>
</ul>
```


Contacts

- George Smith
  george@example.com
- Sally Carpenter
  sally@example.com
- Peter Jason
  peter@example.com
- Lila Carhausen
  lila@example.com

FIGURE 8-4

This is a typical example of how a program jumps in and out of PHP. You could have stayed in PHP and `echo`'d all the HTML. For clarity of the PHP, most of the examples have done that. However, generally speaking, if you are outputting to the screen, it is better to think of it as mostly HTML with PHP helping out where needed.

It's important that you understand the sequence of events for the `for` loop so here's a closer analysis of this example:

➤ You start out in HTML with the `<h1>` tag and then jump into PHP to assign values to the `$contact` array.

➤ You go briefly back into HTML to start the unordered list.

➤ Now you jump back into PHP for the start of the `for` loop.

➤ Next you define the loop. The initialization, condition, and increment of this example are more complex than the earlier examples:

   ➤ The first expression of the initialization sets the starting point of `$i`. By setting `$i` to 0 you are able to use it as the index for the array because array indexes start at 0. The second expression counts the number of elements in the array.

   ➤ The condition compares the variable containing the index of the array to the number of items in the array. Notice when the condition turns false. That is when the looping stops. It turns false when the index of the array matches the number of items in the array. Hmm. Does that mean that you miss the last person? No, because the array index is 0,1,2,3 and number of elements is 4.

   ➤ The increment is the standard increment that you have been using. It adds 1 to `$i` as it finishes each loop. Because this is the first time through, this statement is ignored.

➤ Now that you've finished defining the `for` loop, you get out of PHP.

➤ While in HTML, you define a list tag. You need to define only one `<li>` tag because you are using this single one as you loop through.

➤ To display the name and e-mail of this row of the array you use a PHP `echo` command.

➤ The end of the code in the loop is signaled by the `endfor` statement. At that point the following happens:

   ➤ You are sent back to the `for` statement.

   ➤ The increment statement is processed.

   ➤ The conditional statement is evaluated to see if it is now true or false:

      ➤ If it is true, you continue to do the loop again.

      ➤ If it is false, you jump to the `endfor`, which takes you out of the loop and out of PHP. You then end with the unordered list tag.

FOREACH LOOPS

The `foreach` loop is a simple way of looping through all the elements in an array or object. Although you could do the same thing with a `for` loop, using a `foreach` loop is less complex and gives you advantages. This is the standard syntax:

```
foreach ($array as $element) {
  code to be executed using $element;
}
```

As with most of the other control structures, the `foreach` has an alternative syntax that is helpful when moving in and out of PHP:

```
foreach ($array as $element) :
  code to be executed using $element;
endforeach;
```

It is a convention, when it makes sense, for the array name to be plural and the element to be the singular. The following example loops through the outer array with a `foreach` loop and gives the same results as in Figure 8-4. Here each element happens to be another array.

```php
<h1>Contacts</h1>
<?php
$contacts = array(
        array('name' => 'George Smith', 'email' => 'george@example.com'),
        array('name' => 'Sally Carpenter', 'email' => 'sally@example.com'),
        array('name' => 'Peter Jason', 'email' => 'peter@example.com'),
        array('name' => 'Lila Carhausen', 'email' => 'lila@example.com')
        );
?>
<ul>
<?php foreach ($contacts as $contact) : ?>

  <li><?php echo $contact['name']; ?><br />
  <?php echo $contact['email']; ?></li>

<?php endforeach; ?>
</ul>
```

If the array you are looping through is an associative array you can keep track of the index with the following syntax:

```php
foreach ($array as $key=>$value) {
  code to be executed using $element and/or $key;
}

foreach ($array as $key=>$element) :
  code to be executed using $element and/or ;
endforeach;
```

In the following code you loop through the array and use each key as a label for displaying the value. See Figure 8-5.

```php
<h1>Contact</h1>
<?php
$myArr = array('name' => 'George Smith',
'email' => 'george@example.com',
'phone' => '555-555-1212',
'state' => 'MA');

?>
<ul>
<?php foreach ($myArr as $key=>$value) : ?>

  <li><?php echo ucfirst($key); ?>:
  <?php echo $value; ?></li>

<?php endforeach; ?>
</ul>
```

Contact

- Name: George Smith
- Email: george@example.com
- Phone: 555-555-1212
- State: MA

FIGURE 8-5

You should be aware of these behaviors:

➤   The foreach loop always starts on the first element of the array.

➤   The element variable is a copy of the array element so nothing you do to it affects the actual array. If you want to make changes to the actual array, prefix an & to the element variable as in foreach ($array as &$value). The & makes this a reference of the array rather than a copy. The new variable is just another name for the same variable.

➤   If you reference the array, the last referenced element remains after the foreach finishes so it is recommended that you remove the variable by unsetting with unset($value).

CONTINUE/BREAK

You use continue if you are done with a particular loop iteration and want to jump to the start of the next iteration. In the following example, the email value is skipped:

```
<h1>Contact</h1>
<?php
$myArr = array('name' => 'George Smith',
'email' => 'george@example.com',
'phone' => '555-555-1212',
'state' => 'MA');

?>
<ul>
<?php foreach ($myArr as $key=>$value) :

  if ($key == 'email') :
    continue;
  endif; ?>

  <li><?php echo ucfirst($key); ?>:
  <?php echo $value; ?></li>

<?php endforeach; ?>
</ul>
```

If you have multiple nested loops, you can use a numeric argument to skip to the next iteration of enclosing loops.

You use `continue` to jump to the next iteration of a loop. To jump completely out of the loop, use `break`. You used `break` in Lesson 7 when learning about switches, but you can also use `break` to jump out of loops.

> ✕ *Overuse of* `continue` *and* `break` *can lead to what is called spaghetti code. This is code that is convoluted and jumps around. It is hard to read, hard to debug, and error prone. Breaks are standard on* `switch` *statements and a simple* `continue` *can be very helpful, but be cautious of more than that.*

TRY IT

Available for download on Wrox.com

In the first part of this Try It, you create a program to display the even numbers from an array using the `for` loop.

You continue improving the Case Study in the second part of the Try It. In the previous lesson you moved information from individual variables to arrays and duplicated the code for each element in the array. Now you replace the duplicated code with a loop.

> ✎ *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson08 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the case study, you need your files from the end of Lesson 7. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Remember that the first index of an array is 0, not 1.

Remember that the index of an element is not the same thing as the value of that element.

Don't forget the echo when needed.

Step-by-Step

Create a program to display the even numbers from an array using the `for` loop. See Figure 8-6 for the desired results.

Display the even numbers

- The array value is 2.
- The array value is 4.
- The array value is 6.
- The array value is 8.
- The array value is 10.

FIGURE 8-6

1.   Start by creating a file called **exercise08a.php**.

2.   Enter the basic code for an HTML page:

```
<html>
<head>
    <title>Exercise 8a</title>
</head>

<body>

</body>
</html>
```

3.   Initialize an array called **$myArray** with the numbers 1–10. Insert it on a line just after the `<body>` tag. Remember to go into PHP.

```
<?php
$myArray = array(1,2,3,4,5,6,7,8,9,10);
```

4.   Count the total number of items in the array:

```
$total = count($myArray);
```

5.   Insert a heading that says **Display the even numbers**. Use HTML, rather than PHP.

```
?>
<h1>Display the even numbers</h1>
```

6.   Start an unordered list:

```
<ul>
```

7.   Set up the `for` loop. You are using the index to print the value and to control the loop. Because you want the second value (2) to be the first to print, set a variable called $i with the

beginning value of 1. This is the index for the second value. You want the loop to continue while the index is less than the count of the items. Because you are printing only even numbers, you want to increase $i by 2 at the end of each loop. You use HTML after this statement, so get out of PHP at the end.

```
<?php for ($i=1; $i < $total; $i += 2) : ?>
```

8. Display the value of the element in a list:

```
<li>The array element value is <?php echo $myArray[$i]; ?>.</li>
```

9. End the `for` loop and the unordered list:

```
<?php endfor; ?>
</ul>
```

10. The resulting code should look like this:

```
<html>
<head>
  <title>Exercise 8a</title>
</head>

<body>

<?php
$myArray = array(1,2,3,4,5,6,7,8,9,10);
$total = count($myArray);
?>

<h1>Display the even numbers</h1>
<ul>
  <?php for ($i=1; $i < $total; $i += 2) : ?>
    <li>The array element value is <?php echo $myArray[$i]; ?>.</li>
  <?php endfor; ?>
</ul>

</body>
</html>
```

In the Case Study, add a `foreach` loop to display the list in `content/gents.php`.

1. Open `contents/gents.php`.

2. Find the `<ul class="ulfancy">` line and add the beginning of the `foreach` loop following that line. Use `$i` as a key. That will give you a counter to use for calculating alternating rows:

```
<?php foreach ($lots as $i=>$lot) : ?>
```

3. Add the `endforeach` after the `</li>`:

```
<?php endforeach; ?>
```

4. Within that loop, change `$lots[$i]` to **$lot**.

5. Remove the increment counter `<?php $i++; ?>` because you are letting the `foreach` loop key do that work.

6. Delete the next two `<li></li>` groupings.

7. The `<ul>` group should now match this:

```
<ul class="ulfancy">

<?php foreach ($lots as $lot) : ?>
  <li class="row<?php echo $i % 2; ?>">
    <div class="list-photo"><a href="images/<?php echo $lot['image']; ?>">
      <img src="images/thumbnails/<?php echo $lot['image']; ?>"  alt=""
/></a>
    </div>
    <div class="list-description">
      <h2><?php echo ucwords($lot['name']); ?></h2>
      <p><?php echo htmlspecialchars($lot['description']); ?></p>
      <p><strong>Lot:</strong> #<?php echo $lot['product_id']; ?>
      <strong>Price:</strong> $<?php echo number_format($lot['price'],2);
?></p>
    </div>
    <div class="clearfloat"></div>
  </li>
<?php endforeach; ?>

</ul>
```

8. Back at the end of the first block of PHP code is the line `$i = 0;`. You used that when you were counting the lines manually. Because the `foreach` loop is handling that, you can delete this line.

> *Watch the video for Lesson 8 on the DVD or watch online at* `www.wrox.com/go/24phpmysql`.

9

Learning about Scope

In this lesson you learn about the concept of *scope*. As you learn about user-defined functions in Lesson 10 and objects in Lesson 12, you learn the details of how scope works when you use functions and objects.

Scope refers to where a specific variable, function, or object can be seen. Scope can be local or global.

LEARNING ABOUT LOCAL VARIABLES

Think of a program as a village where the houses have blinds on all their windows. Imagine that a variable is a person taking a walk down the street. He can talk and interact with all the other people outside, but he cannot be seen by anyone inside the houses nor can he see them. If he needs to talk with someone inside a house, he has to knock on the door and be let in first.

This is the way that local scope works. Variables can be seen only where they are created. The houses are the user-defined functions you learn about in Lesson 10. If you create a variable outside a function, it cannot be seen inside the function. Conversely, if you create a variable inside a function, it can be seen only within that function.

Global scope, on the other hand, is as if that village was on a South Seas island and the houses were open-air tents. Everyone can see everyone else at all times.

Scope is local by default in PHP. Having local scope allows for *encapsulation*. Encapsulation means that you can create a function and know that nothing that goes on outside of that function will change anything inside the function unexpectedly. It also means that you are free to make changes inside the function and do not have to worry that you are messing up something outside the function. This makes your code easier to debug and more robust.

Older programming styles made extensive use of global variables so you may see programs using it, but it's not a style you should copy.

At times you may need to create and use global variables, however. In the next section you learn how to define and work with global variables.

LEARNING ABOUT GLOBAL VARIABLES

Setting a variable to global scope is very easy. Just put the word `global` before the variable before you use it:

```php
<?php
global $myVar;
$myVar = '15';
```

In addition to regular globals, PHP has predefined variables called superglobals. These variables are automatically global. You used the superglobals `$_GET`, `$_POST`, and `$_COOKIE` in Lesson 6. The rest of the superglobals work the same way. The superglobals and an explanation of each are listed in Table 9-1.

TABLE 9-1: Superglobal Variables

SUPERGLOBAL VARIABLE	CONTENTS
`$GLOBALS`	All global variables
`$_SERVER`	Server and execution environment information
`$_GET`	Variables passed via URL parameters
`$_POST`	Variables passed via the HTTP POST method
`$_FILES`	Items uploaded via the HTTP POST method (file uploads)
`$_COOKIE`	Variables passed via HTTP cookies
`$_SESSION`	Session variables
`$_REQUEST`	Contains contents of `$_GET`, `$_POST`, and `$_COOKIE`
`$_ENV`	Variables passed via environment method
`$_SERVER`	Server and execution environment information

Notice that `$GLOBALS` does not have an underscore. It is the superglobal that has been around for the longest time.

Globals have undergone many changes in the different versions of PHP. It is handy to be familiar with the old style so that you know not to copy it. Some of the old ways of doing things still work, but will be removed in later versions. That is another reason for moving right to the modern style.

In the old style the superglobals were prefixed with HTTP and suffixed with VARS. Table 9-2 contains a cross-reference from the current values to the terms that are being retired.

$_SERVER ELEMENTS

The `$_SERVER` superglobal has several predefined keys (elements) for the data it contains. Not all elements work on all servers and they might give you unexpected data depending on the server setup. Here are some that may be useful to you:

➤ `PHP_SELF`: Filename of the currently executing script, relative to the document root

➤ `SERVER_NAME`: Name of the server host

➤ `DOCUMENT_ROOT`: Document root directory under which the current script is executing

➤ `HTTP_REFERER`: Address of the page that referred the user agent to the current page (not available with all user agents)

➤ `HTTP_USER_AGENT`: Contents of the User-Agent such as `Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586)`

➤ `SCRIPT_FILENAME`: Absolute pathname of the currently executing script

➤ `SCRIPT_NAME`: Current script's path

➤ `REQUEST_URI`: URI used to access this page; for instance `/index.html`

These are some of the elements listed in the PHP manual. See the full list at `www.php.net/manual/en/reserved.variables.server.php`.

TABLE 9.2 Deprecated Variables

NEW STYLE	DEPRECATED OLD STYLE
$GLOBALS	This has always been $GLOBALS
$_SERVER	$HTTP_SERVER_VARS
$_GET	$HTTP_GET_VARS
$_POST	$HTTP_POST_VARS
$_FILES	$HTTP_POST_FILES
$_COOKIE	$HTTP_COOKIE_VARS
$_SESSION	$HTTP_SESSION_VARS
$_ENV	$HTTP_ENV_VARS

Variables are local by default in PHP. There is a setting in your `php.ini` file where you can change that. If you turn `register_globals` on, all your variables will be global by default. Some older

programs depended on that setting being on to work. Properly run servers do not allow `register_globals` to be on because it is a security risk, so most of those programs have been weeded out. In PHP6, `register_globals` will not exist.

TRY IT

In this Try It, you set a variable to global and use a superglobal to display information about the server.

> *You can download the code and resources for this Try It from the book's web page at* `www.wrox.com`. *You can find them in the Lesson09 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

This Try It does not use the Case Study.

Hints

The superglobal for the server information is `$_SERVER`.

`$_SERVER` works the same way as `$_GET`, which you learned to use in Lesson 6.

Step-by-Step

Create a program that creates a global variable, assigns a value to it, locates the document root, and concatenates the two results.
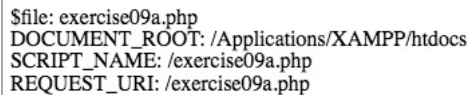
1. Create the file **exercise09a.php**.

2. Create a global variable called **$file**, set it to **exercise09a.php**, and display it:

```php
<?php
global $file;
$file = 'exercise09a.php';
echo '$file: ' . $file .'<br />';
```

3. Look up `DOCUMENT_ROOT`, `SCRIPT_NAME`, and `REQUEST_URI` and display them:

```
$path = $_SERVER['DOCUMENT_ROOT'];
echo 'DOCUMENT_ROOT: ' . $path .'<br />';
$script = $_SERVER['SCRIPT_NAME'];
echo 'SCRIPT_NAME: ' . $script .'<br />';
$uri = $_SERVER['REQUEST_URI'];
echo 'REQUEST_URI: ' . $uri .'<br />';
```

4. Your results should look similar to Figure 9-1. They reflect your server and documents.

```
$file: exercise09a.php
DOCUMENT_ROOT: /Applications/XAMPP/htdocs
SCRIPT_NAME: /exercise09a.php
REQUEST_URI: /exercise09a.php
```

FIGURE 9-1

Watch the video for Lesson 9 on the DVD or watch online at www.wrox.com/
go/24phpmysql.

10

Reusing Code with Functions

Writing code can be time-consuming and error prone. After you have written a piece of code that works well, you want to reuse that code instead of constantly rewriting it when you need to do the same thing again.

You could copy and paste that piece of code everywhere you need it, but then if you found a new bug or thought of an enhancement, you'd need to change it everywhere that you copied it — assuming you could find all the places.

PHP lets you take those pieces of code and create mini-programs called *functions* out of them. Functions make it easier to read your code because functions move extraneous detail out of the main flow of the program. You learned several PHP functions in Lesson 4, such as the function `strlen()` that counts the number of characters:

```php
<?php
$myName = 'Andy';
echo strlen($myName);
```

You give `strlen()` some information (optionally), it does something, and it (optionally) gives you something back. When it gives something back, it is said to *return* something.

In this lesson you learn how to take the valuable pieces of code that you've written and turn them into user functions of your own. You learn how to define the function, how to pass data to the functions, and how to get data back from the functions. Then you take that knowledge to learn how to use the functions in your code.

Another way of reusing code is `include` statements. You are using a simple `include` in the Case Study. In this lesson you also learn about the different types of includes, how they work, and how you can use them to organize your collection of functions.

DEFINING FUNCTIONS

You name functions with the same rules as naming variables. Functions do not have a $ before their name and they are immediately followed by parentheses. The simplest function definition looks like this:

```
function functionname() {
  // PHP code goes here;
}
```

> ✕ PHP, *in general, is very forgiving of whitespace — those blank spaces, tabs, new lines, and blank lines you put in code to make it more readable. However, one place where whitespace is not allowed is between the end of the function name and the parentheses. The* () *must immediately follow the name.*

It's easy to turn code into a function. This code displays "George Smith":

```
<?php
$name = "George Smith";
echo $name;
```

This is how you define a function to display "George Smith":

```
<?php
function getName() {
  $name = "George Smith";
  echo $name;
}
```

This defines the function but does not run the function. To actually run the function and display "George Smith" you need to also call the function, as shown in the following code. See Figure 10-1.

```
<?php
function getName() {
  $name = "George Smith";
  echo $name;
}
?>

<h1>Contacts</h1>
<p><?php getName(); ?></p>
```


FIGURE 10-1

You see that the code in the function is not processed when you define it in the function. It is only when you call the function that "George Smith" is displayed. After you have defined the function you can call it as often as you need it. So if you had a need to echo out George's name three times, you could do it this way (see Figure 10-2):

```
<?php
function getName() {
  $name = "George Smith";
  echo $name;
}
?>
```


FIGURE 10-2

```
<h1>Contacts</h1>
<p><?php getName(); ?></p>
<p><?php getName(); ?></p>
<p><?php getName(); ?></p>
```

Each time you run a function, the variables in your function are new variables. If you change the value of a variable in a function, that change does not exist the next time the function is called. Look at this code and compare it to the results in Figure 10-3:

```
<?php
function getCount() {
  $count++;
  echo "Count: " . $count. "<br />";
}
?>

<h1>Count</h1>
<?php for ($i=0; $i < 5; $i++) :
  getCount();
endfor; ?>
```


Count

Count: 1
Count: 1
Count: 1
Count: 1
Count: 1

FIGURE 10-3

When the getCount() in the preceding code is called, it adds 1 to $count using the $count++ command and then prints it out. So when you loop through the for loop five times, $count prints out as 1 each time because every time getCount() is called $count is initialized to 0. This is a good thing because it keeps your functions clean and contained. You know that your variables start fresh every time you run the function.

Every so often, however, you may want to have a variable stick around through all the times you run the function in a script. Two examples of this would be if you want to do something different the first time you run a function or if you are trying to keep a counter going. To do that you declare your variable as a *static variable*. Static variables maintain their values instead of reinitializing. See how the results change when $count is declared as static in the following code. Figure 10-4 shows the results.

```
<?php
function getCount() {
  static $count;
  $count++;
  echo "Count: " . $count. "<br />";
}
?>

<h1>Count</h1>
<?php for ($i=0; $i < 5; $i++) :
  getCount();
endfor; ?>
```


Count

Count: 1
Count: 2
Count: 3
Count: 4
Count: 5

FIGURE 10-4

PASSING PARAMETERS

All this function can do is print out "George Smith," which is not very useful. If, however, you could tell it to print any name you wanted, it would more useful. The parentheses following the function name are used to hold *parameters*, which are variables that can be passed from the calling

program into the function. You pass different values to the function and the function uses those values when it processes its code. This code displays two different names as shown in Figure 10-5:

```php
<?php
function getName($name) {
  echo $name;
}
?>

<h1>Contacts</h1>
<p><?php getName("George Smith"); ?></
p>
<p><?php getName("Sally Meyers"); ?></p>
```

Contacts

George Smith

Sally Meyers

FIGURE 10-5

PARAMETERS AND ARGUMENTS

You often hear the terms "parameters" and "arguments" used interchangeably. Technically, parameters are the list of variables in the function definition. Arguments are the actual values that are passed to the parameters when the function is used.

You can also pass the information as a variable. This code produces the same results as the prior code:

```php
<?php
function getName($name) {
  echo $name;
}
?>

<h1>Contacts</h1>
<?php
  $name1 = "George Smith";
  $name2 = "Sally Meyers";
?>
<p><?php getName($name1); ?></p>
<p><?php getName($name2); ?></p>
```

You can use all the different arrays and loops that you have learned as well. This code is another way to get the same results:

```php
<?php
function getName($name) {
  echo $name;
}
?>

<h1>Contacts</h1>
<?php
$contacts = array("George Smith", "Sally Meyers");

foreach ($contacts as $contact) :
```

```
?>
  <p><?php getName($contact); ?></p>
<?php endforeach; ?>
```

The name of the variable that you are passing does not need to be the same as the name in the function definition. It ignores the actual name of the variable. Note that in the prior code the variable $contact was passed to the function getName(), but the function defined the parameter as $name.

You can pass more than one parameter to a function. The function loads the information you send it in the same order that the information is received. The following code passes both the name of the contact and the department he or she is in. See the results in Figure 10-6.

```
<?php
function getName($name, $department) {
  echo $name . ' - ' . $department;
}
?>

<h1>Contacts</h1>
<?php
$contacts = array("George Smith", "Sally Meyers");
$department = "Office";

foreach ($contacts as $contact) :
?>
  <p><?php getName($contact, $department); ?></p>
<?php endforeach; ?>
```

Contacts

George Smith - Office

Sally Meyers - Office

FIGURE 10-6

When variables are created within a function, they are automatically created as local variables and can be seen only within the function. Even if a variable has been declared a global variable outside of the function, within the function a variable created with the same name would be a separate local variable and would not know the value in the global variable. If you want a variable in a function to be global, you must specifically declare it as global:

```
global $myVariable;
```

The following example tries to display $department, but it shows as blank. The reason you see the value of the $name within the function is because you passed it through the parentheses. Because you did not do that with $department, you are not passed those values. See the results in Figure 10-7.

```
<?php
function getName($name) {
  echo $name . ' - ' . $department;
}
?>

<h1>Contacts</h1>
<?php
$name = "George Smith";
$department = "Office";
?>

<p><?php getName($name); ?></p>
```

Contacts

George Smith -

FIGURE 10-7

The function is passed a copy of the argument so any changes made to the variable are not made to the original variable. The following example changes the value of $name in the function after it is displayed from within the function. However, when the $name variable is displayed outside the function, it has not changed. See Figure 10-8 for the results of the following code.

```php
<?php
function getName($name) {
  echo $name;
  $name = "Sally Meyers";
}
?>

<h1>Contacts</h1>
<?php
$name = "George Smith";
?>

<p><?php getName($name); ?></p>
<p><?php echo $name; ?>
```


Contacts

George Smith

George Smith

FIGURE 10-8

If you want to make changes to the original variable, you need to pass it by *reference*. When you pass a variable by reference you create a link, a shortcut, or an alias to the original variable rather than making a copy of it. You are creating multiple names for the same thing. Prefix the parameter in the function definition with an ampersand (&) to indicate it should pass by reference, rather than making a copy. In the following example, $name is passed by reference. When the function is called, it prints out the passed name and department and then changes those variables. When you echo $contact after calling the function, you see that the value variable passed by reference, $contact, has changed. On the other hand, $department was not passed by reference, so it is unchanged. See Figure 10-9.

```php
?php
function getName(&$name, $department) {
  echo $name . ' - ' . $department;
  $name = "Sally Meyers";
  $department = "Techs";
}
?>

<h1>Contacts</h1>
<?php
$contact = "George Smith";
$department = "Office";
?>

<p><?php getName($contact, $department); ?></p>
<p><?php echo $contact; ?></p>
<p><?php echo $department; ?></p>
```


Contacts

George Smith - Office

Sally Meyers

Office

FIGURE 10-9

You can also assign a *default* to a parameter. If a parameter has a default value, that value is assigned if no argument is passed. You must pass an argument for every parameter without a default, so always put the arguments with defaults at the end. In the following code, notice that George has no department passed, so he gets the default, while Sally uses the department that is passed. See Figure 10-10.

```php
<?php
function getName($name, $department="Office") {
  echo $name . ' - ' . $department;
}
?>

<h1>Contacts</h1>
<?php
$contact = "George Smith";
$department = "Tech";
?>
<p><?php getName($contact); ?></p>

<?php
$contact = "Sally Meyers";
?>

<p><?php getName($contact, $department); ?></p>
```

Contacts

George Smith - Office

Sally Meyers - Tech

FIGURE 10-10

GETTING VALUES FROM FUNCTIONS

Up until now, the examples of the functions in this lesson have been functions that just perform an action. They print something to the browser. Often, however, you want a function to perform an action and then give you the results back. This "giving the result back" is called *returning*. For instance, you could have a function that takes two arguments and adds them together and returns the result. Give it two and three and it returns five. This example creates a function that adds two numbers together. The result of that function is assigned to the variable $answer, which is then printed out. Here is what that code would look like:

```php
<?php
function addNumbers($number1, $number2) {
  $result = $number1 + $number2;
  return $result;
}

$answer = addNumbers('2', '3');
echo answer;
```

If you want to know if a function performed correctly and it does not have data to return, you would return true or false. You could also return false if there were errors in calculating the data. The following code creates a function that checks to be sure that it received valid numbers and returns false if it received non-numeric data. The program uses the result of the function as the condition in an if statement to determine what to do. See the results in Figure 10-11.

```php
<?php
function addNumbers($number1, $number2) {
  if (is_numeric($number1) AND is_numeric($number2)) {
    $result = $number1 + $number2;
    return $result;
  } else {
    return false;
  }
}
```

Unable to calculate. Non-numeric data.

FIGURE 10-11

```
$answer = addNumbers('2', 'all');

if ($answer) {
  echo $answer;
} else {
  echo 'Unable to calculate. Non-numeric data.';
}
```

If you want to pass multiple bits of data, you use an array. The following code uses an associative array to return the value of the calculation and a separate value for the success or failure. Refer to Figure 10-12.

Unable to calculate. Non-numeric data.

FIGURE 10-12

```
<?php
function addNumbers($number1, $number2) {
  if (is_numeric($number1) AND is_numeric($number2)) {
    $result['answer'] = $number1 + $number2;
    $result['status'] = true;
    $result['message'] = "The answer is ";
    return $result;
  } else {
    $result['answer'] = null;
    $result['status'] = false;
    $result['message'] = "Unable to calculate. Non-numeric data.";
    return $result;
  }
}

$answer = addNumbers('2', 'all');

if ($answer['status']) {
  echo $answer['message'] . $answer['answer'];
} else {
  echo $answer['message'];
}
```

Change `$answer = addNumbers('2', 'all');` to `$answer = addNumbers('2', '3');`, which gives you a valid result. You see a result similar to Figure 10-13.

The answer is 5

FIGURE 10-13

You can use a return (with or without a value) to end a function wherever you want. If you use return without a value, it returns a NULL value.

USING FUNCTIONS

Now that you know how to define functions, how to pass data to them, and how to get data back, it is time to try using them.

You are creating a program that takes a temperature, converts it from Fahrenheit to Celsius (or vice versa), and displays it. See Figure 10-14.

Convert Temperature

Temperature: 70.0°

Type: Fahrenheit to Celsius

Answer: 70.0° Fahrenheit is equal to 21.1° Celsius.

FIGURE 10-14

First, look at the main script for the program:

```php
<?php
// Set up the inputs
$temperature = 70; // Enter the temperature to be converted
$type = 'FtoC'; // Enter FtoC or CtoF for the type of conversion

// Display the Results
?>
<html>
<head>
  <title>Lesson 10t</title>
</head>

<body>
<h1>Convert Temperature</h1>
<p>Temperature: <?php echo formatDeg($temperature); ?></p>
<p>Type: <?php echo expandType($type); ?>
<p>Answer: <?php echo convertTemperature($temperature, $type); ?>
</body>
</html>
```

This code is just the script. If you run it, you receive errors because it is calling several functions that you have not given it yet. Let's go through the script and the functions that need to be added to the start of the program.

This script starts by setting the variables for temperature and for the type of conversion. In Lesson 11 you learn how to get this information from a form, but for now you can just hard-code it.

> **HARDCODING**
>
> You might come across the term hardcoding often in programming. Hardcode just means that you have directly specified the data instead of using a dynamic method of getting it. It is used for testing or when it's deemed too difficult, too time-consuming, too insecure, or just unnecessary to use a more flexible method.

Next, you display the temperature with this code: `echo formatDeg($temperature);`. You are displaying a temperature several times, so you create a function called `formatDeg()` that takes a number and returns it formatted to display with the proper number of decimals and the special HTML entity for a degree symbol. If the value received is not numeric, the function returns 0.

```php
function formatDeg($number) {
  if (is_numeric($number)) {
    return number_format($number, 1) . '&deg;';
  } else {
    return 0 . '&deg;';
  }
}
```

The next line displays the type of conversion: echo expandType($type);. You could just display FtoC or CtoF, but it is easier for the user if you spell out what that means. The expandType() does that for you:

```php
function expandType($type) {
  if ($type=='CtoF') {
    return 'Celsius to Fahrenheit';
  } else {
    return 'Fahrenheit to Celsius';
  }
}
```

And now we come to the real meat of the matter where you display the following answer:

```php
echo convertTemperature($temperature, $type);
```

The convertTemperature() function takes the temperature and the type of conversion and returns an answer with the converted temperature:

```php
function convertTemperature($temperature, $type = "FtoC") {
  switch ($type) {
  case 'CtoF':
    $result = convertCtoF($temperature);
    break;
  case 'FtoC':
  default :
    $result = convertFtoC($temperature);
  }
  return $result;
}
```

Notice that the convertTemperature() function is really just a controlling function that calls the function that does the real work, depending on what type of conversion needs to happen. There are a few defaults happening here as well. The two valid types are CtoF and FtoC. If no type is specified when the function is called, then the type is set to FtoC. If the type that comes in is not one of the two valid types then it is processed as an FtoC.

Finally we come to the two functions that do the conversion calculations and create the answer text, convertFtoC() and convertCtoF(). Notice that these functions also call the formatDeg() function described earlier:

```php
function convertFtoC($temperature) {
  $celsius = ($temperature - 32)* (5/9);
  $result = formatDeg($temperature) . ' Fahrenheit is equal to ' .
          formatDeg($celsius) . ' Celsius.';
  return $result;
}

function convertCtoF($temperature) {
  $fahren = $temperature * (9/5) + 32;
  $result = formatDeg($temperature) . ' Celsius is equal to ' .
          formatDeg($fahren) . ' Fahrenheit.';
  return $result;
}
```

Taking all the preceding code and adding comments, the final code is as follows:

```php
<?php
// FUNCTIONS
/**
 * convertTemperature
 * Convert Temperature
 * @param $temperature
 * @param $type
 */
function convertTemperature($temperature, $type = "FtoC") {
  switch ($type) {
  case 'CtoF':
    $result = convertCtoF($temperature);
    break;
  case 'FtoC':
  default :
    $result = convertFtoC($temperature);
  }
  return $result;
}

/**
 * convertFtoC
 * Convert from Fahrenheit to Celsius
 * @param $temperature
 */
function convertFtoC($temperature) {
  $celsius = ($temperature - 32)* (5/9);
  $result = formatDeg($temperature) . ' Fahrenheit is equal to ' .
           formatDeg($celsius) . ' Celsius.';
  return $result;
}

/**
 * convertCtoF
 * Convert from Celsius to Fahrenheit
 * @param unknown_type $temperature
 */
function convertCtoF($temperature) {
  $fahren = $temperature * (9/5) + 32;
  $result = formatDeg($temperature) . ' Celsius is equal to ' .
           formatDeg($fahren) . ' Fahrenheit.';
  return $result;
}

/**
 * formatDeg
 * Format the numbers to display as Degrees
 * @param unknown_type $number
 */
function formatDeg($number) {
  if (is_numeric($number)) {
```

```php
    return number_format($number, 1) . '&deg;';
  } else {
    return 0 . '&deg;';
  }
}

/**
 * expandType
 * Convert the type to a description
 * @param $type
 */
function expandType($type) {
  if ($type=='CtoF') {
    return 'Celsius to Fahrenheit';
  } else {
    return 'Fahrenheit to Celsius';
  }
}

?>

<?php
// SCRIPT

// Set up the inputs
$temperature = 70; // Enter the temperature to be converted
$type = 'FtoC'; // Enter FtoC or CtoF for the type of conversion

// Display the Results
?>
<html>
<head>
  <title>Lesson 10t</title>
</head>
<body>
<h1>Convert Temperature</h1>
<p>Temperature: <?php echo formatDeg($temperature); ?></p>
<p>Type: <?php echo expandType($type); ?>
<p>Answer: <?php echo convertTemperature($temperature, $type); ?>
</body>
</html>
```

> ✖ *If you forget and put a $ in front of a function such as* `$myVar()`*, PHP calls a function with the name of the value of* `$myVar`*. So the following code calls the function* `foo()`*:*
>
> ```php
>         <?php
>         $myVar = 'foo';
>         echo $myVar();
> ```
>
> *This is called a variable function and is perfectly good PHP, but likely not what you intended to do.*

INCLUDING OTHER FILES

With user functions you have moved from just writing scripts to the more upscale *procedural* programming where your code is in reusable modules. Procedural programming cuts down on redundant code and makes debugging and maintenance easier.

You use the PHP commands `include`, `include_once`, `require`, and `require_once` to organize your files by grouping your functions in their own files and then "including" them in the code you are writing. You did this in Lesson 2 when you moved the content div into files in the content folder and then included them back in.

Includes are different from functions. You can think of an include as an automatic copy/paste. PHP goes out, grabs the file, and plops it right down where the `include` command is. Include files can contain just straight PHP and HTML, which is executed immediately, or they can contain function definitions, which are now available to be called.

These four commands, `include`, `include_once`, `require`, and `require_once`, all have the same syntax: the command and then the filename to be included:

```php
<?php
include 'content/home.php';
```

The difference between include and require is that the two require commands give you a fatal error and stop if the file is missing, whereas the two include commands just issue a warning and keep going. The "once" on the end means that PHP loads the file only once. If the file is already loaded it won't try to load it again.

> *Functions and classes (which you learn about in Lesson 12) can be loaded only once because otherwise they attempt to load duplicate functions or classes, which causes a fatal error. So always use* `include_once` *or* `require_once` *with those files.*

TRY IT

In this Try It, you create a new folder for the Case Study where you put your functions. You create a new function that loads content for different areas of the web page based on parameters passed and from the URL string.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson10 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for down-loading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the case study, you need your files from the end of Lesson 8. Alternatively, you can download the files from the book's website at `www.wrox.com`.

Step-by-Step

Create a function that loads content from the URL string and puts it in a file called `functions.php` in a new folder that contains the includes files.

1. Create a folder called **includes**.

2. Create a file called **functions.php** in that folder. You put your new functions in this file.

3. Create a new function called **loadContent():**

```php
<?php
/**
 * loadContent
 * Load the Content
 * @param $default
 */
function loadContent() {

}
```

4. Open `index.php` and move the code that is in the content div inside the PHP tags to this new function. Your function should now look like this:

```php
<?php
/**
 * loadContent
 * Load the Content
 * @param $default
 */
function loadContent() {
  $content = '';
  // Get the content from the url
  if (isset($_GET['content'])) :
    $content = $_GET['content'];
    // Sanitize it for security reasons
    $content = filter_var($content, FILTER_SANITIZE_STRING);
  endif;
    // Set up the home page as the default
  $content = (empty($content)) ? "home" : $content;
  // Include the chosen page
  include 'content/' . $content . '.php'; }
```

5. In the `index.php` file, add a `require_once` command to the functions file, just below the starting comments:

```php
require_once 'includes/functions.php';
?>
```

6. In the `index.php` file, call the new `loadContent()` function. Your content div should now look like this:

```
<div class="content">
  <?php loadContent(); ?>
</div><!-- end content -->
```

7. Save the `function.php` and `index.php` files and test that the pages are showing up as they did before without errors.

Use the `loadContent` to load the catnav div menu. To do this you add the `$where` and `$default` parameters to the `loadContent` function.

1. Add a `$where` parameter to the function in `function.php` so you know where the request is coming from. Make it the first parameter because there is no default value for it. Add a `$default` parameter with the default of `''`. You use the file in `$default` if you don't get passed a good file name in the `$where` parameter:

```
function loadContent($where, $default='') {
```

2. Change the call directly to the `content` URI parameter with one that locates the URI parameter specified in the `$where` passed to the function. Remove the following code:

```
if (isset($_GET['content'])) :
  $content = $_GET['content'];
  // Sanitize it for security reasons
  $content = filter_var($content, FILTER_SANITIZE_STRING);
endif;
```

3. Replace it with this code, which retrieves the GET parameter whose name is in the `$where` variable:

```
$content = filter_input(INPUT_GET, $where, FILTER_SANITIZE_STRING);
```

4. Filter the `$default` parameter:

```
$default = filter_var($default, FILTER_SANITIZE_STRING);
```

5. Change the hardcoded "home" default to the `$default` variable:

```
$content = (empty($content)) ? $default : $content;
```

6. Change the `include` statement so that you include a file only if there is something in `$content` and return the `include` statement:

```
if ($content) {
  // sanitize the data to prevent hacking.
  $html = include 'content/'.$content.'.php';
  return $html;
}
```

7. Add those same parameters where you call the function in `index.php`. Pass `'content'` and `'home'` as the values:

```
<?php loadContent('content', 'home'); ?>
```

8. Add the parameter **&sidebar=catnav** to the URL of the Gents, Sporting, and Women links in `contents/catnav.php`:

```
<li><a href="index.php?content=gents&sidebar=catnav">Gents</a></li>
<li><a href="index.php?content=sporting&sidebar=catnav">Sporting</a></li>
<li><a href="index.php?content=women&sidebar=catnav">Women</a></li>
```

9. Add the parameter **&sidebar=catnav** to the URL of the Gents, Sporting, and Women links in `contents/categories.php`:

```
<h2><a href="index.php?content=gents&sidebar=catnav">Gents</a></h2>
<p>Gents' clothing from the 18th century to modern times</p>
<a class="button display"
href="index.php?content=gents&sidebar=catnav">Display Lots</a>
...
<h2><a href="index.php?content=sporting&sidebar=catnav">Sporting</a></h2>
<p>Sporting clothing and gear.</p>
<a class="button display"
href="index.php?content=sporting&sidebar=catnav">Display Lots</a>
...
<h2><a href="index.php?content=women&sidebar=catnav">Women</a></h2>
<p>Women's Clothing from the 18th century to modern times</p>
<a class="button display"
href="index.php?content=women&sidebar=catnav">Display Lots</a>
```

10. In `index.php`, replace everything in the sidebar div to a `loadContent()`. with the first argument of `'sidebar'` and the second argument of `''`:

```
<div class="sidebar">
      <?php loadContent('sidebar', ''); ?>
</div><!-- end sidebar -->
```

Watch the video for Lesson 10 on the DVD or watch online at `www.wrox.com/go/24phpmysql`.

11

Creating Forms

Forms are ubiquitous on websites. They include not only obvious "fill in the blank" forms but also are the method used to interact with the user. Search boxes and drop-down filters, for instance, are contained within forms.

In this lesson you review the HTML needed to code a form. Then you learn how to use PHP to process that form and gather the responses. Finally, you learn how to send users to the right page after they have submitted the form.

SETTING UP FORMS

Forms are set up primarily using HMTL code. This is a basic form that asks for your name (see Figure 11-1):

```html
<html>
<head>
  <title>Contact</title>
</head>
<body>
<form action="index.php" method="get">
  <fieldset>
  <legend>Contact</legend>
    <label for="fullname">Name</label>
    <input id="fullname" name="fullname" type="text" />
    <input name="contactForm" type="submit" value="Submit" />
  </fieldset>
</form>
</body>
</html>
```


FIGURE 11-1

The `<form>` tag sets the action that occurs when the Submit button is clicked. This is the URI of the program that processes the form. It can either be a file specifically for processing forms or it can be an existing file that checks for whether form data was sent and automatically processes it. You learn how to do that later in this lesson.

The `<form>` tag also assigns the `method` that is used to send the form data. You learned about the two methods, GET and POST, in Lesson 6. In general, use `get`, which is the default, for inquiries and use `post` for database changes or actions that should not be repeated. GET data is appended to the end of the URL and POST data is not.

The `for` attribute in the `<label>` tag links to the `<input>` tag's `id`. The `name` attribute on the `<input>` tag is the name that is used to identify the data. The `for`, `name`, and `id` attributes are often the same, but only the `for` and the `id` need to match.

The `<input>` tag of `type="submit"` is the Submit button. When the user clicks that button, the form is submitted for processing. This is often used to identify the form. The data is named by the `name` attribute and the value is from the `value` attribute, which is also what is displayed on the button.

> *Use the* `<input>` *tag to create the submit buttons in forms.* `<input type="submit">` *automatically submits the form.* `<input type="button">` *does not automatically submit the form. You need to use JavaScript to submit it.*
>
> *If you use the* `<button>` *tag instead of the* `<input>` *tag, be aware that Internet Explorer passes the information between the button tags and all the other browsers use the value.*

To create radio buttons, you use an `<input>` tag of `type="radio"`:

```
<fieldset>
<legend>What is your gender?</legend>
  <input type="radio" id="genderf" name="gender" value="f" checked="checked"/>
  <label for="genderf"> Female</label>
  <input type="radio" id="genderm" name="gender" value="m"/>
  <label for="genderm"> Male</label>
</fieldset>
```

The `name` attribute matches for each of the radio buttons. This groups the radio buttons together so only one value is sent for the group. The `value` attribute is the value that is submitted, not what is in the label. Because you cannot have duplicate `id` attributes, this is one place where your `for` and `id` attributes do not match your `name` attribute.

Checkboxes are similar to radio buttons, except that more than one box can be marked:

```
<fieldset>
<legend>Where do you live?</legend>
  <input type="checkbox" id="arearural" name="areatypes[]" value="rural" />
  <label for="arearural"> Rural</label>
  <input type="checkbox" id="areasuburb" name="areatypes[]" value="suburb"/>
  <label for="areasuburb"> Suburb</label>
  <input type="checkbox" id="areacity" name="areatypes[]" value="city"/>
  <label for="areacity"> City</label>
</fieldset>
```

Here the `name` attribute for all the checkmarks is the same and has square brackets prefixed to it. The square brackets tell the system to send the checked values as an array. If you leave off the square

brackets, only the last checked value is sent. You could also have each name be different and without the square brackets. In that case each name that was selected is sent as a separate value. With checkboxes, only checked values are sent.

You can also have individual checkboxes that are not part of a group. In the following instance, if the box is checked, the form field `imagesonly` is sent with the value of `yes`. If it is not checked, nothing is sent:

```
<input type="checkbox" id="imagesonly" name="imagesonly" value="yes" />
<label for="imagesonly"> Check here if you only want images.</label>
```

If you want to enter multiple lines of text, you use the `<textarea>` tag instead of a type of `<input>` tag. The following code shows what that looks like:

```
<label for="address">Address</label><br />
<textarea id="address" name="address" rows="3" cols="30"></textarea>
```

To enable a user to select from a drop-down list, you use the `<select>` tag with `<option>` tags as in the following example. This passes the value selected with the name given in the `name` attribute.

```
<label for="level">Level</label>
<select id="level" name="level">
  <option value="">- Select a level -</option>
  <option value="gold">Gold</option>
  <option value="silver">Silver</option>
  <option value="bronze">Bronze</option>
</select>
```

To enable multiple selections, you add the `multiple` attribute and add square brackets to the end of the `name` attribute as shown in the following code. The `multiple` attribute enables the user to select multiple options with his operating system's shortcut using the Shift, Control, or Command keys. The square brackets on the `name` attribute create an array to hold the multiple selections. The `size` attribute determines how many options should be displayed at a time.

```
<label for="interests">What do you like?</label>
<select id="interests" name="interests[]" multiple="multiple" size="3">
  <option value="0">Reading</option>
  <option value="1">Whitewater boating</option>
  <option value="2">Music</option>
</select>
```

If you want to let the user clear his form, you use the `<input>` tag of `type="reset"`. This automatically resets the values for you, as shown in the following code.

```
<input type="reset" value="Clear" />
```

You can also send hidden values. "Hidden" just means the form field is not displayed in the form. It is still displayed in the URL. Use the `<input>` tag with `type="hidden"` to specify hidden values:

```
<input type="hidden" name="id" value="12345" />
```

When you put all these examples together you have a form that looks similar to Figure 11-2. This is the code for that program:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Contact</title>
  <style type="text/css">
    li {list-style:none; margin-bottom: 10px}
  </style>
</head>

<body>
<form action="lesson11b.php" method="get">
<fieldset>
<legend>Contact</legend>

<ul>

<li><label for="fullname">Name</label><br />
<input id="fullname" name="fullname" type="text" /></li>

<li><label for="address">Address</label><br />
<textarea id="address" name="address" rows="3" cols="30"></textarea></li>

<li>
<fieldset>
<legend>What is your gender?</legend>
<input type="radio" id="genderf" name="gender" value="f" checked="checked"/>
<label for="genderf"> Female</label>
<input type="radio" id="genderm" name="gender" value="m"/>
<label for="genderm"> Male</label>
</fieldset>
</li>

<li>
<input type="checkbox" id="imagesonly" name="imagesonly" value="yes" />
<label for="imagesonly"> Check here if you only want images.</label>
</li>

<li>
<fieldset>
<legend>Where do you like to live?</legend>
<input type="checkbox" id="arearural" name="areatypes[]" value="rural" />
<label for="arearural"> Rural</label>
<input type="checkbox" id="areasuburb" name="areatypes[]" value="suburb"/>
<label for="areasuburb"> Suburb</label>
<input type="checkbox" id="areacity" name="areatypes[]" value="city"/>
<label for="areacity"> City</label>
</fieldset>
</li>

<li>
<label for="level">Level</label>
<select id="level" name="level">
<option value="">- Select a level -</option>
```

```
<option value="gold">Gold</option>
<option value="silver">Silver</option>
<option value="bronze">Bronze</option>
</select>
</li>

<li>
<label for="interests">What do you like?</label>
<select id="interests" name="interests[]" multiple="multiple" size="3">
<option value="0">Reading</option>
<option value="1">Whitewater boating</option>
<option value="2">Music</option>
</select>
</li>

</ul>

<input type="submit" value="Submit" name="contactForm" />
<input type="reset" value="Clear" />

<div>
<input type="hidden" name="id" value="12345" />
</div>

</fieldset>
</form>

</body>
</html>
```


FIGURE 11-2

> *You can download the preceding code in this book from the book's web page at* www.wrox.com. *The program is* lesson11b.php *in the Lesson11 folder in the download. "Processing Forms" starts with* lesson11c.php *for the step-by-step files, though the instructions refer to the completed file* lesson11i.php. *It is important to be aware of the name of the file because that must match the action on the* <form> *tag.*

PROCESSING FORMS

You have refreshed your knowledge of coding forms in HTML and you have learned how to set up the information that is passed when the user submits the form. So now it is time to learn how to process the form using the example at the end of the previous section. Figure 11-3 shows the form just before the user submits it.

FIGURE 11-3

When the form is submitted, the data is passed to either the GET or the POST variable, depending on the method attribute. Here is the code used for the <form> tag:

```
<form action="lesson11i.php" method="get">
```

Normally you use a POST because this is data that you are probably using to update a database. I used the GET method here so that you can see the data being passed. Look at the data in the address bar. You see something similar to this:

```
http://localhost/lesson11i.php?fullname=George+Smith&address=123+Anywhich+Street%0D
%0AAnywhere+XX+00000&gender=m&areatypes[]=rural&areatypes[]=city&level=gold&
interests[]=1&interests[]=2&contactForm=Submit&id=12345
```

This is dictated by the HTTP protocol, not PHP itself. Just after the filename `lesson11i.php` is a question mark. That signals the start of a *query string.* The GET variables show as key=value pairs separated by ampersands within the query string. The `name` attribute from the form fields is the key and the `value` attribute (or equivalent) is the value.

Only certain characters are allowed in a URL so characters that are not allowed are encoded to an allowable sequence. Notice that blanks are changed to + and the carriage return between the two lines of address is encoded as `%0D%0A`.

You submit the form with `<input type="submit" value="Submit" name="contactForm" />` so you should be able to find `&contractForm=Submit` in that query string. It's near the end of the string if you are having trouble locating it. You can have your program look for that value shortly.

As well as passing the data, the action on the `<form>` tag is performed. If a URL, either absolute or relative, is listed, that program is called. Your action is to call the same program that contains the form.

You add code to the top of the file to see if a form has been submitted. First, you check to see if there is a GET variable with the same name as the Submit button that contains the value of that Submit button. If you had used the POST method, you would look for it in POST. Because you are just doing a compare with this data, you do not need to worry about filtering it. If a form is submitted, you display a message to the user. It's a good idea to let the user know that she is successful in submitting the form and it also is a quick way to test your `if` statement.

```php
<?php
if($_GET['contactForm'] == "Submit") {
  echo 'Thank you for submitting the form';
}
?>
```

You should not see the message the first time you go to the page. Fill in the form and submit it. Figure 11-4 shows the form after submitting. A message is displayed across the top thanking the user for submitting the form and the form has been reset to the default values.

Next, it is time to collect the rest of the data within the form. You do this using `$_GET`, `$_POST`, and the sanitizing methods you learned at the end of Lesson 6. Normally you would do something constructive with this data, such as update a database. For this example, you just display it on the screen.

FIGURE 11-4

You can assign single values to a variable. The following code gets the value from the `fullname` form field, sanitizes it, and assigns it to `$name`. It does the same thing for the `address`, `gender`, and `level` form fields. You then display the data as a test. See Figure 11-5.

```php
<?php
if($_GET['contactForm'] == "Submit") {
    $name = filter_var($_GET['fullname'], FILTER_SANITIZE_STRING);
    $address = filter_var($_GET['address'], FILTER_SANITIZE_STRING);
    $gender = filter_var($_GET['gender'], FILTER_SANITIZE_STRING);
    $level = filter_var($_GET['level'], FILTER_SANITIZE_STRING);
    echo $name . '<br />';
    echo $address . '<br />';
    echo $gender . '<br />';
    echo $level . '<br />';

    echo 'Thank you for submitting the form';
}
```

Radio buttons, such as gender in the preceding example, are passed only if one is selected. In that example you preselected a choice. If you don't require your user to answer the question, when you process the form you should check first to see if a value was passed. You can do this with the `isset()` function, which is demonstrated in the discussion on checkmark boxes next.

> *Remember that the data from this form is in the address bar. If you get tired of filling in the form while you are testing, just copy and use the full URL. It simulates submitting the form.*

FIGURE 11-5

Next you process the checkboxes. With checkboxes, the field itself is passed only if the box was checked, so you want to see if the field exists. Take your single checkbox named `imagesonly`, which has a value of `Yes`. You check that the parameter was set and that it is equal to the value. If not, you know the checkbox was not checked. See Figure 11-6 to see the results when the box is not checked.

```php
<?php
if (isset($_GET['imagesonly']) && $_GET['imagesonly'] == 'Yes') {
  $imagesonly = 'Yes';
} else {
  $imagesonly = 'No';
}
echo 'Images Only? ' . $imagesonly . '<br />';
```

In the example you also had a group of checkboxes that you want to process as an array. You signified this by using the same name and suffixing the name with square brackets. Because the checkboxes are sent only if they are checked, you use `isset()` to see if any of the `areatype` checkboxes were selected at all. Then you loop through and filter each element of the array and build the filtered `$areatypes` array that you will display. See Figure 11-7 for the results.

```php
if (isset($_GET['areatypes'])) {
  $inputs = array();
  $inputs = $_GET['areatypes'];
  foreach ($inputs as $input) {
    $areatypes[] = filter_var($input, FILTER_SANITIZE_STRING);
  }
  foreach ($areatypes as $areatype) {
    echo $areatype . '<br />';
  }
} else {
  echo "You don't want to live anywhere.<br />";
}
```

FIGURE 11-6

FIGURE 11-7

You can use the same procedure for the multi-select box as well. The values here should be integers so you force them to integers as your filter. See Figure 11-8.

```
if (isset($_GET['interests'])) {
    $inputs = array();
    $inputs = $_GET['interests'];
    foreach ($inputs as $input) {
```

```
      $interests[] = (int) $input;
    }
    foreach ($interests as $interest) {
      echo $interest . '<br />';
    }
  } else {
    echo "You have no interests.<br />";
  }
```

George Smith
123 Anywhich Street Anywhere XX 00000
m
gold
Images Only? No
rural
city
1
2
Thank you for submitting the form
  Contact

    Name*
    [   ]

    Address*
    [   ]
    [   ]

    What is your gender?
    ⦿ Female ⦾ Male

    ☐ Check here if you only want images.
    Where do you like to live?

FIGURE 11-8

Hidden parameters are processed just like regular parameters. In this case your hidden parameter field `id` should be an integer, so you filter it by forcing it to an integer. See Figure 11-9.

```
$id = (int) $_GET['id'];
echo $id . '<br />';
```

Now that you know how to read the data, you can run error checking to see if the form was filled out correctly. In this example, the only checking you need is to be sure that the user entered a name and address. You change the processing for the `fullname` and `address` parameters. You build an array with any errors and then loop through and display those errors. You fill in the same form, but submit it without an address. Your result should be similar to Figure 11-10.

```
// initialize error array
$errors = array();
// see if the form was submitted
if($_GET['contactForm'] == "Submit") {
  // Process required fields
  $name = filter_var($_GET['fullname'], FILTER_SANITIZE_STRING);
  if (!(trim($name))) {
    $errors[] = "You must enter a name";
  }
  $address = filter_var($_GET['address'], FILTER_SANITIZE_STRING);
```

```
if (!(trim($address))) {
  $errors[] = "You must enter an address";
}
if ($errors) {
  foreach ($errors as $error) {
    echo $error . '<br />';
  }
}
```

George Smith
123 Anywhich Street Anywhere XX 00000
m
gold
Images Only? No
rural
city
1
2
12345
Thank you for submitting the form

Contact

Name*

Address*

What is your gender?
⦿ Female ○ Male

☐ Check here if you only want images.

FIGURE 11-9

You must enter an address
George Smith

m
gold
Images Only? No
rural
city
1
2
12345
Thank you for submitting the form

Contact

Name*

Address*

What is your gender?
⦿ Female ○ Male

FIGURE 11-10

REDIRECTING WITH HEADERS

Often when processing forms you need to send the user to another page, depending on the user's response. You can redirect the pages this way, using the `header()` function.

Headers are part of the HTTP protocol that the Web uses to direct traffic and carry data. You see the HTTP every time you call a web page: `http://www.example.com`. HTTP consists of two parts: the headers and the body. The headers contain the address to go to as well as the GET information. The body is where the data goes, including HTML and POST data.

HTTP sends the headers before the body of the message. If it comes across a body before it is given any headers, it automatically creates the headers. Any HTML that you create, even a blank line or an invisible newline character, is seen as part of the body. After headers have been created and the body started, any attempt to add another header results in an error. This is the reason for leaving off the final `?>` tag at the end of PHP files (so no final control characters are seen as output) and why cookies have to be set before any HTML (because setting cookies involves creating headers).

When you use the `header()` function, you are creating headers, so you need to use this before you have created any HTML or `echo`s or stray blank lines. The syntax of the `header()` function is

```
header("Location: filename.php");
```

In this example you are echoing out information so you do not use the redirect. If instead you were saving the data to a database, you could choose to redirect to a different page upon successful completion or stay on the same page to allow the user to correct errors. You can also use redirects if you have a series of forms for the user to go through as in an order entry process.

In most of the examples you have been using GET rather than POST to send your data because then you then can see what is being passed. When you use POST, the parameters do not appear as part of the URL. Some of these example forms would normally have used POST. One of the decisions you need to make for every form you create is whether to use the method GET or POST. Here are some pointers on which one to select when.

Use GET when the same submission of the form can be processed multiple times, such as when making inquiries that don't change a database. GET parameters are part of the address and are displayed in the address bar so they are easily visible to users. Because they are part of the address they can be included in a bookmark.

Use POST when the same submission of the form cannot be processed multiple times without making changes. For instance, a form that is part of registering a new user should be created using POST parameters rather than GET. You may have noticed this difference when you use the back button on your browser and you get a message that warns you that you are about to reprocess POST data. POST data is not part of the address so it is not seen as easily. This also means that you cannot use POST to create different pages that can be bookmarked.

WHERE ARE THE GET PARAMETERS?

You may come across programs that are using GET but you don't see them in the address bar. All the GET parameters in the URL can look messy, which was historically a disadvantage when it came to search engine optimization (SEO). There are ways to rewrite the address so that the GET parameters don't show, but that is beyond the scope of this book.

TRY IT

Available for download on Wrox.com

In this Try It, you create a form that allows people to enter a temperature and have it converted from Fahrenheit to Celsius or vice versa. You use the temperature conversion functions you created in Lesson 10. You do not use the Case Study for this Try It.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson11 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

Hints

The conversion from Fahrenheit to Celsius is C = (F − 32)* (5/9).

The conversion from Celsius to Fahrenheit is F = C * (9/5) + 32.

Step-by-Step

Create a form to get the temperature. The form should have two buttons: one for Fahrenheit to Celsius and the other for Celsius to Fahrenheit.

1. Create a new file called **exercise11a.php**.

2. Enter the following code to create the form:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Convert Temperature</title>
</head>
<body>

<form action="exercise11a.php" method="post">

<fieldset>
<legend>Fahrenheit/Celsius Converter</legend>
  <p><label for="temperature">Temperature</label>
    <input type="text" name="temperature" id="temperature" size="6" />
  </p>
  <p><input type="submit" name="FtoC" value="Fahrenheit to Celsius" /></p>
  <p><input type="submit" name="CtoF" value="Celsius to Fahrenheit" /></p>
</fieldset>
</form>
</body>
</html>
```

Create a file containing functions to convert the temperature.

1. Create a new file called **exercise11b.php**.

2. Add the `convertFtoC()` and `convertCtoF()` functions as shown in Lesson 10.

```
/**
 * convertFtoC
 * Convert from Fahrenheit to Celsius
 * @param $temperature
 */
function convertFtoC($temperature) {
  $celsius = ($temperature - 32)* (5/9);
  $result = formatDeg($temperature) . ' Fahrenheit is equal to ' .
  formatDeg($celsius) . ' Celsius.';
  return $result;
}

/**
 * convertCtoF
 * Convert from Celsius to Fahrenheit
 * @param unknown_type $temperature
 */
function convertCtoF($temperature) {
  $fahren = $temperature * (9/5) + 32;
  $result = formatDeg($temperature) . ' Celsius is equal to ' .
  formatDeg($fahren) . ' Fahrenheit.';
  return $result;
}
```

3. These functions call the `formatDeg()` function that formats the temperature for display, so add that function to the same file:

```
/**
 * formatDeg
```

```
 * Format the numbers to display as Degrees
 * @param unknown_type $number
 */
function formatDeg($number) {
  if (is_numeric($number)) {
    return number_format($number, 1) . '&deg;';
  } else {
    return 0 . '&deg;';
  }
}
```

Back in the `exercise11a.php` file, include the `exercise11b.php` file. Check to see which button was clicked. Based on that, do the conversion.

1. In the `exercise11a.php` file, include the file with the functions using the `require_once()` function:

```
require_once "exercise11b.php";
```

2. Use an `if` statement to check the `$_POST` to see if the FtoC button was clicked:

```
if ($_POST['FtoC'] =="Fahrenheit to Celsius") {
}
```

3. Inside that `if` statement call `convertFtoC()` passing the `$_POST['temperature']` parameter. Force a conversion to float type to prevent a malicious code from getting in.

```
$answer =  convertFtoC((float) $_POST['temperature']);
```

4. Continue the `if` statement with an `elseif` statement to check for the other button and process it:

```
elseif ($_POST['CtoF']) {
    $answer =  convertCtoF((float) $_POST['temperature']);
}
```

Display the answer. Initialize it in the beginning so it is set to nothing.

1. In the `exercise11a.php` file, initialize `$answer` before the `if` statement performing the conversions:

```
$answer = '';
```

2. In the HTML below the form, echo out `$answer`:

```
<p><?php echo $answer; ?></p>
```

3. The full `exercise11a.php` file should look like this:

```
<?php
require_once("exercise11b.php");
$answer = '';
if ($_POST['FtoC']) {
  $answer =  convertFtoC((float) $_POST['temperature']);
} elseif ($_POST['CtoF']) {
  $answer =  convertCtoF((float) $_POST['temperature']);
}
?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Convert Temperature</title>
</head>
<body>

<form action="exercise11a.php" method="post">

<fieldset>
<legend>Fahrenheit/Celsius Converter</legend>
  <p><label for="temperature">Temperature</label>
    <input type="text" name="temperature" id="temperature" size="6" />
  </p>
  <p><input type="submit" name="FtoC" value="Fahrenheit to Celsius" /></p>
  <p><input type="submit" name="CtoF" value="Celsius to Fahrenheit" /></p>
</fieldset>

</form>
<p><?php echo $answer; ?></p>
</body>
</html>
```

4. Test the program. You should see results like Figure 11-11 if you enter 70 and click the Fahrenheit to Celsius button.
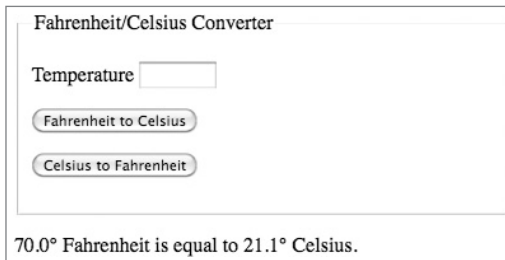
FIGURE 11-11

Watch the video for Lesson 11 on the DVD or watch online at www.wrox.com/go/24phpmysql.

SECTION III
Objects and Classes

The first level in PHP programming is writing scripts, which is just sequential lines of PHP code or bits of PHP interspersed with HTML. The second level is procedural programming, which you learned in Lesson 10 when you started moving code into functions. The next level is object-oriented programming (OOP).

In Lesson 12, you learn the basic concepts of object-oriented programming and why you would want to use it. In the next two lessons you learn how to create classes and use them. The final lesson of this section teaches the more advanced object-oriented techniques.

12

Introducing Object-Oriented Programming

Programmers strive to write code that has fewer errors, is easier to read, and is easier to maintain, and they strive to write that code faster. *Object-oriented programming (OOP)* gives you more tools to do that. In this lesson you learn the reasons for using OOP and the basic concepts behind it.

UNDERSTANDING THE REASONS FOR USING OOP

Object-oriented programming is a way of coding that organizes your programs, encourages consistency, reduces redundancy and complexity, increases flexibility, and promotes better security.

It enables you to create building blocks of basic functionality. You can then reuse these blocks, add on to the blocks, or even override parts of the blocks to create more complex structures. Being able to reuse code in this flexible manner means you have fewer bugs when creating your programs, which makes them more reliable over time.

OOP also uses what is called *encapsulation*. You use encapsulation every time you use local variables in functions because the variables have local scope and can't be seen outside of the function. Encapsulation is the concept that what you do in one section of your program is not affected by and does not affect another section. OOP has similar structures that encapsulate data and actions. You are inside your house with the shades drawn and you control the door through which you receive and disseminate information.

Additionally, OOP is well suited for implementing both design patterns, which are an advanced technique for modeling program designs, and MVC (Model-View-Controller), which is a software design technique for separating database interactions, presentation, and control systems much as you separate content from presentation with HTML and CSS.

INTRODUCING OOP CONCEPTS

One reason people shy away from OOP is that it involves some advanced concepts that can be hard to grasp at first. However, good programming often incorporates OOP so you need to know how to handle it when you come across it. Besides, it is a lot of fun when you understand it.

Objects and Classes

OOP is a way of thinking about what you need to accomplish in terms of objects (nouns) that you need to define and actions (verbs) that you need to perform.

An *object* is an *instantiation* of a *class* that contains *properties* and *methods*. This sounds like so much geek-speak, but you will understand it by the time you are done with this section.

Let's use the example of a cell phone. The cell phone itself is an object. This particular phone is 4.5 inches tall by 2.3 inches wide by .37 inches thick. It has 32GB of storage and weighs 4.8 ounces. It contains specific songs, phone numbers, and ebooks. These are properties that the phone has. Properties are information.

This cell phone can do actions. You can tell it to make phone calls, take pictures, browse the Internet, or play tunes. Each of those types of actions is a method. These are the verbs, the acts that can be performed.

A class corresponds to the blueprint for creating this cell phone. A class is what defines the object.

An object, such as this particular cell phone, is an instance of the class. You can imagine a manufacturing line just churning out instances (objects) of the class. The act of making an instance from a class is called instantiation.

So you could say, "My cell phone was manufactured based on plans and it contains songs and a way for me to play them." An object is an instantiation of a class that contains properties and methods.

To illustrate the concept more clearly, here are some other examples:

Customer Class

➤ **Properties:** First name, last name, company, address, e-mail, phone number

➤ **Methods:** Place an order, inquire about an order, change an e-mail address

Product Class

➤ **Properties:** Product number, description, cost, price, quantity on hand, image of product

➤ **Methods:** Increase quantity when product received, decrease quantity when product shipped, format the price, find an extended price for a given quantity

Article Class

➤ **Properties:** Title, author, abstract, content, ratings, permanent link

➤ **Methods:** Check for proper authority to see the article, save the article to the database, delete the article from the database, format the article for display

To bring it into PHP terms, properties are variables for the class and the methods are functions that are in a class. You learn in the next lesson how to create the classes in PHP.

Extending Classes

You can create a parent class that contains common functions and properties and then build more detailed classes on top of it. For example, a Phone class is able to receive calls and make calls. A Cellphone class extends the phone class so it can automatically receive and make calls, but is also able to retrieve phone numbers from an address book. A Smartphone class extends the Cellphone class and is able to keep track of your calendar, browse the Internet, and play songs.

The classification of animals is another example. The parent class is Animal. Dog, Cat, and Bird all extend Animal, but with different properties and methods. Dog has a Tail Wagging method, whereas Cat has a Purring method and a Length of Fur property, and Bird has a Flying method.

You can also override methods in the parent class with your new class. So the Whale class can override those parts of the Mammal class that it needs to because it swims in the ocean instead of walking on land as most mammals do.

The ability to extend classes (inheritance) and override them makes OOP very powerful. It enables you to create classes that are generic enough to reuse in many programs and yet it enables you to tailor classes for very specific needs.

LEARNING VARIATIONS IN DIFFERENT PHP RELEASES

OOP features are relatively new to PHP. They existed in PHP4 but were more fully developed in PHP5, especially 5.2. Also, PHP 5.3 introduced additional features.

If you are writing your own code in 5.3, these differences are not important. If you are dipping into someone else's code you should be aware of these changes so you can recognize a remnant of an older coding style:

➤ **Pass by Reference:** When you assign an object to a variable it used to create a copy of the object. Now it creates a reference so that changes in either the original or the new object affect both.

➤ **Visibility & Final:** The ability to alter the scope of properties and functions. You learn about visibility in Lesson 15.

➤ **Constructors:** An optional method that is called when you create an object. In PHP4 this was the same name as the class. Now there is a special function, `__construct()`. You learn about this in Lesson 13.

➤ **Class Constants and Static Methods:** This is a way to use classes without creating an object. You learn about this in Lesson 15.

➤ **Abstract Classes:** This is a special type of parent class that you can use to define other classes. You learn about this in Lesson 15.

➤ **The __autoload Function:** This is a way to automatically load your class definitions without needing long lists of `require_once` statements. This is not fully implemented until PHP 5.3. You learn more about this in Lesson 15.

TRY IT

Available for download on Wrox.com

In this Try It, you analyze the contacts in the About Us page of the Case Study to decide what properties and methods you need for a `Contact` object.

> *You can download the code and resources for this Try It from the book's web page at* `www.wrox.com`*. You can find them in the Lesson12 folder in the download. This lesson does not contain any changes to the code.*

Lesson Requirements

If you want to look at the Case Study on your local computer, your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

Hints

This is a conceptual exercise, not a PHP coding exercise.

Step-by-Step

Based on the contacts in Figure 12-1, write down the properties and methods that a `Contact` object might need.

1. The properties are the pieces of information about the `Contact`.

Properties include the following:

- ➤ First name
- ➤ Last name
- ➤ Position
- ➤ Email
- ➤ Phone

2. The methods are functions that the object would need.

- ➤ A method that assembles a full name from first name and last name

About Us

We are all happy to be a part of this. Please contact any of us with questions.

Martha Smith

Position: none
Email: martha@example.com
Phone:

George Smith

Position:
Email: george@example.com
Phone: 515-555-1236

Jeff Meyers

Position: hip hop expert for shure
Email: jeff@example.com
Phone:

Peter Meyers

Position:
Email: peter@example.com
Phone: 515-555-1237

Sally Smith

Position:
Email: sally@example.com
Phone: 515-555-1235

Sarah Finder

Position: Lost Soul
Email: finder@a.com
Phone: 555-123-5555

FIGURE 12-1

Watch the video for Lesson 12 on the DVD or watch online at `www.wrox.com/go/24phpmysql`.

13

Defining Classes

Classes are the heart of object-oriented programming. They define what an object looks like, what information it can store, and what actions and calculations it can perform. In this lesson you learn how to create classes.

The basic syntax of a class looks like this:

```php
<?php
class MyClassname
{
  // Properties
  public $someProperty;
  public $someOtherProperty;

  // Methods
  public function someFunction($input = false) {
    // php code
  }

  public function anotherFunction() {
    // php code
  }
}
```

Begin a class declaration with the word `Class` followed by the name of the class. The body of the class goes in curly brackets. Put property definitions next, followed by the methods.

The class name can be any combination of letters, numbers, or underscores. It can start with either a letter or an underscore, but, according to convention, a class name should begin with a capital letter. The name is case sensitive. Note that like functions, classes do not start with a `$`.

It is standard practice to put each class in a file with the same name as the class. So the class Myclassname would go in a file called `myclassname.php`. This makes it easier to find the right file while you are programming and easier to reuse selected ones in different programs.

If you document your classes with PHPDoc blocks, many editors are able to use them as help text as you program. The block for a class looks like this:

```
/**
 * Class short description
 *
 * Class longer description if needed
 *
 * @package    PackageName
 */
```

If the class is in its own file, you have both the page PHPDoc block and the class PHPDoc block.

Up until now you have been able to run the code in the lessons. Classes do not actually do anything because they are just blueprints. So you are not able to see this code in action until the next lesson.

DEFINING CLASS VARIABLES (PROPERTIES)

Class variables, called properties, are where you put information that is specific to the object. Valid property names are the same as valid variable names. This is the syntax for a property:

```
public $someProperty = 'Mine';
public $someOtherProperty = true;
```

The declaration starts with the scope keyword (public, protected, or private) followed by the property. You learn more about the scope keywords in Lesson 15. For now use public.

You can initialize the property as you do with regular variables except that you can only initialize with a constant, including the Boolean literals. The following code is not valid because you cannot use variables:

```
public $someOtherProperty = $myVar;
```

Notice that these class variables are outside the functions. The functions themselves can have ordinary variables within them, but the properties are separate. When you create an object from a class, a brand new set of properties is initialized and set aside specifically for that object.

To use a property in a method, you prefix the property name with $this->:

```
$this->someProperty
```

Notice that when using $this->, *the property name does not have a* $. *If you put a* $ *in front of the property name, you are telling the program that you want to use the property name that is the value of the property, not that property.*

```
$someProperty = 'anotherProperty';
$anotherProperty = 'Hello';
echo $this->someProperty; // displays 'anotherProperty'
echo $this->$someProperty; // displays 'Hello'
```

It's important when you start working with methods that you recognize the difference between properties, which are defined in the class, and variables, which are defined in a method. The properties require `$this` and keep their value for the life of the object. Variables in methods (like variables in functions) do not use `$this` and start new each time the method is called.

You should document each of your properties with a `PHPDoc` block. The comment includes a short description, any pertinent or confusing information, the `@var` tag to signal that this is a variable, and the type of variable expected, such as string, integer, float, array, Boolean, or object.

```
/**
 * The phone number of this cell phone
 * @var string
 */
```

DEFINING CLASS FUNCTIONS (METHODS)

Although properties have a vital difference from ordinary variables, methods act just like functions and are often called functions. The main difference between methods and functions (other than that one is inside a class and the other is not) is that methods are defined with a scope keyword. This scope affects access to the method, not to anything within the method. You learn more about scope for methods in Lesson 15. For now use `public` scope. This is the syntax for a method:

```
public function myMethod($input) {
  // php code
}
```

You can put anything in this function as you would in a regular function. In addition, you can use the properties by using the `$this->someProperty` form. If you want to call another method in the class you use `$this` as well. So to call `yourMethod()` use `$this->yourMethod()`.

Let's go through a complete class, including the methods. The following is a class describing a cell phone.

Start with the class name and the properties:

```
class Cellphone
{
  // Properties
  public $phoneNumber;
  public $model;
  public $color;
  public $contacts;
  public $songs;
```

Next come the methods. The first is a method to add contacts. The Contact name and Contact phone number are passed in when the method is called. They are then added to the property `contacts`, which is an associative array. Notice the use of `$this->` to reference the property.

```
public function addContact($number, $name) {
  $this->contacts[$name] = $number;
}
```

The following method adds songs to the cell phone. A song is passed into the method when it is called. If the argument is an array, the program loops through the array and adds the songs. If it is not an array, it adds the single song. Remember that empty square brackets on an array automatically add an element with the next available numeric index.

```php
public function addSongs($songs) {
  if (is_array($songs)) {
    foreach ($songs as $song) {
      $this->songs[] = $song;
    }
  } else {
    $this->songs[] = $songs;
  }
}
```

The following method displays the contacts, which are stored in the property `$contacts`. It uses a `foreach` loop that loops through the property `$contacts`, which is an array. Remember that this is just a class — a blueprint — so nothing runs until you create an object from it in the next lesson.

```php
public function displayContacts() {
  // Notice that the property has -> and no $
  // while the array has => and a $
  foreach ($this->contacts as $name=>$number) {
    echo $name . ' - ' . $number . '<br />';
  }
}
```

> It's easy to get confused between -> and => as well as when you need a $ and when you don't.
>
> The class construction uses a dash with the greater-than sign, and constructions working with associative arrays use the equal sign with the greater-than sign.
>
> The associative array uses the normal form for the second variable, so it has a $ in front of the second variable.
>
> There is normally no $ following $this->. You can think of the $ as meaning "use the value of what is inside." If you want to use the property $contacts, it is $this->contacts. You use the $ only if you want to use the value of what is in a property as the name of another property. The following two echo statements both display "George Smith":
>
> ```php
> $this->field = 'contact';
> $this->contact = 'George Smith';
> echo $this->$field;
> echo $this->contact;
> ```

The following method calls two other methods in the same class. It uses the same `$this->` construction as used for properties.

```php
public function addThenDisplayContacts($newname, $newnumber) {
  $this->addContacts($newnumber, $newname);
```

```
      $this->displayContacts();
   }
```

The next method counts the number of songs. It passes the result back via the `return` statement. The earlier methods did what they needed within the method so they did not need to return anything. With this method, the whole point is to return an answer.

```
public function countSongs() {
   $result = count($this->songs);
   return $result;
}
```

And, finally, you need the closing curly bracket for the class:

```
   }
```

Document each of the methods with a `PHPDoc` block. You will find this invaluable when programming if you are using one of the many editors that are able to use it as dynamic help text. This is an example:

```
/**
 * Add contacts
 * @param string $number
 * @param string $name
 * @return integer | boolean
 */
```

The comment starts with a description. Include in the description anything about the method that might trip someone up. List all the parameters coming in, along with their type and parameter name. If you have a return in the method, document it here, along with the type. If it could be more than one type (such as a good value or false if there was an error), separate the types with a pipe symbol (|). Some of the editors even give you a skeleton of the comment. If you are using Eclipse, type a `/**` on the line before a method and Eclipse gives you a comment template with the parameters filled in.

This is what the full `Cellphone` class looks like, complete with documentation:

```
<?php
/**
 * cellphone.php
 *
 * Cellphone class file
 *
 * @version    1.2 2011-02-03
 * @package    Example
 * @copyright  Copyright (c) 2011 Myself
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
/**
 * Cellphone class
 *
 * @package    Example
 */
```

```php
class Cellphone
{
  /**
   * The phone number of this cell phone
   * @var string
   */
  public $phoneNumber;
  /**
   * The model number
   * @var string
   */
  public $model;
  /**
   * The color of the phone, using an id from the color file
   * @var int
   */
  public $color;
  /**
   * Assoc. Array with contact name as the key, the phone number as the value
   * @var array
   */
  public $contacts;
  /**
   * Array with filenames of song mp3 files
   * @var array
   */
  public $songs;

  /**
   * Create a new Contact
   * @param string $number
   * @param string $name
   */
  public function addContact($number, $name) {
    $this->contacts[$name] = $number;
  }

  /**
   * Add an array mp3 filename to the Songs array,
   * if it isn't an array, then just add the single song
   * @param array|string $songs
   */
  public function addSongs($songs) {
    if (is_array($songs)) {
      foreach ($songs as $song) {
        $this->songs[] = $song;
      }
    } else {
      $this->songs[] = $songs;
    }
  }

  /**
```

```
 * Display a list of the Contacts
 */
public function displayContacts() {
  foreach ($this->contacts as $name=>$number) {
    echo  $name . ' - ' . $number . '<br />';
  }
}

/**
 * Create a new contact and then display all the contacts
 * @param string $newname
 * @param string $newnumber
 */
public function addThenDisplayContacts($newname, $newnumber) {
  $this->addContacts($newnumber, $newname);
  $this->displayContacts();
}

/**
 * Count the songs
 * @return int
 */
public function countSongs() {
  $result = count($this->songs);
  return $result;
}

}
```

You have learned how to define a class. In the next lesson you learn how to use this class.

TRY IT

In this Try It, you create a class for the contacts in the Case Study. A contact should include first name, last name, position, email, and phone along with a method that creates a full name out of the first name and last name.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson13 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for down-loading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the case study, you need your files from the end of Lesson 10. Alternatively, you can download the files from the book's website at `www.wrox.com`.

Hints

You do not need the `PHPDoc` block comments for the code to work but it is good to get into a habit of entering it as you go.

Step-by-Step

Create the `includes/classes/contact.php` file.

1. Create a folder called **`classes`** in the `includes` folder.

2. Create a file called **`contact.php`** in the `includes/classes` folder. Enter the page-level documentation:

```php
<?php
/**
 * contact.php
 *
 * Contact class file
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
```

3. Define the class called **`Contact`**:

```php
/**
 * Contact class
 *
 * @package    Smithside Auctions
 */
class Contact
{
}
```

4. Enter the properties in between the curly braces:

```php
    /**
     * First name
     * @var string
     */
    public $first_name;
    /**
     * Last Name
     * @var String
```

```
 */
public $last_name;
/**
 * Position in the company.
 * @var string
 */
public $position;
/**
 * Email
 * @var string
 */
public $email;
/**
 * Phone number, formatted in string
 * @var string
 */
public $phone;
```

5.   Create a function called `name` that concatenates the first and last names with a space in between. Put this method after the properties.

```
/**
 * Creates a full name by concatenating first and last names
 * @return string
 */
public function name() {
  $name = $this->first_name . ' ' . $this->last_name;
  return $name;
}
```

You test this class in the Try It in Lesson 14.

> *Watch the video for Lesson 13 on the DVD or watch online at* `www.wrox.com/go/24phpmysql`.

14

Using Classes

In the previous lesson you learned how to define classes. In this lesson you learn how to use them.

INSTANTIATING THE CLASS

A class is a blueprint for creating objects. When you create an object you are instantiating the class—making an instance of the class. You can create multiple objects from the same class. They all start out the same, with properties empty or equal to a default and with the capability of performing all the actions detailed by the methods. They are all independent so when you change a property in one, it does not affect the property in any of the other objects made from that class. It is just like when you enter a contact in your cell phone; no one else's cell phone is affected.

Instantiating the class is very easy. Make sure that the class code has been included. Use a `require_once` to include it. You need the file, so it should be a `require` rather than an `include`. You also want the class to be included only once because you get an error if you try to define the class a second time, even if it is identical. It is standard practice to include your common classes when you start the program. If you rarely use a class, you can include it before you create an object.

If a class called `Cellphone` is in the `cellphone.php` file, the following code creates an object called `$myPhone`:

```php
<?php
require_once 'cellphone.php';
$myPhone = new Cellphone();
```

The object called `$myPhone` is a regular variable that happens to be an object. You use all the same naming conventions for the variable as you do for other variables.

If you want to create multiple objects, create multiple variables:

```php
<?php
require_once 'cellphone.php';
$myPhone = new Cellphone();
```

```
$sallyPhone = new Cellphone();
$georgePhone = new Cellphone();
```

You can also use a variable to contain the name of the class:

```php
<?php
$type = 'cell';
$classname = $type . 'phone';
$myPhone = new $classname();
$sallyPhone = new $classname();
$georgePhone = new $classname();
```

> ⊗ *Be careful that you only use the $ when using a variable to specify the class name. If you use the actual name of the class, which is usual, do not use a $.*

USING OBJECTS

To access an object's properties or classes use the "dash-greater-than" construction. If `$myPhone` is an object with a property of `$phoneNumber`, the following code assigns 555-555-7777 to that property:

```php
<?php
$myPhone->phoneNumber = '555-555-7777';
```

Assuming that the object has a method called `$displayContacts()`, the following code runs that method:

```php
<?php
$myPhone->displayContacts();
```

Often the purpose of a method is to give you a value, that is, to return a value. You capture this by assigning it to a variable or by using it. Here are two ways of getting the same information from `countSongs()`:

```php
<?php
$numberSongs = $myPhone->countSongs();
echo $numberSongs;
echo $myPhone->countSongs();
```

In the previous lesson you created the class `Cellphone`, which you put in the file `cellphone.php`. You can see that code at the end of Lesson 13, just before the Try It section, or you can download it from this book's website at `www.wrox.com` in the Lesson14 folder. You are now ready to create code using the class so that you can run it and see results.

The first step is to include the class and create an object:
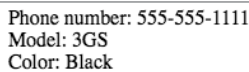
```php
<?php
require_once 'cellphone.php';
$myPhone = new Cellphone();
```

Now put some data into the object. You can put data into the properties directly. This is how you load the cell phone's own number, model, and color:

```
$myPhone->phoneNumber = '555-555-1111';
$myPhone->model = '3GS';
$myPhone->color = 'Black';
```

Echo them back to verify that it worked. The results should look like Figure 14-1. This is what the full code looks like:

```php
<?php
require_once 'cellphone.php';
$myPhone = new Cellphone();
$myPhone->phoneNumber = '555-555-1111';
$myPhone->model = '3GS';
$myPhone->color = 'Black';
echo 'Phone number: ' . $myPhone->phoneNumber . '<br />';
echo 'Model: ' . $myPhone->model . '<br />';
echo 'Color: ' . $myPhone->color . '<br />';
```

```
Phone number: 555-555-1111
Model: 3GS
Color: Black
```

FIGURE 14-1

There are also a couple of methods that enable you to enter data into the properties: addContact() and addSongs(). Use addContact() to add to the $contacts property array by giving the name and phone number of the contact. Display the array to see that the contacts were added properly. Use print_r() to display an array rather than echo. The results look like Figure 14-2.

```
$myPhone->addContact('555-555-1212', 'Sally Strange');
$myPhone->addContact('555-555-1515', 'George Mason');
print_r($myPhone->contacts);
```

```
Phone number: 555-555-1111
Model: 3GS
Color: Black
Array ( [Sally Strange] => 555-555-1212 [George Mason] => 555-555-1515 )
```

FIGURE 14-2

Use a method that prints the contacts. Replace the print_r() statement with a call to displayContacts(). See Figure 14-3.

```
$myPhone->displayContacts();
```

```
Phone number: 555-555-1111
Model: 3GS
Color: Black
Sally Strange - 555-555-1212
George Mason - 555-555-1515
```

FIGURE 14-3

The `addSongs()` property is looking for either an array of filenames of songs or a single filename. Add an array of songs and then display the property to verify it. Putting the `<pre>` tags around `print_r()` makes the results easier to read when testing. See Figure 14-4.

```
$myPhone->addSongs(array('ibelieve.mp3','heaven.mp3','song3.mp3'));
echo '<pre>';print_r($myPhone->songs);echo '</pre>';
```

```
Phone number: 555-555-1111
Model: 3GS
Color: Black
Sally Strange - 555-555-1212
George Mason - 555-555-1515

Array
(
    [0] => ibelieve.mp3
    [1] => heaven.mp3
    [2] => song3.mp3
)
```

FIGURE 14-4

The `countSongs()` method counts the number of songs and returns the number. It does not display the number, though, so you need to display it so you can see that you received it. Instead of printing out the song names, display how many songs are on the phone. See Figure 14-5.

```
echo 'My phone has ' . $myPhone->countSongs(). ' songs.<br />';
```

```
Phone number: 555-555-1111
Model: 3GS
Color: Black
Sally Strange - 555-555-1212
George Mason - 555-555-1515
My phone has 3 songs.
```

FIGURE 14-5

Updating a property directly is convenient but best practices recommend using a method. Using a method enables you to handle any other actions that need to be done when you change the property. This could be error checking, filtering, or adding subsidiary information. Even if you only need to change the value now, at some point in the future you might need to add filtering. If you are updating through a method, the only place you need to change is in the class. If you are updating directly, you need to locate all those places and make changes to all of them.

Two special methods help you fill in the properties when you create the object. The `__construct()` method is automatically called when you create an object. You pass arguments to the method, which

can be used to update the methods or perform any other initializing tasks you need. As an example, add the following method to the cellphone class definition:

```php
public function __construct($phoneNumber, $model, $color) {
  $this->phoneNumber = $phoneNumber;
  $this->model = $model;
  $this->color = $color;
}
```

Note that the method begins with a double underscore. You are taking the phone number, model, and color and using them to update those properties.

Now create a new file and create some objects from the class and then display them. See Figure 14-6.

```php
<?php
require_once 'cellphone.php';
$myPhone = new Cellphone('555-555-1111', 'iPhone', 'Black');
$yourPhone = new Cellphone('555-555-2222', 'Droid', 'Purple');
$hisPhone = new Cellphone('555-555-3333', 'Blackberry', 'Pink');
echo 'Phone number: ' . $myPhone->phoneNumber . '<br />';
echo 'Model: ' . $myPhone->model . '<br />';
echo 'Color: ' . $myPhone->color . '<br />';
echo 'Phone number: ' . $yourPhone->phoneNumber . '<br />';
echo 'Model: ' . $yourPhone->model . '<br />';
echo 'Color: ' . $yourPhone->color . '<br />';
echo 'Phone number: ' . $hisPhone->phoneNumber . '<br />';
echo 'Model: ' . $hisPhone->model . '<br />';
echo 'Color: ' . $hisPhone->color . '<br />';
```

```
Phone number: 555-555-1111
Model: iPhone
Color: Black
Phone number: 555-555-2222
Model: Droid
Color: Purple
Phone number: 555-555-3333
Model: Blackberry
Color: Pink
```

FIGURE 14-6

If PHP does not find a __construct() method, it looks for a method with the same name as the class. This is the old style and you should avoid it in new coding.

TRY IT

In this Try It, you replace the static HTML code on the About Us page of the Case Study with objects for each of the contacts.

First you add the __construct() method to the Contact class that you created in Lesson 13. Then you update the properties in the objects with the names and contact information. Finally you use those objects to display them on the About Us page.

When you start using databases in Section V, you will be able to fill these objects from the database.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson14 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 13. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

The __construct method is called automatically when you create an object.

Because objects are just another type of variable, they can be array elements. By putting objects into an array you can use the power of the array to loop through the objects.

Step-by-Step

So that you can instantiate the Contact class, add an include to the index.php file and add a __construct() method to the class.

1. Include the contact.php file so that the class is available. In index.php, add the following code immediately following the PHPDoc block comments:

    ```
    require_once 'includes/classes/contact.php';
    ```

2. In the contact.php file, add a __construct() method as the first method. The following code shows what it should look like.

    ```
    /**
     * Initialize the Contact with first name, last name, position
     * email, and phone
     * @param array
     */
    public function __construct($input = false) {
      if (is_array($input)) {
        foreach ($input as $key => $val) {
          // Note the $key instead of key.
          // This will give the value in $key instead of 'key' itself
          $this->$key = $val;
        }
    ```

```
    }
  }
```

Create an object for each of the contacts and fill with the information on each of the contacts.

1. In `about.php`, just below the `PHPDoc` comment block, create an object called **`$item`** from the `Contact` class. Pass an associative array with the property names as the keys and the values from the first contact in the list.

```
$item = new Contact(array('first_name'=>'Martha',
  'last_name'=>'Smith',
  'position'=>'none',
  'email'=>'martha@example.com',
  'phone'=>''));
```

2. Change the first item in the unordered list to use the object instead of the hardcoded information about Martha Smith:

```
<li class="row0">
  <h2><?php echo $item->name(); ?></h2>
  <p>Position: <?php echo $item->position; ?><br />
  <?php echo $item->email; ?><br />
  Phone: <?php echo $item->phone; ?><br /></p>
</li>
```

3. Run your program and verify that your About Us page looks similar to Figure 14-7.
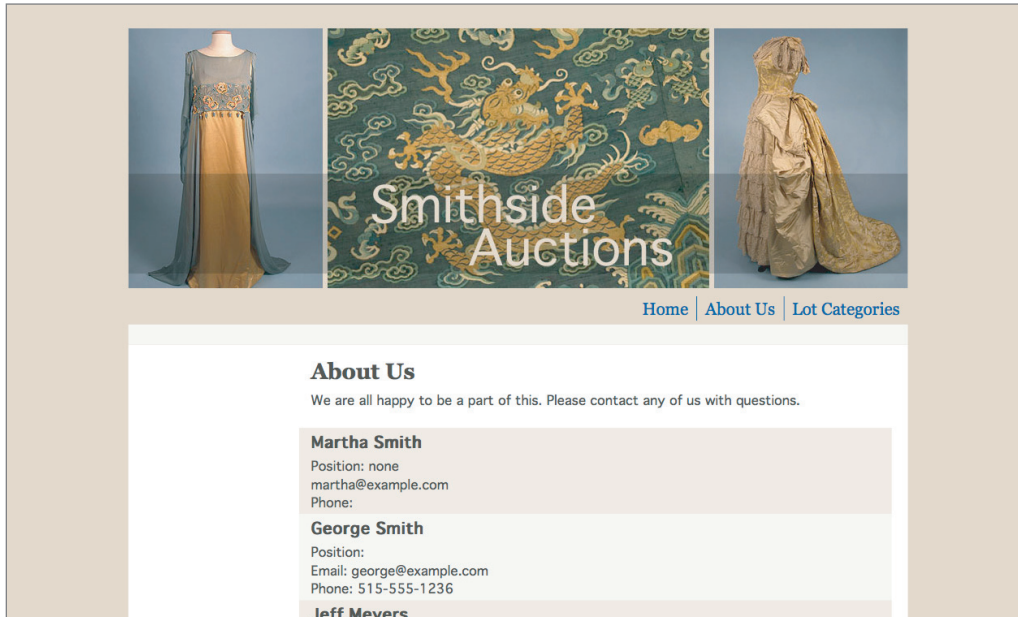
FIGURE 14-7

4. Now that you have successfully created and used your first object, you create an array of objects and loop through the array to display all the contacts. In the `about.php` file, just before you create the `$item` object, initialize the array `$items`:

```
$items = array();
```

5. Change the variable `$item` to **`$items[]`** where you create your object. This adds an element to the `$items` array instead of using a single variable. This is what the altered code looks like:

```
$items[] = new Contact(array('first_name'=>'Martha',
 'last_name'=>'Smith',
 'position'=>'none',
 'email'=>'martha@example.com',
 'phone'=>''));
```

6. Create objects for the rest of the contacts, adding them to the `$items` array:

```
$items[] = new Contact(array('first_name'=>'George',
 'last_name'=>'Smith',
 'position'=>'none',
 'email'=>'george@example.com',
 'phone'=>'515-555-1236'));
$items[] = new Contact(array('first_name'=>'Jeff',
 'last_name'=>'Meyers',
 'position'=>'hip hop expert for shure',
 'email'=>'jeff@example.com',
 'phone'=>''));
$items[] = new Contact(array('first_name'=>'Peter',
 'last_name'=>'Meyers',
 'position'=>'none',
 'email'=>'peter@example.com',
 'phone'=>'515-555-1237'));
$items[] = new Contact(array('first_name'=>'Sally',
 'last_name'=>'Smith',
 'position'=>'none',
 'email'=>'sally@example.com',
 'phone'=>'515-555-1235'));
$items[] = new Contact(array('first_name'=>'Sarah',
 'last_name'=>'Finder',
 'position'=>'Lost Soul',
 'email'=>'finder@a.com',
 'phone'=>'555-123-5555'));
```

7. Put a `foreach` loop around the first item in the unordered list to loop through the `$items` array:

```
<?php foreach ($items as $item) : ?>
  <li class="row0">
    <h2><?php echo $item->name(); ?></h2>
    <p>Position: <?php echo $item->position; ?><br />
    <?php echo $item->email; ?><br />
    Phone: <?php echo $item->phone; ?><br /></p>
  </li>
<?php endforeach; ?>
```

8. Notice that with this loop the class on the `<li>` tag is always `"row0"`. You want that to alternate between `"row0"` and `"row1"` so that your CSS background colors alternate. To do this, add the index of the array in the `foreach` loop. Because you created the array by automatically adding each element, the index counts starting from 0. You then use the formula `$i % 2`, which divides the index by 2 and returns the remainder, thus alternating between 0 and 1. Change your `foreach` statement and the `<li>` tag to the following code:

```php
<?php foreach ($items as $i=>$item) : ?>
  <li class="row<?php echo $i % 2; ?>">
```

9. Now remove the remaining hardcoded contacts in the unordered list. Your complete `<ul>` group should look like the following:

```php
<ul class="ulfancy">
  <?php foreach ($items as $i=>$item) : ?>
    <li class="row<?php echo $i % 2; ?>">
      <h2><?php echo $item->name(); ?></h2>
      <p>Position: <?php echo $item->position; ?><br />
      <?php echo $item->email; ?><br />
      Phone: <?php echo $item->phone; ?><br /></p>
    </li>
  <?php endforeach; ?>
</ul>
```

10. Your About Us page should still look the same. See Figure 14-8. You have changed the internal workings of the page but not what it looks like.
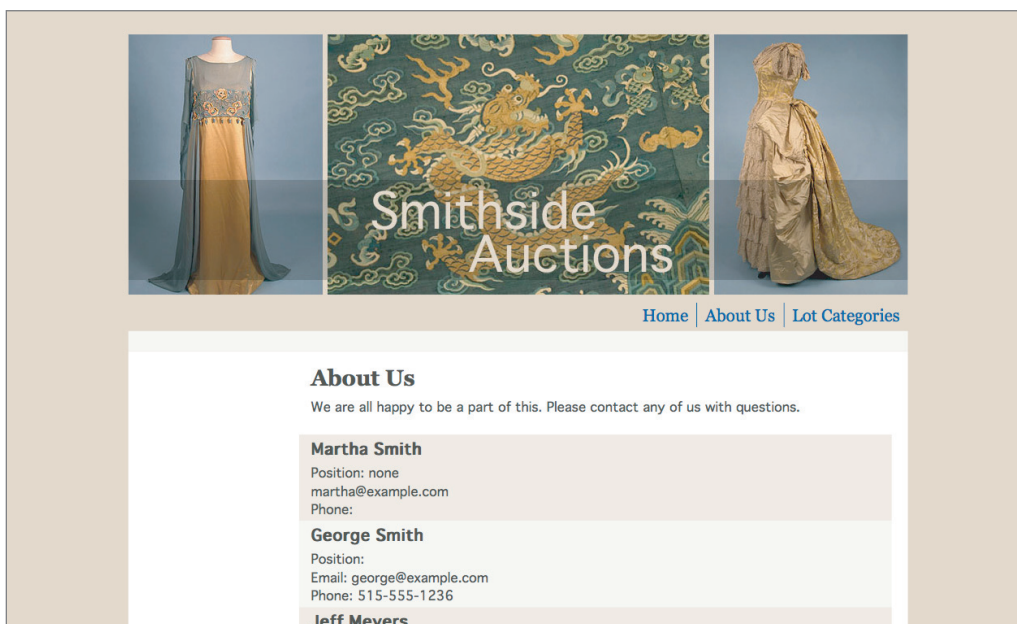
FIGURE 14-8

> Watch the video for Lesson 14 on the DVD or watch online at www.wrox.com/go/24phpmysql.

15

Using Advanced Techniques

Now that you have a grasp of the basic concepts of using classes and objects in PHP, you are ready to learn more advanced techniques. In this lesson you learn easier ways of including the code for the class files and best practice techniques for updating and accessing properties. You learn how to use scope with classes to make your code more secure.

You also learn how to extend classes so that you can build on stable classes, and how to organize the functionality in your program by using static functions and classes.

INITIALIZING THE CLASS

Best practice dictates that you create a new file for each of your classes. You also have to include any of those files once, and only once, to define the class before you can use it. You can use different techniques to do this.

As you learned in the previous chapter, you can use `require_once` statements at the beginning of your program or just before you use your classes. This, however, can result in a lot of tedious code or a lot of extra code that might never be needed.

PHP has what are called "magic methods." These are functions that are called automatically at certain times. One magic method you are familiar with is `__contruct()`, which is called when you create a new object. Magic methods start with a double underscore.

PHP has a magic function called `__autoload()` that is automatically called if you try to use a class that has not been defined. You define this function at the beginning of your program and you can use it to include your class files when they are needed.

The `Cellphone` class was included in the previous lesson with this code:

```php
<?php
require_once 'cellphone.php';
```

The following code takes the class that is being instantiated, adds the `.php` extension, and includes that file if this is the first time it has been looked for:

```php
<?php
function __autoload($class_name) {
  require_once $class_name . '.php';
}
```

PHP calls this magic function if you have created it, but you have control over what happens when `__autoload` is called because you write the function. You need to include this function at the beginning of your code because it needs to exist or PHP will not call it.

That code works if your server is case insensitive, but `Cellphone.php` and `cellphone.php` are considered two different files on most servers. So you need to take that class name and convert it to lowercase before you use it in as a filename. This assumes that you are following the convention of using all lowercase for your filenames. The following code shows the addition of this functionality:

```php
<?php
function __autoload($class_name) {
  require_once strtolower($class_name) . '.php';
}
```

Obviously if you have one class, this is overkill. But if you have several classes using the `__autoload()` function is easier than remembering to go into your first program and adding a `require_once` for each new class. It is also more efficient in terms of performance than including all the files at the beginning of the program if you might not use them all.

UNDERSTANDING SCOPE

Scope dictates who can see what; what has *visibility*. You learned in Lesson 9 that variables have local scope (they can only been seen where they are created) unless they are declared as global. You have more options when using scope with classes.

Properties

When you create a property in a class, you can use it in any of the methods in an object using the `$this->` construction. This gives you the value for the property in the particular object you are in, as opposed to the values of the same property in other instances of the same class. You can think of it as "`$this` gives you the value in this object". Remember that variables that are not properties (those ordinary variables in the methods) do not use `$this`. You can have a property and a variable with the same name, so in a method `$this->myVar` and `$myVar` refer to different variables. The first is a property and its value is accessible anywhere in the object; it is still there the next time you go into the same object. The second is an ordinary variable and is available only in that method; it is initialized every time you use the method.

When you declare a property at the beginning of the class you preface the declaration with the scope. The scope keywords are `public`, `protected`, and `private`. So far in this book, you have been using `public` as the scope:

```php
public $phoneNumber;
```

When a property is public, it can be accessed directly from outside the class. You indicate the object that the property is in to locate it. The following code gives you the public `$phoneNumber` from objects created from the `Cellphone` class. See Figure 15-1.

```php
<?php
require_once 'cellphone.php';
$myPhone = new Cellphone('555-555-1111', 'iPhone', 'Black');
$yourPhone = new Cellphone('555-555-2222', 'Droid', 'Purple');
echo 'Phone number: ' . $myPhone->phoneNumber . '<br />';
echo 'Phone number: ' . $yourPhone->phoneNumber . '<br />';
```

Phone number: 555-555-1111
Phone number: 555-555-2222

FIGURE 15-1

> *You may see properties using the* var *scope. This is a holdover from PHP 4 and is the same as* public *if there is no scope keyword.*

Protected properties cannot be seen outside the class except in inherited classes or parent classes. You learn about inheritance later in this lesson. The following code is how you define a protected property in the class.

```php
protected $_phoneNumber;
```

Private properties are available only within the class itself. This code is an example of defining a private property:

```php
private $_phoneNumber;
```

> *Private and protected properties often start with a single underscore. This was the convention used in earlier versions of PHP, which did not have scope keywords. It was a signal to programmers not to use the property outside of the class. Many programmers continue the convention because it is a good reminder of which properties are private.*
>
> *However, just starting a variable name with an underscore does not make it private. You need to add the* protected *or* private *keyword.*

The `protected` and `private` keywords are not necessarily used in the same sense that you keep your Social Security number private. They are used for control and encapsulation. As an example, say that whenever you change a customer's e-mail address, you want to send an e-mail to the old address as confirmation. If you can change the customer's address directly, you have to know and remember that you also need to send the e-mail. Instead, you can make the e-mail property private and force programmers to use a method to update the e-mail address. You send the e-mail confirmation from within that method. So then any time someone changes the e-mail, the message is automatically sent.

If you need to access these properties outside of the class, you use what are commonly called *getter* and *setter* methods. These are methods you write that return the property value (getters) or change the property value (setters). By convention, the method name begins with get or set followed by the property name with the first letter capitalized. The following is the earlier example in which you directly retrieved the public property `$phoneNumber`:

```php
<?php
require_once 'cellphone.php';
$myPhone = new Cellphone('555-555-1111', 'iPhone', 'Black');
$yourPhone = new Cellphone('555-555-2222', 'Droid', 'Purple');
echo 'Phone number: ' . $myPhone->phoneNumber . '<br />';
  echo 'Phone number: ' . $yourPhone->phoneNumber . '<br />';
```

This next example uses a protected phone number and retrieves the property with a getter method instead. This is the class definition that defines the protected property `$phoneNumber` and the new getter function `getPhoneNumber()`:

```php
<?php
class Cellphone
{
  protected $_phoneNumber;
  public $model;
  public $color;

  public function __construct($phoneNumber, $model, $color) {
    $this->_phoneNumber = $phoneNumber;
    $this->model = $model;
    $this->color = $color;
  }

  public function getPhoneNumber() {
    return $this->_phoneNumber;
  }
}
```

The following is the script that retrieves the protected property. Remember that the method requires parentheses.

```php
<?php
require_once 'cellphone_v2.php';
$myPhone = new Cellphone('555-555-1111', 'iPhone', 'Black');
$yourPhone = new Cellphone('555-555-2222', 'Droid', 'Purple');
echo 'Phone number: ' . $myPhone->getPhoneNumber() . '<br />';
echo 'Phone number: ' . $yourPhone->getPhoneNumber() . '<br />';
```

The getter method is simply returning the value of the `$_phoneNumber` property. However, at a later date, you could add more functionality to that method without needing to change any of the places that are calling it and they would all get the new version.

> *You may have noticed that in the previous example the* `require_once` *file was* `cellphone_v2.php` *but the class name was* `Cellphone`. *The filename and the class name do not have to be the same, although if you want to use* `__autoload()` *to load the classes for you, it's easier if they are.*

Methods

Methods have the same scope keywords as properties: `public`, `protected`, and `private`. If they have no scope keyword, `public` is assumed. Older code doesn't include the scope because it has been available only since PHP 5. Get into the habit of always using a scope keyword.

If you use a method as a getter or setter for a property, the method should be public because only public methods work outside the class.

Private and protected methods are used when they are needed only within the class or they are using variables that are available only within the class. In the following example, the `_formatDeg()` method is only needed to format the information in the other class methods. See Figure 15-2 for the results. Though you usually put the classes in a separate file, for the sake of demonstration, the class and script to use the class are in one file.

```php
<?php
class Converter
{

  public function convertFtoC($temperature) {
    $celsius = ($temperature - 32)* (5/9);
    $result = $this->_formatDeg($temperature) . ' Fahrenheit is equal to '
      . $this->_formatDeg($celsius) . ' Celsius.';
    return $result;
  }

  public function convertCtoF($temperature) {
    $fahren = $temperature * (9/5) + 32;
    $result = $this->_formatDeg($temperature) . ' Celsius is equal to '
      . $this->_formatDeg($fahren) . ' Fahrenheit.';
    return $result;
  }

  private function _formatDeg($number) {
    if (is_numeric($number)) {
      return number_format($number, 1) . '&deg;';
    } else {
      return 0 . '&deg;';
    }
  }

} // end of class

// script to use the class
$newTemp = new Converter;
echo $newTemp->convertFtoC(70);
```

70.0° Fahrenheit is equal to 21.1° Celsius.

FIGURE 15-2

Classes

Classes do not use scope keywords, but you can prevent people from instantiating the class by making the __construct() method and the __clone() methods private or protected. The __construct() method is used to create the object so if it is not accessible, the object cannot be created. You don't need a __construct() method in your class to create an object, but if there is a __construct() method then it needs to be available. So if you don't need a __construct() method but don't want people to instantiate the class from outside the class, just create an empty protected or private __construct() method. You are still able to create an object from within itself or an inherited or parent class, depending on the scope. If you are wondering how you could create an object inside the class if you cannot create an object, you find out when you learn about static methods later in this lesson. The __clone() method is used to create a copy of an object, so if you need to prevent anyone from creating a copy you need to make that method protected or private.

UNDERSTANDING INHERITANCE

One of the powerful features of using classes is that you can extend them. You can make a base class and then create subclasses that inherit all the public and protected properties and methods in addition to their own properties and methods. You can also override existing parent methods with special ones for the child class by using the same name. Here is an example:

```
class Child extends Parent
```

> *It is the classes that are being extended, not the objects. Another way of saying it is that the blueprints are extended, and you build houses based on those extended blueprints. So inheriting properties does not inherit any property values because you have your own values.*

The terms base class and parent class are interchangeable as are subclass and child class. They are just descriptive terms that indicate that one class is the class being extended (base, parent) and the other is the extended class (subclass, child class). You may also come across talk of grandparents and siblings.

The following class, Smartphone, extends the Cellphone class. In addition to the properties and methods it inherits from the Cellphone class, it has apps and a method to list the apps. There are three files: one each for the classes and the script file. In the interest of space I have left off the documentation blocks.

The first file is the base file for the Cellphone class. This is the same file used earlier at the end of the discussion on property scope.

```php
<?php
class Cellphone
{
```

```php
  protected $_phoneNumber;
  public $model;
  public $color;

  public function __construct($phoneNumber, $model, $color) {
    $this->_phoneNumber = $phoneNumber;
    $this->model = $model;
    $this->color = $color;
  }

  public function getPhoneNumber() {
    return $this->_phoneNumber;
  }
}
```

The following code defines the `Smartphone` class. You add a public property for `$apps` that contains the names of apps stored on the phone in an array. In the `__construct()` method, you bring in all four properties; the one you add in this class, plus the three inherited from the `Cellphone` class. You can use all the inherited properties just as if you had created them in the child class. `$app` should be an array, so you use the shortcut `if` statement to cast it to an array if it isn't already.

You also add a public function called `displayApps()` that creates an unordered list of the apps. The period before the `=` is a concatenation sign. It appends what is on the right side to the value on the left. This is a common way of building a long, complex string in an easy-to-read, easy-to-create way.

```php
<?php
class Smartphone extends Cellphone
{
  public $apps;

  public function __construct($phoneNumber, $model, $color, $apps) {
    $this->_phoneNumber = $phoneNumber;
    $this->model = $model;
    $this->color = $color;
    $this->apps = is_array($apps) ? $apps : array($apps);
  }

  public function displayApps() {
    $result = '<ul>';
    foreach ($this->apps as $key=>$app) {
      $result .=  '<li>' . ($key + 1)  . ' - ' . $app . '</li>';
    }
    $result .= '</ul>';
    return $result;
  }
}
```

TRYING TO USE UNAVAILABLE PRIVATE PROPERTIES

If the phone number were private, rather than protected, in the `Cellphone` class, the `Smartphone` class would not see the `Cellphone` class phone number. Because PHP does not require declarations of properties, it would create _phoneNumber as a public property in `Smartphone`.

This next part is a little tricky, but see if you can follow it. If you use `$this->_ phoneNumber` in `Smartphone`, it works as you expect because PHP created a property. However, when you use the methods that you inherit from `Cellphone` you have a problem. `$this->` refers to the class it is in, which in this case is `Cellphone`. Because you put the value into `Smartphone`'s _phoneNumber, not `Cellphone`'s _ phoneNumber, the `Cellphone` method to display the phone number displays blanks.

The moral of this story is that you need to be aware of the scope of the variables and PHP's overly helpful tendencies.

The following is the script that uses the `Smartphone` class. First, include the code for the two classes. The parent class code must exist before the child class. Then populate two instances of `Smartphone`, my phone and your phone, with data. You can display the phone number using a method inherited from `Cellphone` and display the apps list from a method that you added in `Smartphone`. See the results in Figure 15-3.

```php
<?php
require_once 'cellphone_v2.php';
require_once 'smartphone_v1.php';

$applist = array("Angry Birds", "Tetris", "Pandora");
$myPhone = new Smartphone('555-555-1111', 'iPhone', 'Black', $applist);

$applist = array("CNN", "Angry Birds");
$yourPhone = new Smartphone('555-555-2222', 'Droid', 'Purple', $applist);

echo 'Phone number: ' . $myPhone->getPhoneNumber() . '<br />';
echo 'List Apps: '. $myPhone->displayApps() . '<br />';
echo 'Phone number: ' . $yourPhone->getPhoneNumber() . '<br />';
echo 'List Apps: '. $yourPhone->displayApps() . '<br />';
```

```
Phone number: 555-555-1111
List Apps:

    • 1 - Angry Birds
    • 2 - Tetris
    • 3 - Pandora


Phone number: 555-555-2222
List Apps:

    • 1 - CNN
    • 2 - Angry Birds
```

FIGURE 15-3

Take another look at the `__construct()` methods in `Cellphone` and `Smartphone`.

From `Cellphone`:

```
public function __construct($phoneNumber, $model, $color) {
    $this->_phoneNumber = $phoneNumber;
    $this->model = $model;
    $this->color = $color;
}
```

From `Smartphone`:

```
public function __construct($phoneNumber, $model, $color, $apps) {
    $this->_phoneNumber = $phoneNumber;
    $this->model = $model;
    $this->color = $color;
    $this->apps = is_array($apps) ? $apps : array($apps);
}
```

The only thing different is that the `Smartphone` pulls in the $apps and updates it. What you want to do is extend the `__construct()` method itself. You cannot extend methods but you can call the parent's version of a method.

To call a parent's version of a method, use the scope resolution operator, which is a double colon (`::`), known affectionately as a *Paamayim Nekudotayim*. You may see this name in error messages. This calls a parent's `__construct()` method:

```
parent::__construct($phoneNumber, $model, $color);
```

You could also use the class name instead of the keyword `parent`, but using a generic keyword is better in case your inheritance tree changes. The only change you need to make is to change the `Smartphone`'s `__construct()` to the following:

```
public function __construct($phoneNumber, $model, $color, $apps) {
    parent::__construct($phoneNumber, $model, $color);
    $this->apps = is_array($apps) ? $apps : array($apps);
}
```

You call the parent's `__construct()`, passing it the three parameters it is expecting. Then you use the fourth parameter to update the new property. Normally when you override a parent's method you should keep the same parameters. The `__construct()` method is the exception to that rule.

You can override any of the parent's methods and call the parent's version of the method as well, if you need it.

> **FINAL KEYWORD**
>
> If you have a method in the parent that you do not want children to override, add the `final` keyword to the definition in the parent class:
>
> ```
> final protected MyClassMethod()
> ```
>
> Because it is a `protected` method, the children can use it and because it is `final` they can't override it. As a reminder, `private` methods can't be seen by the children.

Often the classes that you extend are ordinary classes. When you are creating an application, however, you may find it makes sense to use *abstract* classes to extend. Abstract classes are classes that are only blueprints; you cannot create objects from them. They are there to act as a base for other classes, not to be used themselves. They are a template for creating classes. You declare a class abstract with the abstract keyword:

```
abstract class MyBaseClass
```

An abstract class can contain both regular and abstract methods. Child classes inherit the regular methods. Abstract classes are empty in the parent and must be defined in the child class. In the following example, MyBaseClass is an abstract class. It requires that child classes create a method called getItem() and a method called quantity(), which has a parameter of $qty. This abstract class also has a regular method that the child classes inherit called listItem().

```php
<?php
abstract class MyBaseClass
{
    abstract protected function getItem();
    abstract protected function quantity($qty);

    public function listItem() {
      $result = '<p>' . $this->getItem() . '</p>';
        return $result;
    }
}
```

The child class in the example, MyChildClass, defines the two abstract classes, one of which it changes to public scope so that it can be called from outside the class:

```php
class MyChildClass extends MyBaseClass
{
    protected function getItem() {
        return "This is an Item";
    }

    public function quantity($qty) {
        return '<p>Your quantity is ' . $qty . '.</p>';
    }
}
```

The script that uses these classes starts by instantiating (creating an object) from the MyChildClass:

```php
$myObject = new MyChildClass
```

Next, the script echoes out (displays on the screen) the result from $myObject->quantity(5). The object $myObject uses the public method in MyChildClass to create an HTML paragraph to display.

```php
echo $myObject->quantity(5);
```

Finally, the script displays the result from the inherited public method listItem(), which calls a method in the child class and creates another HTML paragraph. The results are shown in Figure 15-4.

```php
echo $myObject->listItem();
```

Your quantity is 5.

This is an Item

FIGURE 15-4

UNDERSTANDING STATIC METHODS AND PROPERTIES

So far I have emphasized that the class is just a blueprint and the objects created from the class are the actual items that you use. *Static* methods and properties are accessible without creating an object. As such, they do not have access to regular properties because it is the object that holds the property values.

You might want to use static methods if you have a function that is related to a class but does not require the data from an object. For instance, take the `Customer` class example from Lesson 12:

➤ **Properties:** First name, last name, company, address, e-mail, phone number

➤ **Methods:** Place an order, inquire about an order, change an e-mail address

Say you have a function that pulls a list of all the customers from the database. You could just leave it as a function but if you put it in the class, you know where to find it and don't need to include more files. You do not want to make a customer object to get the list, however, because a customer object is one customer and what you want is an array with a list of customers.

To move back to the `Converter` class from earlier in the lesson, notice that it has no properties; all that is happening is calculations based on data that passed to it. If you turn those into static methods, you do not have to take the expense of creating objects when you want to use the methods:

```php
<?php
class Converter
{

  static public function convertFtoC($temperature) {
    $celsius = ($temperature - 32)* (5/9);
    $result = self::_formatDeg($temperature) . ' Fahrenheit is equal to '
      . self::_formatDeg($celsius) . ' Celsius.';
    return $result;
  }

  static public function convertCtoF($temperature) {
    $fahren = $temperature * (9/5) + 32;
    $result = self::_formatDeg($temperature) . ' Celsius is equal to '
      . self::_formatDeg($fahren) . ' Fahrenheit.';
    return $result;
  }

  static private function _formatDeg($number) {
    if (is_numeric($number)) {
      return number_format($number, 1) . '&deg;';
    } else {
      return 0 . '&deg;';
    }
  }

} // end of class
```

To convert this to static, you add the `static` keyword to the method definitions. Because you do not have an object to work with, you cannot use `$this->` because that refers to the object data. You use the `self` construction with the scope resolution parameter `self::` instead of `$this->`. So calling the formatting class changes from `$this->_formatDeg($celsius)` to `self::_formatDeg($celsius)`.

Now that there are static methods to call, you do not need to create an object. You can reference the class itself, using the same scope resolution parameter. See the results in Figure 15-5.

70.0° Fahrenheit is equal to 21.1° Celsius.

FIGURE 15-5

```
// script to use the class
echo Converter::convertFtoC(70);
```

Static properties can be used as a substitution for global variables. Because they can be accessible from anywhere and retain their values, they can hold or transfer data that needs to be available to many classes and programs. They can also be used as counters. The following is a class that is used to hold site-wide information:

```php
<?php
class Sitewide
{
  public static $copyright ='&copy; 2011';
  private static $site = 'Counting Site';
  private static $count;

  public static function getSite() {
    return self::$site;
  }
  public static function getCopyright() {
    return self::$copyright;
  }
  public static function getCount() {
    self::$count++; // add one to count
    return self::$count; // return count
  }
}
```

The following program could then reference it. You need to include the data either with a `require_once` or the autoloader. See Figure 15-6 for the results.

```html
<html>
  <title><?php echo Sitewide::getSite(); ?></title>
<body>
<h1><?php echo Sitewide::getSite(); ?></h1>
<ul>
  <li><?php echo Sitewide::getCount(); ?></li>
  <li><?php echo Sitewide::getCount(); ?></li>
  <li><?php echo Sitewide::getCount(); ?></li>
</ul>
<p><?php echo Sitewide::$copyright; ?></p>
</body>
</html>
```

Counting Site

- 1
- 2
- 3

(c) 2011

FIGURE 15-6

You can also use the `static` keyword on variables within functions or methods as well as on properties. Static variables remember their value across multiple opens of the functions or methods. Ordinary variables are initialized each time a function or method is used, but static variables remember their value. So when you use a function again, it still remembers the value of any static variables.

TRY IT

In this Try It, you improve the Case Study by adding autoloading of the classes. With only one class this is not needed yet but you add more classes in the later lessons. Without autoloading you would need to change the initialization for each class to include a `require_once` for the class file.

You also change the scope of appropriate properties and methods, adding needed getter methods in the `Contact` class.

> *You can download the code and resources for this Try It from the book's web page at* `www.wrox.com`*. You can find them in the Lesson15 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 14. Alternatively, you can download the files from the book's website at `www.wrox.com`.

Hints

Move the initial processing out of the `index.php` to a separate file to keep `index.php` less complex.

To autoload classes, use the magic method `__autoload()` that you used in the "Initializing the Class" section. You need to add the path to where you put the classes in the Case Study.

Step-by-Step

Create a file called `includes/init.php` to put in all the initial processing including autoloading the classes. Move the initialization code out of `index.php`.

1. Create a file `includes/init.php` with an initial `PHPDoc` block comment:

```php
<?php
/**
 * init.php
```

```
 *
 * Initialization file
 *
 * @version   1.2 2011-02-03
 * @package   Smithside Auctions
 * @copyright Copyright (c) 2011 Smithside Auctions
 * @license   GNU General Public License
 * @since     Since Release 1.0
 */
```

2. Add the magic function for autoloading class files:

```
/**
 * Auto load the class files
 * @param string $class_name
 */
function __autoload($class_name) {
  require_once 'includes/classes/' . strtolower($class_name) . '.php';}
```

3. Move the `require_once` for the functions file from the `index.php` file and paste it into the `includes/init.php` file:

```
// include required files
require_once 'includes/functions.php';
```

4. In the `index.php` file, change the `require_once` for the `contact.php` file to the `init.php`. The PHP section before the <DOCTYPE> now looks like this:

```
<?php
/**
 * index.php
 *
 * Main file
 *
 * @version   1.2 2011-02-03
 * @package   Smithside Auctions
 * @copyright Copyright (c) 2011 Smithside Auctions
 * @license   GNU General Public License
 * @since     Since Release 1.0
 */
require_once 'includes/init.php';
?>
```

Change the `Contact` class to use protected properties. Add the getter methods to return the values.

1. In the `includes/classes/contact.php` file change all the properties to protected. They should look like the following when you are done:

```
/**
 * First name
 * @var string
 */
protected $first_name;

/**
 * Last Name
 * @var String
```

```
 */
protected $last_name;

/**
 * Position in the company.
 * @var string
 */
protected $position;

/**
 * Email
 * @var string
 */
protected $email;

/**
 * Phone number, formatted in string
 * @var string
 */
protected $phone;
```

2.   In the `includes/classes/contact.php` file, add public methods to return each of the properties. Use the `getProperty` naming convention for the methods. Put the following code after the `__construct` method:

```
/**
 * Return First Name
 * @return string
 */
public function getFirst_name() {
  return $this->first_name;
}

/**
 * Return Last Name
 * @return string
 */
public function getLast_name() {
  return $this->last_name;
}

/**
 * Return Position
 * @return string
 */
public function getPosition() {
  return $this->position;
}

/**
 * Return Email
 * @return string
 */
public function getEmail() {
  return $this->email;
```

```
          }

          /**
           * Return Phone
           * @return string
           */
          public function getPhone() {
            return $this->phone;
          }
```

3.   In the `content/about.php` file, change `$item->position`, `$item->email`, and `$item->phone` to use the getter methods. The name is already a function so it does not change. The `<li>` group should look like this:

```
    <li class="row<?php echo $i % 2; ?>">
      <h2><?php echo $item->name(); ?></h2>
      <p>Position: <?php echo $item->getPosition(); ?><br />
      <?php echo $item->getEmail(); ?><br />
      Phone: <?php echo $item->getPhone(); ?><br /></p>
    </li>
```

4.   Test your changes. Your About Us page should look just the same as it did in the previous lesson. See Figure 15-7.

FIGURE 15-7

Watch the video for Lesson 15 on the DVD or watch online at `www.wrox.com/go/24phpmysql`.

SECTION IV
Preventing Problems

▶ **LESSON 16:** Handling Errors

▶ **LESSON 17:** Writing Secure Code

You want a program that runs smoothly. To do that you need to minimize the errors that can occur and handle gracefully errors that do happen.

In the first lesson in this section, you learn how to test for possible errors so that you can fix the problem, bypass it, or gently inform the user. You learn how to set up PHP to monitor for specific error conditions and react to them. In the second lesson, you learn how to secure your code against malicious people.

16

Handling Errors

Errors come in different types and levels. The first type, which you likely have become very familiar with, is programming errors. This is when you use a wrong syntax or do something incorrectly in PHP. If you have `display_errors` set to `on` in your `php.ini` file, PHP is not shy about letting you know there is a problem. You receive big orange warnings with a lot of barely comprehensible stack information. These PHP errors have levels from minor notices where the code still works, to warnings where there is an error but the code continues to run after the error, to fatal errors where processing stops.

> You should display all of these errors while you are developing, but not when your program goes into production. You can have them posted to a log file instead where you can see them if needed but they do not inconvenience your user.

The next type is those errors that happen not because of intrinsic problems with your code, but because of data and resources outside your code. For example, the variable you want to divide by happens to be zero; the e-mail given by the user is not an e-mail address; the image file you want to display is missing; or the database is not accessible.

You learn how to handle this second type of error in the first part of this lesson. You also learn how to incorporate this in the standard error reporting of PHP. In the second part of this lesson, you learn the new error processing techniques that PHP has added to use with objects.

TESTING FOR ERRORS
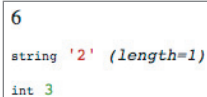
Testing for errors can be divided into two groups. One is to test for conditions that will produce the errors, so you can prevent them before they happen. The other is to test whether an error has happened.

Here are conditions that could produce errors that you should check for:

➤ **Variable types and values:** Is the variable the type you expected and is the value within the range you expected? Are you trying to use a `foreach` loop on a variable that isn't an array or an object? Are you trying to divide by 0? Does the variable contain the name of a valid file?

➤ **Existence of a resource:** Are you trying to include a file that doesn't exist? Are you trying to display an image that doesn't exist? Does the parameter exist? Is the variable NULL or not set?

➤ **Validity of user supplied data:** Did the user fill in all the required inputs on the form? Did the user inject malicious code? Did the user give you data that needs to be encoded before it can be displayed in HTML?

PHP is a *loosely typed* programming language. A variable can switch between containing text or numbers. PHP automatically converts text to numbers, if it can, if the calculation requires numbers. It uses numbers as text if the operation calls for text. You were introduced to this in Lesson 6.

In the following example, $a has a value of 2 and a type of string. $b has a value of 3 and a type of integer. When you multiply them together, PHP converts $a to a type of integer because 2 is a valid integer. The result of the calculation is 6. PHP does not change the type of the variable; it just converts it while it uses it in the calculation. See Figure 16-1.


```
6
string '2' (length=1)
int 3
```

FIGURE 16-1

```php
<?php
$a = '2';
$b = 3;
echo $a * $b;
var_dump($a);
var_dump($b);
```

This works even if both of the variables are string. Change the assignment to `$b = '3';` and you get the same result. In fact, change that same assignment to `$b = '3xyz';` and you still get the answer 6. PHP sees the 3, converts that, and ignores the rest. However, if you change the assignment to `$b = 'xyz3';`, PHP uses 0 for the variable and the result is 0. It is hard to predict how a string with mixed numbers and letters will convert.

Although it is convenient that PHP automatically converts the type for you, it can make it more difficult to know if you have the right type of data for a particular situation. PHP has built-in functions that enable you to check for variable types or for certain values in the variable. The PHP function `is_numeric()` checks whether a variable is a number (2) or a valid numeric string ('2'). If it is either, then it returns true. You saw this function in use in the temperature converter. If the variable is a number, it is formatted. If it is not a number, a 0 is returned.

```php
function formatDeg($number) {
  if (is_numeric($number)) {
    return number_format($number, 1) . '&deg;';
  } else {
    return 0 . '&deg;';
  }
}
```

Table 16-1 lists other common functions for verifying types and values in variables.

TABLE 16-1: Checking Variable Values and Types

FUNCTION	DESCRIPTION
is_numeric()	True if number or numeric string
ctype_digit()	True if all digits are numeric characters
is_bool()	True if variable is a Boolean
is_null()	True if variable is NULL
is_float()	True if variable type is a float
is_double()	True if variable type is a double
is_int()	True if variable type is integer
is_string()	True if variable type is string
is_object()	True if variable is an object
is_array()	True if variable is an array

You should check that a variable is not 0 before attempting to divide by it. The following example shows different ways of checking for the condition as well as different variable values that PHP would convert to 0. See Figure 16-2.

```php
<?php
$b = 3;
$c = 0;
$d = '0';
$e = 'xyz3';
if ($c != 0) {
  echo $b/$c . '<br />';
} else {
  echo 'Cannot divide by 0. <br />';
}
echo ($c != 0) ? $b/$c : 'Cannot divide by 0.<br />';
echo ($d != 0) ? $b/$d : 'Cannot divide by 0.<br />';
echo ($e != 0) ? $b/$e : 'Cannot divide by 0.<br />';
```


Cannot divide by 0.
Cannot divide by 0.
Cannot divide by 0.
Cannot divide by 0.

FIGURE 16-2

You can handle errors in different ways. Here you just displayed a message to the user. In the converter example, you took steps in the program to handle the error so that the user never saw it. Another option is to use the PHP error reporting system. Rather than just displaying a message, you use the trigger_error() function. This is the syntax:

```php
trigger_error($error_msg, $error_level);
```

This posts your error as using the same system that PHP uses. If you have display_errors on, the user sees the error message. If you are logging errors, it is logged. You can use the E_USER_NOTICE level to post informational notices that do not affect the processing, E_USER_WARNING level for errors that allow processing to continue, or E_USER_ERROR to stop the processing. If you do not

specify a level when you create the message, the level defaults to E_USER_NOTICE. Remember that it is good practice to display errors only while testing so this would be more useful for logging. See Figure 16-3.

```php
<?php
$b = 3;
$c = 0;
$d = '0';
$e = 'xyz3';

if ($c != 0) {
  echo $b/$c . '<br />';
} else {
  trigger_error('The value of $c is ' . $c .'. You cannot divide by it ',
    E_USER_NOTICE);
}

if ($d != 0) {
  echo $b/$d . '<br />';
} else {
  trigger_error('The value of $d is ' . $d .'. You cannot divide by it ',
    E_USER_WARNING);
}

if ($e != 0) {
  echo $b/$e . '<br />';
} else {
  trigger_error('The value of $e is ' . $e .'. You cannot divide by it ',
    E_USER_ERROR);
}

echo 'You will never see this because E_USER_ERROR stops the program';
```

Notice: The value of $c is 0. You cannot divide by it in /Users/andytarr/Documents/php24/wphp24/php24/lesson16code/lesson16h.php on line 9

Call Stack				
#	Time	Memory	Function	Location
1	0.0005	327588	{main}( )	../lesson16h.php:0
2	0.0005	328064	trigger_error ( )	../lesson16h.php:9

Warning: The value of $d is 0. You cannot divide by it in /Users/andytarr/Documents/php24/wphp24/php24/lesson16code/lesson16h.php on line 15

Call Stack				
#	Time	Memory	Function	Location
1	0.0005	327588	{main}( )	../lesson16h.php:0
2	0.0007	328144	trigger_error ( )	../lesson16h.php:15

Fatal error: The value of $e is xyz3. You cannot divide by it in /Users/andytarr/Documents/php24/wphp24/php24/lesson16code/lesson16h.php on line 21

Call Stack				
#	Time	Memory	Function	Location
1	0.0005	327588	{main}( )	../lesson16h.php:0
2	0.0008	328148	trigger_error ( )	../lesson16h.php:21

FIGURE 16-3

You can create a custom class for handling errors so all errors, including PHP errors, are processed by your custom class. You could make that class user-friendly enough that you would use it for displaying errors to the user during production. However, that is beyond the scope of this book.

Sometimes it is not the value that you are concerned about but whether the item exists. To see if a variable exists, use the `isset()` function. This example prints the variables if they exist. See Figure 16-4.

```php
<?php
$b = 3;
$c = 0;

if (isset($a)) {
  echo '$a equals ' . $a . '<br />';
}
if (isset($b)) {
 echo '$b equals ' . $b . '<br />';
}
if (isset($c)) {
 echo '$c equals ' . $c . '<br />';
}
```



```
$b equals 3
$c equals 0
```

FIGURE 16-4

To find out if a file exists, you use either `file_exists()` or `is_file()`. `file_exists()` looks for either a directory or a file. `is_file()` is faster and works better if you are working with relative paths but fails on very large files. It only locates files. Use `is_dir()` as the alternative for directories. The following example checks for the existence of the image before trying to display it. See Figure 16-5.

```php
<?php
$name = "Sally Meyers";
$phone = "515-555-1222";
$image = "sally-meyers-t.jpg"
?>
<html>
<head>
<title>Contact</title>
</head>
<body>

<p>
  <?php  if (file_exists($image)) : ?>
    <img src="<?php echo $image; ?>" />
  <?php endif; ?>
  <?php echo $name; ?> :
  <?php echo $phone; ?>
</p>

</body>
</html>
```
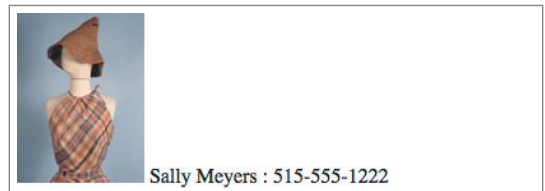

FIGURE 16-5

> **FILE SYSTEM PATH VERSUS URL**
>
> When dealing with files and folders, you need to remember when you are giving the location based on where it is in the folder and file structure on the computer or servers (file system path), or when based on the URL. Some functions are looking for the URL and others for the file system path. If your website does not start at your web root the relative paths are different.

You should check all data that is coming from users or gets or posts for validity or sanitize it. As a reminder, checking for validity means to see if a value meets certain parameters and sanitizing means to automatically make changes to values to render them harmless or in a proper state. See Lesson 6 for more information.

Check the data from users as early as possible in the processing so that you can try to get the proper data before you have done anything that cannot be undone. For instance, if you have required fields, check that you have data for those fields while you can still go back to the user for more information. Be specific about what the customer has to do differently when you display an error message for the user so that he has an easier time fixing it. Some people suggest that you be very non-specific to users about errors over which the user has no control. However, this can make it difficult to track down bugs. You may want the message language to be non-specific but have an error number that identifies the actual problem.

USING TRY/CATCH

With the proliferation of object-oriented abilities in PHP 5, a new type of error handling has been introduced. Most of PHP's internal errors still use the old system, but its object-oriented expressions have started using `try/catch` and the `Exception` class to handle errors.

This new system has four parts: `try`, `throw`, `catch`, and the `Exception` class. The `try/catch` has a syntax similar to the `if/else`. Your main code goes in the `try` block, where you throw an error if you find one. It is then caught in the `catch` block where you handle the error.

```php
<?php
try {
  // your code goes here that might have an error
  // when you find an error you throw an exception
  // by creating an object in Exception class,
  // passing it the error message
  throw new Exception('Divide by Zero');
} catch (Exception $e) {
  // Here's where you handle the error
  echo 'Found an error:', $e->getMessage();
}
```

Here is an example where you check to see if the number you are going to divide by is 0. If it is 0, you throw an `Exception` with the message "Divide by Zero". See Figure 16-6.

```php
<?php
$b = 3;
$c = 0;

try {
  if ($c != 0) {
    echo $b/$c . '<br />';
  } else {
    throw new Exception('Divide by
Zero');
  }
} catch (Exception $e) {
  echo 'Found an error: ', $e->getMessage();
}

echo '<p>And then the code continues.</p>';
```

Found an error: Divide by Zero

And then the code continues.

FIGURE 16-6

You throw only objects and the object must be the `Exception` class or a subclass that you have extended from the `Exception` class. If you throw an object, be sure that it will be caught or you will get an error.

TRY IT

In this Try It, you add error checking to the Case Study. You check for the existence of images files in the display of the gents lots before displaying them.

You add a `try/catch` block around the autoloading of classes.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson16 folder in the download. You find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 15. Alternatively, you can download the files from the book's website at `www.wrox.com`.

Hints

You can use either `is_file()` or `file_exists()` to check for the existence of a file.

`Try/catch` only works with autoloading if you are using at least PHP 5.3. If you are still on PHP 5.2, substitute with a check for the existence of the file and a `trigger_error()`.

Step-by-Step

Check for the existence of the image files before trying to display them in the `content/gents.php` file.

1. Decide what to do if you have no image files. You want something to display as a thumbnail because it is part of the design. So if you do not have a good thumbnail, you use a place-holder photo that exists in the thumbnail folder. You could check to see if that exists each time, but because that is unlikely on a production system and the result of a missing image is not catastrophic, assume your placeholder image exists.

The larger image is a link from the thumbnail. If there is no larger image, show the thumbnail with no link.

2. Open `content/gents.php`.

3. Create variables to hold the calculated paths. If there is no thumbnail, replace it with the `nophoto.jpg`. Add the following code immediately following `<div class="list-photo">`:

```php
<?php // Set up the images
$image = 'images/'. $lot['image'];

$image_t = 'images/thumbnails/'. $lot['image'];
if (!is_file($image_t)) :
  $image_t = 'images/thumbnails/nophoto.jpg';
endif;
```

4. Add an `if` statement to see if the larger image file exists. You can use either the `is_file()` or `file_exists()`. Get out of PHP at the end because you are going back to HTML.

```php
if (is_file($image)) :
?>
```

5. Change the `<a>` tag and `<img>` tag to reference the `$image` and `$image_t` variables:

```html
<a href="<?php echo $image; ?>">
<img src="<?php echo $image_t; ?>"  alt="" />
</a>
```

6. Display just the thumbnail if the larger image did not exist:

```php
<?php else : ?>
  <img src="<?php echo $image_t; ?>"  alt="" />
<?php endif; ?>
```

Your results should look the same as they did in the previous lesson. See Figure 16-7.
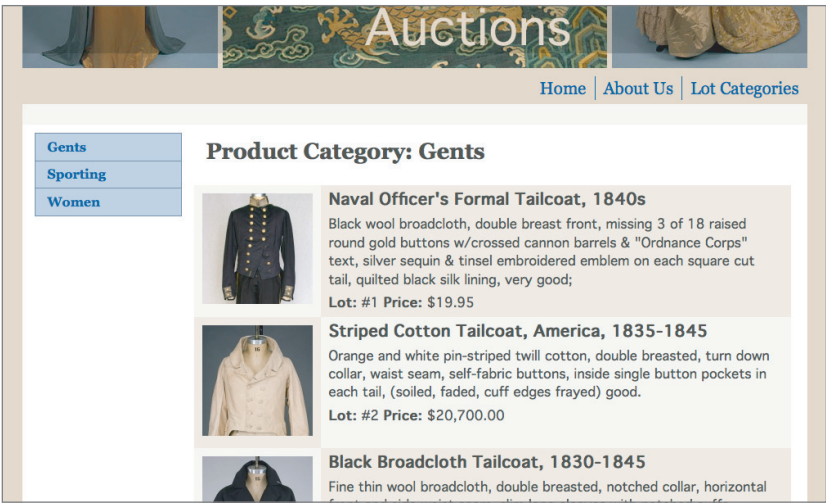
FIGURE 16-7

7. To see what it looks like if there is no photo, temporarily change one of the filenames, such as the following. Your results should look similar to Figure 16-8.

```
$lots[1]['image'] = "gents-striped-8-26XXX.jpg";
```
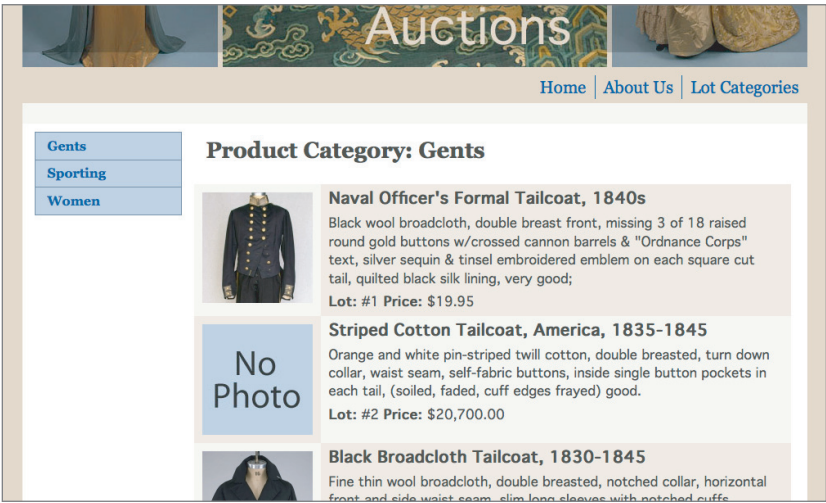

FIGURE 16-8

8. Change the filename back to the correct name.

Add a `try/catch` block to the autoloading of the classes. Note: If you are not running at least PHP 5.3, leave off the `try/catch` structure and use a `trigger_error()` function instead of throwing an error.

1. Open the `includes/init.php` file.

2. Add the `try/catch` structure around the `require_once` statement in the `__autoload()` function:

```
try {
  require_once 'includes/classes/' . strtolower($class_name) . '.php';
} catch (Exception $e) {
  echo 'Exception caught: ',  $e->getMessage(), "\n";
}
```

3. Add a check to see if the file exists around the `require_once`. Because there are a lot of calculations to get the filename, assign it to a variable first:

```
$class_file = 'includes/classes/' . strtolower($class_name) . '.php';
if (is_file($class_file)) {
  require_once $class_file;
}
```

4. Add an `else` to the `if` statement to throw an error if the file was not found:

```
} else {
    throw new Exception("Unable to load class $class_name in file $class_
file.");
  }
```

5. The completed `__autoload()` function should look like this:

```
function __autoload($class_name) {
  try {
    $class_file = 'includes/classes/' . strtolower($class_name) . '.php';
    if (is_file($class_file)) {
      require_once $class_file;
    } else {
      throw new Exception("Unable to load class $class_name in file $class_
file.");
    }
  } catch (Exception $e) {
    echo 'Exception caught: ',  $e->getMessage(), "\n";
  }
}
```

6. To test it, go to the About Us page. It should look similar to Figure 16-9 and contain no errors.
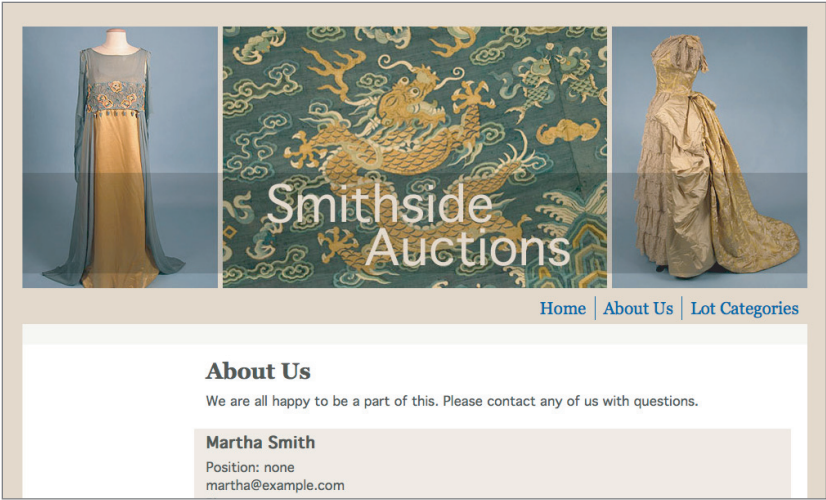
FIGURE 16-9

7. To see what an error looks like, temporarily change the path to an invalid path. Your results should look similar to Figure 16-10.

```
$class_file = 'includes/classesXXX/' . strtolower($class_name) . '.php';
```
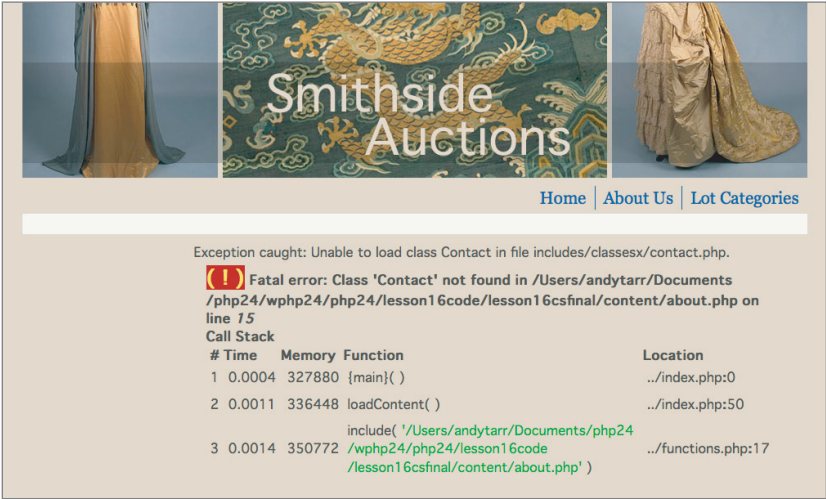

FIGURE 16-10

8. Change your path back to the correct path.

> Watch the video for Lesson 16 on the DVD or watch online at www.wrox.com/ go/24phpmysql.

17

Writing Secure Code

One of the most important things you can learn about PHP and MySQL is how to prevent your code from being an easy target to those who are malicious. There is no way to make your code completely hack-proof, but you can go a long way to securing it by following certain practices. This is not an exhaustive lesson in all the ways that a hacker can get into your site, but it is the equivalent of keeping your car safe by removing your keys and locking your doors.

You might think that the chance of your site being hacked is slight, but remember that hackers can find your site and its vulnerabilities the same way that Google scans your site for search indexes.

In the first section of this lesson you learn what is meant by three common threats: cross-site scripting, cross-site request forgery, and SQL injection. You learn proper coding habits in the second part, which mitigate those and other threats.

UNDERSTANDING COMMON THREATS

Cross-site scripting (XSS), a type of code injection, embeds malicious code inside innocent code that is later output; for instance, when a user enters a search term it is usually displayed on the screen with the results. If, instead of an innocent word, the data entered were JavaScript, that code would be run when the search term was output to the screen. Hackers can install programs that track your keystrokes and track where you go.

Cross-site request forgeries (CSRF, XSRF) work by allowing an attacker to hijack a user's session so that the hacker can use an authenticated user's authority or identity. Requests from the attacker look like they are legitimate responses from forms on your website. The attacker is able to do such things as post comments as a different person, transfer funds to another person's account, or do a distributed password-guessing attack. Attackers can alter your website to trick your users into linking to their site where the hacker can then have control.

SQL Injections are where a hacker injects his own code to alter your database queries, enabling him to access, alter, or even destroy your database. The dynamic power of the PHP/MySQL combination is using PHP variables and expressions when creating queries and updates to the

database. If you use input directly from a user in creating those queries, a malicious user can effectively change your innocent queries into different queries that give him direct access to your database. You learn more specifics about preventing this type of attack starting in the next lesson.

USING PROPER CODING TECHNIQUES

The first rule of writing secure code is to never trust your users. They will give you data you do not expect, either intentionally or unintentionally. You need to check all data that a user submits or could intercept. This includes information from forms or data from POSTs, GETs, or cookies. You should check variables for the proper type of data, for malicious data, and for any character substitutions required, such as changing `&` to `&` before displaying in HTML.

You already know several ways of sanitizing your data, such as these:

```php
<?php
$myVar = htmlspecialchars($myInput); // Lesson 4
$myVar = filter_var($myInput, FILTER_SANITIZE_STRING); // Lesson 6
$myVar = filter_input(INPUT_GET, $where, FILTER_SANITIZE_STRING); // Lesson 10
$myVar = (int) $myInput; // Lesson 11
```

When displaying any user output to the browser, use `htmlspecialchars()` if you have not verified that it is an integer. Before saving any data to a database you need to escape it properly for that database. With MySQL you use `mysql_real_escape_string`, which you learn in Lesson 22.

When given an option of using quotes or not, use quotes because it makes it more difficult for a hacker to break out. You still need to sanitize the variable. Here is a case where using valid XHTML makes your code more secure. Take, for instance, the following insecure code:

```php
<?php $myClass = $_GET['class']; ?>
<div class=<?php echo $myClass; ?>>Text goes here</div>
```

Call that code with `?class=red` at the end of the URL and your source code evaluates to `<div class=red>Text goes here</div>`. However, if you call the code with `?class=g><b>BAD STUFF HERE </b`, you see results similar to Figure 17-1.

FIGURE 17-1

If you enclose the class with quotes, which is the valid method in XHTML, your results look like Figure 17-2.

```php
<?php $myClass = $_GET['class']; ?>
<div class="<?php echo $myClass; ?>">Text goes here</div>
```


FIGURE 17-2

You should also sanitize any input coming from your GET so your final code will look like this:

```
<?php $myClass = filter_input(INPUT_GET, 'class', FILTER_SANITIZE_STRING); ?>
<div class="<?php echo $myClass; ?>">Text goes here</div>
```

If you validate forms using a client-side validation such as JavaScript, always back it up with server-side validation such as PHP. Using JavaScript enables you to create a better user experience, but it is easier to bypass than server-side validations or could be turned off, so using a combination is the better solution.

How your PHP is set up is important for security. Global variables should be off in php.ini. On a production site, turn off display_errors. You can do this via php.ini for the whole site or use .htaccess if you are using a site for both production and development.

Initialize your variables if you are not setting the variable in all cases. Do not assume they start empty because users can add variable assignments to URLs.

```
<?php
$myVar = '';
if ($someCondition) {
  $myVar = '1';
}
```

Do not include a file directly from a get request. Use a two-step process. First verify that there is no invalid data in the name itself using filter_input or filter_var. Then you need to verify that the file is one of your files and not from some remote site. You can do this by checking against a list of valid files or only displaying files from valid folders. An example is the loadContent function in the Case Study:

```
function loadContent($where, $default='') {
  $content = filter_input(INPUT_GET, $where, FILTER_SANITIZE_STRING);
  $default = filter_var($default, FILTER_SANITIZE_STRING);
  $content = (empty($content)) ? $default : $content;
  if ($content) {
    $html = include 'content/'.$content.'.php';
    return $html;
  }
}
```

Do not let users list your file directories. There are ways to prevent this with the .htaccess file, but if your program is run where you do not have control of the .htaccess file you should take measures yourself. The easiest way to do that is to create a skeleton index.html file and add it to each of your folders that does not have an index.php file. A typical index.html file contains this code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title></title>
</head>
<body>
</body>
</html>
```

To prevent cross-site request forgeries when using forms, use randomly generated tokens on the form and verify those tokens when you use the data.

In the following example, a token is generated on the form. That token is then passed to both the *session* and to the form as a hidden value. Remember that hidden just means that it does not display on the form page. If the form uses the GET method, the hidden field is displayed in the URL.

A session is used for storing data such as gets, posts, and cookies. The session file is stored on the server, not in the user's browser, so is not generally viewable by the user. A cookie holds a session ID that is used to link to the session file. A session lasts until you destroy it or your user closes her browser.

To start a session or access an existing session, put this code at the very beginning of your program:

```
<?php session_start(); ?>
```

On your form, create a random value. For security, you want to *salt* that value. To salt a value is to add an additional piece to it that makes it more difficult to decode. There are many ways of creating salts. You can use a randomly created salt or put a salt constant in a configuration file. Randomly generated salts are the safest if you are using a salt with a password. This token is created by choosing a random number between 1 and 1,000,000 that is concatenated to the `$salt` value. That is then encrypted as a `sha1()` hash:

```
<?php $token = sha1(mt_rand(1,1000000) . $salt); ?>
```

`$token` is assigned to both a session variable `'token'` and a hidden input in the form:

```
<?php $_SESSION['token'] = $token; ?>
<input type='hidden' name='token' value='<?php echo $token; ?>'/>
```

In the program that reads your form values, start the session and compare the session value with the POST/GET value. If the token parameter is not set in either the POST or the SESSION or if it is empty or if they do not match then do not trust the input.

```
<?php
session_start();
if (!isset($_POST['token'])
  || !isset($_SESSION['token'])
  || empty($_POST['token'])
  || $_POST['token'] !== $_SESSION['token']) {
  die('Bad token');
} else {
  $name = filter_input(INPUT_POST, 'name', FILTER_SANITIZE_STRING);
  unset($_SESSION['token']);
}
```

For the best user experience, you want to control how you end a program. However, when dealing with security, it is sometimes better just to get out of the program entirely if you detect an insecurity. In these cases, the `die()` function ends the program immediately and optionally displays a message to a user. You may also want to be wary of how much information you give a user at this point because the user could use that information maliciously.

An additional security precaution against CSRF is to add a confirmation page and to check that both the original request and the confirmation page are processed.

TRY IT

In this Try It, you add an additional security feature to the Case Study. You have been using some security features already, such as sanitizing data from forms. Here you add `index.html` files to the folders in the Case Study. This prevents attackers from seeing your folders and files if they enter a folder name in the URL.

For the second part of the Try It, you create a form with a generated token and check for the valid token before processing the form.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson17 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 16. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Don't forget to add the `index.html` file to subfolders.

You don't need an `index.html` file in your root directory because you already have the `index.php` file there instead. Depending on your server setup, an `index.html` file might be displayed instead of your `index.php` file.

When you use sessions, you need to do a `session_start()` at the very beginning of any file where you want to use the session variables.

Step-by-Step

Prevent users from seeing your folders and files.

1. Create a file called **index.html** that displays a blank screen using the following code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title></title>
</head>
<body>
</body>
</html>
```

2.   Add a copy of that file to the following folders. The only folder that doesn't need an
     `index.html` file is the root folder, which already has the `index.php` file.

     ➤   content

     ➤   css

     ➤   images

     ➤   images/thumbnails

     ➤   includes

     ➤   includes/classes

Create a form and then add a generated token to verify the form before processing in order to pre-
vent CSFR.

1.   Create a standard HTML input form called **exercise17a.php** that asks for a name
     input. The form action is **exercise17b.php** and the method is **post**. The results look
     like Figure 17-3.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Test Form</title>
</head>
<body>

<form action="exercise17b.php" method="post">

<fieldset>
  <legend>Test Form</legend>
  <p><label for="name">Name</label>
  <input type="text" name="name" id="name"  />
  </p>
  <p><input type="submit" name="testform" value="Submit" /></p>
</fieldset>

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

2.   Create the file **exercise17b.php** that processes the form and then displays the name. Run
     `exercise17a.php`, enter a name, and submit. Your results should look similar to Figure 17-4.

```
<?php
$name = filter_input(INPUT_POST, 'name', FILTER_SANITIZE_STRING);
?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Test Form</title>
</head>
<body>

<p>My name is <?php echo $name; ?></p>

</body>
</html>
```

Test Form

Name Andy

Submit

FIGURE 17-3

My name is Andy

FIGURE 17-4

3. Now that you have a form that works, you can add the CSFR prevention features to it. At the very beginning of exercise17a.php, start a session:

```
<?php session_start(); ?>
```

4. Create a token just before the </fieldset>:

```
<?php
$salt = 'SomeSalt';
$token = sha1(mt_rand(1,1000000) . $salt);
```

5. Add that token to the session and to a hidden input:

```
$_SESSION['token'] = $token;
?>
<input type='hidden' name='token' value='<?php echo $token; ?>'/>
```

6. In exercise17b.php, start a session at the very beginning of the file:

```
<?php
session_start();
```

7. Initialize $message to nothing and $badToken to true:

```
$message = '';
$badToken = true;
```

8. Check to see if the token parameter is missing in either the POST or the SESSION, if the token value is empty, and if the tokens from the POST and the SESSION are not equal. If any
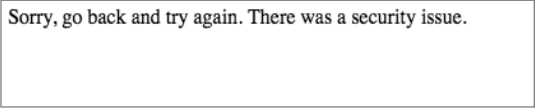
of those conditions exist, set the "bad token" message; otherwise set $badToken to false and set up the $name variable. Unset the session token so that it cannot be used again.

```
if (!isset($_POST['token'])
  || !isset($_SESSION['token'])
  || empty($_POST['token'])
  || $_POST['token'] !== $_SESSION['token']) {
  $message = 'Sorry, go back and try again. There was a security issue.';
  $badToken = true;
} else {
  $badToken = false;
  $name = filter_input(INPUT_POST, 'name', FILTER_SANITIZE_STRING);
  unset($_SESSION['token']);
}
?>
```

9. Change the output so that it displays the name only if there is a good token:

```
<?php  if (!$badToken) : ?>
  <p>My name is <?php echo $name; ?></p>
<?php else : ?>
  <p><?php echo $message; ?></p>
<?php endif ?>
```

10. Run `exercise17a.php` and your results should look the same as before.

11. To test what happens if a bad token is found, temporarily change the session in `exercise17a.php`. To do that, change `$_SESSION['token'] = $token;` to **`$_SESSION['token'] = '12345';`** and rerun `exercise17a.php`. Your results should look similar to Figure 17-5.

Sorry, go back and try again. There was a security issue.

FIGURE 17-5

Watch the video for Lesson 17 on the DVD or watch online at www.wrox.com/go/24phpmysql.

SECTION V
Using a Database

Variables in PHP hold information, but only for the length of the program. You can fill in a form to get information, but you need to store it somewhere. You need a place to hold your data that is still there after the program ends. This section introduces you to what databases are and how to design a database. You learn how to create a MySQL database, how to set up the appropriate tables, and how to use the MySQL scripting language within PHP to create, update, query, and delete information.

18

Introducing Databases

In this lesson you learn about databases, what they are, how to figure out what to put in them, and how to organize them so that you can retrieve the information you need when you need it. Most of the information is true for most databases, but the information covered is in the context of the relational database system MySQL.

WHAT IS A DATABASE?

Databases have terminology all their own. Several terms are also used interchangeably:

➤ **Database:** A database is an organized collection of data. In MySQL you often create separate databases for each of your projects.

➤ **Table:** A table is a collection of similar information. In MySQL you might have a Customers table that contains data about your customers, a Products table that has data about your products, an Order Headers table that contains header and totals about your orders, and an Order Details table that contains the line items on the orders.

➤ **Row:** Inside a table, you have rows. Each row is a related set of data. In the Customers table, each customer is in a row.

➤ **Record:** A record is another word for a row.

➤ **Column:** Inside your table you also have columns. Columns are the types of information you are storing in your table. For instance in the Customers table, name, street, and city would all be columns.

➤ **Field:** A field is another word for a column. Sometimes used to refer to a specific row's column.

➤ **Value:** A value is what is in a given cell. In the Customers table, for instance, you would have a row for George Smith where the value of the cell in the name column is "George Smith."

➤ **Relationship:** A relationship is a link between two tables. For instance, an Order Details table would link to the Order Headers so that you can associate the line items with the correct order.

➤ **Key:** A key is a field, or fields, that link the tables. In the Order Details table you have an order number field that matches an order number field in the Order Headers table. The order number field is a key or key field.

➤ **Index:** An index is an internal system that a database system uses to locate information more quickly. In MySQL you can specify that certain columns, usually keys, are indexes.

There are different types of databases. The simplest are flat files. A `.csv` file is a flat file as are `.ini` files such as the `php.ini` file you may have had to configure in the first lesson. These are in a single file, each row is a complete record, and there are no relationships. Hierarchical databases have hierarchical relationships such as you see in the Windows Explorer folders and files lists. Relational database management systems, such as MySQL, have multiple tables with dynamic relationships that you define and a query language for extracting data in different formats and groupings without having to reorganize the tables.

For your database to be successful, it needs to contain all the data that the business needs. It needs to contain the business rules for processing the data and to protect the data security and integrity. Your database also needs to be able to access the data effectively, yet be flexible enough to handle exceptions and questions you didn't think of when you first created the database. It should have the ability to allow for growth and change.

Research, and a lot of trial and error, has gone into determining the most likely way to meet these goals. The rest of this lesson shows you the best practices in designing your database.

GATHERING INFORMATION TO DEFINE YOUR DATABASE

Before you can design your tables, you need to gather certain information. This part is not technical. It is finding out about the business. You need to know the problem you are trying to solve and the scope. You need to know the purpose and objectives of the database. You need to know the data that has to be retained and what the database needs to do. Ideally, the client provides this information for you, but if he doesn't, you need to find it out.

Find out what you already have. What are people already using? Are there existing databases? What do they contain? What are they missing? Are there spreadsheets that people are using? Forms? Reports? These all give you a handle on the data that that needs to be in the database as well as the type of data it is. MySQL has data types, just as PHP does, but they are more rigid so you need to know what type of data you can expect.

If you have a chance, observing people gathering and working with the data is invaluable. It helps answer the "What did you know and when did you know it?" question. You need to know if there is a specific order when the users find out what certain data is or when they use the data, so you should be familiar with the standard workflow of the data. For instance, if you find out there is a vital required piece of data that is not known until late in the workflow, you know you won't always have access to it. You also need to understand the exceptions that can occur.

After you have all this information, you are ready to start designing your tables.

DESIGNING YOUR TABLES

At this point you have analyzed your data needs. Now it is time to organize a list of the different data pieces you have collected into tables and fields. The nouns (things) are likely to become tables. Adjectives and aspects of a thing are likely to become fields. Data that can be calculated generally does not belong in a database because you can create functions in PHP to do the calculations when they are needed.

Based on the information showing on the Case Study website, Table 18-1 shows you what a database would look like to provide the information needed:

TABLE 18-1: Tables and Fields for the Case Study

TABLE	FIELDS
Contacts	First name, last name, position, e-mail, phone
Lots	Category, lot name, description, image, lot number, price
Categories	Category, description, image

The next task is to determine the characteristics of each of the fields. For instance, which fields are text fields and which are numeric fields? How long do your text fields need to be? Is there any validation that you need? Which fields do you have to have information in from the start? MySQL assigns data types to each field. In Lesson 21, you use the information about what type of data is stored in each of the fields to assign specific data types to the fields when you create your tables. You also use this information to set up business rules such as whether an email is required for all contacts or whether a phone number is a formatted numeric field in the form xxx-xxx-xxxx or a freeform text field. This is what the Contacts table could look like:

Table: Contacts

First name: Text, up to 50 characters long, required

Last name: Text, up to 50 characters long, required

Position: Text, up to 50 characters long

Email: Text, up to 255 characters long

Phone: Text, up to 20 characters long

SETTING UP RELATIONSHIPS BETWEEN TABLES

You link between tables by including in one table a field that identifies a record in another table. For instance, in the Case Study, lots are assigned to a category. So in the Lots table you need a field that matches a uniquely identifying field in the Categories table.

Those fields are called *keys*. The key in the Lots table is called a *foreign key* because it links to something outside the table. This key in the Categories table is the *primary key*, which uniquely identifies the category record. The primary key is often referred to as the id.

Requirements exist for determining what fields can be used as primary keys. These requirements are not forced on you by the database, but if you do not follow them you will have trouble keeping your database working correctly.

- ➤ **Unique:** The primary key must be unique. If it is not unique then you are not able to match to a single record.

- ➤ **Not null and not optional:** There must always be something in the field.

- ➤ **Not changeable:** There should be no reason you should need to change the field. If you use a person's name for the primary key and the person changes his name, then all the tables trying to link with the old name won't work.

- ➤ **Should not violate security policies:** In other words, do not use something as a key that needs to be private, such as a Social Security number or a password.

Many database developers create a new field whose only purpose is to be the primary key. These are called *artificial keys*. Integers are usually used because they are fast to process and do not take up much room in the database. In MySQL, you learn how to create such keys with `auto_increment`. Because these keys are arbitrary and divorced from the business logic, there is no need for them to change. They can be assigned when the record is created, and they can be unique.

Not all foreign keys have to match to a primary key. They could match to a different field that also identifies the record.

Relationships can be one-to-one (driver to driver's license) or one-to-many (teacher to students) or many-to-many (students to courses). The most common relationship between tables is a one-to-many relationship. In a one-to-many relationship, one record in a table links to many records in a second table. In the Case Study the link between the Categories table and the Lots table is a one-to-many.

When tables have a many-to-many relationship, the relationship can be defined with a separate file that contains only the foreign keys. So if you allowed lots to be in multiple categories, your files might look like this:

Table: Lots

Fields: Lot id, lot name, description, image, lot number, price

Table: Categories

Fields: Category id, category name, description, image

Table: Categories-Lots

Fields: Category id, Lot id

There would be multiple records in the Categories-Lots table; one for each category/lot combination that exists.

INSTITUTING THE BUSINESS RULES

Business rules are policies that a business uses to make its business run smoothly and profitably. There are different ways of enforcing business rules. Some are enforced just in the way that you design your database. The Case Study has a business rule that each lot needs to be in one and only one category. By putting a foreign key in the Lots table to the Categories table, you are enforcing that rule.

Data typing is another way to enforce business rules. If a business rule is that prices have to be numeric, you set the data type for the price field to be numeric. Data types in MySQL can also include lengths or sizes. If you have a business rule that no comment can be longer than 100 characters, you can specify that the field is no more than 100 in size.

Some database systems allow you to enforce other aspects of the data. MySQL enables you to force certain fields to be unique or to not be empty. It also enables you to set a default value for fields.

You use validation files if you need to restrict a field to containing only specific values. An example is if you want to restrict a State field to the two-character abbreviation of the state. Validation files in MySQL are just like any other table. They can be a small table with one or two columns or they can be a regular table, such as the Categories table.

Then, finally, you can use program validation in place of or in addition to database validation. MySQL enforces the integrity of the database, but if you try to give it something invalid it gives you an error message. Because you do not want error messages going to your user, you want to verify that the data about to go into the database meets the criteria the database is looking for, and fix it first.

NORMALIZING THE TABLES

If you hang around with people dealing with databases, you might hear them talking about *normalization* or about *normalizing a database*. Normalization is the reorganization of the database so it meets certain design standards. Like the list of requirements for a primary key, these are rules established so that the database ends up more usable. Normalization aims to design databases that are more robust and that will be useful longer. It does this by designing for flexibility, so that you can use the database for a variety of general queries and tasks rather than designing it so that it is optimized for a single task but is unfit for handling future tasks.

Normalization reduces redundancy. Redundancy requires more storage space and introduces maintenance errors. If, instead of just having a key to the categories table in the Lots table, you had the description in all the lots records, you would use more storage. Any time the description changed, you would have to change it in each of the lots.

Normalization ensures that data that is independent in reality is independent in the database. If you have the category description in the Lots table instead of in a separate Categories table, you have to create a lot before you can have a category. If you delete all the lots in a particular category, you lose that category altogether.

To normalize a database, you apply a series of rules to the tables and change them as necessary to pass the rules. There is a balance to be maintained among flexibility, performance, and understandability. There are six levels of normalization, but most databases use only the first three levels, achieving what is called *third normal form*.

First normal form (1NF) states that a field must provide a fact about a key, that there can be no repeating elements, and no multivalued elements. Take the following table:

Table: Lots

Fields: Lots id, lot name, description, image, lot number, price, category id1, category id2, category id3

Category id1, category id2, and category id3 are repeating elements. They allow up to three categories to be assigned to the Lots table. However, if the business rule changes and you now need to assign a fourth category, you have to change the database and all the programs that use that table. Instead use the many-to-many relationship file as shown earlier in this lesson.

> Table: Lots
>
> Fields: Lot id, lot name, description, image, lot number, price
>
> Table: Categories
>
> Fields: Category id, category title, description
>
> Table: Categories-Lots
>
> Fields: Category id, Lot id

Now if a fourth category is needed, there is no problem.

Second normal form (2NF) states that you need the whole key in order to retrieve the data. You need to worry about second normal form only if you have a composite key, which is a key made up of more than one field.

Third normal form (3NF) states that you cannot have any hidden dependencies and no duplicate or calculated fields. Category description in the following table is a hidden dependency because it depends on category id rather than lot id.

> Table: Lots
>
> Fields: Lot id, Lot name, description, image, lot number, price, category id, category description

TRY IT

In this Try It, you design the three database tables for the Case Study website, contacts, categories, and lots. You use this design when you create the tables in Lesson 21. There is no coding for this exercise, but you may want to display the Case Study to see the information that you need to store.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson18 folder in the download.*

Lesson Requirements

There is no coding in this Try It. If you use the figures in the "Step-by-Step" section, you don't need your computer. If you want to look at the Case Study on your computer, you have the following two requirements:

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

If you are following along with the Case Study, you need your files from the end of Lesson 17. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

I used the Case Study tables as examples in this lesson.

Look at the About Us page, Lot Categories page, and the Display Lots pages to see the tables to be created or look at the figures in the Try It.

The lots in the Gents, Sporting, and Women pages can all be in one table.

Step-by-Step

Look at the About Us page. Based on that, design the Contacts table, showing the fields and characteristics. See Figure 18-1.

   1. List the fields needed in the Contacts table, including an artificial key for the primary key: Contact id, first name, last name, position, e-mail, phone.

   2. Write down characteristics to each of the fields.

> Table: Contacts
>
> Contact id: Integer, positive number, required, primary key
>
> First name: Text, up to 50 characters long, required
>
> Last name: Text, up to 50 characters long, required
>
> Position: Text, up to 50 characters long
>
> E-mail: Text, up to 255 characters long
>
> Phone: Text, up to 20 characters long

Look at the Lot Categories page. Based on that, design the Categories table, showing the fields and characteristics. See Figure 18-2.

   1. List the fields needed in the Categories table, including an artificial key for the primary key: Category id, category name, category description, category image.

   2. Write down the characteristics for each of the fields.

> Table: Categories
>
> Category id: Integer, positive number, required, primary key

Category name: Text, up to 50 characters long, required

Category Description: Text, up to 5 or 6 lines of text

Category Image: Text, up to 255 for the name of the file

Look at the Gents, Sporting, and Women pages. Based on that, design the lots table, showing the fields and characteristics. See Figure 18-3, Figure 18-4, and Figure 18-5.

FIGURE 18-1

FIGURE 18-2

FIGURE 18-3

FIGURE 18-4

FIGURE 18-5

1. List the fields needed in the Lots table, including an artificial key for the primary key: Lot id, lot name, description, image, lot number, price. Add a key to link to the Categories table.

2. Write down the characteristics to each of the fields.

> Table: Lots
>
> Lots id: Integer, positive number, required, primary key
>
> Lot name: Text, up to 50 characters long, required
>
> Description: Text, up to 5 or 6 lines of text
>
> Image: Text, up to 255 for the name of the file
>
> Lot number: Integer, positive number
>
> Price: Numeric, up to $100,000.00
>
> Category id: Integer, positive number, link to categories table

---

Watch the video for Lesson 18 on the DVD or watch online at `www.wrox.com/go/24phpmysql.`

19

Introducing MySQL

MySQL is a free open-source relational database management system. It's pronounced either as My S-Q-L or as My Sequel. It is a standard for many shared hosting services and is part of the standard (L)AMP stack of Apache web server, MySQL database, and PHP scripting that runs many of the Internet's sites. Access to MySQL data is based on SQL, a query language developed in the 1970s. Other SQL databases include Oracle (the current owner of MySQL), PostgreSQL, MS SQL Server, and SQLite. Although all are based on SQL, differences exist in the implementation, so the different databases and SQL commands are not interchangeable.

MySQL runs on many platforms including various Unix/Linux versions, Windows, and Mac OS X. It can be used in many different programming languages, including PHP, Java, C#, Visual Basic, ASP, and ColdFusion. Using MySQL in PHP is a combination of running the MySQL statements and using special PHP functions written to interact with MySQL.

MySQL as shipped allows for manipulation at a command line and does not have a graphical interface. The most widespread graphical interface is phpMyAdmin. Because it is generally the interface that is available and because it displays the MySQL statements for the actions you do, it is a good choice to use when learning MySQL.

In the first part of this lesson you learn your way around phpMyAdmin. As you create a database and a table with phpMyAdmin, you get exposure to MySQL statements. In the second part of this lesson you learn the basics of the SQL syntax as used by MySQL.

USING PHPMYADMIN

You installed phpMyAdmin in Lesson 1 as part of XAMPP. You run phpMyAdmin either by going to `http://localhost/xampp` and clicking the phpMyAdmin link on the left as shown in Figure 19-1, or by going directly to `http://localhost/phpMyAdmin` where you see a page similar to Figure 19-2.

FIGURE 19-1
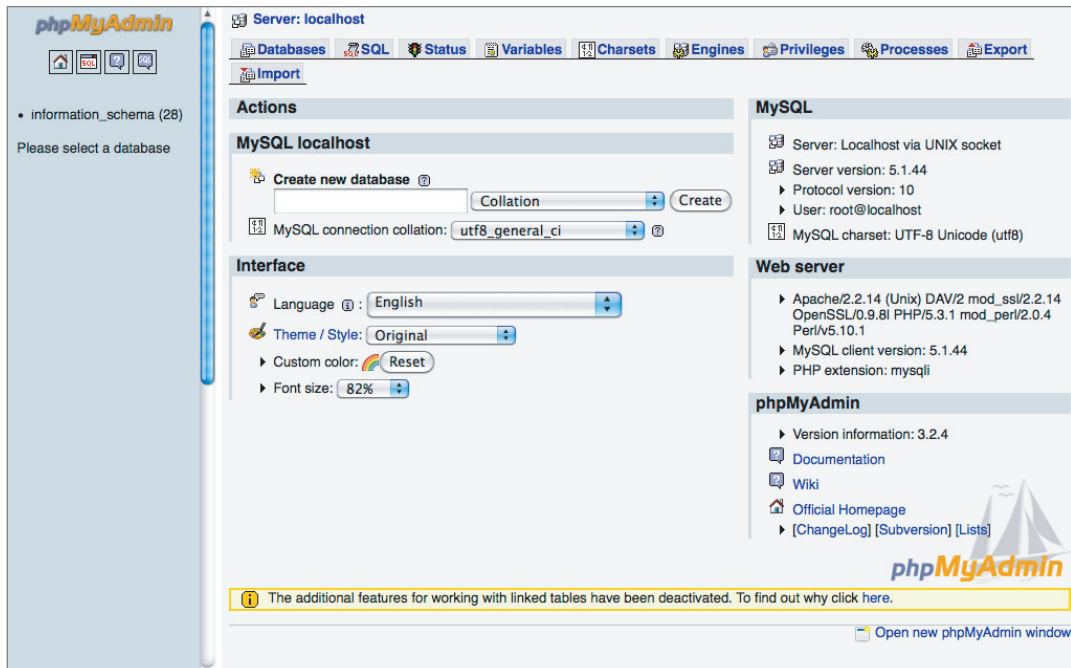
FIGURE 19-2

If you did not install XAMPP and do not have phpMyAdmin, you can freely download it from `www.phpmyadmin.net`.

Creating Databases

On the left in Figure 19-2 is a list of your databases. If at any point you want to return to this screen, click the icon that looks like a house just above the list. You likely have only one database showing, information_schema. This is the database that contains all the information about the MySQL databases on your server. You do not want to touch this.

There are several tabs across the top of the window. This list of tabs and what they refer to changes depending on whether you are at the base level, as you are now, in a selected database, or in a selected table.

At this level the only tab you are likely to use is called Privileges. This is where you add or remove users or change passwords as you learned in Lesson 1. When you are working with MySQL in PHP, you specify a single user for the program to use. You do not create a user for everyone who runs the PHP program. In a production program it is more secure to use a user that has only the privileges needed to run the program.

To create a new user, click on the Privileges tab. On the page that appears, find the Add a New User link as shown in Figure 19-3 and click it.

Add a new User

FIGURE 19-3

Fill in the screen as shown in Figure 19-4. To match the sample code, use `php24sql` for the User Name. Your Host is most likely `localhost`. To match the sample code, use `hJQV8RTe5t` for the Password and Re-type fields.

In the Global privileges section, check the following boxes in the Data section: SELECT, INSERT, UPDATE, DELETE. This gives your user (that is, the program you run) the ability to update the data in the database. Because you have not given the user any privileges in the Structure section, he isn't able to change the structure of your database. Because you haven't given him any privileges in the Administration section, he won't be able to do any administrative tasks. This is important because if a hacker is able to run code through your program, he is more limited in what he can do.

Click the Go button to create this user.

This is the user you use when you need to connect to the database within your program. This does not affect the user you use to run MySQL itself or when you are in phpMyAdmin, so don't change anything in your XAMPP configuration.

FIGURE 19-4

Next, you need to create a database. To create a database you need to know two things: the name of the database and the default *character set* and *collation* the database will use.

A character set is a list of the symbols and their internal representation. At its simplest, you can think of a character set as the alphabet, numbers, and punctuation along with the encoding of those characters. A collation is the set of rules for comparing and ordering those characters. The collation decides not only that A comes before B and 1 comes before 2, but whether or not A and a are equivalent and whether letters come first or numbers.

Character sets can contain not just the characters used in English, but also characters for accented characters used in other languages or the thousands of characters used in other systems of writing. The early common character sets were limited in scope and although they work with English, they

cannot handle many of the world's languages. Use a UTF-8 such as `utf8_general_ci` collation to allow for more languages.

The character set/collation is used only on text data. Fields that are numeric types have no character sets or collations. You can also have text data that has no character set or collation. These are called *binary* strings. An example of a binary string is a field that contains the bytes for an image.

MySQL is flexible with its use of character sets and collations. You can mix and match different character sets and collations on the same server, the same database, and even within the same table. Unless you know what you are doing, however, you can really get into trouble. If possible, use the same character set and collation at all the levels.

Follow these steps as illustrated in Figure 19-5 to create a database called `test`.
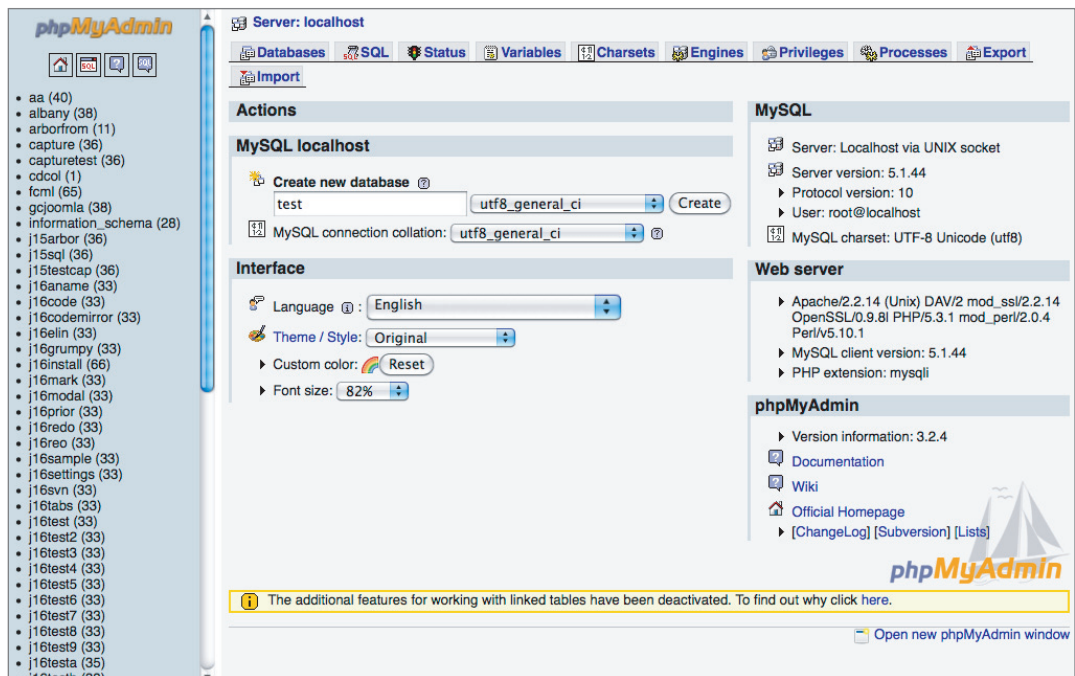
FIGURE 19-5

1. Type `test` in the input box labeled Create New Database. (If you do not see that input box, click the house icon to return to the Home page first.)
2. Click the down arrow in Collation and select `utf8_general_ci`.
3. Click the Create button.

Your results should look similar to Figure 19-6.
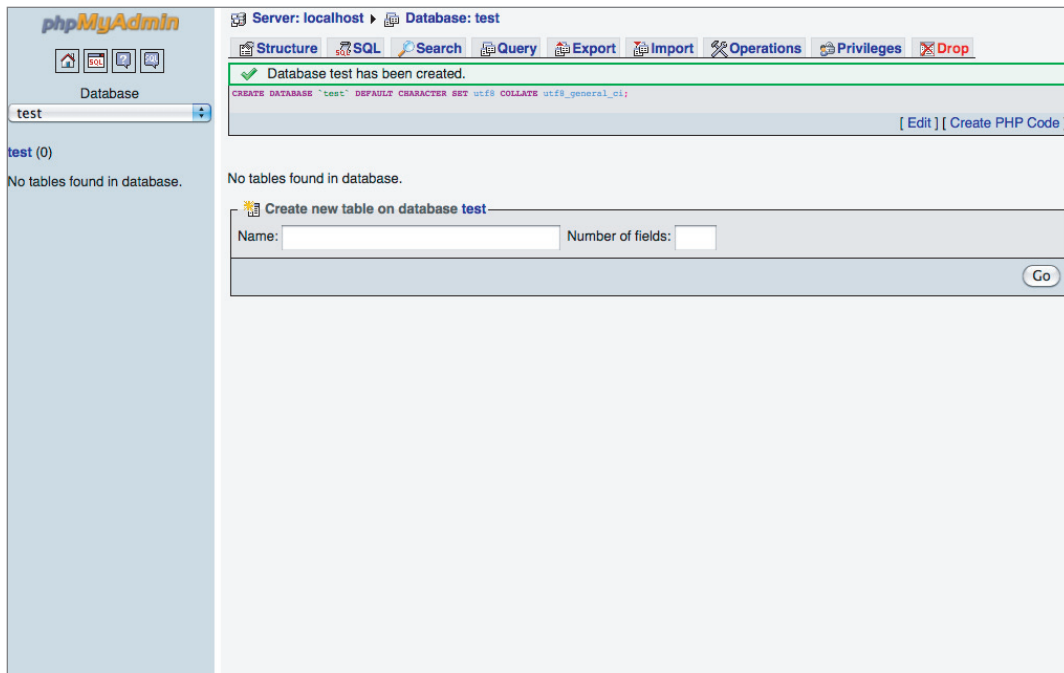
FIGURE 19-6

Defining Tables and Columns

You are now working in your test database and the list on the left in phpMyAdmin displays any tables in the database, rather than a list of databases. Notice the breadcrumbs at the top of the window. This helps you keep track of whether or not you are in a database. Later you see that it also shows if you are in a particular table in the database.

There is a new set of tabs that are based on the actions you perform on a specific database. Just beneath the tabs is a message. This message is either a success message or a specific error message. The action that you just performed is displayed in the box below. What you see here is the actual statement that you use on a command line to perform the same action. If you click the Create PHP Code link, you see the same statement transformed into an assignment to a PHP variable by adding the variable, double quotes around the statement, and a semicolon. Either version can be easily copied with the standard keyboard shortcuts of Ctrl+A, Ctrl+C (on a PC) or Command+A, Command+C (on a Mac).

> *If you click Create PHP Code and then try to return by clicking Without PHP Code, the program tries to rerun your code. Use the back arrow for the browser instead.*

Next comes a list of the tables in this database. There are no tables yet, so you need to create one now:

1. Enter `table1` as the name.

2. Enter `2` for the number of fields (columns) to create.

3. Click the Go button and a window similar to Figure 19-7 is presented.

FIGURE 19-7

4. Fill in the field ID by giving it the name `id`, selecting PRIMARY from the Index drop-down, and clicking the AUTO_INCREMENT checkbox. Selecting Primary for the index tells MySQL that this is a main field that is used to identify records in the table. MySQL creates an index for the field in order to retrieve the data more quickly. Flagging the field as `auto_increment` means that MySQL automatically creates a unique sequential number in this field when a record is added.

5. Fill in the field description by giving it the name `description`, changing the Type drop-down to TEXT, and clicking the Null checkbox to allow nulls to exist. Allowing nulls to exist means that the field is not required. Your window looks like Figure 19-8.

6. Click the Save button to add these two columns to table1 as shown in Figure 19-9.

FIGURE 19-8

FIGURE 19-9

The table table1 is now listed in the left column. To work with a specific table, click that table on the left. If there are records in the table, the Browse tab is activated; otherwise, as in this case, the Structure tab is active as it was in Figure 19-7. You add or change columns (not the data in the columns) in the Structure tab. You can add a column either at the end of the table, at the beginning of the table, or after a given column as shown in Figure 19-10.

FIGURE 19-10

After you click the Go button, the window where you create your new column displays as shown in Figure 19-11. This example creates an integer column called code, where any new rows default to 42 if not set to a different number.

FIGURE 19-11

Clicking the Save button adds the column to table1 as shown in Figure 19-12.
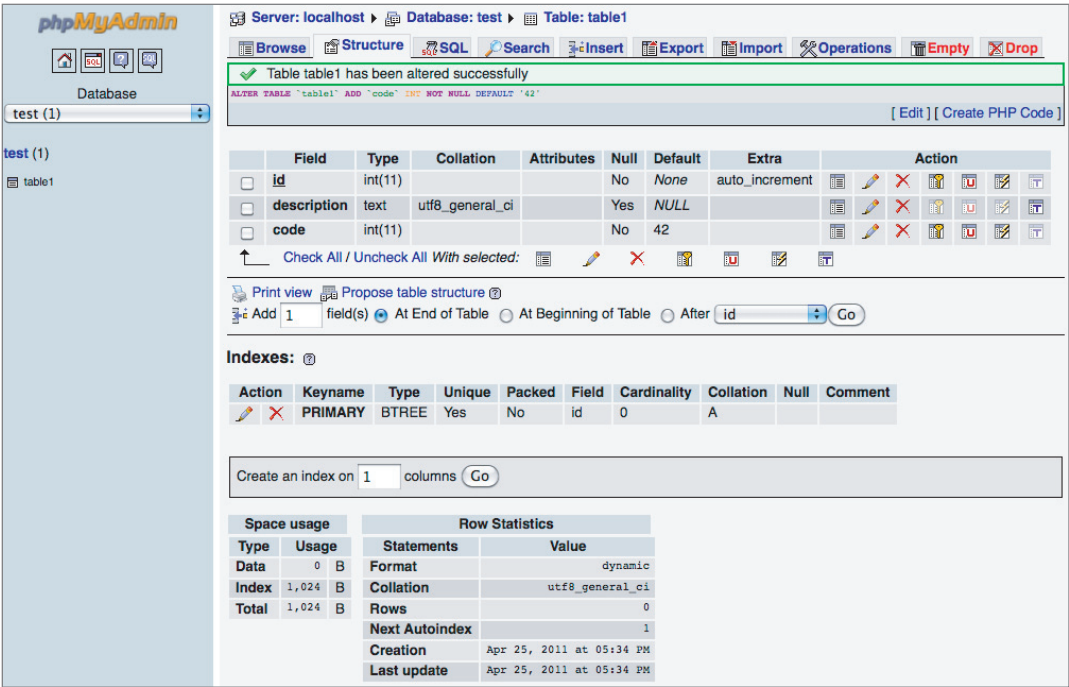
FIGURE 19-12

Entering Data

You now have a database called test, which contains a single table called table1, which has three fields: id, description, and code. At this point you can enter data into your database. To do so, be sure that the table is selected either by clicking the table name on the left or checking the bread-crumb at the top of the window. Clicking the Insert tab opens a window with forms to enter two records. Figure 19-13 shows the forms filled in and ready to be saved.

The first field is id, which is the primary key that is flagged as an `auto_increment` field. When left blank, MySQL automatically assigns the next number in sequence. The next field is description, which is a text field where text can be entered. The code field displays the default of 42 to start, but can that can be changed to a different number. To save both the records, click the Go button in either form. The program jumps to the SQL tab where it displays a status message and the SQL command used to add the records as shown in Figure 19-14.

Now that you have records in the table, clicking the Browse tab displays those records as shown in Figure 19-15.

To delete the records in a table, but leave the structure intact, click the Empty tab. You are asked if you want to TRUNCATE TABLE. Click the OK button to continue with the deletion of the records.

To delete the entire table, including the structure, click the Drop tab. You are asked if you want to DROP TABLE. Click the OK button to continue to delete the table completely.

FIGURE 19-13

FIGURE 19-14

FIGURE 19-15

> ⊗ *Make sure that it says DROP TABLE, not DROP DATABASE. If you are at the database level instead of in a table, the Drop tab deletes the database including all the tables and all the data.*

Backing Up and Restoring

After you have a database, you need to know how to back up that database. Databases are not backed up by creating a copy of a file, but by creating commands that enable you to re-create that database. In phpMyAdmin you can back up the entire database, a single table, or a selected group of tables.

> ⊗ *If you need to back up the entire database, be sure that you are at the database level, not in a table.*

Click the database name on the left side of the window and then click the Export tab to see the window shown in Figure 19-16.

FIGURE 19-16

The form is divided into three sections. On the left is the Export fieldset that enables you select the tables to include and the format to use. On the right is the Option fieldset that lists a series of options that change depending on the format. At the bottom is where you specify how you want to save.

At the top of the Export fieldset is a list of the tables in the database. All the tables are automatically selected, but you can change that to only specific tables. Below that is a list of formats that can be used to export the database. For a normal backup you use SQL.

The Options for the SQL format are displayed on the right. The defaults work well for a standard backup. The Structure fieldset refers to setting up the table: what columns it has, how those columns are defined, and so on. It coordinates with the Structure tab. Here are explanations on common items you might change:

➤ Add DROP TABLE deletes an existing table with the same name if one exists. Check this if you want to replace an existing table.

➤   Add IF NOT EXISTS creates a table only if there is not one. If it finds a table with the same name it leaves it intact. If you use the default Export type of INSERT in the Data fieldset, the data also stays.

➤   Add AUTO_INCREMENT value copies over the current value. If you are bringing over the data as well, you usually need to bring over the `auto_increment` value so that new records start on the right value. If you are just exporting the structure, uncheck this so that new records start at the beginning value.

The Data fieldset deals with the actual data in tables—the records. Leave these at the defaults.

The bottom fieldset is where you tell the program whether you want to save the statements it creates in a file or to display them in the SQL tab box. When saving as a file you have the option of giving a specific name or of setting up a template that automatically creates a name for you. The default template is to use the database name, which is what `_DB_` stands for. In this case, where you are exporting as in the SQL format, the database test is saved as `test.sql`. Files in the `.sql` format are simple text files that contain SQL statements.

To restore a database from an SQL Export, you use the Import tab at a database level. You can restore the tables to the same database or a different database. If you restore into the same database, you need to remove the tables first, either manually with the Drop tab on the table or by including the DROP table in the backup. To restore into a new database, you create a new database and select it.

To restore your test database into a new database, create a new database called testbackup. Click the Import tab. You see a window that looks similar to Figure 19-17.
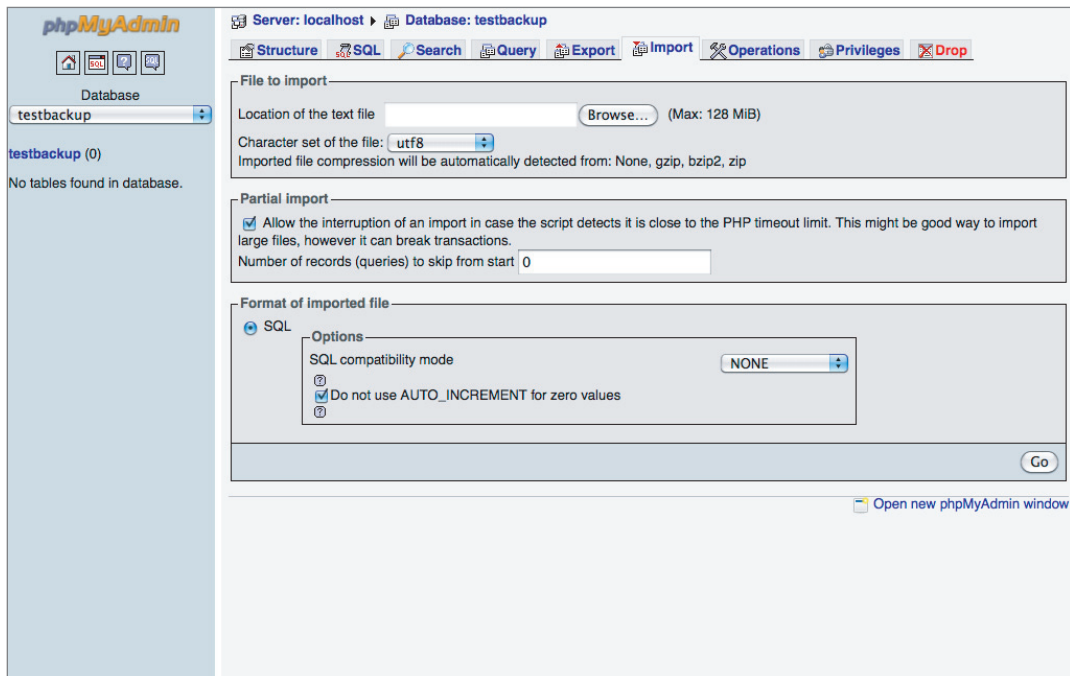
FIGURE 19-17

Click the Browse button for the Location of the Text File input, select the `.sql` file from your hard drive, and click the Go button. The results look similar to Figure 19-18. You have successfully re-created your database.
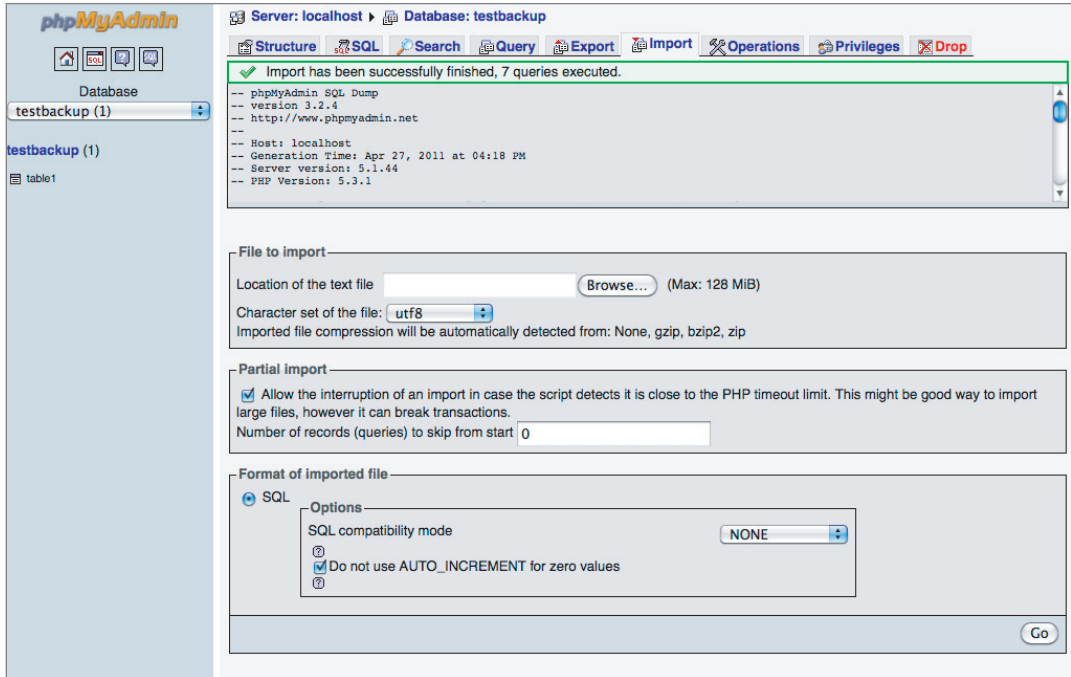
FIGURE 19-18

LEARNING THE SYNTAX

MySQL has an online reference manual for each of the versions of MySQL. You can find the manual for version 5.6 at `http://dev.mysql.com/doc/refman/5.6/en/`. The manual contains a list of all the statements, along with the syntax for each. The manual is for MySQL as used on a command line or in the SQL box of phpMyAdmin. Before trying to use a statement in PHP it can be helpful to be sure that it is a valid MySQL command that works as you expect it to. In this section you learn the common language structure for the statements and the conventions of the manual.

A command is a MySQL statement followed by a semicolon. You can have multiple commands on a single line and a single command can be on multiple lines.

There is a difference between `.sql` files and statements within PHP programs. In the .sql files the only things in the file are MySQL statements. When you use MySQL in PHP programs, the actual MySQL statement is contained within other PHP code and may have variables inside it.

Literal Values

Text strings in MySQL are quoted. You can use either single quotes or double quotes. As in PHP, single quotes and double quotes are usually, though not always, interchangeable. Standard SQL uses

only single quotes, so that is preferred. If you have a quote as part of the text, you either need to enclose the text with the other type of quote mark, escape the quote with a backslash (\), or double it.

Numbers are not enclosed in quotes. They can be preceded by a - or a + to show whether the number is negative or positive. Number types that are allowed to have decimals can also have a decimal separator (.). Scientific notation is also allowed.

Dates and times can either be quoted strings or numbers depending on the circumstances. You learn more about dates and times in Lesson 21.

The Boolean True and False evaluate to 1 and 0 and are not case sensitive.

Identifiers

Identifiers are the names of objects in MySQL such as databases, tables, indexes, and columns. They can be up to 64 characters long. Identifiers that contain only alphanumeric characters, underscores (_), and $ do not have to be enclosed in quotes unless they contain a reserved word. There is a list of reserved words in the manual at `http://dev.mysql.com/doc/refman/5.6/en/reserved-words.html`. Special characters are encoded so that could cause issues if the character set changes.

The quote mark for identifiers is the backtick (`). If you are using a QWERTY keyboard, this is the little ticky-mark under the tilde (~) usually in the far upper-left side of your keyboard.

> *The backtick (`) is not the same as the single quote ('). You will get errors if you use the wrong one in the wrong place. The easiest way to remember which to use is that the backtick goes on the names of things (databases, tables, columns) whereas the single quote goes on values and around statements and commands.*

Identifiers can be qualified or unqualified as long as the unqualified identifier is unambiguous. For example, if you have a database named test and a table named table1 and a column named id it could be written as

```
id
table1.id
test.table1.id
`id`
`table1`.`id`
`test`.`table1`.`id`
```

Notice that if you quote the names in a qualified identifier, you quote each name separately. If an identifier is unqualified, the defaults are used.

Technically databases and tables are not case sensitive but they map to files in the filesystem on the host. Depending on your operating system, files in the filesystem are case sensitive. Windows is not case sensitive but if you develop on Windows and then move your work to an online host that is Linux, your code could fail. For that reason, it is recommended that you code as if your identifiers are case sensitive.

Comments

If you are using MySQL through your PHP program you usually only use MySQL comments if you create a backup from phpMyAdmin or if you have a `.sql` file to create your database structure and set up data when doing an install of your program.

Comments come in different flavors in MySQL:

➤ Use a `#` character to comment everything from the character to the end of the line.

➤ Use `--` (double-dash plus space/tab/newline) to comment everything from the dashes to the end of the line.

➤ Use `/*` to `*/` for commenting multiple lines as in PHP.

Do not nest comments. There are times when it works, but many more when it doesn't. So if you have a block of statements to comment out, don't try to do it by surrounding the block with `/* */` if there are comments within the statements.

TRY IT

In this Try It, you create a database for the Case Study. Based on part of the database analysis from the previous lesson, you create the Contacts table and fill it with data. You back up the database and restore it in a second database.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com*. You can find them in the Lesson19 folder in the download. You will find code for the* smithside.sql *file.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a working copy of phpMyAdmin, which comes as part of XAMPP. You can download a free copy at www.phpmyadmin.net.

Hints

See the database analysis done for the Case Study in the Try It section of Lesson 18.

Make sure that you are in the right level, whether it is global, database, or table. Use the indicators on the left column or the breadcrumbs at the top of the window.

Step-by-Step

Create the Case Study database.

1. Open phpMyAdmin.

2. In the input box labeled Create New Database enter **smithside**.

3. Change the Collation drop-down from `Collation` to `utf8-general-ci`.

4. Click the Create button. Your results look similar to Figure 19-19.
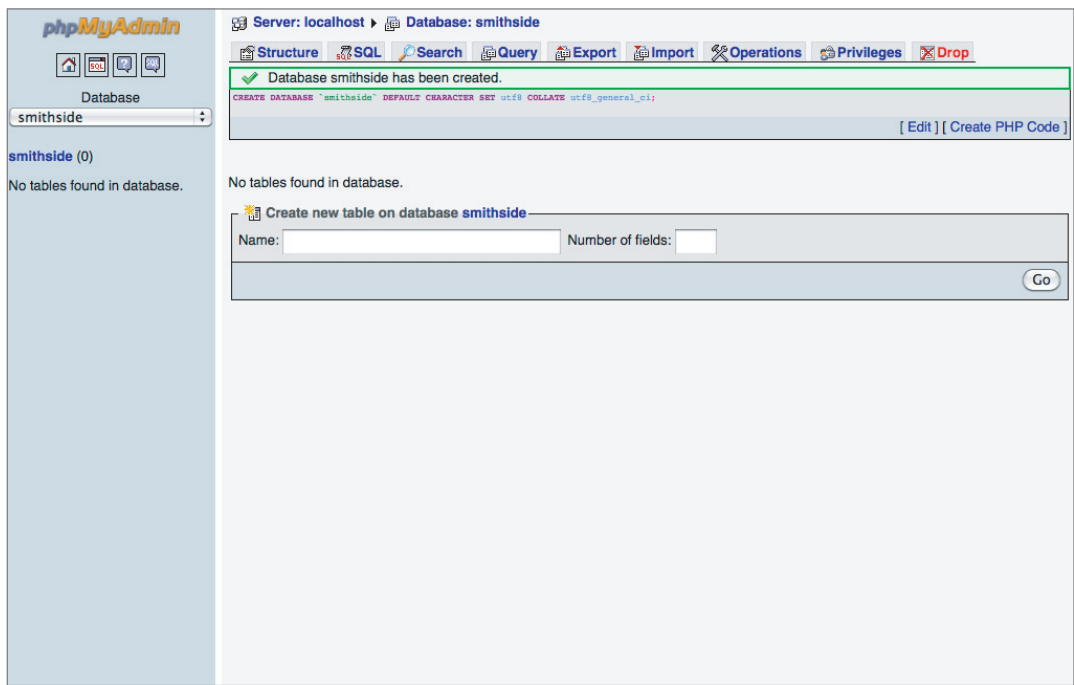
FIGURE 19-19

Create the contacts table with the fields id, first name, last name, position, email, and phone.

1. In the fieldset called Create New Table on Database smithside, type **contacts** as the name.

2. Type **6** for the Number of Fields.

3. Click Go to see a window that looks similar to Figure 19-20. Because there are several fields, each of the field definitions stretches horizontally, instead of vertically in the lesson example.

4. The first field is id. Fill in the following information, scrolling the screen to see the rest of the window as necessary. Steps 4 through 9 are shown in Figure 19-21.

   Field: **id**

   Type: **INT**

Length: **11**

Default: **None**

Attributes: **UNSIGNED**

Index: **PRIMARY**

A_I: Check this box for AUTO_INCREMENT
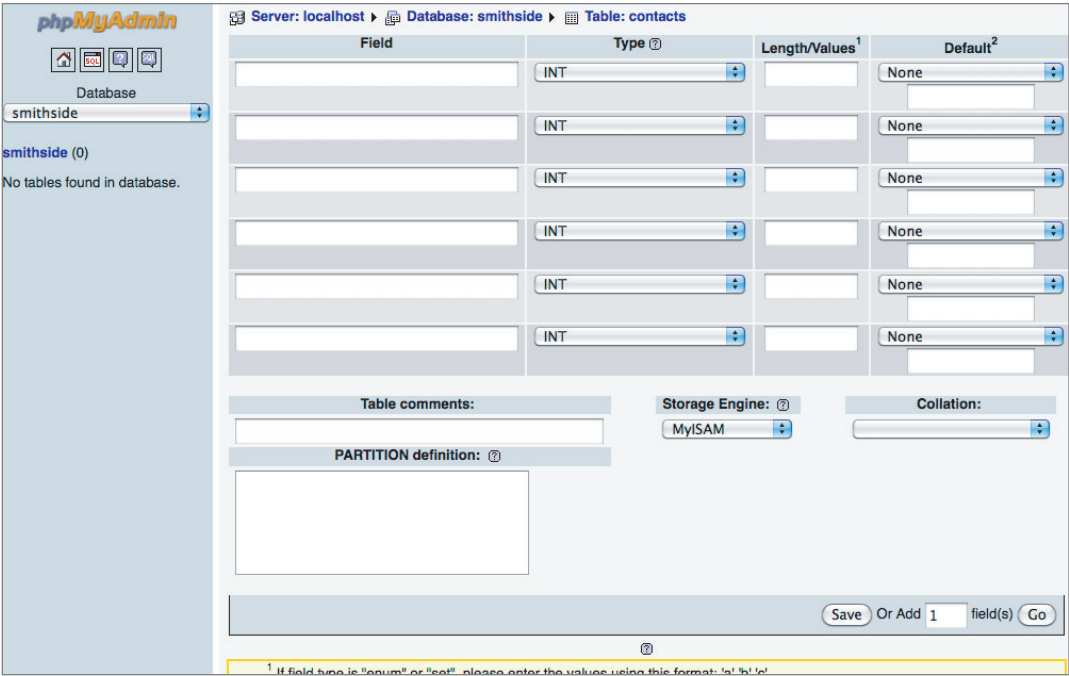
FIGURE 19-20

5. Type in the information for the next field, first name. This is a text field of up to 50 charac-
ters. VARCHAR stands for Variable Characters and lets you specify a maximum number of
character positions. You learn more about types in Lesson 21.

Field: **first_name**

Type: **VARCHAR**

Length: **50**

Default: **None**

6. Type in the information for the next field, last name.

Field: **last_name**

Type: **VARCHAR**

Length: **50**

Default: **None**

7.   Type in the information for the next field, position.

Field: **position**

Type: **VARCHAR**

Length: **50**

Default: **None**

Null: Because position is not a required field, check the Null box to indicate that a null is permitted.

8.   Type in the information for the next field, email.

Field: **email**

Type: **VARCHAR**

Length: **255**

Default: **None**

Null: Because email is not a required field, check the Null box to indicate that a null is permitted.

9.   Type in the information for the next field, phone.

Field: **phone**

Type: **VARCHAR**

Length: **20**

Default: **None**

Null: Because phone is not a required field, check the Null box to indicate that a null is permitted.

Field	Type ⑦	Length/Values[1]	Default[2]	Collation	Attributes	Null	Index	A_I
id	INT	11	None			☐	PRIMARY	☑
first_name	VARCHAR	50	None			☐	---	☐
last_name	VARCHAR	50	None			☐	---	☐
position	VARCHAR	50	None			☑	---	☐
email	VARCHAR	255	None			☑	---	☐
phone	VARCHAR	20	None			☑	---	☐

FIGURE 19-21

10.   Click the Save button. Your results should look similar to Figure 19-22.
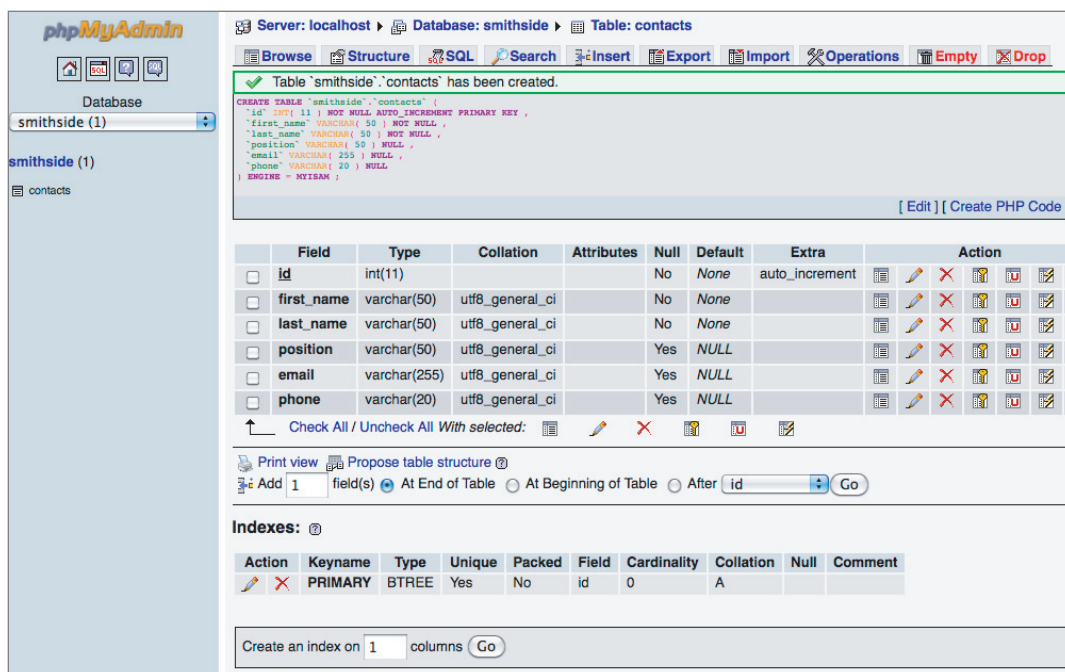
FIGURE 19-22

Create six contact records. You can use the data from the Case Study or create your own.

1. In phpMyAdmin in the contacts table, click the Insert tab.

2. At the bottom of the page, change Restart Insertion With to 10 rows.

3. Enter the contacts. Leave the ids blank so that they are automatically calculated. Do not click the Go button until you have entered all six contacts. Use the Tab key to navigate. If you press Enter, whatever you have entered so far is saved. If that happens, click the Insert tab and insert the remaining contacts. If you were in the middle of a contact, see step 4 to fix that contact. Your result should look like Figure 19-23.

➤ `Martha Smith, position: none, martha@example.com`

➤ `George Smith, postion: none, george@example.com, 515-555-1236`

➤ `Jeff Meyers, hip hop expert for shure, jeff@example.com`

➤ `Peter Meyers, position: none, peter@example.com, 515-555-1237`

➤ `Sally Smith, position: none, sally@example.com, 515-555-1235`

➤ `Sarah Finder, Lost Soul, finder@a.com, 555-123-5555`

4. Click the Browse tab to see your contact data as shown in Figure 19-24. If you need to change any of the data, click the pencil at the beginning of the row you need to change.
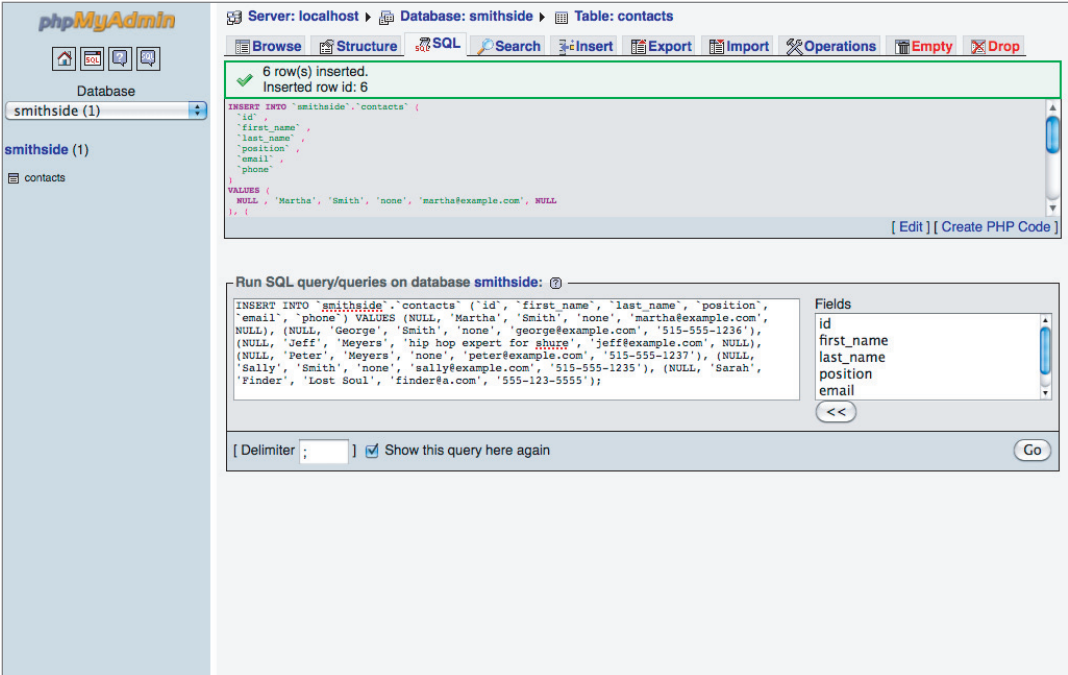
FIGURE 19-23
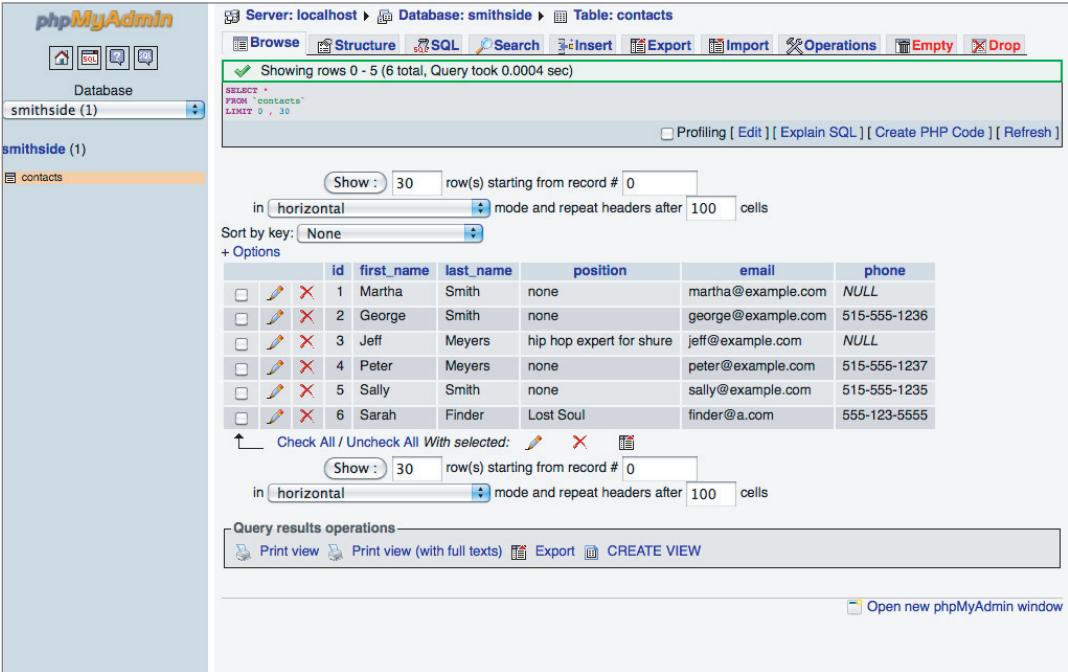
FIGURE 19-24

Back up your database and restore it in a different database.

1. Click the database smithside on the left column.

2. Click the Export tab to see the window shown in Figure 19-25.
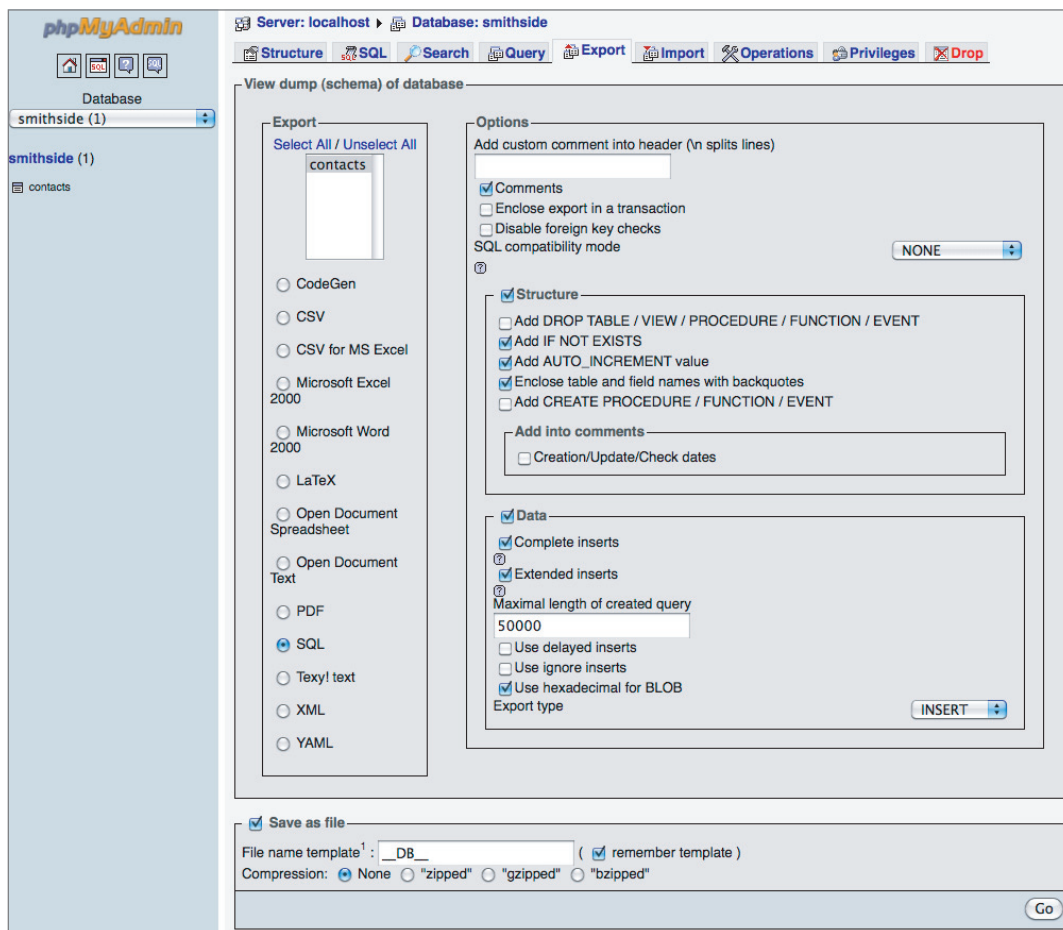
FIGURE 19-25

3. Save the `smithside.sql` file.

4. Click the house icon on the upper-left side.

5. Create a database called **ssbackup** with the `utf8_general_ci` collation. You should see Database: ssbackup appear in the breadcrumbs at the top of the window.

6. Click the Import tab to see the window shown in Figure 19-26.

7. Click the Browse tab. Find and select the `smithside.sql` file you saved.

FIGURE 19-26

8. Click the Go button. You get a message saying the import was successful and the table(s) in the database display on the left column.

9. Click the contacts table in the left column to see the six contacts you added.

> Watch the video for Lesson 19 on the DVD or watch online at `www.wrox.com/ go/24phpmysql`.

20

Creating and Connecting to the Database

In this lesson, you learn the interfaces that PHP has to communicate with a MySQL database. The original method was with an extension to PHP called mysql. You can still find this in older code, but it has been replaced by the mysqli extension for PHP 5. Mysqli is an improved, more secure version that takes advantage of features added to newer versions of MySQL. It is recommended for new work.

PDO (PHP Data Objects) is an extension to PHP for connecting to various databases, not just MySQL. The same PHP code enables you to connect to MySQL, PostreSQL, and SQLite databases, among others. You use different drivers to switch from one database type to the other.

CONNECTING WITH MYSQL/MYSQLI

Mysql was the traditional way to communicate with MySQL. When mysqli came along in PHP 5, it added the following features:
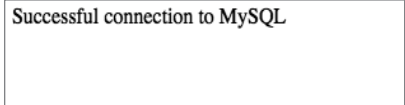
➤ **Object-oriented interface:** Mysqli has the `mysqli` class, the `mysql_stmt` class for queries, and the `mysqli_result` class for results. Each of these has properties that give you information on the connection, the request you are making, or the data you have retrieved. They also have methods that enable you to perform actions. There is still a procedural interface similar to mysql if you prefer to use that.

➤ **Support for prepared statements:** With prepared statements you set up the request once and then send the particulars for the actual request. Reusing the same type of requests works faster than creating a new request each time. More importantly, it is more secure because it cuts down on injection attacks.

➤ **Support for multiple statements:** You can process multiple statements at a time rather than one by one.

➤ **Support for transactions:** You use transactions when you have multiple changes to a database that should be thought of as a group. An example is when you transfer money from one account to another. A transaction ties the subtraction from one account to the addition to the other so that if one action fails the other fails as well.

To connect to MySQL you need to know the hostname (which is usually localhost), username, and password. You create an instance of the class `mysqli` to establish a connection. The following code connects to MySQL running on localhost and uses the username `root` and the password `12345`. The object `$connection` can be called anything. Other common variable names are `$db`, `$dbh`, and `$mysqli`.

```php
<?php
$connection = new mysqli('localhost', 'php24sql', 'hJQV8RTe5t');
```

If there is an error with the connection, the error is put in the property `connect_error` for the object you just created. Use the `if` statement to check for errors. The following example displays the error message if there is an error. If you are in a production site you should give a message to the user without details because the details could be used to hack the

Successful connection to MySQL

FIGURE 20-1

system. If there is no error a success message is displayed. The `@` before the `new` suppresses the normal error handling. Without the `@`, errors with the connection are automatically displayed. Use the `@` if you want to handle the errors yourself. Remember to change the configuration information to match your setup. Your results should look similar to Figure 20-1.

```php
<?php
$connection = @new mysqli('localhost', 'php24sql', 'hJQV8RTe5t');
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL';
}
```

You should have one place to go for the connection information. That way you have one place to go to if you change a username or password and you have the ability to store the information separately such as in a configuration file. One way is to use constants as the following code does:

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");

$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL';
}
```

You have successfully connected to MySQL, but before you can use a database, you need to connect to the database. The database must already exist before you can connect to it. In the previous lesson,

you created a database called smithside in phpMyAdmin. Adding the database name as the fourth parameter selects that particular database.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "smithside");

if ($connection = new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB)) {
  echo 'Successful connection to MySQL';
}
```

You are now at the same point as you were in the previous lesson when you opened phpMyAdmin and clicked on a database. In phpMyAdmin you could see a list of the tables. You can use your new database connection object to get a list of the tables in the database.

The `mysqli` class has a method called `query()`. You pass it a MySQL statement and it returns an object of the `mysqli_result` class. You then use the properties and methods of that object to see your results. The MySQL command to see a list of tables is SHOW TABLES. The MySQL command names are not case sensitive, but it is standard practice to capitalize them. Assuming that `$connection` is your connection object, the following code executes the SHOW TABLES command and creates `$result` as an object based on the `mysqli_result` class:

```php
$result = $connection->query("SHOW TABLES");
```

The `mysqli_result` class property `num_rows` contains the number of rows. Because `$result` is based on the `mysqli_result` class it also has `num_rows` as a property.

```php
$count = $result->num_rows;
```

The `mysqli_result` class method `fetch_array()` returns the results in the form of an array for each record, which in this case is each table. The first element in the array contains the table name.

```php
$row = $result->fetch_array();
echo $row[0];
```

This finds only the first table. To get a list of all the tables you use a `while` loop. The script continues to loop through the results until it reaches the end.

```php
while ($row = $result->fetch_array()) {
  echo $row[0]. '<br />';
}
```

The following is what the full code looks like to create a connection and list the tables in the database. Your results should look similar to Figure 20-2.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "smithside");
```



```
Successful connection to MySQL
Tables: (1)
contacts
```

FIGURE 20-2

```php
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';
  if ($result = $connection->query("SHOW TABLES")) {
    $count =$result->num_rows;
    echo "Tables: ($count)<br />";
    while ($row = $result->fetch_array()) {
      echo $row[0]. '<br />';
    }
  }
}
```

You have been learning the object-oriented style of mysqli. There is also a procedural type of mysqli. The following is the same program using the procedural style:

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "smithside");

$connection = @mysqli_connect(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if (mysqli_connect_error()) {
  die('Connect Error: ' . mysqli_connect_error());
} else {
  echo 'Successful connection to MySQL <br />';
  if ($result = mysqli_query($connection, "SHOW TABLES")) {
    $count = mysqli_num_rows($result);
    echo "Tables: ($count)<br />";
    while ($row = mysqli_fetch_array($result)) {
      echo $row[0]. '<br />';
    }
  }
}
```

> 🖉 *For a complete list of the properties and methods of the mysqli classes see the documentation at* `www.php.net/manual/en/mysqli.summary.php`.

Whether you use the object-oriented style or the procedural style, you use the variable that contains the connection (in this case $connection) to access the database throughout your program. Many older programs use a global variable to hold the connection. The connection itself goes in an initial program such as the following code:

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "smithside");
```

```php
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
}
```

When the database is needed, the `$connection` variable is declared as global:

```php
<?php
global $connection;
if ($result = $connection->query("SHOW TABLES")) {
  $count = $result->num_rows;
  echo "Tables: ($count)<br />";
  while ($row =$result->fetch_array()) {
    echo $row[0]. '<br />';
  }
}
```

The other way is to create a database class that contains the connection. You can also use this class to contain other database-related functions. Start by defining the database connection information as properties. The properties are private because they should be seen only inside the class. Make them static. Static variables are associated with the class rather than a specific object.

```php
<?php
class Database
{
  private static $_mysqlUser = 'php24sql';
  private static $_mysqlPass = 'hJQV8RTe5t';
  private static $_mysqlDb = 'smithside';
  private static $_hostName = 'localhost';
```

Add a property for the connection variable, initializing it to NULL:

```php
protected static $_connection = NULL;
```

Create a method to get the connection. Make this a public static function that returns the connection property. Remember that you address static properties and functions with `self::` instead of `$this->`.

```php
public static function getConnection() {
  self::$_connection = new mysqli(self::$_hostName, self::$_mysqlUser,
    self::$_mysqlPass, self::$_mysqlDb);
  if (self::$_connection->connect_error) {
    die('Connect Error: ' . self::$_connection->connect_error);
  }
  return self::$_connection;
}
```

If you leave the method as it is, every time you create a new object you create a new database connection, which uses extra memory. The solution is to make the class a *singleton*. A singleton class is limited to a single instance. Each time you use the class, you get the same instance instead of creating a new object every time you use the database.

Check first thing to see if the connection already exists. If it doesn't, then create the object and do the error checking. Whether the connection already existed or you just created it, return the

connection. That way, you only make the connection the first time through. All the rest of the time you just return the $_connection.

```
public static function getConnection() {
    if (!self::$_connection) {
        self::$_connection = new mysqli(self::$_hostName, self::$_mysqlUser,
          self::$_mysqlPass, self::$_mysqlDb);
        if (self::$_connection->connect_error) {
          die('Connect Error: ' . self::$_connection->connect_error);
        }
    }
    return self::$_connection;
}
```

To prevent programmers from creating an object by using new Database you make the $__construct() method private:

```
private function __construct(){}
```

Outside of the class, call the getConnection method to get the $connection property. To call this static method in the Database class, use Database::getConnection as shown in the following code:

```
<?php
$connection = Database::getConnection;
if ($result = $connection->query("SHOW TABLES")) {
  $count = $result->num_rows;
  echo "Tables: ($count)<br />";
  while ($row = $result->fetch_array()) {
    echo $row[0]. '<br />';
  }
}
```

PHP closes the database connection when the script is done, but you can close it yourself if you are done with the database. The object-oriented version looks like this:

```
<?php
$connection = Database::getConnection();
$connection->close();
```

This is the procedural style for closing a database:

```
<?php
$connection = Database::getConnection();
mysqli_close($connection);
```

> *Mysqli has an object-oriented style and a procedural style. You also have the choice of sharing the connection using a global variable or using the object-oriented method. These two decisions are separate. You can use the object-oriented mysqli with the global variable or with a static singleton class.*

CONNECTING WITH PDO

Mysqli uses an extension that is created specifically to talk to the MySQL database. As such it is able to take advantage of all the features of the MySQL database. However, if you want PHP programs that are more portable — that are able to be easily adapted to other flavors of databases — then the PDO extension is the extension to use. The following is a list of PDO features.

➤ **Object-oriented interface:** PDO has the main `PDO` class, the `PDOStatement` class for prepared statements, and the `PDOException` class for errors.

➤ **Support for prepared statements:** With prepared statements you set up the request once and then send the particulars for the actual request. Reusing the same type of requests works faster than creating a new request each time. More importantly, it is more secure because it cuts down on injection attacks. Prepared statements in PDO are easier to use than the mysqli prepared statements.

➤ **Support for transactions:** You use transactions when you have multiple changes to a database that should be thought of as a group. An example is when you transfer money from one account to another. A transaction ties the subtraction from one account to the addition to the other so that if one action fails the other fails as well.

The concepts for using PDO are similar to using mysqli. The following code connects to MySQL running on the `localhost` and uses the username `php24sql` and the password `hJQV8RTe5t`. It sets up a connection to the smithside database and displays a message when it successfully connects. Your results should look similar to Figure 20-3.

> Successful connection to MySQL

FIGURE 20-3

```php
<?php
if ($connection = new PDO('mysql:host=localhost;dbname=smithside', 'php24sql',
  'hJQV8RTe5t')) {
  echo 'Successful connection to MySQL';
}
```

As with mysqli, you can use constants to set up the connection:

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "smithside");

if ($connection = new PDO('mysql:host='.HOSTNAME.';dbname=' . MYSQLDB,
  MYSQLUSER, MYSQLPASS)) {
  echo 'Successful connection to MySQL';
}
```

The `PDO` class has a method called `query()`. You pass it a MySQL statement and it returns an object of the `PDO` class. You then use the properties and methods of that object to see your results.
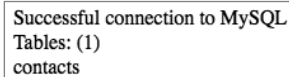
The MySQL command to see a list of tables is SHOW TABLES. Assuming that $connection is your connection object, this code executes the SHOW TABLES command and creates $result as an object based on the PDO class:

```
$result = $connections->query("SHOW TABLES")
```

The PDO class method fetch(PDO::FETCH_NUM) returns the results in the format specified, which in this case is a numeric array. The format is in the form of an array for each record, which in this case is each table. The first element in the array contains the table name.

```
$row = $result->fetch(PDO::FETCH_NUM);
echo $row[0];
```

This finds only the first table. To get a list of all the tables you use a while loop. The script continues to loop through the results until it reaches the end. The following is what the full code looks like to create a connection and list the tables in the database. Your results should look similar to Figure 20-4.

```
Successful connection to MySQL
Tables: (1)
contacts
```

FIGURE 20-4

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("MYSQLDB", "smithside");
define("HOSTNAME", "localhost");

// set up the Database connection
if ($connection = new PDO('mysql:host='.HOSTNAME.';dbname=' . MYSQLDB,
  MYSQLUSER, MYSQLPASS)) {
    echo 'Successful connection to MySQL<br />';
  if ($result = $connection->query("SHOW TABLES")) {
    echo "Tables:<br />";
    while ($row =$result->fetch(PDO::FETCH_NUM)) {
      echo $row[0]. '<br />';
    }
  }
}
```

> *For a complete list of the properties and methods of the PDO classes, see the documentation at* www.php.net/manual/en/book.pdo.php.

PDO can share the connection using either a global variable or the singleton class described in the mysqli section.

CREATING THE DATABASE

In the previous lesson, you created a database using phpMyAdmin. That is the normal way to create databases. Because you need more privileges to create databases than you do to use them, the creation is often a separate function from standard programs. In this example you need to use your root user or any user that has the privilege of creating databases.

To create a database, connect to MySQL and run the `CREATE DATABASE` command. This is the MySQL command to create a database called mydatabase:

```
CREATE DATABASE 'mydatabase';
```

After you have created the database, you need to select it for use before you can use it. Running the following code gives you a result similar to Figure 20-5:
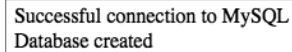
```
<?php
define("MYSQLUSER", "root");
define("MYSQLPASS", "p##V89Te5t");
define("HOSTNAME", "localhost");

if ($connection = new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS)) {
  echo 'Successful connection to MySQL <br />';
  if ($result = $connection->query("CREATE DATABASE 'mydatabase'")) {
    $connection->select_db('mydatabase'); // use the database
    echo "Database created";
  } else {
    echo "Problem creating the database. Is the user not allowed
      to create database or does the database already exist?";  }
}
```

> Successful connection to MySQL
> Database created

FIGURE 20-5

Note that the preceding code uses an equal sign in the `if` statement:

```
if ($result = $connection->query("CREATE DATABASE 'mydatabase'")) {
```

The way that this statement is processed is that the statement on the right is evaluated first, which attempts to create the database. That function returns a value, which in this case is `TRUE` or `FALSE`. That value is then assigned to `$result`, which is then evaluated to determine if the code enclosed by the `if` statement should be run.

TRY IT

In this Try It, you create a database class for the Case Study. It holds a mysqli object-oriented type of connection to the database.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson20 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 17. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Static properties and functions belong to the class, not to individual objects. Normally when you refer to properties or functions within a class, you use the $this->*propertyname* construction. The $this refers to the objects' values so when you don't have an object, you have no $this. Use self::$ instead.

Step-by-Step

In the Case Study, create a database class to connect to the smithside database.

1. Create a file called **database.php** in the includes/classes folder.

2. Create a class called **Database**:

```php
<?php
/**
 * Database class
 * For one point of database access
 */
class Database
{
// rest of the code will go here
}
```

3. Within the class, create properties for the username, password, database, and hostname. Default the properties to the appropriate values. These values won't change and they should only be seen inside the class, so make them private and static. Because they are private, start the class names with an underscore.

```php
    /**
     * User name to connect to database
     * @var string $_mysqlUser
     */
    private static $_mysqlUser = 'php24sql';
    /**
     * Password to connect to database
     * @var string $_mysqlPass
     */
    private static $_mysqlPass = 'hJQV8RTe5t';
    /**
     * Database name
     * @var string $_mysqlDb
     */
    private static $_mysqlDb = 'smithside';
    /**
     * Hostname for the server
     * @var string $_hostName
```

Try It | **273**

```
  */
 private static $_hostName = 'localhost';
```

4. You also need a property called `$_connection` to hold the actual connection:

```
/**
 * Database connection
 * @var Mysqli $connection
 */
private static $_connection = NULL;
```

5. Set up a public static function called `getConnection()`. Connect to the database using the object-oriented mysqli extension. The new object is assigned to the `$_connection` property. If there is an error, print the error and end the program. Return the connection. Because this is a static method, use the `self::$propertyname` construction to refer to the properties.

```
/**
 * Get the Database Connection
 *
 * @return Mysqli
 */
public static function getConnection() {
  self::$_connection =
    new mysqli(self::$_hostName, self::$_mysqlUser, self::$_mysqlPass,
      self::$_mysqlDb);
  if (self::$_connection->connect_error) {
      die('Connect Error: ' . self::$_connection->connect_error);

  }
  return self::$_connection;
}
```

6. You want to do the connection only once so surround the connection with an `if` statement that checks to be sure there is no connection before it tries to connect. The full `getConnection()` method looks like the following code:

```
public static function getConnection() {
  if (!self::$_connection) {
    self::$_connection = @new mysqli(self::$_hostName, self::$_mysqlUser,
      self::$_mysqlPass, self::$_mysqlDb);
    if (self::$_connection->connect_error) {
      die('Connect Error: ' . self::$_connection->connect_error);
    }
  }
  return self::$_connection;
}
```

7. You want to control this class so that it cannot be used to create an object using the `new Database` construct. Making the `__construct()` method a private method prevents that.

```
/**
 * Constructor
 */
private function __construct(){
}
```

To test your database class, temporarily display a statement on the Home page on the Case Study to show what database you are connected to.

1. In the `index.php` file, find `<div class="message">`. Enter all the following code in that div. Get the connection by making a static call to the `Database` class `getConnection()` method.

   ```php
   <?php
   $connection = Database::getConnection();
   ```

2. `SHOW DATABASE()` is the MySQL command to display the database name. Enter the following data to run the command:

   ```php
   if ($result = $connection->query("SELECT DATABASE()")) {
     $row = $result->fetch_array(MYSQLI_NUM);
     echo '<p>*** Using database ' .  $row[0] . ' ***</p>';
   } ?>
   ```

3. Run the program. Your results should look similar to Figure 20-6.
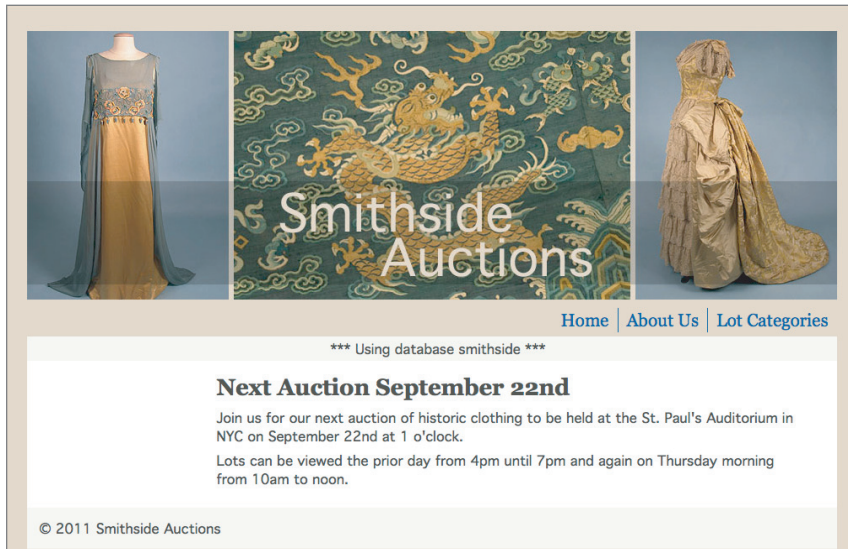
FIGURE 20-6

4. Remove the test message from `index.php` that you just added in steps 1 and 2.

> Watch the video for Lesson 20 on the DVD or watch online at `www.wrox.com/go/24phpmysql`.

21

Creating Tables

In this lesson you learn how to set up detail specifications of MySQL tables. First you learn about the different data types and attributes and then you learn how to use that information to create the tables.

You learn what the different data types are and how to assign them to fields. You learn how to set up a field so that it is automatically assigned a value when a row is added, whether that is a MySQL-calculated value to be used as an arbitrary key or a constant value to be used as a default.

You also learn advanced functions in phpMyAdmin to create and change tables and how to do the same thing in PHP. Finally, you learn how to create and use a text file to perform MySQL commands.

UNDERSTANDING DATA TYPES

Just as in PHP, MySQL has different data types for the fields. The data types in MySQL are stricter than in PHP and there is not a one-to-one correlation.

Strings

There are two types of strings in MySQL. The first is text strings, which have character sets and collations. This is the type of string that you use most often. Text strings are further defined as follows:

➤ `CHAR`: This is the character data type. You define exactly how many characters are stored. For example, if you want a field to be exactly six characters long, you define it as `CHAR(6)`. If you pass it data that is less than that, it pads with spaces at the end. If you pass it more, the extra characters are truncated. Whether you are truncating blanks or non-blank characters and what error reporting you have set determines what, if any, errors you see. You can go all the way up to 255 characters. Note that some character

sets require more than 1 byte to store some characters. The size limits for the text strings are based on the number of characters, not the number of bytes. Trailing spaces are removed when you retrieve the data.

➤ VARCHAR: This data type has a variable number of characters. You specify the maximum number of characters, up to 65,535. If you have a field that could contain up to 50 characters but would likely contain less, you define it as VARCHAR(50). There is a little overhead when using VARCHAR rather than CHAR because 1 or 2 bytes are used to store the length. Trailing spaces are not removed when you retrieve the data.

➤ TEXT: There are four TEXT types — TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. Like VARCHAR, the TEXT types contain a variable number of characters. The difference between the four types is the maximum number of characters. The type defines the maximum number of characters; you do not. See Table 21-1.

TABLE 21-1: TEXT Type Sizes

TEXT TYPE	MAXIMUM CHARACTERS
TINYTEXT	255
TEXT	64K
MEDIUMTEXT	16M
LONGTEXT	4G

The second type of string is binary strings, which have no character sets or collations. Character strings contain text, whereas binary strings contain raw data such as images and other media. The binary types are subdivided in the same way that the text strings are, but the size limits are based on the number of bytes, not the number of characters.

➤ BINARY: This is the binary data type. You define exactly how many bytes are stored. For example, if you want a field to be exactly 6 bytes long, you define it as BINARY(6). You can go all the way up to 255 bytes.

➤ VARBINARY: This data type has a variable number of bytes. You specify the maximum number of bytes, up to 65,535. If you have a field that could contain up to 50 bytes but would likely contain less, you define it as VARBINARY(50). There is a little overhead when using VARBINARY rather than CHAR because 1 or 2 bytes are used to store the length.

➤ BLOB: There are four BLOB types — TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. Like VARBINARY, the BLOB types contain a variable number of bytes. The difference between the four types is the maximum number of bytes. The type defines the maximum number of bytes; you do not. See Table 21-2.

TABLE 21-2: BLOB Type Sizes

BLOB TYPE	MAXIMUM BYTES
TINYBLOB	255
BLOB	64K
MEDIUMBLOB	16M
LONGBLOB	4G

Numeric

As in PHP, numbers that do not have a decimal point are integers. MySQL has different integer types that are based on the size of the integer. Additionally, if the integer is SIGNED — that is, has both negative and positive values — the range starts in the negative numbers. If the field is flagged as UNSIGNED, the values start at 0 and go twice as high, as you see in Table 21-3.

TABLE 21-3: Integer Types

INTEGER TYPE	RANGE IF SIGNED	RANGE IF UNSIGNED
TINYINT	–128 to 127	0 to 255
SMALLINT	–32,768 to 32,767	0 to 65,535
MEDIUMINT	–8,388,608 to 8,388,607	0 to 16,777,215
INT (or INTEGER)	–2,147,483,648 to 2,147,483,647	0 to 4,294,967,295
BIGINT	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0 to 18,446,744,073,709,551,615

You may see these types written with a length such as TINYINT(1) or TINYINT(4). This refers to the number of digits to be displayed. It does not affect the value that is stored or the space needed to store the value.

If you need decimals you use either a floating-point data type or a fixed-point data type. The floating-point types are similar to PHP floating-point types. FLOAT uses 4 bytes of storage and DOUBLE (also called DOUBLE PRECISION or REAL) takes 8. MySQL allows you to specify the total number of digits and the number of digits after the decimal point. So to specify a number between –999.9999 and 999.9999 you use FLOAT(7,4). MySQL rounds the decimal when storing it rather than truncating it if it is too long.

Just as in PHP, floating-point numbers are inexact because of how the computer stores them. You use the exact value data type of DECIMAL (also called NUMERIC) if you need to store exact numbers. Exact values take up more room than the floating-point numbers. Currency is often stored using the decimal format. As with the FLOAT data type, you can specify the total number of digits and the number of digits after the decimal point. DECIMAL(10,2) has a range of –99999999.99 to 99999999.99.

> *The* DECIMAL *data type has been evolving in MySQL to come closer to the SQL standard. You may come across different behaviors in earlier versions of MySQL.*

Date and Time

MySQL stores dates and times in the format of YYYY-MM-DD HH:MM:SS, unlike PHP. Your MySQL server dictates where you can store invalid dates or whether all invalid dates should be converted to zeros.

➤   DATETIME contains the date and the time. It has a range from the year 1000 through the year 9999.

➤   DATE contains just the date value.

➤   You can use TIMESTAMP to automatically contain the initial value or automatically update when something changes on the row. It has a range from 1970 through early 2038. It stores all values as of the UTC time zone.

➤   TIME displays the time portion of a date or an elapsed time.

➤   YEAR displays the year. It can be either YEAR(2) or YEAR(4) for two- or four-digit representation of the year. It has a range from the year 1901 through 2155. Two digits between 00 and 69 are converted to 2000 through 2069 and 70 to 99 are converted to 1970 through 1999.

You can specify dates and times in several ways. Here are examples for January 8, 2013, at 8:30 a.m. Remember that the # starts a comment.

```
'2013-01-08 08:30:00' # the full information
'13-01-08 08:30:00'   # two digit year is fine
'13-1-8 8:30'         # you do not need leading zeros
'13^1+8 8/30'         # any punctuation works
'130108083000'        # no delimiters is fine
'13.01.08' # just the date, if that's all you need
'13/01/00' # you can use 00 for a missing part (not TIMESTAMP)
```

If you are updating TIME, using colons indicates a time rather than elapsed time:

```
'0830' # is an elapsed time of 8 mins, 30 secs
'08:30' # is 8:30 am
```

Other Data Types

MySQL has a data type ENUM that restricts the field to values from an enumerated list of values. Although you can use numbers as values, it is not recommended because errors can easily occur. Numbers can be misinterpreted as an index of a value instead of the value itself.

```
ENUM('small', 'medium', 'large')
```

Although you can put your business logic here, if there's any chance it might change, it would be better to put the value checking in your program where it would be easier to make changes.

MySQL has two other data types that you may come across:

➤   SET includes zero or more values from a defined list.

➤   BIT stores data at the bit level using binary values.

USING AUTO_INCREMENT

You learned in Lesson 18 that the primary key for a table should have the following characteristics:

➤   Unique

➤   Not Null

➤   Not optional

➤   Never needs to be changed

➤   Does not violate security policies

In addition, a short simple key that can be retrieved quickly helps performance. It can be difficult to find a data field that meets all of these requirements. For that reason, tables are often given artificial keys — arbitrary keys that have no meaning other than to be a primary key.

MySQL supports this policy with the AUTO_INCREMENT attribute. You assign this attribute to a field and MySQL generates a unique sequential number for each new row. You can assign AUTO_INCREMENT to either an integer or a floating-point data type, though an integer is the most common. Make sure that the data type you choose is large enough to hold the highest number you need. The following snippet of code shows the typical specifications for an artificial primary key. The name of the field is id; it is an integer data type that is unsigned, is a required field, will be automatically filled by MySQL, and is assigned as the primary key.

```
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

The table keeps track of the next number to be assigned. It starts with 1 unless you tell it differently when you create the table.

In Lesson 22, you learn how to add data to the tables. When you add the data, if you do not assign a value to id for new rows, or you assign a NULL or 0, a value is automatically assigned. MySQL

has a function, `LAST_INSERT_ID()`, that contains the last `AUTO_INCREMENT` value. PHP also has a function that can retrieve this number if you need it.

UNDERSTANDING DEFAULTS

Default values are used to automatically assign a value to a field when you create a new row and you do not assign a value to that field. If you have a required field (`NOT NULL`), you should either always assign a value through your program, assign a default, or, in appropriate cases, set it up as `AUTO_INCREMENT`.

Defaults must be constants. You cannot use MySQL functions, except that you can define a field of the `TIMESTAMP` data type to default to `CURRENT_TIMESTAMP`. The following line of code, included when you create a table, creates a field called `dept` that is up to 20 characters long, is required, and defaults to `Office` on new rows unless a different value is assigned:

```
`dept` VARCHAR(20) NOT NULL DEFAULT 'Office',
```

The following code is for a field called `startdate` that is a date, is required, and defaults to zeros on new rows unless a different value is assigned:

```
`startdate` DATE NOT NULL DEFAULT '0000-00-00'
```

A field does not need to be required to have a default. The following code is for a field called `display_number` that is an integer with a maximum range of 255 that defaults to 20:

```
`display_number` TINYINT UNSIGNED NULL DEFAULT '20'
```

Defaults are not allowed on the `TEXT` or the `BLOB` types. If you need to assign a default and the size permits it, create the field as a `VARCHAR` or `VARBINARY` data type instead.

There are implicit defaults on some data types. These implicit defaults are used if you don't assign a value or you assign the value of `NULL` and you have no explicit default. If you have `NOT NULL` specified and the implicit default has to be used, you get an error depending on the strictness of your error reporting. These are the implicit defaults:

➤ Numeric data types default to 0

➤ Date and time (except for `TIMESTAMP`) are set to the appropriate version of `0000-00-00 00:00:00`

➤ `TIMESTAMP` defaults to the current date and time

➤ Strings default to an empty string

➤ `ENUM` defaults to the first value

CREATING TABLES IN PHPMYADMIN

Now that you understand the details behind creating fields, go back to phpMyAdmin and create another table in the test database. You did this in Lesson 19, but now you should understand what you are doing. Open phpMyAdmin and select the test database or create a new database. Your window should look similar to Figure 21-1.
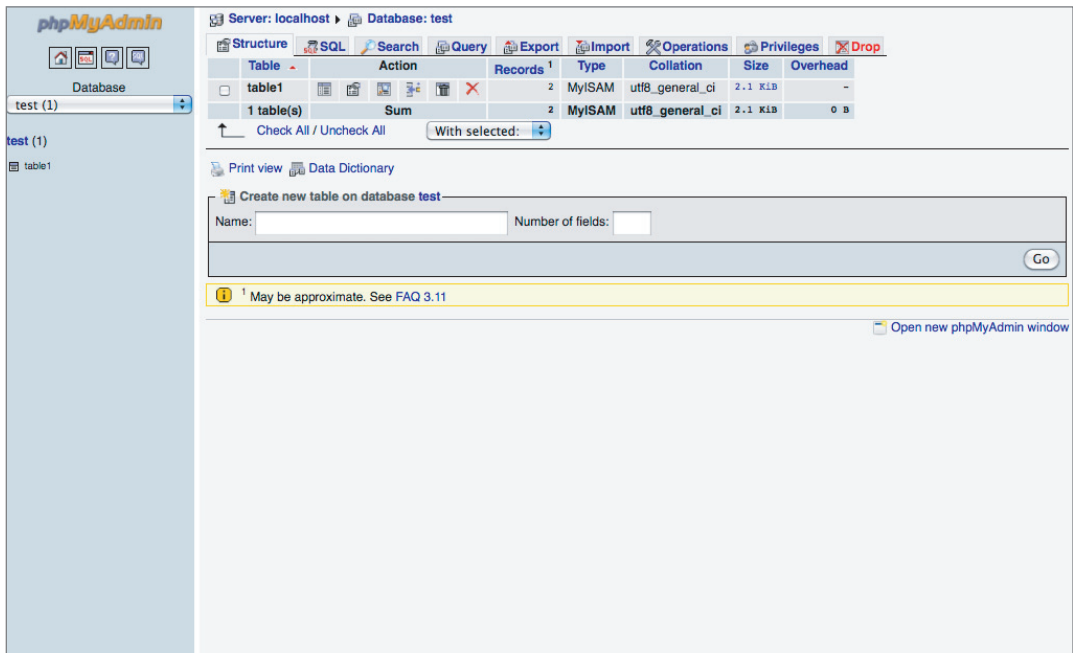
FIGURE 21-1

You create a table called `products` with the following fields: `id`, `product`, `description`, `source`, `date_created`.

Type in the new table name **products** and the number of fields as **5** and click the Go button.

Enter the following information in the five fields as shown in Figure 21-2:

➤ `id`: integer, unsigned, required, auto increment (primary key)

  ➤ Select `UNSIGNED` under the Attribute drop-down.

  ➤ Required is the same as `NOT NULL`, so do not check the Null checkbox.

  ➤ `AUTO_INCREMENT` is the A.I. checkbox. Check that box.

  ➤ This is the primary key, so select `PRIMARY` from the Index drop-down.

➤ `product:` up to 20 characters, required

➤ `description`: long description, required

➤ `source`: up to 20 characters, default to `External`

  ➤ To specify the default, select `As Defined` in the Default drop-down and then type **External**.

  ➤ This field is not required, so check the Null checkbox.

➤ `date_created`: timestamp when created

  ➤ To specify the default, select `CURRENT_TIMESTAMP` in the Default drop-down.

  ➤ This field is not required, so check the Null checkbox.

FIGURE 21-2

Below the list of fields are the attributes for the table itself. You should already be familiar with the collation from creating databases. Usually you can leave this blank. It defaults to the database collation.

The Storage Engine either defaults to MyISAM or to InnoDB. You can think of storage engines like the engines in a car. Different engines are good for different things. MySQL has several engines each with their own capabilities; the most common are MyISAM and InnoDB. MyISAM has been the default for many years. It's a very good basic engine with very good performance. InnoDB has more capabilities, but those capabilities can come with a performance hit. If you create applications with complex transactions, you should use InnoDB. For the database you are creating for this website, MyISAM is a good choice. The newest versions of MySQL are now shipping with InnoDB as the default.

Click the Save button to create the table. You see a window similar to Figure 21-3.
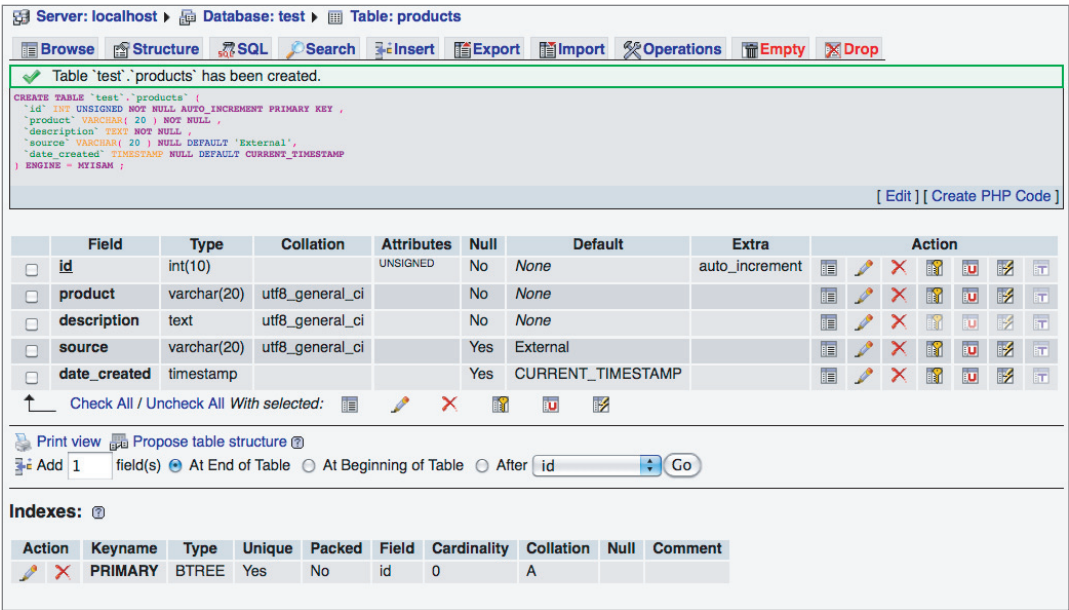
FIGURE 21-3

The MySQL command that was used to create the table is displayed at the top of the window. Copy that and paste it into a text file for the next section.

If you need to change fields after you have created them, go to the Structure tab and click the pencil icon for the field you want to change. Make your changes and click the Save button. To add fields, use the Add fields form below the list of fields as shown in Figure 21-4.

FIGURE 21-4

To add fields, select the number of fields that you want to add and whether they are to be added at the end of the table, at the beginning of the table, or after a given existing field. Click the Go button to go to the form to add fields. Enter the information for the fields and click the Save button to create the fields.

To change table attributes, go to the Operations tab, where you can change the name, the engine, the next AUTO_INCREMENT value to use, and the collation.

USING .SQL SCRIPT FILES

In the preceding section you created a table using phpMyAdmin. You copied the following code to a text file:

```
CREATE TABLE `test`.`products` (
`id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY ,
```

```
`product` VARCHAR( 20 ) NOT NULL ,
`description` TEXT NOT NULL ,
`source` VARCHAR( 20 ) NULL DEFAULT 'External',
`date_created` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE = MYISAM ;
```

The online MySQL manual uses 200 lines to show the syntax of the CREATE TABLE command at `http://dev.mysql.com/doc/refman/5.6/en/create-table.html`. This is a simplified version:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name (
    col_name data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string'],
    col_name data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string']
) table_options;
```

Here are the valid data types:

```
      BIT[(length)]
    | TINYINT[(length)] [UNSIGNED] [ZEROFILL]
    | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
    | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
    | INT[(length)] [UNSIGNED] [ZEROFILL]
    | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
    | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
    | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
    | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
    | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
    | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
    | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
    | DATE
    | TIME
    | TIMESTAMP
    | DATETIME
    | YEAR
    | CHAR[(length)]
        [CHARACTER SET charset_name] [COLLATE collation_name]
    | VARCHAR(length)
        [CHARACTER SET charset_name] [COLLATE collation_name]
    | BINARY[(length)]
    | VARBINARY(length)
    | TINYBLOB
    | BLOB
    | MEDIUMBLOB
    | LONGBLOB
    | TINYTEXT [BINARY]
        [CHARACTER SET charset_name] [COLLATE collation_name]
    | TEXT [BINARY]
        [CHARACTER SET charset_name] [COLLATE collation_name]
    | MEDIUMTEXT [BINARY]
        [CHARACTER SET charset_name] [COLLATE collation_name]
    | LONGTEXT [BINARY]
        [CHARACTER SET charset_name] [COLLATE collation_name]
    | ENUM(value1,value2,value3,...)
        [CHARACTER SET charset_name] [COLLATE collation_name]
```

```
| SET(value1,value2,value3,...)
    [CHARACTER SET charset_name] [COLLATE collation_name]
| spatial_type
```

Here are some common table options:

```
ENGINE [=] engine_name
AUTO_INCREMENT [=] value
[DEFAULT] CHARACTER SET [=] charset_name
[DEFAULT] COLLATE [=] collation_name;
```

Words in uppercase are keywords and words in lowercase are to be replaced with the actual values. The keywords do not need to be in uppercase, though it is convention to use uppercase for them. In some cases, such as `tbl_name` and `col_name`, you can put anything as long as it conforms to the proper naming requirements. In other cases, such as `data_type` and `engine_name`, you are restricted to actual data types and engines.

Items in the square brackets (`[]`) are optional, choices are separated by the pipe symbol (`|`), and the list of fields is contained within the parentheses. This example shows two fields but you can enter as many fields as you need. Commas separate the fields and there is no comma before the final parenthesis. After the list of fields come the table attributes, which are separated by commas. The command ends with a semicolon.

You can create temporary tables by using the `TEMPORARY` keyword. The table is automatically deleted when the connection is closed and can be seen only by that connection.

If you try to create a table and it already exists you get an error unless you have specified `IF NOT EXISTS`. If the table exists and you have added `IF NOT EXISTS`, the new table is not created and you get no error message. Any commands that follow — for instance, to add rows — work on the already existing table. There is no way in MySQL to verify that the table has the same structure. If you want to totally replace any table that exists, use the `DROP TABLE tbl_name` command before `CREATE TABLE IF NOT EXISTS tbl_name`.

The table name can be just the table name, if it is in the default database, or you can specify the database using the dot notation:

```
CREATE TABLE `mydatabase`.`myproduct`
```

If you quote the names, remember that the quote is a back tick (`` ` ``) not a single quote (`'`) and that you quote the database and the table separately.

In the text file you created, change the name of the `products` table to **products2** and save the file with a `.sql` extension. The extension is just a convention. You could call it anything you want, but the `.sql` extension lets you and other programmers know that the file consists of SQL commands. The export file that you created in the prior lesson was the same type of `.sql` file.

Go to phpMyAdmin, select the database, and go to the Import tab. Browse for the text file you just saved. You see a window that looks similar to Figure 21-5.

Click the Go button to run the commands in the `.sql` file to create the `products2` table. Click the `products2` table that appears on the left column to see the list of fields shown in Figure 21-6.

By putting your commands in a file, you can save them, modify them, and rerun them when needed. This is a good place to put those commands that you need to create your database for your application in case you need to re-create it.
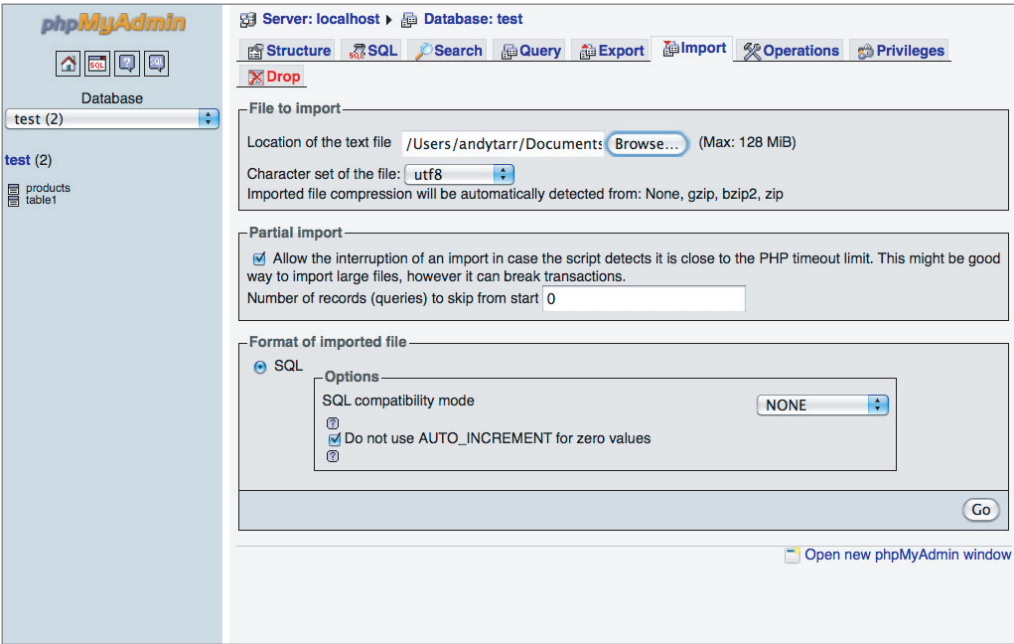
FIGURE 21-5

To change a table after it has been created, use the ALTER TABLE command. Many of the specifications are the same as for creating a table. For the details, see the online MySQL manual at `http://dev.mysql.com/doc/refman/5.6/en/alter-table.html`.
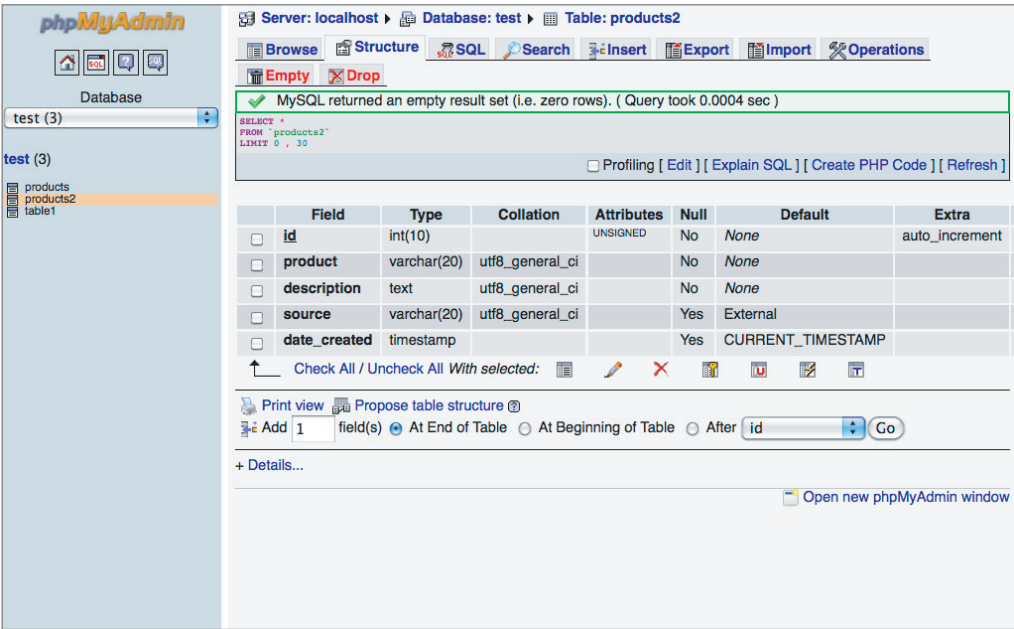
FIGURE 21-6

ADDING MYSQL TABLES TO PHP

Normally you create the tables for your application ahead of time either through a program like phpMyAdmin or by running a `.sql` file. However, there are times when you want to create them within your PHP program. If you do create tables in a PHP program, the MySQL user needs the structure privilege of CREATE. For this example, use your root user. Use the following steps:

1. Make a connection to the database.
2. Create a safe query with the command.
3. Run the query.

The following code uses the same MySQL command as in the previous section to create the table `products3`. You could just enclose the command in double quotes and assign it to the `$query` variable. It is not good practice to have extremely long lines and in practice you might need to add in PHP variables. You can use double quotes around appropriate sections of the command and put them back together with the concatenation operator (`.`). The `query()` method does only one statement at a time and does not use a semicolon at the end of the MySQL statement. Your results should look similar to Figure 21-7.

```php
<?php
define("MYSQLUSER", "root");
define("MYSQLPASS", "p##V89Te5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';
  // Create the MySQL command by copying the command and
  // splitting into shorter lines and concatenating with periods
  // Drop the final semicolon on the MySQL commmand
  // but don't forget the semicolon for ending the PHP command
  $query = "CREATE TABLE `test`.`products3` ( "
      . "`id`  INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY , "
      . "`product` VARCHAR( 20 ) NOT NULL , "
      . "`description` TEXT NOT NULL , "
      . "`source` VARCHAR( 20 ) NULL DEFAULT 'External', "
      . "`date_created` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP"
    . ") ENGINE = MYISAM";
  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "Unable to create table<br />";
  } else {
    echo "Table successfully created<br />";
  }
  // Show the tables
  if ($result = $connection->query("SHOW TABLES")) {
    $count = $result->num_rows;
    echo "Tables: ($count)<br />";
    while ($row =$result->fetch_array()) {
```

```
        echo $row[0]. '<br />';
      }
    }
  }
```

```
Successful connection to MySQL
Table successfully created
Tables: (4)
products
products2
products3
table1
```

FIGURE 21-7

TRY IT

In this Try It, you create the tables for the lots and lot categories in the Case Study using the database analysis you created in Lesson 18.

You use the classes corresponding to the tables in the Case Study, so this is a good time to start creating the classes for the lots and lot categories.

You created a class for the contacts in a Lesson 13 and then in Lesson 19 you created the table. In this Try It, you add the missing table fields to the Contact class.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson21 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 20. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Code to create the contacts, categories, and lots tables is in the `install.sql` file in the download-able code.

You can use the `includes/classes/contact.php` file as a basis for your new classes.

Step-by-Step

Create the `categories` and `lots` tables.

1. Based on the database analysis from Lesson 18, create the `categories` table with the following characteristics:

Table: `categories`

`cat_id`: Integer, positive number, required, primary key, auto increment

`cat_name`: Text, up to 50 characters long, required

`cat_description`: Text, up to 5 or 6 lines of text

`cat_image`: Text, up to 255 for the name of the file

You can use the graphical interface in phpMyAdmin, copy the following code into the SQL tab for the Case Study database, or copy the code to a `.sql` file and import it.

```
CREATE TABLE `smithside`.`categories` (
`cat_id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
`cat_name` VARCHAR(50) NOT NULL,
`cat_description` TEXT NULL,
`cat_image` VARCHAR(255) NULL
) ENGINE = MYISAM;
```

2. Based on the database analysis from Lesson 18, create the `lots` table with the following characteristics:

Table: `lots`

`lot_id`: Integer, positive number, required, primary key, auto increment

`lot_name`: Text, up to 50 characters long, required

`lot_description`: Text, up to 5 or 6 lines of text

`lot_image`: Text, up to 255 for the name of the file

`lot_number`: Integer, positive number

`lot_price`: Numeric, up to $100,000.00, default to 0

`cat_id`: Integer, positive number, link to `categories` table

You can use either the graphical interface in phpMyAdmin, copy the following code into the SQL tab for the Case Study database, or copy the code to a `.sql` file and import it.

```
CREATE TABLE `lots` (
  `lot_id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `lot_name` varchar(50) NOT NULL,
  `lot_description` TEXT NULL,
  `lot_image` VARCHAR(255) NULL,
  `lot_number` INT(11) UNSIGNED NULL,
  `lot_price` DECIMAL(10,2) DEFAULT '0' NULL,
  `cat_id` INT(11) UNSIGNED NULL
) ENGINE=MyISAM
```

Create `category.php` and `lot.php` in the includes/classes folder with classes for each of the tables.

1. Create the file `includes/classes/category.php`.

2. Enter the file documentation and the `Category` class:

```php
<?php
/**
 * category.php
 *
 * Category class file
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
/**
 * Category class
 *
 * @package    Smithside Auctions
 */
class Category
{
}
```

2. Inside the class, add a property for each of the fields in the `categories` table. Make these protected properties.

```php
/**
 * Category ID
 * @var int
 */
protected $cat_id;

/**
 * Category Name
 * @var string
 */
protected $cat_name;

/**
 * Category Description
 * @var string
 */
```

```
      protected $cat_description;

      /**
       * Category Image path
       * @var string
       */
      protected $cat_image;
```

3. Following the properties, add a __construct method to fill the properties based on an array input:

```
/**
   * Initialize the Item
   * @param array
   */
  public function __construct($input = false) {
    if (is_array($input)) {
      foreach ($input as $key => $val) {
        // Note the $key instead of key.
        // This will give the value in $key instead of 'key' itself
        $this->$key = $val;
      }
    }
  }
```

4. Add in methods to get the protected properties:

```
/**
   * Return Category ID
   * @return int
   */
  public function getCat_id() {
    return $this->cat_id;
  }

  /**
   * Return Category Name
   * @return string
   */
  public function getCat_name() {
    return $this->cat_name;
  }

  /**
   * Return Category Description
   * @return string
   */
  public function getCat_description() {
    return $this->cat_description;
  }

  /**
   * Return Category Image path
   * @return string
   */
  public function getCat_image() {
    return $this->cat_image;
  }
```

5. Create a similar file for the `lots` table. This file is `includes/classes/lot.php`.

```php
<?php
/**
 * lot.php
 *
 * Lot class file
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
/**
 * Lot class
 *
 * @package    Smithside Auctions
 */
class Lot
{
  /**
   * Lot ID
   * @var int
   */
  protected $lot_id;

  /**
   * Lot Name
   * @var string
   */
  protected $lot_name;

  /**
   * Lot Description
   * @var string
   */
  protected $lot_description;

  /**
   * Lot Image path
   * @var string
   */
  protected $lot_image;

 /**
   * Lot Number
   * @var int
   */
  protected $lot_number;

  /**
   * Lot Price path
```

```php
 * @var float
 */
protected $lot_price;

/**
 * Lot Catalog ID
 * @var int
 */
protected $cat_id;

/**
 * Initialize the Item
 * @param array
 */
public function __construct($input = false) {
  if (is_array($input)) {
    foreach ($input as $key => $val) {
      // Note the $key instead of key.
      // This will give the value in $key instead of 'key' itself
      $this->$key = $val;
    }
  }
}

/**
 * Return Lot ID
 * @return int
 */
public function getLot_id() {
  return $this->lot_id;
}

/**
 * Return Lot Name
 * @return string
 */
public function getLot_name() {
  return $this->lot_name;
}

/**
 * Return Lot Description
 * @return string
 */
public function getLot_description() {
  return $this->lot_description;
}

/**
 * Return Lot Image path
 * @return string
 */
public function getLot_image() {
```

```
    return $this->lot_image;
  }

  /**
   * Return Lot Price path
   * @return string
   */
  public function getLot_price() {
    return $this->lot_price;
  }

  /**
   * Return Lot Category ID
   * @return string
   */
  public function getCat_id() {
    return $this->cat_id;
  }

}
```

Add the `id` field to the `Contact` class in `includes/classes/contact.php`.

1. Add `id` as a protected property at the beginning of the `Contact` class:

```
/**
 * ID
 * @var int
 */
protected $id;
```

2. Add a public getter method to access the `id` property. Add this at the start of the rest of the getter methods.

```
/**
 * Return ID
 * @return int
 */
public function getId() {
  return $this->id;
}
```

> *Watch the video for Lesson 21 on the DVD or watch online at* `www.wrox.com/go/24phpmysql.`

22

Entering Data

In this lesson you learn how to enter data into the MySQL tables you built. You learn to use both the MySQL INSERT command to add rows and to load a file of information with the LOAD DATA command.

The main way that you get — data from users on websites is through forms. In this lesson you go through the whole process from creating a form, error checking the data, adding the data to the tables, and informing the user of the success or failure of the update. You learn how to use the MySQL commands in a PHP program.

UNDERSTANDING THE INSERT COMMAND

The examples in this section are based on the table1 table that you created in Lesson 19. This is the command that creates that table:

```
CREATE TABLE IF NOT EXISTS 'table1' (
   'id' int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
   'description' text,
   'code' int(11) NOT NULL DEFAULT '42'
) ENGINE=MyISAM
```

The MySQL INSERT command adds new rows to a table. You tell MySQL the table to use and then give it the values for all the fields in the new row. The following code creates a new row, assigning 1 as the value of the first field, abc as the value of the second field, and 15 as the value of the third field:

```
INSERT INTO 'table1' VALUES ('101','abc','15');
```

The name of the table is enclosed with back ticks, which is optional for simple names. Enclose the value in quotes, regardless of whether the value is text or numeric. The quotes can be either single or double quotes. If there is a quote as part of the value, you must either use the other type of quote to enclose it, double the quote, or escape the quote in the value. To escape

a quote, prefix it with a backslash (\). All of the following methods work when the value of the second field is `a'bc`:

```
INSERT INTO 'table1' VALUES ("102","a'bc","15");
INSERT INTO 'table1' VALUES ('103','a''bc','15');
INSERT INTO 'table1' VALUES ('104','a\'bc','15');
```

To add more than one row at a time, add additional parentheses groups separated by commas. This code adds three more rows:

```
INSERT INTO 'table1' VALUES ('105','def','23'), ('106', 'ghi','23'), ↵
('107', 'jkl','15');
```

If a field has the AUTO_INCREMENT attribute, MySQL automatically assigns the next number if you assign a 0, empty quotes, or the keyword NULL. Keywords are not enclosed in quotes. Assuming that the first field has the AUTO_INCREMENT attribute, the following code creates rows with the values of 108, 109, and 110. You can use multiple lines to make the code easier to read.

```
INSERT INTO 'table1' VALUES
  ('0','mno','15'),
  ('', 'pqr','23'),
  (NULL, 'stu','42');
```

Use the DEFAULT keyword to indicate that the default should be used. If the default for the third field is 42, the following code inserts 42 as the value for the third field:

```
INSERT INTO 'table1' VALUES (NULL,'vwx',DEFAULT);
```

Figure 22-1 shows the rows added by the code in this section as seen in phpMyAdmin.

☐	🖉	✕	101	abc	15
☐	🖉	✕	102	a'bc	15
☐	🖉	✕	103	a'bc	15
☐	🖉	✕	104	a'bc	15
☐	🖉	✕	105	def	23
☐	🖉	✕	106	ghi	23
☐	🖉	✕	107	jkl	15
☐	🖉	✕	113	mno	15
☐	🖉	✕	114	pqr	23
☐	🖉	✕	115	stu	42
☐	🖉	✕	116	vwx	42

FIGURE 22-1

Rather than listing a value for all of the fields, you can list the fields to be added. Any fields not listed use the default as defined in the table definition or the implied default based on the data type. If the missing field is defined as NOT NULL and there is no appropriate default, you get an error. The version of MySQL and the level of error reporting determine exactly what produces errors and whether those errors prevent the command from completing. The following code adds a row with the field description set to yza. The first field defaults with the AUTO_INCREMENT and the last field defaults to the explicit default of 42.

```
INSERT INTO 'table1' ('description') VALUES ('yza');
```

If you try to add a row that has the same primary key as an existing row, you get an error and the row is not added. To prevent the error, add the IGNORE keyword. If a duplicate is found, the new line is ignored without issuing errors or killing the command in the middle. Assuming that a row with the primary key 101 already exists and row 118 does not, the following command ignores the request to add 101 and still adds row 118:

```
INSERT IGNORE INTO 'table1' VALUES ('101','bcd','99'), ('118','efg','23');
```

If you want to do something special if you get a duplicate key, specify ON DUPLICATE KEY UPDATE. MySQL updates the existing record for any duplicates based on what you specify after the UPDATE keyword.

```
INSERT INTO 'table1' VALUES ('101','bcd','15'), ('118','efg','23') ↵
ON DUPLICATE KEY UPDATE code = '99';
```

To replace the row with the new values, you use the REPLACE statement. The REPLACE statement has the same syntax as the INSERT statement. The only difference is that if you try to add a row with the same primary key, the existing row is replaced rather than causing an error message or being ignored.

EXECUTING MYSQL COMMANDS IN PHP

You won't very often create databases or tables using PHP because then the MySQL used to make the database connection has to have extensive privileges, which is unsafe. However, you will frequently add data to MySQL tables using PHP. Entering data through a PHP program using MySQL command takes three steps:

1. Make a connection to the database.
2. Create a safe query with the command.
3. Run the query.

The following code uses the MySQL INSERT command to add rows to the table table1. You could enclose the command in double quotes and assign it to the $query variable. However, it is not good practice to have extremely long lines. Use the double quotes around appropriate sections of the command and put them back together with the concatenation operator (.). The query() method does only one statement at a time and does not use a semicolon at the end of the MySQL statement. Your results look like Figure 22-2. When you look in phpMyAdmin you see the rows in Figure 22-3.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';
```

> Successful connection to MySQL
> Rows successfully added

FIGURE 22-2

```php
// Set up the query
$query = "INSERT INTO 'table1' ('description', 'code') VALUES "
 . " ('hij','15'), "
 . " ('klm','23'), "
 . " ('nop', DEFAULT)";

// Run the query and display appropriate message
if (!$result = $connection->query($query)) {
  echo "Unable to add rows<br />";
} else {
  echo "Rows successfully added<br />";
}
}
```


FIGURE 22-3

You can use variables to create the query statement. In the following examples I am hardcoding in the values of the variables to make the examples clearer. In real life these variables may be coming from forms, parameters or other sources. Even though you can see what the value is, assume the value could be anything. The following code is an example where the data passed to MySQL comes from variables. You add the row shown in Figure 22-4.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  $desc = "qrs";
  $code = "15";

  // Set up the query
  $query = "INSERT INTO 'table1' ('description', 'code') VALUES "
   . " ('$desc', '$code')";

  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "Unable to add rows<br />";
  } else {
    echo "Rows successfully added<br />";
  }
}
```

FIGURE 22-4

Earlier in this lesson you learned that you need to put quotes around the values and that you can use either single quotes or double quotes and if you have a quote within the value, you need to either enclose with the opposite type of quote or escape the quote in the value. In the preceding example you have the line `$desc = "qrs";`. If you replace that with `$desc = "qr's";` you might think you are following all the rules and that it will work. However, when you do that, as demonstrated in the following code, the rows do not update because that single quote in `$desc` is then enclosed by single quotes in `$query`, which is invalid in MySQL as shown in Figure 22-5.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  $desc = "qr's";
  $code = "15";

  // Set up the query
  $query = "INSERT INTO 'table1' ('description', 'code') VALUES "
   . " ('$desc', '$code')";

  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "Unable to add rows<br />";
  } else {
    echo "Rows successfully added<br />";
  }
}
```


FIGURE 22-5

You could fix this by using double quotes around `$desc` in the `$query` assignment. However, with a dynamic site you do not know whether the data contains single or double quotes (or both) so you need a solution that properly prepares the variables for insertion into the database. The answer to that is to escape any quotes in the variables by prefixing each quote character with a backslash (\). PHP has a function that does this for you. In mysqli it is the method `mysqli::real_escape_string` if you are using the object-oriented style or the function `mysqli_real_escape_string()`. This code escapes any quotes in `$desc` so that it works in the MySQL statement. The added line seen in phpMyAdmin looks similar to Figure 22-6.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");
```

```
// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  $desc = "qr's";
  $code = "15";

  $desc = $connection->real_escape_string($desc);

  // Set up the query
  $query = "INSERT INTO 'table1' ('description', 'code') VALUES "
    . " ('$desc', '$code')";
echo $query;

  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "Unable to add rows<br />";
  } else {
    echo "Rows successfully added<br />";
  }
}
```
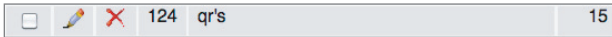

FIGURE 22-6

If you are using the procedural style the command looks like this instead:

```
$desc = mysqli_real_escape_string($connection, $desc);
```

There is one more wrinkle to consider — magic quotes. PHP recognized that this need to escape quotes was inconvenient and inexperienced programmers would neglect to do it. So PHP started doing it automatically. You used to be able to turn this behavior on or off in the `php.ini` file by turning on or off `magic_quotes_gpc`. Unfortunately this led to more problems and the use of magic quotes is now strongly discouraged; it will eventually be removed. However, if it happens to be on when you use the recommended `mysqli::real_escape_string` or `mysqli_real_escape_string` you get undesirable results. Before you escape your variables, you need to see if `magic_quotes_gpc` is on and if so, you need to strip out the escapes it added before using the recommended function. Use the `get_magic_quotes_gpc()` function to see if it is active:

```
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';
```

```
    $desc = "qr's";
    $code = "15";

    if (get_magic_quotes_gpc()) {
      $desc = stripslashes($desc);
    }
    $desc = $connection->real_escape_string($desc);

    // Set up the query
    $query = "INSERT INTO 'table1' ('description', 'code') VALUES "
     . " ('$desc', '$code')";

    // Run the query and display appropriate message
    if (!$result = $connection->query($query)) {
      echo "Unable to add rows<br />";
    } else {
      echo "Rows successfully added<br />";
    }
  }
```

Another reason for escaping the quotes is that it helps prevent SQL injection. Notice that you only escaped the `$desc` variable and not `$code`. `$desc` is a string field and `$code` is an integer. Best practices deal with different types of fields differently. If a field is an integer data type, the best check is to be sure that it is an integer.

```
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  $desc = "qr's";
  $code = "15";

  if (get_magic_quotes_gpc()) {
    $desc = stripslashes($desc);
  }
  $desc = $connection->real_escape_string($desc);
  $code = (int) $code;


  // Set up the query
  $query = "INSERT INTO 'table1' ('description', 'code') VALUES "
   . " ('$desc', '$code')";

  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "Unable to add rows<br />";
```

```
    } else {
      echo "Rows successfully added<br />";
    }
  }
```

These examples are inserting only one row at a time. If you are inserting more than one row, the same principles apply. Add in other error-checking for the fields and you can see that there is significant preparation for each field that is used to update a database field. This is a place where creating functions for preparing variables for insertion is helpful.

PROCESSING DATA ENTRY FORMS IN PHP

In this section you take everything that you've learned about processing forms, passing tokens, preparing data, and updating the database and use it to program a form to add rows to a database table. Start by creating a basic input form with two required fields for getting the description and code as shown in Figure 22-7. Add a spot for any messages in $message to be displayed.

FIGURE 22-7

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title></title>
</head>
<body>
<h1>Data Entry</h1>

<p><?php echo $message; ?></p>

<form action="lesson22j.php" method="post" name="maint" id="maint">

  <fieldset class="maintform">
    <legend>Add a Row</legend>
    <ul>
      <li><label for="desc">Description *</label><br />
        <input type="text" name="desc" id="desc" /></li>
      <li><label for="code">Code *</label><br />
        <input type="text" name="code" id="code" /></li>
    </ul>

    <input type="submit" name="save" value="Save" />
    <a class="cancel" href="lesson22j.php">Cancel</a>
```

```
    </fieldset>
  </form>

  </body>
  </html>
```

Because you are updating a database, you want to add the security of a token. Start a session at the very beginning of the file:

```php
<?php
session_start();
?>
```

Within the form fieldset, create a token and post it to the session as a hidden input:

```php
<?php
// create token
$salt = 'SomeSalt';
 $token = sha1(mt_rand(1,1000000) . $salt);
 $_SESSION['token'] = $token;
 ?>
 <input type='hidden' name='token' value='<?php echo $token; ?>'/>
```

You can either create a file for the form and a different one for the processing or do everything in one file. In this example, you use one file so the next task is to add the processing of the form. After `session_start()`, initialize the `$message` variable, add a check to see if there is a form to process, and then check for a good token, as in the following code:

```php
$message = '';

if (isset($_POST['save']) AND $_POST['save'] == 'Save') {
  // check the token
  $badToken = true;
  if (empty($_POST['token']) || $_POST['token'] !== $_SESSION['token']) {
    $message = 'Sorry, try it again. There was a security issue.';
    $badToken = true;
  } else {
    $badToken = false;
    unset($_SESSION['token']);
```

Next make your connection to the database and get the data from the POST array. For the description input, sanitize the data coming from the form as string data. To allow quotes without encoding them (such as ' remaining as a ' and not changing to '), add the FILTER_FLAG_NO_ENCODE_ QUOTES filter. Filter the code input by forcing it to an integer. Verify that data exists in both `$desc` and `$code`. If the data passes all those requirements, prepare the data for insertion into the database and then set up the query and add it to the database. Errors or success messages are posted to `$message`, which is then displayed. Close out all of the `if` statements.

```php
        define("MYSQLUSER", "php24sql");
        define("MYSQLPASS", "hJQV8RTe5t");
        define("HOSTNAME", "localhost");
        define("MYSQLDB", "test");

        // Make connection to database
        $connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
        if ($connection->connect_error) {
          die('Connect Error: ' . $connection->connect_error);
        } else {
```

```
      // Get the data
      $desc = filter_input(INPUT_POST,'desc',
FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES);
      $code = (int) $_POST['code'];

      // Verify the data
      if (!$desc OR !$code) {
        $message .= 'Description and Code are required <br />';
      } else {

        // Prepare the data
        if (get_magic_quotes_gpc()) {
          $desc = stripslashes($desc);
        }
        $desc = $connection->real_escape_string($desc);
        $code = (int) $code;

        // Set up the query
        $query = "INSERT INTO 'table1' ('description', 'code') VALUES "
         . " ('$desc', '$code')";

        // Run the query and display appropriate message
        if (!$result = $connection->query($query)) {
          $message .= "Unable to add rows<br />";
        } else {
          $message .= "Row successfully added<br />";
        }
      }
    }
  }
}
?>
```

Enter data into the form as shown in Figure 22-8. Click Save. You get a message that the row has been successfully saved, as shown in Figure 22-9.

FIGURE 22-8

FIGURE 22-9

Step-by-Step

Create a form (see Figure 22-11) to enter contacts. Add processing to add the data from the form to the database and handle the messaging and page redirections.

FIGURE 22-11

1. In `content/about.php` add a link in the `<h1>` to the data entry form you are creating.

```
<h1>About Us<a class="button" href="index.php?content=contactmaint&id=0 ↵
 ?>">Add</a></h1>
```

2. Create a new file called `content/contactmaint.php`. This contains the HTML form. Use an empty `Contact` object to define the data fields. This isn't required for this window but is used when you turn it into a maintenance page. Pass a hidden input for the `id` for the row. This is blank for new contacts. Pass a hidden input for a task so you know what form you are processing. Create and pass a token to confirm this is a legitimate form.

```php
<?php
/**
 * contactmaint.php
 *
 * Maintenance for Contacts
 *
```

```
 * @version   1.2 2011-02-03
 * @package   Smithside Auctions
 * @copyright Copyright (c) 2011 Smithside Auctions
 * @license   GNU General Public License
 * @since     Since Release 1.0
 */
$item = new Contact;
?>
<h1>Contact Maintenance</h1>

<form action="index.php?content=about" method="post" name="maint" id="maint">

  <fieldset class="maintform">
    <legend>Add a Contact</legend>
    <ul>
      <li><label for="first_name" class="required">First Name</label><br />
        <input type="text" name="first_name" id="first_name" class="required"
        value="<?php echo $item->getFirst_name(); ?>" /></li>
      <li><label for="last_name" class="required">Last Name</label><br />
        <input type="text" name="last_name" id="last_name" class="required"
        value="<?php echo $item->getLast_name(); ?>" /></li>
      <li><label for="position">Position</label><br />
        <input type="text" name="position" id="position" class="required"
        value="<?php echo $item->getPosition(); ?>" /></li>
      <li><label for="email" >Email</label><br />
        <input type="text" name="email" id="email"
        value="<?php echo $item->getEmail(); ?>" /></li>
      <li><label for="phone" >Phone</label><br />
        <input type="text" name="phone" id="phone"
        value="<?php echo $item->getPhone(); ?>" /></li>
    </ul>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="id" id="id" value="<?php echo $item->getId();
?>" />
    <input type="hidden" name="task" id="task" value="contact.maint" />
    <input type='hidden' name='token' value='<?php echo $token; ?>'/>
    <input type="submit" name="save" value="Save" />
    <a class="cancel" href="index.php?content=about">Cancel</a>
  </fieldset>
</form>
```

3. Create a constant in `includes/init.php` called `MAGIC_QUOTES_ACTIVE` that contains the result of `get_magic_quotes_gpc()`. This contains a `1` if magic quotes are on. Creating a constant once rather than running the function each time you need it is quicker.

```
define('MAGIC_QUOTES_ACTIVE', get_magic_quotes_gpc());
```

4. Create a public static method called `prep()` in `includes/classes/database.php` to prepare the data for insertion into the database. The data is passed to the function as a parameter. To refer to the connection property, use the static construction of `self::$_connection`.

```php
public static function prep($value) {
  if (MAGIC_QUOTES_ACTIVE) {
    // If magic quotes is active, remove the slashes
    $value = stripslashes($value);
  }
  // Escape special characters to avoid SQL injections
  $value = self::$_connection->real_escape_string($value);
  return $value;
}
```

5. In the `includes/init.php` file, move the `require_once 'includes/functions.php';` to just below the magic quotes constant.

6. Add processing to the `includes/init.php` file. This file works as the traffic cop to decide what task needs to be done. Up until now the implied task has been to display pages. Set up a switch to check for tasks with a `case` block where the condition is equal to `contact.maint`. Break out at the end of the `case` block.

```php
// Process based on the task. Default to display
$task = filter_input(INPUT_POST,'task', FILTER_SANITIZE_STRING);
switch ($task) {
  case 'contact.maint' :
  break;
}
```

7. Within the `case` block you created in step 5, run a function to process the task. The `$results` return consists of an array where the first element is a redirect to different content page, if any. The second element contains a message, which is assigned the `$message` variable. If there is a redirect page, move the message to a `SESSION` variable and redirect. Change the code to this code:

```php
// Process based on the task. Default to display
$task = filter_input(INPUT_POST,'task', FILTER_SANITIZE_STRING);
switch ($task) {
  case 'contact.maint' :
  // process the maint
  $results = maintContact();
  $message .= $results[1];
  // If there is redirect information
  // redirect to that page
  if ($results[0] == 'contactmaint') {
    // pass on new messages
    if ($results[1]) {
      $_SESSION['message'] = $results[1];
    }
    header("Location: index.php?content=contactmaint");
    exit;
```

```
        }
      break;
    }
```

8. Because you are using a SESSION variable you need to start the session at the very beginning of the file, just after the documentation:

```
session_start(); // starts new or resumes existing session
```

9. Add processing to check the SESSION for a message and move it to $message and then remove it from the SESSION with an unset. Put this before the task processing.

```
// Initialize message coming in
$message = '';
if (isset($_SESSION['message'])) {
  $message = htmlentities($_SESSION['message']);
  unset($_SESSION['message']);
}
```

10. Add the maintContact() function in includes/functions.php. Start by initializing the $results variable. If the Save button was clicked and equals Save, check the token. If there is a problem, put a message in the result. Otherwise, filter the data from the form and put it in an array called $item. Use that array to create a new Contact object. Run the addRecord() method of the object to add the row to the database, putting the results in $results.

```
function maintContact() {
  $results = '';
  if (isset($_POST['save']) AND $_POST['save'] == 'Save') {
    // check the token
    $badToken = true;
    if (!isset($_POST['token'])
       || !isset($_SESSION['token'])
       || empty($_POST['token'])
       || $_POST['token'] !== $_SESSION['token']) {
      $results = array('','Sorry, go back and try again. There was a ↵
security issue.');
      $badToken = true;
    } else {
      $badToken = false;
      unset($_SESSION['token']);
      // Put the sanitized variables in an associative array
      // Use the FILTER_FLAG_NO_ENCODE_QUOTES to allow names like O'Connor
      $item  = array (  'id' => (int) $_POST['id'],
                 'first_name' => filter_input(INPUT_POST,'first_name',
FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES),
                 'last_name'  => filter_input(INPUT_POST,'last_name',
FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES),
                 'position'   => filter_input(INPUT_POST,'position',
FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES),
                 'email'      => filter_input(INPUT_POST,'email',
FILTER_SANITIZE_STRING),
```

```
                        'phone'        => filter_input(INPUT_POST,'phone',
    FILTER_SANITIZE_STRING)
                );

          // Set up a Contact object based on the posts
          $contact = new Contact($item);
          $results = $contact->addRecord();
        }
      }
      return $results;
    }
```

11. Add the public method `addRecord()` in `includes/classes/contact.php`. This function calls the method `_verifyInput()`. If the data is verified, get the database connection. Set up the data by creating the `INSERT` statement. Use the `Database::prep()` method to prepare the data. Create the array `$result()` where the first element is blank for normal processing, or `contactmaint` if there is an error where the user should stay on the `contactmaint` page. The second element contains a success message or an error message.

```
    public function addRecord() {

      // Verify the fields
      if ($this->_verifyInput()) {

        // Get the Database connection
        $connection = Database::getConnection();

        // Prepare the data
        $query = "INSERT INTO contacts(first_name, last_name, position, email,
    phone)
          VALUES ('" . Database::prep($this->first_name) . "',
          '" . Database::prep($this->last_name) . "',
          '" . Database::prep($this->position) . "',
          '" . Database::prep($this->email) . "',
          '" . Database::prep($this->phone) . "')";
        // Run the MySQL statement
        if ($connection->query($query)) {
          $return = array('', 'Contact Record successfully added.');

          // add success message
          return $return;
        } else {
          // send fail message and return to contactmaint
        $return = array('contactmaint', 'No Contact Record Added. Unable to ↵
    create record.');
        return $return;
        }
      } else {
        // send fail message and return to contactmaint
        $return = array('contactmaint', 'No Contact Record Added. Missing ↵
    required information.');
        return $return;
      }

    }
```

12. Add the protected function `_verifyInput()` in `includes/classes/contact.php`. This method checks that the required fields have been filled in. It returns false if there is an error.

```
protected function _verifyInput() {
  $error = false;
  if (!trim($this->first_name)) {
    $error = true;
  }
  if (!trim($this->last_name)) {
    $error = true;
  }
  if ($error) {
    return false;
  } else {
    return true;
  }
}
```

13. In `index.php` change the `<div class="message">` section to just display the `$message` variable:

```
<div class="message">
  <?php echo $message; ?>
</div><!-- end message -->
```

> *Watch the video for Lesson 22 on the DVD or watch online at* www.wrox.com/ go/24phpmysql.

23

Selecting Data

In this lesson you learn how to retrieve data from the database. The SELECT command is argu-
ably the most common MySQL command you use in PHP programs. It is also one of the most
complex, with clauses that enable you to choose what table(s) you use, which columns are
returned, what conditions must be met before a row is selected, what order to sort the data in,
and whether and how to group and summarize the data.

You work with a single table at a time in this lesson. In the next lesson you learn how to use
multiple tables. The table used to illustrate this lesson is shown in Figure 23-1 and is created
from the following code:

```
CREATE TABLE IF NOT EXISTS 'table1' (
   'id' int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
   'description' text,
   'code' int(11) NOT NULL DEFAULT '42'
) ENGINE=MyISAM;

INSERT INTO 'table1' ('id', 'description', 'code') VALUES
(101, 'abc', 99),
(102, 'a''bc', 15),
(103, 'a''bc', 15),
(104, 'a''bc', 15),
(105, 'def', 23),
(106, 'ghi', 23),
(107, 'jkl', 15),
(108, 'mno', 15),
(109, 'pqr', 23),
(110, 'stu', 42),
(111, 'vwx', 42),
(112, 'yza', 42),
(118, 'efg', 99);
```

id	description	code
101	abc	99
102	a'bc	15
103	a'bc	15
104	a'bc	15
105	def	23
106	ghi	23
107	jkl	15
108	mno	15
109	pqr	23
110	stu	42
111	vwx	42
112	yza	42
118	efg	99

FIGURE 23-1

USING THE SELECT COMMAND

The easiest form of the SELECT command is to select all fields and rows from a single table:

```
SELECT * FROM 'table1';
```

The asterisk (*) indicates that all fields are to be selected and FROM 'table1' tells which table to use. By default all rows are selected.

To select only some of the fields to show, specify those fields instead of using an * as in the following code and shown in Figure 23-2:

```
SELECT 'id', 'description' FROM 'table1';
```

Whatever order you put the fields in is the order in which they are displayed as shown in the following code and Figure 23-3. This change in order is just for the display. The SELECT command does not change anything in the database itself.

```
SELECT 'description', 'id' FROM 'table1';
```

id	description
101	abc
102	a'bc
103	a'bc
104	a'bc
105	def
106	ghi
107	jkl
108	mno
109	pqr
110	stu
111	vwx
112	yza
118	efg

description	id
abc	101
a'bc	102
a'bc	103
a'bc	104
def	105
ghi	106
jkl	107
mno	108
pqr	109
stu	110
vwx	111
yza	112
efg	118

FIGURE 23-2   **FIGURE 23-3**

> It is very tempting to always use the asterisk (*) to select your fields instead of writing out just the ones you need. You will see this done a lot, especially by programmers who don't know MySQL well. However, the * is more resource intensive and you should avoid it if it is not needed.

The SELECT statement is made up of *clauses*. These clauses are the building blocks for creating the statement. You are already using two clauses: the *select expression* where you chose the fields and

the `FROM` clause. There is a list of the clauses in the MySQL documentation at `http://dev.mysql`
`.com/doc/refman/5.6/en/select.html`. The other common clauses include `WHERE`, `ORDER BY`,
and `LIMIT`.

> ✖ *It is important that the clauses are in the right order. If you have problems with a*
> `SELECT` *command, check the manual to see that you have the different clauses in*
> *the right order. In general you start with the* `SELECT` *expression and then follow*
> *with* `FROM`, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`, `PROCEDURE`, *and* `INTO`.
> *It's outside the scope of this book to go over all the different clauses.*

The `SELECT` expression clause has many possibilities. You can rename using *aliases*. This code
renames the `id` field to `myid`. See Figure 23-4.

```
SELECT 'id' AS 'myid', 'description' FROM 'table1';
```

myid	description
101	abc
102	a'bc
103	a'bc
104	a'bc
105	def
106	ghi
107	jkl
108	mno
109	pqr
110	stu
111	vwx
112	yza
118	efg

FIGURE 23-4

A common select expression is `COUNT()`. If you use an `*`, the result is a count of all the
rows selected. If you use a field then the result is the count of all the rows selected
where that field is not NULL. See Figure 23-5.

COUNT(*)
13

FIGURE 23-5

```
SELECT COUNT(*) FROM 'table1';
```

Another keyword that comes in handy is the `DISTINCT` keyword. This
indicates that if duplicates exist, only one is used. In the following
example you count the different descriptions that are in the table. See
Figure 23-6.

COUNT(DISTINCT `description`)
11

FIGURE 23-6

```
SELECT COUNT(DISTINCT 'description') FROM 'table1';
```

You can make complex expressions. You can use string functions on text fields. String functions are listed at `http://dev.mysql.com/doc/refman/5.6/en/string-functions.html`. Some common functions are `CONCAT()` to join fields, `CONCAT_WS` to concatenate with a separator, `TRIM()` to remove leading and trailing spaces, and `SUBSTR()` or `SUBSTRING()` to return part of the field. With numeric fields you can use numeric functions. Numeric functions are listed at `http://dev.mysql.com/doc/refman/5.6/en/numeric-functions.html`. MySQL automatically makes simple conversions but occasionally you may need to change the data type of the field. Use the cast functions that are listed at `http://dev.mysql.com/doc/refman/5.6/en/cast-functions.html`. The following code takes the description and appends it with a comma, a space, and the first letter of the description and then calls the result `item`. The result is shown in Figure 23-7.

item
abc, a
a'bc, a
a'bc, a
a'bc, a
def, d
ghi, g
jkl, j
mno, m
pqr, p
stu, s
vwx, v
yza, y
efg, e

FIGURE 23-7

```
SELECT CONCAT( 'description', ', ', SUBSTRING('description',1,1 )) AS
'item'
FROM 'table1';
```

When you use MySQL with PHP, you can choose to perform some actions in either the MySQL statement or in PHP. Which way you choose depends on how comfortable you are with each language and what the performance implications are. Some people always select all fields with the * rather than take the time to enter just the fields that they need. This has an advantage if you later want to use another field from that selection, but it means that you are taking a performance hit on the query. If your tables are small, the hit is irrelevant. However, if you have blobs of data or a large number of records, it could become significant. On the other hand, complex selections can take longer than simple selections so you could be better off doing the manipulations in PHP.

If you entered some of the commands in this lesson in phpMyAdmin, you may have noticed that they often append the `LIMIT` clause. The `LIMIT` clause is a way to limit the number of rows returned at one time. There are limits, in both MySQL and PHP, in the amount of server processing time used before an error is thrown. Using `LIMIT` allows you to receive your results in digestible bits. `LIMIT` is frequently used for pagination as well. There are two different syntaxes for `LIMIT`. The following code gives three ways to select the first five rows:

```
SELECT * FROM 'table1' LIMIT 5;
SELECT * FROM 'table1' LIMIT 0, 5;
SELECT * FROM 'table1' LIMIT 5 OFFSET 0;
```

And here are two ways to select the next five rows:

```
SELECT * FROM 'table1' LIMIT 5, 5;
SELECT * FROM 'table1' LIMIT 5 OFFSET 5;
```

The limiting is based on selected, ordered rows.

The `ORDER BY` clause enables you to return the rows in a given sequence. You specify the field or alias and whether the order should be ascending or descending. If you don't specify a direction,

ascending is assumed. The following code returns the rows in order by the description in ascending order as shown in Figure 23-8.

```
SELECT * FROM 'table1' ORDER BY 'description' ASC
```

You can string together multiple fields/directions to create more complex orders. The following code orders by the code in reverse order and then by the description (see Figure 23-9):

```
SELECT * FROM 'table1' ORDER BY 'code' DESC, 'description' ASC
```

id	description ▲	code
102	a'bc	15
103	a'bc	15
104	a'bc	15
101	abc	99
105	def	23
118	efg	99
106	ghi	23
107	jkl	15
108	mno	15
109	pqr	23
110	stu	42
111	vwx	42
112	yza	42

FIGURE 23-8

id	description	code
101	abc	99
118	efg	99
110	stu	42
111	vwx	42
112	yza	42
105	def	23
106	ghi	23
109	pqr	23
102	a'bc	15
103	a'bc	15
104	a'bc	15
107	jkl	15
108	mno	15

FIGURE 23-9

USING WHERE

The `WHERE` clause is the clause that you use to pick which rows to select. If there is no `WHERE` clause then all rows are selected. A `WHERE` clause can be as simple as it is in the following command in which it selects the row with the `id` of 105 as shown in Figure 23-10:

id	description	code
105	def	23

FIGURE 23-10

```
SELECT * FROM 'table1' WHERE 'id' = 105;
```

A `WHERE` clause can use most of the functions and operators that MySQL uses. For complete information, see the following pages in the online manual:

➤ **Expression Syntax:** This explains things like how to use expressions such as `AND` and `OR` (`http://dev.mysql.com/doc/refman/5.6/en/expressions.html`).

➤ **Comparison Functions and Operators:** This explains the different ways that you can compare values. See Table 23-1 for common operators or the manual for all the functions and operators (`http://dev.mysql.com/doc/refman/5.6/en/comparison-operators .html#operator_not-between`).

TABLE 23-1: MySQL Comparison Operators

OPERATOR	DESCRIPTION
=	Equal (Notes: PHP uses a double equal sign but MySQL uses a single; If a NULL is on either side, the result is NULL.)
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
!=	Not equal to
<>	Not equal to
<=>	Equal to (NULL is safe. NULL<=>NULL is true. NULL<=> other things is false.)
IS NULL	Is the value NULL
IS NOT NULL	Is the value not NULL
IS	IS TRUE, IS FALSE, IS NOT TRUE, IS NOT FALSE
LIKE	Match to a pattern character by character % wildcard matches any number of characters including zero; _ (underscore) wildcard matches any single character
NOT LIKE	Does not match to the pattern
BETWEEN... AND...	Checks to see if a value is within this range This does data type casting, but you should use $CAST()$ to explicitly convert date and time values to the same type
NOT BETWEEN... AND...	Checks to see if a value is not within this range

➤ **Literal Values Syntax:** This explains things such as when you need to use quotes and how to specify dates and times. (Hint: In general, you don't need quotes for numbers but you do for strings.) See the details at `http://dev.mysql.com/doc/refman/5.6/en/literals.html`.

➤ **Functions and Operators:** The table of contents for all the functions and operators is at `http://dev.mysql.com/doc/refman/5.6/en/functions.html`.

SELECTING DATA IN PHP

You frequently select data in a MySQL table using PHP. Selecting data through a PHP program using a MySQL command takes four steps:

1. Make a connection to the database.

2. Create a safe query with the command.

3. Run the query.

4. Read the results.

The following code makes the connection and creates a safe query. Rather than just running the query, use it as the right side of an assignment. This creates a `mysqli_result` object that you use to read the results via methods in the result object. This example takes each row, one at a time, and puts it into an associative array. It uses a `while` loop to loop through the results and prints each row as it retrieves it. See the results in Figure 23-11.

```
Successful connection to MySQL
Array ( [id] => 102 [description] => a'bc [code] => 15 )
Array ( [id] => 103 [description] => a'bc [code] => 15 )
Array ( [id] => 104 [description] => a'bc [code] => 15 )
Array ( [id] => 107 [description] => jkl [code] => 15 )
Array ( [id] => 108 [description] => mno [code] => 15 )
```

FIGURE 23-11

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  // Set up the query
  $query = "SELECT * FROM 'table1' "
   . " WHERE 'code' = 15"
   . " ORDER BY 'description' ASC "
   ;

  // Run the query
  $result_obj = '';
  $result_obj = $connection->query($query);

  // Read the results
  // loop through the result object, row by row
  // reading each row into an associative array
  while($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
```

```
      // display the array
      print_r($result);
      echo '<br />';
  }
}
```
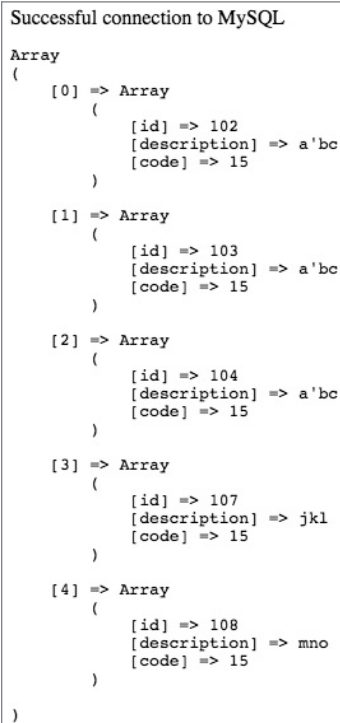
The example prints out the array, but you can do anything with it at that moment before it is over-
written with the next row. This code copies the array to another array so you end up with an array
of arrays, which it prints after it has finished collecting all the rows. See Figure 23-12.

```
// Read the results
// loop through the results, row by row
// reading each row into an associative array
while($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
  // collect the array
  $item[] = $result;
}
// print array when done
echo '<pre>';
print_r($item);
echo '</pre>';
```

```
Successful connection to MySQL

Array
(
    [0] => Array
        (
            [id] => 102
            [description] => a'bc
            [code] => 15
        )

    [1] => Array
        (
            [id] => 103
            [description] => a'bc
            [code] => 15
        )

    [2] => Array
        (
            [id] => 104
            [description] => a'bc
            [code] => 15
        )

    [3] => Array
        (
            [id] => 107
            [description] => jkl
            [code] => 15
        )

    [4] => Array
        (
            [id] => 108
            [description] => mno
            [code] => 15
        )

)
```

FIGURE 23-12

The online PHP manual lists the different methods that you can use to read the results at `www.php`
`.net/manual/en/class.mysqli-result.php`. Here are examples of some of them.

➤ `fetch_array(MYSQLI_ASSOC)`: This returns an associative array. Loop through to get all the rows. Same as `fetch_assoc()`.

➤ `fetch_array(MYSQLI_NUM)`: This returns a numeric array. Loop through to get all the rows. Same as `fetch_row()`.

➤ `fetch_array(MYSQLI_BOTH)`: This returns both an associative array and a numeric array with the same data. Loop through to get all the rows. This is the default if no type is specified.

➤ `fetch_all(MYSQLI_ASSOC)`: This returns all the rows as an associative array.

➤ `fetch_all(MYSQLI_NUM)`: This returns all the rows as a numeric array.

➤ `fetch_all(MYSQLI_BOTH)`: This returns all the rows both as an associative array and a numeric array with the same data.

➤ `fetch_object(`*`$class_name`*`)`: This returns an object of the row. Loop through to get all the rows. If you give it a class name, it uses that class to create the object. If there is no class name it will create a `stdClass` object, which is a predefined class.

These are also available as ordinary functions, which need the mysqli result variable as a parameter. You don't need it for the object method form because the object already knows that information.

➤ `mysqli_fetch_array($result, MYSQLI_ASSOC)`: This returns an associative array. Loop through to get all the rows. Same as `mysqli_fetch_assoc($result)`.

➤ `mysqli_fetch_array($result, MYSQLI_NUM)`: This returns a numeric array. Loop through to get all the rows. Same as `mysqli_fetch_row($result)`:

➤ `mysqli_fetch_array(MYSQLI_BOTH)`: This returns both an associative array and a numeric array with the same data. Loop through to get all the rows. This is the default if no type is specified.

➤ `mysqli_fetch_all($result, MYSQLI_ASSOC)`: This returns all the rows as an associative array.

➤ `mysqli_fetch_all($result, MYSQLI_NUM)`: This returns all the rows as a numeric array.

➤ `mysqli_fetch_all($result, MYSQLI_BOTH)`: This returns all the rows both as an associative array and a numeric array with the same data.

➤ `mysqli_fetch_object($result, `*`$class_name`*`)`: This returns an object of the row. Loop through to get all the rows. If you give it a class name, it uses that class to create the object. If there is no class name, it creates a `stdClass` object, which is a predefined class.

TRY IT

In this Try It, you retrieve data from the Case Study database and use it to populate the website instead of using hardcoded data. You start with the About Us page, pulling the information from the `contacts` table. Then you do the same for the Lot Categories page. Because you don't have any data in the `categories` table, you also create a maintenance page to add data to the `categories` table.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson23 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 22. Alternatively, you can download the files from the book's website at www.wrox.com. To re-create the database tables, create an empty database and then import the install.sql file in phpMyAdmin.

Hints

Use empty square brackets to add a value as a new element in an array.

The Lot Category maintenance page is set up the same way you set up the Contacts maintenance page in the previous lesson.

Step-by-Step

Change the About Us page to get the contact information from the database.

1. In contents/about.php the information is currently hardcoded in to the $items array. Replace those 30 lines with a static call to the Content class method getContacts():

```
// Get the contact information
$items = Contact::getContacts();
?>
```

2. When the information was hardcoded, you had total control over what it was. By pulling information from the database, you don't have the same control. There could be malicious information and there could be characters, such as ampersands, that should be encoded as HTML entities, such as &. To fix this, pass all string information through the htmlspecialchars() function:

```
<h2><?php echo htmlspecialchars($item->name()); ?></h2>
<p>Position: <?php echo htmlspecialchars($item->getPosition()); ?><br />
<?php echo htmlspecialchars($item->getEmail()); ?><br />
Phone: <?php echo htmlspecialchars($item->getPhone()); ?><br /></p>
```

3. In `includes/classes/contact.php` add the `getContacts()` public static method to retrieve the rows and fill the array. Use the `fetch_object()` method with the `Contact` class to read the rows into objects.

```php
static function getContacts() {
  // clear the results
  $items = '';
  // Get the connection
  $connection = Database::getConnection();
  // Set up query
  $query = 'SELECT * FROM 'contacts' ORDER BY first_name, last_name';
  // Run the query
  $result_obj = '';
  $result_obj = $connection->query($query);
  // Loop through the results,
  // passing them to a new version of this class,
  // and making a regular array of the objects
  try {
    while($result = $result_obj->fetch_object('Contact')) {
      $items[]= $result;
    }
    // pass back the results
    return($items);
  }

  catch(Exception $e) {
    return false;
  }
}
```

Create a maintenance page so you can add lot categories into the `categories` table. Add the lot categories into the `categories` table. Change the Lot Categories page to get the information from the database.

1. In `contents/categories.php` add documentation at the beginning of the file:

```php
<?php
/**
 * categories.php
 *
 * Content for Categories page
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
```

2. Follow that with the code to load the `$items` variable using the static `getCategories()` method from the `Category` class. If nothing is returned, initialize `$items` to an array so that later use of the variable doesn't create errors.

```php
// Get the category information
$items = Category::getCategories();
```

```
if (empty($items)) {
  $items = array();
}
?>
```

3. In the `<h1>` header, add a link to the new data entry page. Give it the class `button` for the CSS styling.

```
<h1>Categories<a class="button"
href="index.php?content=categorymaint&cat_id=0">Add</a></h1>
```

4. Wrap the first `<li>...</li>` in a `foreach` loop to loop through the `$items` array.

```
<?php foreach ($items as $i=>$item) : ?>
  <li class="row0">
  ...
  </li>
<?php endforeach; ?>
```

5. Change the `<li>` class to calculate the class for either `row0` or `row1` based on the `$i` key variable:

```
<li class="row<?php echo $i % 2; ?>">
```

6. Set up the images. The table has a field for the name of the file in the images folder. There is another file with the same name in the images/thumbnail folder. You want to check to see if these files exist before trying to display them, to prevent broken links and also verify that what you display is an actual filename in a specific folder. If a file doesn't exist, display a generic image instead.

```
<?php
  $image = 'images/'. $item->getCat_image();
  if (!is_file($image)) {
    $image = 'images/nophoto.jpg';
  }
  $image_t = 'images/thumbnails/'. $item->getCat_image();
  if (!is_file($image_t)) {
    $image_t = 'images/thumbnails/nophoto.jpg';
  }
?>
```

7. Change the image references to the new image variables:

```
<div class="list-photo">
  <a href="<?php echo $image; ?>">
  <img alt="" src="<?php echo $image_t; ?>"/></a>
</div>
```

8. Change the `<h2>` link to assign the `content=` parameter based on the category name. Change `<h2>` text to the category name as well. Change the description to use the category description and change the `content=` parameter in the `<a>` tag link to the category name in lowercase.

```
<div class="list-description">
  <h2>
  <a href="index.php?content=<?php echo ↵
htmlspecialchars($item->getCat_name()); ?>&amp;sidebar=catnav">
```

```
<?php echo htmlspecialchars (strtolower($item->getCat_name())); ?></a>
</h2>
<p><?php echo htmlspecialchars($item->getCat_description()); ?></p>
<a class="button display" href="index.php?content=<?php echo ↵
htmlspecialchars (strtolower($item->getCat_name())); ?>&amp;sidebar=catnav">
    Display Lots</a>
</div>
```

9.   Remove the rest of the `<li>` groups.

10.   In `includes/classes/category.php` add the `getCategories()` public static method to retrieve the rows and fill the array. Rather than directly creating the `Category` object, use `fetch_array(MYSQLI_ASSOC)` to retrieve the row. Use that array to create a new `Category` object that is added as an element in the `$items` array. This is a different way of creating the same `$items` array as is created in the `Contact` class using the `fetch_object()`.

```
static public function getCategories() {
  // clear the results
  $items = '';
  // Get the connection
  $connection = Database::getConnection();
  // Set up the query
  $query = 'SELECT * FROM 'categories' ORDER BY cat_name';

 // Run the query
  $result_obj = '';
  $result_obj = $connection->query($query);
  // Loop through getting associative arrays,
  // passing them to a new version of this class,
  // and making a regular array of the objects
  try {
    while($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
      $items[]= new Category($result);
    }
    // pass back the results
      return($items);
  }

  catch(Exception $e) {
    return false;
  }

}
```

11.   Add the `addRecord()` method to add rows to the `categories` table:

```
public function addRecord() {

  // Verify the fields
  if ($this->_verifyInput()) {

    // Get the Database connection
    $connection = Database::getConnection();

    // Prepare the data
```

```php
            $query = "INSERT INTO categories(cat_name, cat_description, cat_image)
              VALUES ('" . Database::prep($this->cat_name) . "',
                '" . Database::prep($this->cat_description) ."',
                '" . Database::prep($this->cat_image) . "')";

            // Run the MySQL statement
            if ($connection->query($query)) {
              $return = array('', 'Category Record successfully added.');

              // add success message
              return $return;
            } else {
              // send fail message and return to categorymaint
            $return = array('contactmaint', 'No Category Record Added.
              Unable to create record.');
            return $return;
            }
          } else {
            // send fail message and return to categorymaint
            $return = array('categorymaint', 'No Category Record Added.
              Missing required information.');
            return $return;
          }
```

12. Add the `_verifyInput()` method to verify that a category name was entered:

```php
        protected function _verifyInput() {
          $error = false;
          if (!trim($this->cat_name)) {
            $error = true;
          }
          if ($error) {
            return false;
          } else {
            return true;
          }
        }
```

13. Create `content/categorymaint.php`, which is the form for data entry of the categories:

```php
    <?php
    /**
     * categorymaint.php
     *
     * Maintenance for the Categories table
     *
     * @version  1.2 2011-02-03
     * @package  Smithside Auctions
     * @copyright  Copyright (c) 2011 Smithside Auctions
     * @license  GNU General Public License
     * @since    Since Release 1.0
     */
    $item = new Category;
    ?>
    <h1>Category Maintenance</h1>
```

```
 <form action="index.php?content=categories" method="post" name="maint"
id="maint">

   <fieldset class="maintform">
     <legend>Add a Category</legend>
     <ul>
     <li><label for="cat_name" class="required">Category</label><br />
       <input type="text" name="cat_name" id="cat_name" class="required"
       value="<?php echo $item->getCat_name(); ?>" /></li>
     <li><label for="cat_description">Description</label><br />
       <textarea rows="5" cols="60" name="cat_description"
       id="cat_description"><?php echo $item->getCat_description(); ?>
       </textarea></li>
     <li><label for="cat_image" >Image</label><br />
       <input type="text" name="cat_image" id="cat_image"
       value="<?php echo $item->getCat_image(); ?>" /></li>
     </ul>

     <?php
     // create token
     $salt = 'SomeSalt';
     $token = sha1(mt_rand(1,1000000) . $salt);
     $_SESSION['token'] = $token;
     ?>
     <input type="hidden" name="cat_id" id="cat_id"
       value="<?php echo $item->getCat_id(); ?>" />
     <input type="hidden" name="task" id="task" value="category.maint" />
     <input type='hidden' name='token' value='<?php echo $token; ?>'/>
     <input type="submit" name="save" value="Save" />
     <a class="cancel" href="index.php?content=categories">Cancel</a>
   </fieldset>
 </form>
```

14. In `includes/init.php` add a `case` block in the `switch` statement for `category.maint` to process the category maintenance form:

```
case 'category.maint' :
// process the maint
$results = maintCategory();
$message .= $results[1];
// If there is redirect information
// redirect to that page
if ($results[0] == 'categorymaint') {
  // pass on new messages
  if ($results[1]) {
    $_SESSION['message'] = $results[1];
  }
  header("Location: index.php?content=categorymaint");
  exit;
}
break;
```

15. Add the function `maintCategory()` to `includes/functions.php`:

```
function maintCategory() {
  $results = '';
```

```
          if (isset($_POST['save']) AND $_POST['save'] == 'Save') {
            // check the token
            $badToken = true;
            if (!isset($_POST['token'])
                || !isset($_SESSION['token'])
                || empty($_POST['token'])
                || $_POST['token'] !== $_SESSION['token']) {
              $results = array('','Sorry, go back and try again.
                There was a security issue.');
              $badToken = true;
            } else {
              $badToken = false;
              unset($_SESSION['token']);
              // Put the sanitized variables in an associative array
              // Use the FILTER_FLAG_NO_ENCODE_QUOTES
              // to allow quotes in the description
              $item  = array ('cat_id' => (int) $_POST['cat_id'],
                'cat_name' => filter_input(INPUT_POST,'cat_name',
                  FILTER_SANITIZE_STRING),
                'cat_description' => filter_input(INPUT_POST,'cat_description',
                  FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES),
                'cat_image' => filter_input(INPUT_POST,'cat_image',
                  FILTER_SANITIZE_STRING)
              );

              // Set up a Category object based on the posts
              $category = new Category($item);
              $results = $category->addRecord();
            }
          }
          return $results;
        }
```

16. Go to the new entry screen and add the lot categories shown in Figure 23-13.

FIGURE 23-13

17. Change `content/catnav.php` to use the categories in the `categories` table rather than hardcoded links:

```php
<?php
/**
 * catnav.php
 *
 * Menu for the categories
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
$items = Category::getCategories();
?>
<h3 class="element-invisible">Lot Categories</h3>
<ul class="catnav">
  <?php foreach ($items as $i=>$item) : ?>
<li><a href="index.php?content=<?php echo
  htmlspecialchars(strtolower($item->getCat_name())); ?>&sidebar=catnav">
  <?php echo htmlspecialchars($item->getCat_name()); ?></a></li>
  <?php endforeach; ?>
</ul>
```

> *Watch the video for Lesson 23 on the DVD or watch online at* `www.wrox.com/go/24phpmysql.`

24

Using Multiple Tables

In this lesson, you work with multiple tables. MySQL does not explicitly link tables together. Instead you design the tables so that they have fields that contain the information you need to link to another table. You then use that information in separate SELECT statements for each table or by using multiple tables in a SELECT statement.

The JOIN clause in the SELECT statement specifies what the links are between the tables. Subqueries let you create compound commands by using the result of one statement nested in another statement.

The examples in this lesson are based on three tables: the authors table that contains the author's name, the types table that contains the valid types of books, and the books table that lists the title along with a field whose value matches to the author's primary key plus a field whose value matches to the type's primary key. This is a simplified database where there is only one author per book and each book is in only one type.

> You can download the code for this example from the book's web page at
> www.wrox.com. You can find them in the Lesson24 folder in the download in
> a file labeled lesson24a.sql.

The first table is the authors table as shown in Figure 24-1.

The second table is the types table as shown in Figure 24-2.

The third table is the books table as shown in Figure 24-3.

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Peter	Gabriel
5	Dale	Mercer

FIGURE 24-1

type_id	type_name
1	History
2	Suspense
3	Science Fiction

FIGURE 24-2

id	title	author	type_id
1	A Long Day in Spring	3	1
2	Fifteen Hours in March	2	2
3	Green Trees Go Wild	1	3
4	And Then It Happened	1	1
5	Missing in Action	5	2
6	Fourteen Days in February	2	2

FIGURE 24-3

USING THE JOIN CLAUSE

With what you already know, you can work with multiple tables simply by using the information from one table as the basis for the WHERE clause for the other table. For instance, you know that the following command lists the author George Smith. See Figure 24-4.

first_name	last_name
George	Smith

FIGURE 24-4

```
SELECT first_name, last_name FROM authors WHERE id = '2';
```

The link between the books table and the authors table is that the author field in the books table and the id field (the primary key) of the authors table contain the same value. To list the books that George Smith wrote, you select from the books table all the rows where the author field is equal to 2, as shown in the following SELECT command and in Figure 24-5:

title
Fifteen Hours in March
Fourteen Days in February

FIGURE 24-5

```
SELECT title FROM books WHERE author = '2';
```

You can use more than one table in the SELECT statement. This command combines the books table and the types table. See the results in Figure 24-6.

```
SELECT title, type_name FROM books, types;
```

title	type_name
A Long Day in Spring	History
A Long Day in Spring	Suspense
A Long Day in Spring	Science Fiction
Fifteen Hours in March	History
Fifteen Hours in March	Suspense
Fifteen Hours in March	Science Fiction
Green Trees Go Wild	History
Green Trees Go Wild	Suspense
Green Trees Go Wild	Science Fiction
And Then It Happened	History
And Then It Happened	Suspense
And Then It Happened	Science Fiction
Missing in Action	History
Missing in Action	Suspense
Missing in Action	Science Fiction
Fourteen Days in February	History
Fourteen Days in February	Suspense
Fourteen Days in February	Science Fiction

FIGURE 24-6

These are probably not the results that you were intending. Every book is listed along with every type. The JOIN clause is used to specify how the tables should be merged. In this case you want to merge rows based on when the type_id in both tables is equal. Because the field name type_id is present in both tables, you need to specify the table when using the field. The following code displays a merged row of the title and type for each book as shown in Figure 24-7:

title	type_name
A Long Day in Spring	History
Fifteen Hours in March	Suspense
Green Trees Go Wild	Science Fiction
And Then It Happened	History
Missing in Action	Suspense
Fourteen Days in February	Suspense

FIGURE 24-7

```
SELECT title, type_name
FROM books JOIN types ON books.type_id = types.type_id
```

It is common practice to use aliases as a shortcut when you have to qualify names with the table. This is another way of writing the previous command:

```
SELECT title, type_name
FROM books AS b JOIN types AS t ON b.type_id = t.type_id
```

In the previous example, the linking fields had the same name. The names of the fields are irrelevant. The `books` table and the `authors` table both use the field name `id` for their primary keys. The fields, even though they have the same name, refer to different things. Qualifying the fields with the table name or alias removes the ambiguity.

The following code creates rows consisting of the author's last name, first name, and the book's title in order by author and then title. It joins the `authors` table, giving it an alias of `a`, and the `books` table, giving it an alias of `b`, based on the `id` in the `authors` table matching the `author` field in the `books` table. The author's name is concatenated into a single field. The result is shown in Figure 24-8.

full_name	title
Gabriel, Peter	A Long Day in Spring
Meyers, Sally	And Then It Happened
Meyers, Sally	Green Trees Go Wild
Smith, George	Fifteen Hours in March
Smith, George	Fourteen Days in February

FIGURE 24-8

```
SELECT CONCAT(last_name, ', ', first_name) AS full_name, title
FROM authors AS a
JOIN books AS b ON a.id = author
ORDER BY full_name, title;;
```

You can join more than two files. This example takes the previous example and adds the type of book to the list. See Figure 24-9.

```
SELECT CONCAT(last_name, ', ', first_name) AS full_name, title, type_name
FROM authors AS a
JOIN books AS b ON a.id = author
JOIN types AS t ON b.type_id = t.type_id
ORDER BY full_name, title;
```

full_name	title	type_name
Gabriel, Peter	A Long Day in Spring	History
Meyers, Sally	And Then It Happened	History
Meyers, Sally	Green Trees Go Wild	Science Fiction
Smith, George	Fifteen Hours in March	Suspense
Smith, George	Fourteen Days in February	Suspense

FIGURE 24-9

> *MySQL statements end with the semicolon, not the end of a line. Best practice is to move to a new line as needed for readability and to keep the lines from getting too long.*

Notice that the author Dale Mercer is not on the list nor is the book *Missing in Action*. Only rows with matches in both tables are selected with the plain JOIN. There are different types of joins. The default is a plain JOIN, which is the same as INNER JOIN.

If you want all the rows of a table, regardless of whether they find a match in the other file, you use one of the OUTER JOIN keywords. To keep all rows of the first (left) table, use a LEFT OUTER JOIN. To keep all the rows of the second (right) table, use a RIGHT OUTER JOIN. These are the same as a LEFT JOIN and a RIGHT JOIN. Figure 24-10 illustrates the three types of joins.
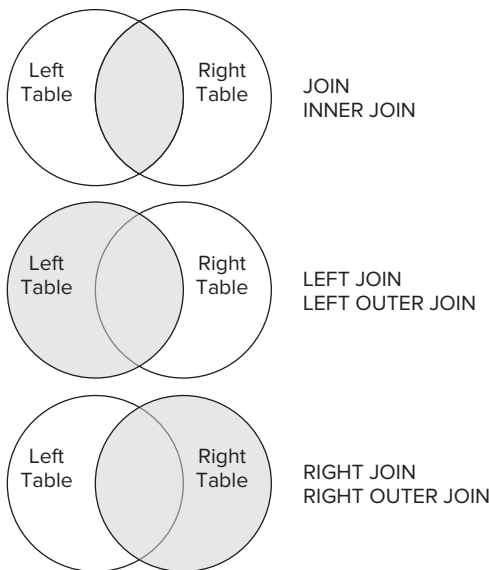
FIGURE 24-10

In this example, Dale Mercer has no books, but using a LEFT JOIN includes him in the results. Any fields that should come from the other table are NULL. See the results in Figure 24-11.

last_name	first_name	title
Meyers	Sally	Green Trees Go Wild
Meyers	Sally	And Then It Happened
Smith	George	Fifteen Hours in March
Smith	George	Fourteen Days in February
Gabriel	Peter	A Long Day in Spring
Mercer	Dale	*NULL*

FIGURE 24-11

```
SELECT last_name, first_name, title
FROM authors AS a
LEFT JOIN books AS b ON a.id = author;
```

When you use a RIGHT JOIN in the same command, all the books are listed, even if they did not match to an author in the authors table. See the results in Figure 24-12.

```
SELECT last_name, first_name, title
FROM authors AS a RIGHT JOIN books AS b ON a.id = author;
```

last_name	first_name	title
Gabriel	Peter	A Long Day in Spring
Smith	George	Fifteen Hours in March
Meyers	Sally	Green Trees Go Wild
Meyers	Sally	And Then It Happened
NULL	*NULL*	Missing in Action
Smith	George	Fourteen Days in February

FIGURE 24-12

When I said that all the rows from a table are included, that means all the rows that meet the other selection criteria in the other clauses such as the `WHERE` and `LIMIT` clauses. The following query asks for all authors whose last names start with an M. See the results in Figure 24-13.

last_name	first_name	title
Meyers	Sally	Green Trees Go Wild
Meyers	Sally	And Then It Happened
Mercer	Dale	*NULL*

FIGURE 24-13

```
SELECT last_name, first_name, title
FROM authors AS a LEFT JOIN books AS b ON a.id = author
WHERE last_name LIKE 'M%';
```

Use the `JOIN` clause in PHP as part of ordinary `SELECT` commands. The `JOIN` clause is also used in `UPDATE` and `DELETE` statements that you learn in later lessons.

USING SUBQUERIES

Subqueries are `SELECT` statements nested inside other statements. These can be a handy substitute for `JOIN` and can, in some cases, do things that `JOIN` cannot. However, if you are using large files there can be unexpected performance costs.

author
3
2
1
1
5
2

This code gives you a list of the author IDs in the `books` table as shown in Figure 24-14.

```
SELECT author FROM books;
```

FIGURE 24-14

The `WHERE` *field* `IN` *list* clause selects rows where the *field* is found in the *list*. You can use the `SELECT` statement in the preceding code to generate the list. The following example lists all the authors that have a book in the `books` table. See the results in Figure 24-15.

```
SELECT *
FROM authors
WHERE id IN (SELECT author FROM books);
```

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Peter	Gabriel

You need to be aware whether the subquery returns a single value, a list of single values, or a list of multiple values because it needs to match

FIGURE 24-15

what the outer statement expects. For instance, if you use WHERE `field = value` then `value` must be a single value, not a list. The following code returns the error "Subquery returns more than 1 row."

```
SELECT * FROM authors WHERE id = (SELECT author FROM books);
```

To fix this use either the keyword ALL or ANY. ALL means that all of the rows returned need to meet the criteria. In this example, using ALL returns an empty list of authors (but no errors), because no author wrote all the books in the `books` table. However, ANY means that the condition is satisfied as long as at least one of the rows satisfied the criteria. So this code also shows the result in Figure 24-15:

```
SELECT * FROM authors WHERE id = ANY (SELECT author FROM books);
```

You also get the same results writing the same statement with JOIN without using subqueries:

```
SELECT DISTINCT a.*
FROM authors AS a JOIN books
WHERE a.id = author;
```

You can use the subqueries in other statements, such as the INSERT statement. This example uses SELECT queries of other tables and uses the results as the values for creating a new row. The updated `books` table looks like Figure 24-16.

id	title	author	type_id
1	A Long Day in Spring	3	1
2	Fifteen Hours in March	2	2
3	Green Trees Go Wild	1	3
4	And Then It Happened	1	1
5	Missing in Action	5	2
6	Fourteen Days in February	2	2
7	Sixteen Seconds in March	2	2

FIGURE 24-16

```
INSERT INTO books (title, author, type_id)
VALUES ('Sixteen Seconds in March',
(SELECT id FROM authors WHERE first_name = 'George' AND last_name = 'Smith'),
(SELECT type_id FROM types WHERE type_name = 'Suspense'));
```

TRY IT

Available for download on Wrox.com

In this Try It, you do not use the Case Study. You create MySQL commands using SELECT JOINs in phpMyAdmin and then run those commands in PHP.

> You can download the code and resources for this Try It from the book's web page at www.wrox.com. You can find them in the Lesson24 folder in the download. You will find code for both before and after completing the exercises.

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

Hints

Setting up complex MySQL statements first in phpMyAdmin can help you locate errors before using them in a PHP program.

If the command works in phpMyAdmin and you get errors in PHP, add a `var_dump($query);` line just after you assign the MySQL statement to `$query`. This displays the command that is processed. You can copy that and run it in the SQL tab in phpMyAdmin to find issues.

Remember that `$query` can contain only one command and no semicolons.

Step-by-Step

Create the tables to be used in this Try It. Perform the queries on the files.

1. If you followed along with the lesson, you can skip to step 4. The code for steps 1–3 is in the `Lesson24/exercise24a.sql` file and you can import it into your database in phpMyAdmin.

Create and fill the `authors` table in phpMyAdmin as shown in Figure 24-17.

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Peter	Gabriel
4	Dale	Mercer

FIGURE 24-17

```sql
CREATE TABLE `authors` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(50) NOT NULL,
  `last_name` varchar(50) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;

INSERT INTO `authors` (`id`, `first_name`, `last_name`) VALUES
(1, 'Sally', 'Meyers'),
(2, 'George', 'Smith'),
(3, 'Peter', 'Gabriel'),
(4, 'Dale', 'Mercer');
```

2. Create and fill the `types` table in phpMyAdmin as shown in Figure 24-18.

type_id	type_name
1	History
2	Suspense
3	Science Fiction

FIGURE 24-18

```sql
CREATE TABLE `types` (
  `type_id` int(11) NOT NULL AUTO_INCREMENT,
  `type_name` varchar(20) NOT NULL,
  PRIMARY KEY (`type_id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;

INSERT INTO `types` (`type_id`, `type_name`) VALUES
(1, 'History'),
(2, 'Suspense'),
(3, 'Science Fiction');
```

3. Create and fill the `books` table in phpMyAdmin as shown in Figure 24-19.

id	title	author	type_id
1	A Long Day in Spring	3	1
2	Fifteen Hours in March	2	2
3	Green Trees Go Wild	1	3
4	And Then It Happened	1	1
5	Missing in Action	5	2
6	Fourteen Days in February	2	2
7	Sixteen Seconds in March	2	2

FIGURE 24-19

```
CREATE TABLE `books` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(50) NOT NULL,
  `author` int(11) DEFAULT NULL,
  `type_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;

INSERT INTO `books` (`id`, `title`, `author`, `type_id`) VALUES
(1, 'A Long Day in Spring', 3, 1),
(2, 'Fifteen Hours in March', 2, 2),
(3, 'Green Trees Go Wild', 1, 3),
(4, 'And Then It Happened', 1, 1),
(5, 'Missing in Action', 5, 2),
(6, 'Fourteen Days in February', 2, 2),
(7, 'Sixteen Seconds in March', 2, 2);
```

4. List the title of the book and the first and last name of the author in phpMyAdmin as shown in Figure 24-20.

title	first_name	last_name
Green Trees Go Wild	Sally	Meyers
And Then It Happened	Sally	Meyers
Fifteen Hours in March	George	Smith
Fourteen Days in February	George	Smith
Sixteen Seconds in March	George	Smith
A Long Day in Spring	Peter	Gabriel

FIGURE 24-20

```
SELECT title, first_name, last_name
FROM books JOIN authors AS a ON author = a.id
```

5. Take the MySQL statement in step 4, sort it by title, and display it using PHP as shown in Figure 24-21.

```
A Long Day in Spring by Peter Gabriel
And Then It Happened by Sally Meyers
Fifteen Hours in March by George Smith
Fourteen Days in February by George Smith
Green Trees Go Wild by Sally Meyers
Sixteen Seconds in March by George Smith
```

FIGURE 24-21

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {

  // Set up the query
  $query = "SELECT title, first_name, last_name "
    . " FROM books JOIN authors AS a ON author = a.id "
    . " ORDER BY `title` ASC "
    ;

  // Run the query
   $result_obj = '';
   $result_obj = $connection->query($query);

  // Read the results
  // loop through the results, row by row
  // reading each row into an associative array
  while($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
    // collect the array
    $items[] = $result;
  }
  // print array when done
  foreach ($items as $item) {
    echo $item['title']. ' by ' . $item['first_name']. ' ' . $item['last_
name'];
    //print_r($item);
    echo '<br />';
  }
}
```

6. List the type name, title, and full name of the author for any titles that contain "Day." Put in sequence by the type in descending sequence, then the title. The results are shown in Figure 24-22.

```
SELECT type_name, title, CONCAT(last_name, ', ', first_name) AS full_name
FROM books AS b
JOIN authors AS a ON author = a.id
JOIN types AS t ON b.type_id = t.type_id
WHERE title LIKE '%Day%'
ORDER BY type_name DESC, title;
```

type_name ▾	title	full_name
Suspense	Fourteen Days in February	Smith, George
History	A Long Day in Spring	Gabriel, Peter

FIGURE 24-22

7. Take the MySQL statement in step 6 and put it into a PHP statement to display as shown in Figure 24-23.

Suspense: Fourteen Days in February by Smith, George
History: A Long Day in Spring by Gabriel, Peter

FIGURE 24-23

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {

  // Set up the query
  $query = "SELECT type_name, title,
    CONCAT(last_name, ', ', first_name) AS full_name "
    . " FROM books AS b "
    . " JOIN authors AS a ON author = a.id "
    . " JOIN types AS t ON b.type_id = t.type_id "
    . " WHERE title LIKE '%Day%' "
    . " ORDER BY type_name DESC, title "
    ;

  // Run the query
   $result_obj = '';
   $result_obj = $connection->query($query);

  // Read the results
  // loop through the results, row by row
  // reading each row into an associative array
  while($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
```

```
        // collect the array
        $items[] = $result;
      }
      // print array when done
      foreach ($items as $item) {
        echo $item['type_name'] . ': ' . $item['title']. ' by ' . $item['full_
name'];
        echo '<br />';
      }
    }
```

Watch the video for Lesson 24 on the DVD or watch online at `www.wrox.com/`
`go/24phpmysql.`

25

Changing Data

In this lesson, you learn how to change the data in your database using the UPDATE command.

There are multiple ways to process MySQL data commands. You have been using variations on similar methods. In this lesson you learn a new way with *prepared statements* where you set up the statements first and then you supply the field data separately. Prepared statements are inherently safer because they discourage SQL injections.

The examples from this lesson use the same tables used in the examples in Lesson 24.

> *You can download the code for this example from the book's web page at* www.wrox.com. *You can find them in the Lesson25 folder in the download in a file labeled* lesson25a.sql.

The first table is the authors table as shown in Figure 25-1.

The second table is the types table as shown in Figure 25-2.

The third table is the books table as shown in Figure 25-3.

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Peter	Gabriel
5	Dale	Mercer

FIGURE 25-1

type_id	type_name
1	History
2	Suspense
3	Science Fiction

FIGURE 25-2

id	title	author	type_id
1	A Long Day in Spring	3	1
2	Fifteen Hours in March	2	2
3	Green Trees Go Wild	1	3
4	And Then It Happened	1	1
5	Missing in Action	5	2
6	Fourteen Days in February	2	2
7	Sixteen Seconds in March	2	2

FIGURE 25-3

USING THE UPDATE COMMAND

The UPDATE command is used to change the data in your tables. To change row 1 in the authors table from Sally Meyers to Sarah Meyers, use the following code. The result is shown in Figure 25-4.

id	first_name	last_name
1	Sarah	Meyers
2	George	Smith
3	Peter	Gabriel
4	Dale	Mercer

FIGURE 25-4

```
UPDATE authors SET first_name = 'Sarah' WHERE id = '1';
```

If you leave off the WHERE clause, all rows are updated. See Figure 25-5 for the results of the following code:

```
UPDATE authors SET first_name = 'Sarah';
```

id	first_name	last_name
1	Sarah	Meyers
2	Sarah	Smith
3	Sarah	Gabriel
4	Sarah	Mercer

FIGURE 25-5

> ✖ *If you do not specify a WHERE clause, the default is to select all rows. With the UPDATE command that means that you could accidently update all rows if you forget your WHERE clause.*

Unlike INSERT, there is no way to update multiple rows with different data, so to restore the first names you need four rows:

```
UPDATE authors SET first_name = 'Sally' WHERE id = '1';
UPDATE authors SET first_name = 'George' WHERE id = '2';
UPDATE authors SET first_name = 'Peter' WHERE id = '3';
UPDATE authors SET first_name = 'Dale' WHERE id = '4';
```

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Peter	Gabriel
4	Jeff	Baamer

FIGURE 25-6

You can, however, update more than one field at a time. This code gives the results shown in Figure 25-6:

```
UPDATE authors SET first_name = 'Jeff', last_name = 'Baamer' WHERE id = '4';
```

You can use clauses you learned for the SELECT statement in the UPDATE statement. The following code updates the title field in the books table by adding an asterisk (*) to the end of the title for any books written by someone with the name of Sally. See the results in Figure 25-7.

```
UPDATE books JOIN authors AS a ON author = a.id
   SET title = CONCAT(title, '*') WHERE first_name = 'Sally';
```

id	title	author	type_id
1	A Long Day in Spring	3	1
2	Fifteen Hours in March	2	2
3	Green Trees Go Wild*	1	3
4	And Then It Happened*	1	1
5	Missing in Action	5	2
6	Fourteen Days in February	2	2
7	Sixteen Seconds in March	2	2

FIGURE 25-7

You can also use subqueries, though the table you are updating cannot be in the subquery. This code appends a double asterisk (**) to any titles in books that have no entry in the authors table. See the results in Figure 25-8.

```
UPDATE books SET title = CONCAT(title, '**')
  WHERE author NOT IN (SELECT id FROM authors);
```

id	title	author	type_id
1	A Long Day in Spring	3	1
2	Fifteen Hours in March	2	2
3	Green Trees Go Wild*	1	3
4	And Then It Happened*	1	1
5	Missing in Action**	5	2
6	Fourteen Days in February	2	2
7	Sixteen Seconds in March	2	2

FIGURE 25-8

When you INSERT new rows you get an error if you try to insert a row that already exists. This INSERT fails with a Duplicate Entry error:

```
INSERT INTO authors (id, first_name, last_name) VALUES ('4', 'Jane', 'Smith');
```

If you add an ON DUPLICATE KEY UPDATE clause to the INSERT statement, you specify that you want MySQL to update the row if it already exists. See the following code and the results in Figure 25-9.

```
INSERT INTO authors (id, first_name, last_name)
VALUES ('4', 'Jane', 'Smith')
ON DUPLICATE KEY UPDATE first_name = 'Jane', last_name = 'Smith';
```

UPDATING DATA IN PHP

Updating data through a PHP program using MySQL commands takes three steps:

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Peter	Gabriel
4	Jane	Smith

FIGURE 25-9

1. Make a connection to the database.

2. Create a safe query with the command.

3. Run the query.

The following code uses the MySQL UPDATE command to change fields in the table authors. See the result in Figure 25-10.

```
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
```

```
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  $first_name = "Liam";
  $last_name = "O'Reilly";
  $id = 4;

  // Prepare the data
  if (get_magic_quotes_gpc()) {
    $first_name = stripslashes($first_name);
    $last_name = stripslashes($last_name);
  }
  $first_name = $connection->real_escape_string($first_name);
  $last_name = $connection->real_escape_string($last_name);
  $id = (int) $id;

  // Set up the query
    $query = "UPDATE `authors` "
    . " SET `first_name`= '$first_name', `last_name` = '$last_name' "
    . " WHERE `id` = '$id'";

  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "No rows updated<br />";
  } else {
    echo $result . " row(s) successfully updated<br />";
  }
}
```

Like the INSERT you learned in Lesson 22, you need to prepare the data before you post it to the database using PHP with UPDATE and REPLACE. The preparation consists of making it safe against SQL injections and to deal with quotes in string values because the wrong type of quote in the wrong place might be seen as a MySQL control character and invalidate the command.

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Peter	Gabriel
4	Liam	O'Reilly

FIGURE 25-10

Compare these lines of code:

```
$result_obj = $connection->query("SELECT first_name FROM authors");
$result = $connection->query("UPDATE authors SET first_name='Sally' WHERE id=4");
```

In the first line of code, the mysqli::query() method returns an object because it processes a SELECT command. In this example, I am assigning it to a variable called $result_obj as a reminder that it is an object, though it can be named anything. This object then needs to be read to get the results of the SELECT statement. You learned several ways to read the result object in Lesson 23.

In the second line of code, the mysqli::query() method returns a Boolean (TRUE or FALSE) that indicates the success of the action, in this case, the UPDATE command.

An object is returned by mysqli::query() when the command is SELECT, SHOW, DESCRIBE, or EXPLAIN. The last three commands are used to show information about the structure of the

database or internal details that can be used to optimize performance. All other commands, such as INSERT, UPDATE, REPLACE, and DELETE just return TRUE or FALSE.

You have been primarily using the object-oriented method of the mysqli connection, but the code can also be written with the procedural version as in this code and as shown in Figure 25-11:

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @mysqli_connect(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if (mysqli_connect_error()) {
  die('Connect Error: ' . mysqli_connect_error());
} else {
  echo 'Successful connection to MySQL <br />';

  $first_name = "Danny";
  $last_name = "O'Murphy";
  $id = 4;
  var_dump($connection);

  // Prepare the data
  if (get_magic_quotes_gpc()) {
    $first_name = stripslashes($first_name);
    $last_name = stripslashes($last_name);
  }
  $first_name = mysqli_real_escape_string($connection, $first_name);
  $last_name = mysqli_real_escape_string($connection, $last_name);
  $id = (int) $id;

  // Set up the query
    $query = "UPDATE `authors` "
    . " SET `first_name`= '$first_name', `last_name` = '$last_name' "
    . " WHERE `id` = '$id'";

  // Run the query and display appropriate message
  if (!$result = mysqli_query($connection, $query)) {
    echo "No rows updated<br />";
  } else {
    echo "Row(s) successfully updated<br />";
  }
}
```

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Peter	Gabriel
4	Danny	O'Murphy

FIGURE 25-11

USING PREPARED STATEMENTS

Prepared statements are relatively new to PHP. With prepared statements, you define a statement with placeholders and then you can rerun the statement multiple times and just give it the place-holder information. Using prepared statements has three main benefits:

➤ They are more secure. SQL injection takes advantage of input variables to pass embedded code that is unconsciously run by the processor. Prepared statements separate running code from the variable values so the embedded code has no place to run.

➤ They can have better performance. This is only true if you reuse the same prepared statement multiple times. The program does the heavy lifting when it sets up the statement and then it's cheap to send new placeholder values.

➤ They are more convenient to write. Just like templates in a word processing program, the constant data is separated out from the dynamic data, which makes the statements easier to work with. Additionally, you don't need to worry about escaping quotes because the data is separate from the command statement.

MYSQLI

The following code creates a prepared statement and then runs it to update the name of the author in row 4 and then in row 3 of the `authors` table. See the result in Figure 25-12 and Figure 25-13.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  $first_name = "Gilly";
  $last_name = "O'Donal";
  $id = 4;

  // Set up the prepared statement
  $query = "UPDATE `authors` "
    . " SET `first_name`= ?, `last_name` = ? "
    . " WHERE `id` = ?";
  // Prepare the statement
  $statement = $connection->prepare($query);
  // Bind the parameters
  $statement->bind_param('ssi',$first_name, $last_name,$id);
  // Run the query
  if (!$result = $statement->execute()) {
    echo "No rows updated<br />";
  } else {
echo "First row successfully updated<br />";
  }

  // Assign new values to the bound variables
  $first_name = "Meg";
  $last_name = "Mitchell";
  $id = 3;

  // Rerun the statement
```

```
    if (!$result = $statement->execute()) {
      echo "No rows updated<br />";
    } else {
      echo "Second row successfully updated<br />";
    }
    // Close the statement
    $statement->close();


  }
```

Successful connection to MySQL
First row successfully updated
Second row successfully updated

FIGURE 25-12

			id	first_name	last_name
☐	✎	✗	1	Sally	Meyers
☐	✎	✗	2	George	Smith
☐	✎	✗	3	Meg	Mitchell
☐	✎	✗	4	Gilly	O'Donal

FIGURE 25-13

As you see in the example, first you set up the prepared statement. To do that, replace the variables with a ? (? is the placeholder). Certain restrictions exist on what you can replace as placeholders. For the most part you can use them where you put data but not as table names or field names.

After you have set up the statement, you prepare the statement with `mysqli::prepare()`. You could do both these steps in one step, but setting up the statement in a variable first makes it easier to debug all the quotes and concatenations because you can place a `var_dump($query);` after you've created it. That displays a MySQL command that you can examine for errors.

Now that you've prepared the statement, you bind the placeholders to the statement with `mysqli::bind_param()`. This assigns the values you want to use for this processing of the prepared statement. Binding requires two pieces of information for each placeholder: the data type and the value. The placeholder parameters must be specified in the same order as they appear in the statement. In the example, the type is `'ssi'` because `$first_name` and `$last_name` are strings and `$id` is an integer. The valid codes are listed in Table 25-1.

TABLE 25-1: Parameter Data Types

CODE	DATA TYPE
i	Integer
d	Double (numeric with decimals)
s	String (Text)
b	Blob (Lots of binary characters)

After you bind the placeholder parameters, you execute the statement. Because you are binding the variables themselves, you can change the value of the variables and execute again with the different values. When you are done, close the statement. You can have only one prepared statement open at a time.
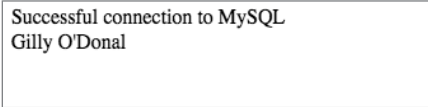
You can use prepared statements for SELECT commands as well. Here you bind the results as well as any parameters. See the results in Figure 25-14.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  $id = 4;

  // Set up the prepared statement
  $query = "SELECT `first_name`, `last_name` "
    . " FROM `authors`"
    . " WHERE `id` = ?";
  // Prepare the statement
  $statement = $connection->prepare($query);
  // Bind the parameters
  $statement->bind_param('i',$id);
  // Execute the statement
  $statement->execute();
  // Bind the results
  $statement->bind_result($first,$last);
  // Run the query
  $statement->fetch();
  echo $first . ' ' . $last . '<br />';
  // Close the statement
  $statement->close();
}
```

```
Successful connection to MySQL
Gilly O'Donal
```

FIGURE 25-14

PHP Data Objects (PDO)

PHP Data Objects (PDO) has its own version of prepared statements, which have some advantages over the mysqli version. You have the ability to use named parameters as well as unnamed parameters. To use named parameters, replace the question mark (?) with a name prefixed with a colon (:). The PDO_Statement::bindParm() method assigns the variable that supplies the data, as well as output data. The data type is specified with predefined PDO Constants, which are listed in Table 25-2.

TABLE 25-2: PDO Data Types

CONSTANT	DESCRIPTION
PDO::PARAM_INT	Integers
PDO::PARAM_STR	Strings
PDO::PARAM_BOOL	Boolean
PDO::PARAM_LOB	Blobs

Run the following code to see the results in Figure 25-15 and Figure 25-16:

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
if (!$connection =
  new PDO('mysql:host='.HOSTNAME.';dbname=' . MYSQLDB, MYSQLUSER, MYSQLPASS)) {
  die('Connect Error');
} else {
  echo 'Successful connection to MySQL <br />';

  $first_name = "Paddy";
  $last_name = "O'Brian";
  $id = 4;

  // Set up the prepared statement
  $query = "UPDATE `authors` "
    . " SET `first_name`= :first_name, `last_name` = :last_name "
    . " WHERE `id` = :id";
  // Prepare the statement
  $statement = $connection->prepare($query);
  // Bind the parameters
  $statement->bindParam(':first_name', $first_name, PDO::PARAM_STR, 50);
  $statement->bindParam(':last_name', $last_name, PDO::PARAM_STR, 50);
  $statement->bindParam(':id', $id, PDO::PARAM_INT);
  // Run the query
  if (!$result = $statement->execute()) {
    echo "No rows updated<br />";
  } else {
    echo "First row successfully updated<br />";
  }
  // Change the value in the variable and rerun
  $id = 3;
  $first_name = 'Nancy';
  $last_name = 'Misson';

  // Rerun the statement
  if (!$result = $statement->execute()) {
```

```
      echo "No rows updated<br />";
   } else {
      echo "Second row successfully updated<br />";
   }
}
```

Successful connection to MySQL
First row successfully updated
Second row successfully updated

FIGURE 25-15

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Nancy	Misson
4	Paddy	O'Brian

FIGURE 25-16

TRY IT

Available for download on Wrox.com

In this Try It, you add maintenance to the `contacts` and `categories` tables in the Case Study. You modify the add pages for the contacts and lot categories so that you can edit contacts and lot categories.

> *You can download the code and resources for this Try It from the book's web page at* `www.wrox.com`. *You can find them in the Lesson25 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 23. Alternatively, you can download the files from the book's website at `www.wrox.com`. To re-create the database tables, create an empty database and then import the `install.sql` file in phpMyAdmin.

Hints

When adding rows, you don't need to know the primary key of the row because the primary key is automatically created. When editing rows, you need to pass that key along so that you always know what it is.

Step-by-Step

Change the contact add page so that it edits existing records.

1. In the `contents/about.php` file add an edit button to link to the maintenance page with the `id` of the current contact in the URL. This is the current `<h2>` line:

```
<h2><?php echo htmlspecialchars($item->name()); ?></h2>
```

Replace it with this new `<h2>` line:

```
<h2><?php echo htmlspecialchars($item->name()); ?>
<a class="button"
  href="index.php?content=contactmaint&id=<?php echo $item->getId(); ?>">
Edit</a></h2>
```

2. In the `contents/contactmaint.php` file check for the `id` in the URL. If it is not 0 then get the row from the table to create the `Contact` object. Do this by changing `$item = new Contact;` to the following code:

```
$id = (int) $_GET['id'];
// Is this an existing item or a new one?
if ($id) {
  // Get the existing information for an existing item
  $item = Contact::getContact($id);
} else {
  // Set up for a new item
  $item = new Contact;
}
```

3. Change the legend to show the `id` if this is an existing contact:

```
<legend><?php echo ($id) ? 'ID: '. $id : 'Add a Contact' ?></legend>
```

4. Add the `getContact()` method in the `includes/classes/contact.php` file to create a `Contact` object from a row in the `contacts` table:

```
public static function getContact($id) {
  // Get the database connection
  $connection = Database::getConnection();
  // Set up the query
  $query = 'SELECT * FROM `contacts` WHERE id="'. (int) $id.'"';
  // Run the MySQL command
  $result_obj = '';
    try {
      $result_obj = $connection->query($query);
      if (!$result_obj) {
        throw new Exception($connection->error);
      } else {
        $item = $result_obj->fetch_object('Contact');
        if (!$item) {
```

```
                    throw new Exception($connection->error);
                } else {
                    // pass back the results
                    return($item);
                }
            }
        }
        catch(Exception $e) {
            echo $e->getMessage();
        }
    }
```

5. You need to change the processing of the `contact.maint` task. In the `includes/init.php` file, locate the `contact.maint case`. The header redirect needs the id of the row. That id is added to the `$result` array from the `editRecord()` method and passed through the `editContact()` function.

    ```
    header("Location: index.php?content=contactmaint&id=$results[2]");
    ```

6. In the `includes/functions.php` file change the `maintContact()` function to call the `editRecord()` function if the id already exists. This is the current code:

    ```
    // Set up a Contact object based on the posts
    $contact = new Contact($item);
    $results = $contact->addRecord();
    ```

    Replace it with this new code:

    ```
    // Set up a Contact object based on the posts
    $contact = new Contact($item);
    if ($contact->getId()) {
     $results = $contact->editRecord();
    } else {
     $results = $contact->addRecord();
    }
    ```

7. Back in the `includes/classes/contact.php` file add the `editRecord()` method to update the table. You need the id to create the URL to redirect to the correct page if there is an error, so add a third element to the return array.

    ```
        public function editRecord() {
            // Verify the fields
            if ($this->_verifyInput()) {

                // Get the Database connection
                $connection = Database::getConnection();

                // Set up the prepared statement
                $query = 'UPDATE `contacts`
                    SET first_name=?, last_name=?, position=?, email=?, phone=?
                    WHERE id=?';
                $statement = $connection->prepare($query);
                // bind the parameters
                $statement->bind_param('sssssi',
                    $this->first_name, $this->last_name, $this->position,
    ```

```
        $this->email, $this->phone, $this->id);
    if ($statement) {
      $statement->execute();
      $statement->close();
      // add success message
      $return = array('', 'Contact Record successfully added.', '');
      return $return;
    } else {
      $return = array('contactmaint',
        'No Contact Record Added. Unable to create record.' ,
        (int) $this->id);
      return $return;
    }

  } else {
    // send fail message and return to contactmaint
    $return = array('contactmaint',
      'No Contact Record Added. Missing required information.' ,
     (int) $this->id);
    return $return;
  }

}
```

8. Go to the About Us page as shown in Figure 25-17.
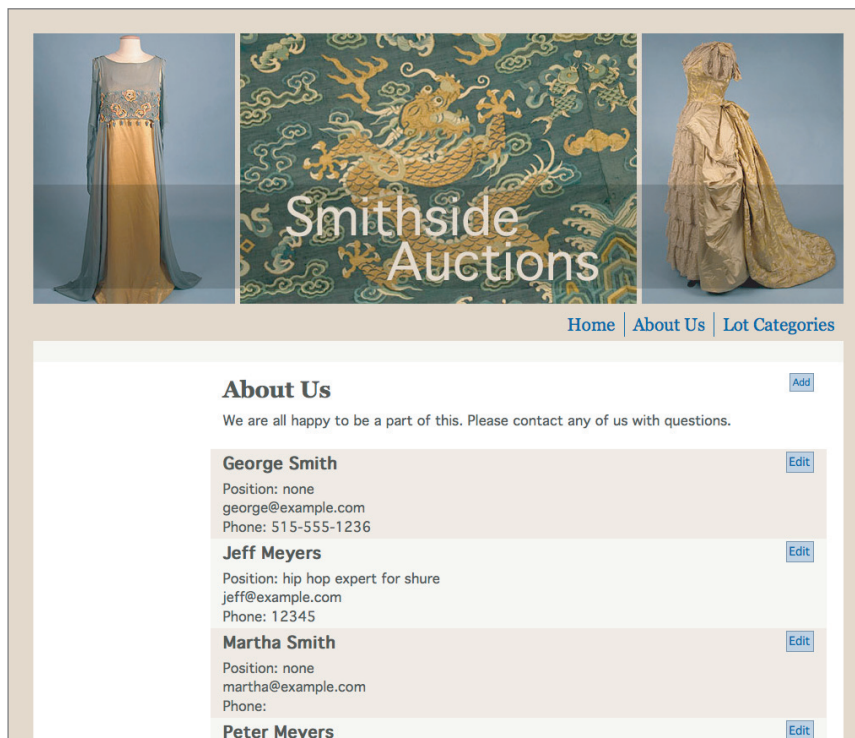
FIGURE 25-17

9. Click the Edit button on George Smith and you see a page similar to Figure 25-18. Make changes and click Save to save the changes.
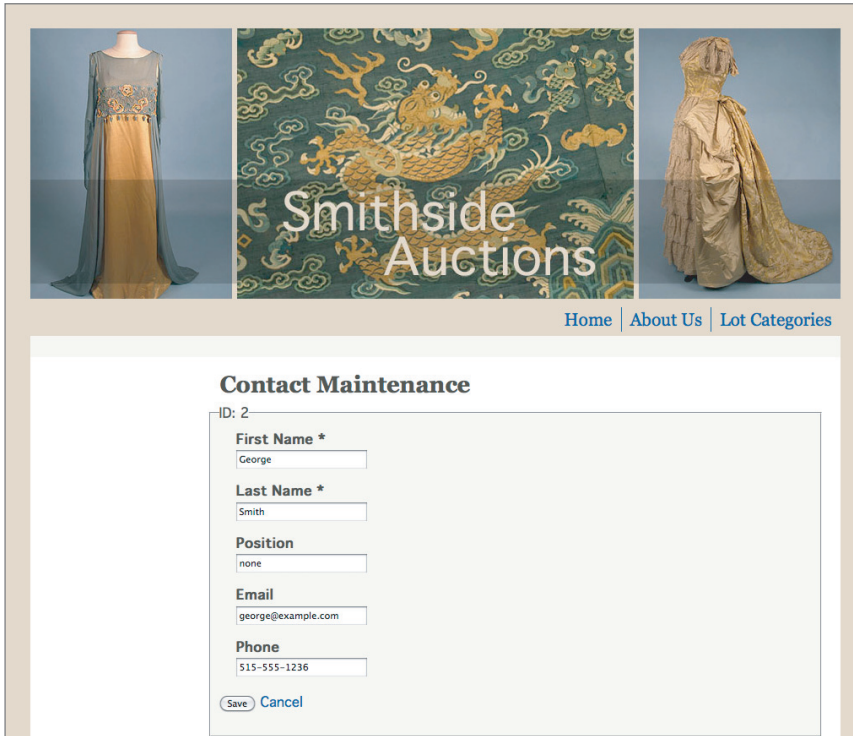
FIGURE 25-18

10. In step 7 you added a third element to the array that returns out of `editRecord()`. You need to add that element to the array that `addRecord()` returns, as shown in the following code from `includes/classes/contact.php`.

```
  // Run the MySQL statement
  if ($connection->query($query)) {
  $return = array('', 'Contact Record successfully added.', '' );

  // add success message
  return $return;
  } else {
  // send fail message and return to contactmaint
  $return = array('contactmaint', 'No Contact Record Added. Unable to ↵
 create record.', '');
  return $return;
  }
} else {
  // send fail message and return to contactmaint
  $return = array('contactmaint', 'No Contact Record Added. Missing ↵
required information.', '0');
  return $return;
}
```

Change the category add page so that it edits existing records.

1. In the `contents/categories.php` file add an Edit button to link to the maintenance page with the id of the current category in the URL. Put it below the Display Lots button:

```
<a class="button edit"
  href="index.php?content=categorymaint&cat_id=<?php echo $item->getCat_id();
?>">
 Edit</a>
```

2. In the `contents/categorymaint.php` file, check for the category id in the URL. If it is not 0 then get the row from the table to create the `Category` object. Do this by changing `$item = new Category;` to the following code:

```
$id = (int) $_GET['cat_id'];
// Is this an existing item or a new one?
if ($id) {
  // Get the existing information for an existing item
  $item = Category::getCategory($id);
} else {
  // Set up for a new item
  $item = new Category;
}
```

3. Change the legend to show the id if this is an existing category:

```
<legend><?php echo ($id) ? 'ID: '. $id : 'Add a Category' ?></legend>
```

4. Add the `getCategory()` method in the `includes/classes/category.php` file to create a `Category` object from a row in the `categories` table:

```
public static function getCategory($cat_id) {
  // Get the DB connection
  $connection = Database::getConnection();
  // Prepare the query
  $query = 'SELECT * FROM `categories` WHERE cat_id="'. (int) $cat_id.'"';

  // Run the MySQL command
  $result_obj = $connection->query($query);
  try {
  while($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
      $item = new Category($result);
    }
      // pass back the results
    return($item);
  }
  catch(Exception $e) {
      return false;
  }
}
```

5. You need to change the processing of the `category.maint` task. In the `includes/init.php` file locate the `category.maint case`. The header redirect needs the id of the row. That id is added to the `$result` array from the `editRecord()` method and passed through the `editCategory()` function.

```
header("Location: index.php?content=categorymaint&cat_id=$results[2]");
```

6. In the `includes/functions.php` change the `maintCategory()` function to call the `editRecord()` function if the id already exists. This is the current code:

```
// Set up a Category object based on the posts
$category = new Category($item);
$results = $category->addRecord();
```

Replace it with this new code:

```
// Set up a Category object based on the posts
$category = new Category($item);
if ($category ->getGet_id()) {
 $results = $category ->editRecord();
} else {
 $results = $category ->addRecord();
}
```

7. Back in the `includes/classes/category.php` file, add the `editRecord()` method to update the table. You need the id to create the URL to redirect to the correct page if there is an error, so add a third element to the return array.

```
public function editRecord() {
  // Verify the fields
  if ($this->_verifyInput()) {

    // Get the Database connection
    $connection = Database::getConnection();

    // Set up the prepared statement
    $query = 'UPDATE `categories`
      SET cat_name=?, cat_description=?, cat_image=?
      WHERE cat_id=?';
    $statement = $connection->prepare($query);
    // bind the parameters
    $statement->bind_param('sssi',
      $this->cat_name, $this->cat_description, $this->cat_image,
      $this->cat_id);
    if ($statement) {
      $statement->execute();
      $statement->close();
      // add success message
      $return = array('', 'Category Record successfully added.', '');
      return $return;
    } else {
      $return = array('categorymaint',
        'No Category Record Added. Unable to create record.',
        (int) $this->cat_id);
      return $return;
    }

  } else {
    // send fail message and return to categorymaint
    $return = array('contactmaint',
      'No Contact Record Added. Missing required information.',
```

```
        (int) $this->cat_id);
      return $return;
    }

  }
```

8. Go to the Lot Categories page as shown in Figure 25-19.
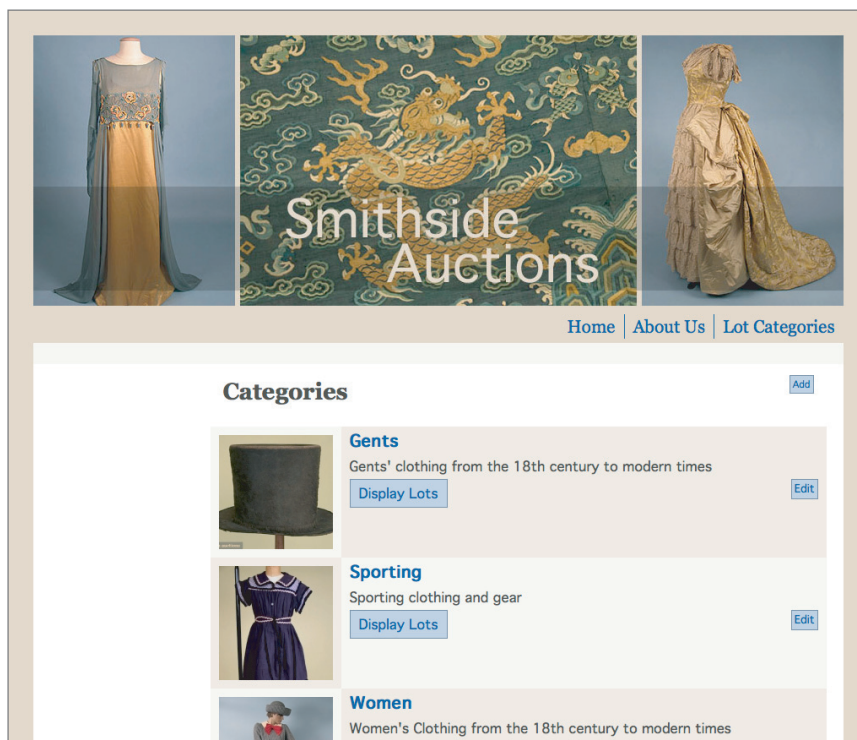
FIGURE 25-19

9. Click the Edit button on the Gents category and you see a page similar to Figure 25-20. Make changes and click Save to save the changes.

10. In step 7, you added a third element to the array that returns out of `editRecord()`. You need to add that element to the array that `addRecord()` returns as shown in the following code from `includes/classes/contact.php`.

```
// Run the MySQL statement
if ($connection->query($query)) {
  $return = array('', 'Category Record successfully added.','');

  // add success message
  return $return;
} else {
  // send fail message and return to categorymaint
```

```
  $return = array('contactmaint', 'No Category Record Added. Unable to ↵
create record.','');
  return $return;
  }
} else {
  // send fail message and return to categorymaint
  $return = array('categorymaint', 'No Category Record Added. Missing ↵
required information.','0');
  return $return;
}
```


FIGURE 25-20

26

Deleting Data

In this lesson you learn how to delete rows from tables in the MySQL database. The examples from this lesson use the same tables used in the examples in Lesson 25.

> *You can download the code for this example from the book's web page at* www.wrox.com. *You can find it in the Lesson 26 folder in the download in a file labeled* lesson26a.sql.

The first table is the authors table as shown in Figure 26-1.

The second table is the types table as shown in Figure 26-2.

The third table is the books table as shown in Figure 26-3.

id	first_name	last_name
1	Sally	Meyers
2	George	Smith
3	Nancy	Misson
4	Paddy	O'Brian

FIGURE 26-1

type_id	type_name
1	History
2	Suspense
3	Science Fiction

FIGURE 26-2

id	title	author	type_id
1	A Long Day in Spring	3	1
2	Fifteen Hours in March	2	2
3	Green Trees Go Wild*	1	3
4	And Then It Happened*	1	1
5	Missing in Action**	5	2
6	Fourteen Days in February	2	2
7	Sixteen Seconds in March	2	2

FIGURE 26-3

USING THE DELETE COMMAND

Later on in this lesson I ask you to restore the example tables back to this point. If you are entering the examples, then you should make a backup by exporting the files now. If you are using the downloaded file lesson26a.sql, then you can just use that file when you are asked to restore.

> *If you don't remember how to back up MySQL tables by exporting them, see "Backing Up and Restoring" in Lesson 19.*

The DELETE command is used to remove rows from your tables. To remove row 2 from the types table you use the following code. The results are shown in Figure 26-4.

type_id	type_name
1	History
3	Science Fiction

```
DELETE FROM types WHERE type_id = '2';
```

FIGURE 26-4

If you don't specify a WHERE clause, all rows are deleted from the table unless you use a LIMIT clause, in which case, the rows selected with the LIMIT clause are deleted. The following code deletes all the rows in the types table:

```
DELETE FROM types;
```

> *If you do not specify a WHERE clause, the default is to select all rows. With the DELETE command, that means that you could accidently delete all rows if you forget your WHERE clause.*

If you want to delete all the rows in a table, best practice is to use the TRUNCATE command as shown in the following code:

```
`TRUNCATE types`
```

See what happens when you add a row back in to the types table as shown in Figure 26-5.

type_id	type_name
4	Fantasy

```
INSERT INTO types VALUES (NULL, 'Fantasy');
```

FIGURE 26-5

Notice that even though there is only one row in the table, the value automatically assigned to type_id is 4. If you are using auto_increment, deleting rows, even all the rows, does not affect it. If the next row was to be assigned 4 before you deleted all the rows, the next row you add is still assigned a 4. This is true for the MyISAM tables and, for the most part, InnoDB tables as well.

There's no way to undelete deleted rows. They are gone. You need to re-insert them if you want to restore them. If you need to synchronize the primary key with other files, you need to specify the correct value rather than letting auto_increment assign it. The following code explicitly lists the values to be used for the type_id field. As you see in Figure 26-6, these inserted rows are created with the specified values rather than being automatically generated by the next higher number.

type_id	type_name
4	Fantasy
1	History
2	Suspense
3	Science Fiction

FIGURE 26-6

```
INSERT INTO `types` (`type_id`, `type_name`) VALUES
(1, 'History'),
(2, 'Suspense'),
(3, 'Science Fiction');
```

You can use the `LIMIT` clause to limit how many records are deleted. You can use this if you have a lot of rows to delete that might exceed the time limit. You would run the command multiple times until all the required rows are deleted. The following example deletes one row. See the results in Figure 26-7.

FIGURE 26-7

```
DELETE FROM types LIMIT 1;
```

The `ORDER BY` clause determines the order in which rows are deleted. The following code deletes the first two books when the titles are in alphabetical order, as shown in Figure 26-8:

```
DELETE FROM books ORDER BY title ASC LIMIT 2;
```


FIGURE 26-8

The `WHERE` clause is the same used by the `SELECT` statement, so it can be as complex as you need. For instance, you can use more than one condition in selecting the rows to be deleted. This example deletes any books written by author 2 that have "Days" in the title. See Figure 26-9.

```
DELETE FROM books WHERE author = '2' AND title LIKE '%Days%';
```


FIGURE 26-9

This example uses a subquery to delete any book that has an invalid author. See Figure 26-10.

```
DELETE FROM books WHERE author NOT IN (SELECT id FROM authors);
```


FIGURE 26-10

After all this deleting from the `books` table, only two authors in the `authors` table still have books in the `books` table. So here you use the `JOIN` clause to locate the orphan authors and remove them.

The LEFT JOIN selects all authors (because they are on the "left" side). Any author without any matching books has the book fields assigned as NULL values. The field author (from the books table) should always be equal to the primary key, id, in the authors table because that is what merges the rows, so if the field author is NULL, you know that no match was found. See Figure 26-11.

```
DELETE authors FROM authors LEFT JOIN books ON authors.id = author
    WHERE author IS NULL;
```

DELETING DATA IN PHP

Now that you've succeeded in deleting most of the database, restore it to the state it was in at the beginning of the lesson so you can delete it using PHP. Restore using either the lesson26a.sql file you imported at the beginning of the lesson or the file you exported at the beginning of the lesson.

id	first_name	last_name
1	Sally	Meyers
2	George	Smith

FIGURE 26-11

> *If you don't remember how to import a .sql file, see "Backing Up and Restoring" in Lesson 19.*

Deleting data using PHP is similar to the way you INSERT data, though you usually don't have as many fields to prepare because the only relevant fields are those you need for the selection process. Easiest of all is deleting rows when you know the primary key and that key is an integer type. The following example deletes the book with the id of 6. Figure 26-12 shows what running the PHP program looks like and Figure 26-13 shows the books table in phpMyAdmin after the deletion has taken place.
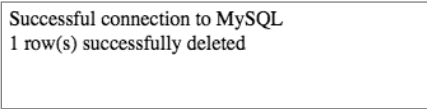
```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  // assign an id value for the test
  $id = 6;

  // Set up the query
  $query = 'DELETE FROM `books` WHERE id="'. (int) $id.'"';
  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "No rows deleted<br />";
  } else {
    echo $connection->affected_rows . " row(s) successfully deleted<br />";
```

```
        }
    }
```

The `mysqli->affected_rows` is a property in the `mysqli` class that tells you the number of affected rows in the last MySQL command. In this example this is represented by the `$connection->affected_rows`.

Successful connection to MySQL
1 row(s) successfully deleted

FIGURE 26-12

id	title	author	type_id
1	A Long Day in Spring	3	1
2	Fifteen Hours in March	2	2
3	Green Trees Go Wild*	1	3
4	And Then It Happened*	1	1
5	Missing in Action**	5	2
7	Sixteen Seconds in March	2	2

FIGURE 26-13

TRY IT

In this Try It, you add delete capabilities to the About Us and Lot Categories pages by adding Delete buttons and a confirmation page. The Lesson26 folder on the website contains the interim files as of the end of this part.

You have been updating the About Us and Lot Categories pages piece by piece through the last several chapters to use the database instead of hardcoded data. As they say in computerese, you have created the CRUD for those tables: Create, Review, Update, and Delete. The last table, and related pages, is the `lots` table.

In this Try It, you bring the website all the way from multiple hardcoded gents, women, and sporting pages to full display and maintenance in integrated lots pages. You preselect which lots display based on the lot category they are in. You also create a drop-down input select based on the `categories` table that is used for selection of the appropriate category.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson26 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 25. Alternatively, you can download the files from the book's website at `www.wrox.com`.

Hints

The steps that you take to delete a row are the same basic steps you need to take to change a row: select the row and display it, process the input from the user, and then either update the table or give the user an error message.

When you display the information from the row, don't use label and input tags because you don't want the user thinking that he can change the data.

When you process the form, the only vital piece of information is the id of the row in the table. Because that is an integer, you can use `(int)` to force the information from the user into an integer as your safety processing.

Step-by-Step

Add delete capabilities to the About Us page.

1. In the `contents/about.php` files add a Delete button in the `<h2>` tag:

```
<h2><?php echo htmlspecialchars($item->name()); ?>
<a class="button"
href="index.php?content=contactdelete&id=<?php echo
$item->getId(); ?>">Delete</a>
<a class="button"  href="index.php?content=contactmaint&id=<?php echo ↵
$item->
    getId(); ?>">Edit</a>
</h2>
```

2. Create the `contents/contactdelete.php` file. This is similar to the `contactmaint.php` file except that you just display the data instead of using labels and inputs. You won't have new items on this page, so you only need to look for existing items. The submit button is a Delete button instead of a Save and the task is `contact.delete`.

```
<?php
/**
 * contactdelete.php
 *
 * Delete the Contacts
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
$id = (int) $_GET['id'];
// Get the existing information for an existing item
$item = Contact::getContact($id);
```

```
?>
<h1>Contact Delete</h1>

<form action="index.php?content=about" method="post" name="maint" id="maint">
  <fieldset class="maintform">
    <legend><?php echo 'ID: '. $id ?></legend>
    <ul>
      <li><strong>First Name:</strong>
        <?php echo htmlspecialchars($item->getFirst_name()); ?></li>
      <li><strong>Last Name:</strong>
        <?php echo htmlspecialchars($item->getLast_name()); ?></li>
      <li><strong>Position:</strong>
        <?php echo htmlspecialchars($item->getPosition()); ?></li>
      <li><strong>Email:</strong>
        <?php echo htmlspecialchars($item->getEmail()); ?></li>
      <li><strong>Phone:</strong>
        <?php echo htmlspecialchars($item->getPhone()); ?></li>
    </ul>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="id" id="id" value="<?php echo $item->getId();
?>" />
    <input type="hidden" name="task" id="task" value="contact.delete" />
    <input type='hidden' name='token' value='<?php echo $token; ?>'/>
    <input type="submit" name="delete" value="Delete" />
    <a class="cancel" href="index.php?content=about">Cancel</a>
  </fieldset>
</form>
```

3. In the includes/init.php file, add the contact.delete case to process the new task:

```
case 'contact.delete' :
// process the delete
$results = deleteContact();
$message .= $results[1];
// If there is redirect information
// redirect to that page
if ($results[0] == 'contactdelete') {
  // pass on new messages
  if ($results[1]) {
    $_SESSION['message'] = $results[1];
  }
  header("Location: index.php?content=contactdelete&id=$results[2]");
  exit;
}
break;
```

4. In the includes/functions.php file, add the deleteContact() function. This function checks the tokens and evokes the static deleteRecord() method in the Contact class.

Because the only piece of information you need to delete the row is the id, there is no need to create a whole object so you can use the static class method.

```php
function deleteContact() {
  $results = '';
  if (isset($_POST['delete']) AND $_POST['delete'] == 'Delete') {
    // check the token
    $badToken = true;
    if (!isset($_POST['token'])
    || !isset($_SESSION['token'])
    || empty($_POST['token'])
    || $_POST['token'] !== $_SESSION['token']) {
      $results = array('',
        'Sorry, go back and try again. There was a security issue.');
      $badToken = true;
    } else {
      $badToken = false;
      unset($_SESSION['token']);

      // Delete the Contact from the table
      $results = Contact::deleteRecord((int) $_POST['id']);
    }
  }
  return $results;
}
```

5. In the `includes/classes/contact.php` file add the `deleteRecord()` method. This method deletes the row based on the id that is passed in as a parameter.

```php
public static function deleteRecord($id) {
    // Get the Database connection
    $connection = Database::getConnection();
    // Set up query
    $query = 'DELETE FROM `contacts` WHERE id="'. (int) $id.'"';
    // Run the query
    if ($result = $connection->query($query)) {
      $return = array('', 'Contact Record successfully deleted.', '');
      return $return;
    } else {
      $return = array('contactdelete', 'Unable to delete Contact.', (int)
$id);
      return $return;
    }
  }
```

6. Check your changes and verify that they look similar to Figure 26-14 and Figure 26-15. Test that you can delete items.
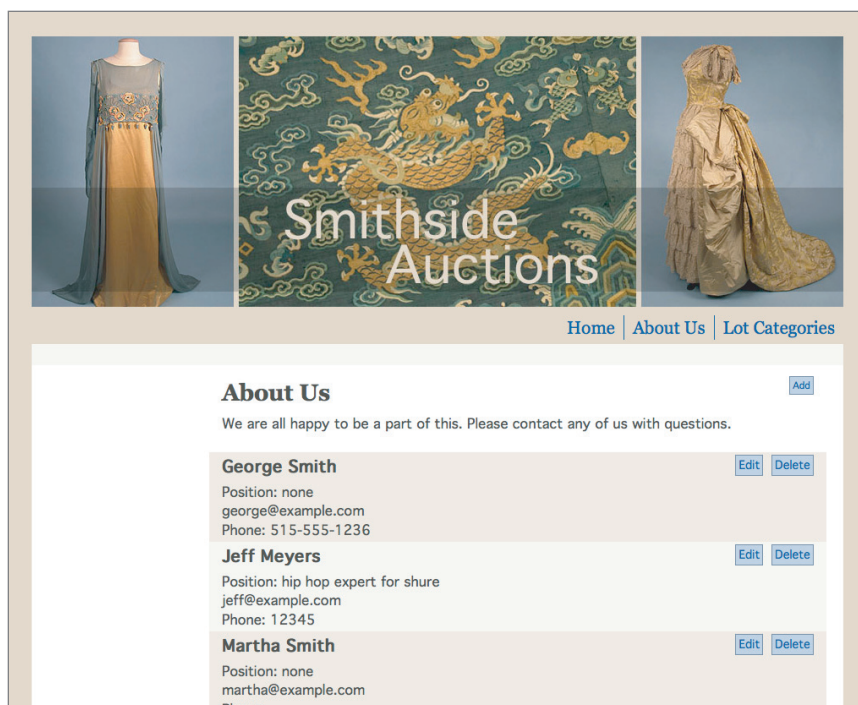
FIGURE 26-14

FIGURE 26-15

Now, repeat the same steps to add delete capabilities to the Lot Categories page.

1. In the `contents/categories.php` files add a Delete button just above the Edit button:

```
<a class="button edit"
  href="index.php?content=categorydelete&cat_id=<?php echo
  $item->getCat_id(); ?>">Delete</a>
```

2. Create the `contents/categorydelete.php` file. This is similar to the `categorymaint.php` file except that you just display the data instead of using labels and inputs. You won't have new items on this page, so you only need to look for existing items. The submit button is a Delete button instead of a Save and the task is `category.delete`.

```php
<?php
/**
 * categorydelete.php
 *
 * Delete for the Categories table
 *
 * @version  1.2 2011-02-03
 * @package  Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license  GNU General Public License
 * @since    Since Release 1.0
 */
$id = (int) $_GET['cat_id'];
// Get the existing information for an existing item
$item = Category::getCategory($id);

?>
<h1>Category Delete</h1>

<form action="index.php?content=categories" method="post" name="maint"
id="maint">

  <fieldset class="maintform">
    <legend><?php echo 'ID: '. $id ?></legend>
    <ul>
    <li><strong>Category:</strong>
      <?php echo htmlspecialchars($item->getCat_name()); ?></li>
    <li><strong>Description</strong><br />
      <?php echo htmlspecialchars($item->getCat_description()); ?></li>
    <li><strong>Image:</strong>
      <?php echo htmlspecialchars($item->getCat_image()); ?></li>
    </ul>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="cat_id" id="cat_id"
      value="<?php echo $item->getCat_id(); ?>" />
```

```
                <input type="hidden" name="task" id="task" value="category.delete" />
                <input type='hidden' name='token' value='<?php echo $token; ?>'/>
                <input type="submit" name="delete" value="Delete" />
                <a class="cancel" href="index.php?content=categories">Cancel</a>
            </fieldset>
        </form>
```

3. In the `includes/init.php` file, add the `category.delete` case to process the new task:

```
        case 'category.delete' :
        // process the maint
        $results = deleteCategory();
        $message .= $results[1];
        // If there is redirect information
        // redirect to that page
        if ($results[0] == 'categorydelete') {
          // pass on new messages
          if ($results[1]) {
            $_SESSION['message'] = $results[1];
          }
          header("Location: index.php?content=categorydelete&cat_
  id=$results[2]");
          exit;
        }
        break;
```

4. In the `includes/functions.php` file, add the `deleteCategory()` function. This function checks the tokens and evokes the static `deleteRecord()` method in the `Category` class. Because the only piece of information you need to delete the row is the id, there is no need to create a whole object so you can use the static class method.

```
  function deleteCategory() {
    $results = '';
    if (isset($_POST['delete']) AND $_POST['delete'] == 'Delete') {
      // check the token
      $badToken = true;
      if (!isset($_POST['token'])
      || !isset($_SESSION['token'])
      || empty($_POST['token'])
      || $_POST['token'] !== $_SESSION['token']) {
        $results = array('',
          'Sorry, go back and try again. There was a security issue.');
        $badToken = true;
      } else {
        $badToken = false;
        unset($_SESSION['token']);

        // Delete the Category from the table
        $results = Category::deleteRecord((int) $_POST['cat_id']);
      }
    }
    return $results;
  }
```

5. In the `includes/classes/category.php` file, add the `deleteRecord()` method. This method deletes the row based on the id that is passed in as a parameter.

```
public static function deleteRecord($id) {
  // Get the Database connection
  $connection = Database::getConnection();
  // Set up query
  $query = 'DELETE FROM `categories` WHERE cat_id="'. (int) $id.'"';
  // Run the query
  if ($result = $connection->query($query)) {
    $return = array('', 'Category Record successfully deleted.', '');
    return $return;
  } else {
    $return = array('categorydelete', 'Unable to delete Category.', (int)
$id);
    return $return;
  }
}
```

6. Check your changes and verify that they look similar to Figure 26-16 and Figure 26-17. Test that you can delete items.

FIGURE 26-16
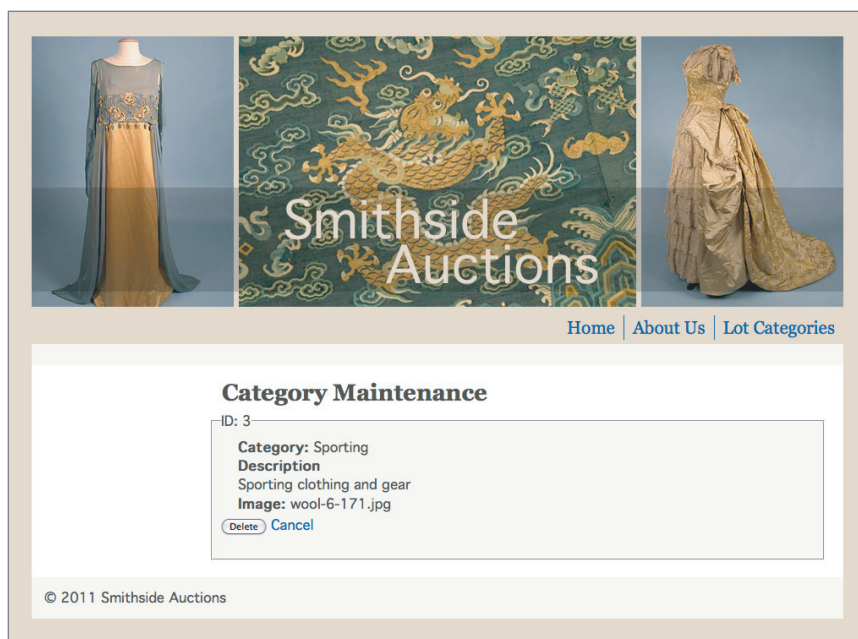
FIGURE 26-17

The rest of this Try It section turns the pages dealing with the lots from static to dynamic. First you change the multiple pages that display lots into a single page and then you add the CRUD for the tables.

Merge the `content/gents.php`, `content/sporting.php`, and `content/women.php` pages into a single `content/lots.php` page where you get the information from the database. Change the lot categories links in the `content/categories.php` file and in the `content/catnav.php` file to work with the new `content/lots.php`.

1. Copy `contents/gents.php` to a new file called `contents/lots.php` and add documentation at the beginning of the file:

```
/**
 * lots.php
 *
 * Content for Lots pages
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
```

2. The category id, that is, the value of the primary key from the `categories` table and the matching value in the `lots` table, is in the URL. Retrieve it with `$_GET['cat_id']`. The lots are all in this category that you use when you retrieve the lots. `Category::getCategory` `($cat_id_in)` creates an object containing the information from the `categories` table for that category.

```php
// Get the Category
$cat_id_in = (int) $_GET['cat_id'];
$category = Category::getCategory($cat_id_in);
```

3. Follow that with the code to load the `$lots` array using the static `getLots()` method from the `Lot` class. This replaces the hardcoded assignments of the existing `$lots` array. Pass in the category id so you can use it in the `getLots()` method. If nothing is returned, initialize `$lots` to an array so that later use of the variable doesn't create errors.

```php
// Get the lot information
$lots = Lot::getLots($cat_id_in);
if (empty($lots)) {
  $lots = array();
}
?>
```

4. In the `<h1>` header, replace the "Gents" text with the category name and add a link to the new data entry page. Give it the class `button` for the CSS styling.

```php
<h1>Product Category: <?php echo $category->getCat_name(); ?>
<a class="button"
   href="index.php?content=lotmaint&cat_id=<?php echo $cat_id_in; ?>&lot_
id=0">
   Add</a>
</h1>
```

5. In the `<li>` block change all the `$lot` array notations to use get methods for the object. For instance, change `$lot['image']` to `$lot->getLot_image()`. Add an Edit button linking to the maintenance page and a Delete button to link to the delete page.

```php
<div class="list-photo">
  <?php // Set up the images
  $image = 'images/'. $lot->getLot_image();

  $image_t = 'images/thumbnails/'. $lot->getLot_image();
  if (!is_file($image_t)) :
    $image_t = 'images/thumbnails/nophoto.jpg';
  endif;

  if (is_file($image)) :
  ?>
    <a href="<?php echo $image; ?>">
    <img src="<?php echo $image_t; ?>"  alt="" />
    </a>
  <?php else : ?>
    <img src="<?php echo $image_t; ?>"  alt="" />
  <?php endif; ?>
</div>
<div class="list-description">
```

```
      <h2><?php echo ucwords($lot->getLot_name()); ?></h2>
      <p><?php echo htmlspecialchars($lot->getLot_description()); ?></p>
      <p><strong>Lot:</strong> #<?php echo $lot->getLot_number(); ?>
        <strong>Price:</strong> $
        <?php echo number_format($lot->getLot_price(),2); ?>
        <a class="button edit"
  href="index.php?content=lotdelete&cat_id=<?php echo $cat_id_in; ?>&lot_
  id=<?php ↵
    echo $lot->getLot_id(); ?>">Delete
        </a>
        <a class="button edit"
  href="index.php?content=lotmaint&cat_id=<?php echo $cat_id_in; ?>&lot_
  id=<?php ↵
  echo $lot->getLot_id(); ?>">Edit
        </a></p></div>
```

6. In the `includes/classes/lot.php` file, add the `getLots()` public static method to retrieve the rows and fill the array. Use the parameter of `$cat_id` to select the correct lots. Rather than directly creating the `Category` object, use the `fetch_array(MYSQLI_ASSOC)` to retrieve the row. Use that array to create a new `Lot` object that is added as an element in the `$items` array.

```
static public function getLots($cat_id) {
  // clear the results
  $items = '';
  // Get the connection
  $connection = Database::getConnection();
  // Set up the query
  $query = 'SELECT * FROM `lots`
    WHERE cat_id="'. (int) $cat_id.'" ORDER BY lot_id';

  // Run the query
  $result_obj = '';
  $result_obj = $connection->query($query);
  // Loop through getting associative arrays,
  // passing them to a new version of this class,
  // and making a regular array of the objects
  try {
    while($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
      $items[]= new Lot($result);
    }
    // pass back the results
      return($items);
  }

  catch(Exception $e) {
    return false;
  }

}
```

7. Change the links in the `content/categories.php` file to use the lots page along with the category id to the URL parameters:

```
<h2>
  <a href="index.php?content=lots&cat_id=<?php echo (int) $item->getCat_id(); ↵
?>&amp;sidebar=catnav">
```

```
...

<a class="button display"
href="index.php?content=lots&cat_id=<?php echo (int) $item->getCat_id(); ↵
?>&amp;sidebar=catnav">Display Lots</a>
```

8. Change the link in the `content/catnav.php` file to use the lots page along with the category id in the URL parameters:

```
<li><a href="index.php?content=lots&cat_id=<?php echo
(int) $item->getCat_id(); ?>&sidebar=catnav"><?php echo
htmlspecialchars($item->getCat_name()); ?></a></li>
```

Now, create a lots maintenance page so you can add and change lots in the `lots` table.

1. Create `content/lotmaint.php`, which is the form for adding the lots. The category id is in the URL. Use `$_GET` to get the id and save it. This is used to select the right category to the new lot and for navigating back to the right lot page when saving or canceling. Because it's not displayed on the form, insert it in a hidden `<input>` so that it is available when you update the database. You also display a drop-down list of valid categories, which you create next in this Try It section. The rest of the file is similar to the maintenance files for contacts and categories.

```php
<?php
/**
 * lotmaint.php
 *
 * Maintenance for the Lots table
 *
 * @version  1.2 2011-02-03
 * @package  Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license  GNU General Public License
 * @since    Since Release 1.0
 */
// Get the Category the new lot will be in
$cat_id_in = (int) $_GET['cat_id'];
// Get the lot id. If it doesn't exist or is 0, then this is a new lot
$id = (int) $_GET['lot_id'];
// Is this an existing item or a new one?
if ($id) {
  // Get the existing information for an existing item
  $item = Lot::getLot($id);
  // set up the category dropdown
  $cat_dropdown = Category::getCat_DropDown($item->getCat_id());
} else {
  // Set up for a new item
  $item = new Lot;
  // set up the category dropdown
  $cat_dropdown = Category::getCat_DropDown($cat_id_in);
}
?>
<h1>Lot Maintenance</h1>
```

```
<form
    action="index.php?content=lots&cat_id=<?php echo $cat_id_in;
?>&sidebar=catnav"
    method="post" name="maint" id="maint">

    <fieldset class="maintform">
      <legend><?php echo ($id) ? 'Id: '. $id : 'Add a Lot' ?></legend>
      <ul>
        <li><label for="lot_name" class="required">Lot Name</label><br />
          <input type="text" name="lot_name" id="lot_name" class="required"
            value="<?php echo $item->getLot_name(); ?>" /></li>
        <li><label for="lot_description">Lot Description</label><br />
          <textarea rows="5" cols="60" name="lot_description"
           id="lot_description"><?php echo $item->getLot_description(); ?></
textarea>
        </li>
        <li><label for="lot_image" >Lot Image File</label><br />
          <input type="text" name="lot_image" id="lot_image"
          value="<?php echo $item->getLot_image(); ?>" /></li>
        <li><label for="lot_number">Lot Number</label><br />
          <input type="text" name="lot_number" id="lot_number"
            value="<?php echo $item->getLot_number(); ?>" /></li>
        <li><label for="lot_price" >Lot Price</label><br />
          <input type="text" name="lot_price" id="lot_price"
          value="<?php echo $item->getLot_price(); ?>" /></li>
        <li><?php echo $cat_dropdown; ?></li>
      </ul>

      <?php
      // create token
      $salt = 'SomeSalt';
      $token = sha1(mt_rand(1,1000000) . $salt);
      $_SESSION['token'] = $token;
      ?>
      <input type="hidden" name="cat_id_in" id="cat_id_in"
        value="<?php echo $cat_id_in; ?>" />
      <input type="hidden" name="lot_id" id="lot_id"
        value="<?php echo $item->getLot_id(); ?>" />
      <input type="hidden" name="task" id="task" value="lot.maint" />
      <input type='hidden' name='token' value='<?php echo $token; ?>'/>
      <input type="submit" name="save" value="Save" />
      <a class="cancel"
        href="index.php?content=lots&cat_id=<?php echo $cat_id_in; ↵
    ?>&sidebar=catnav">Cancel
        </a>
      </fieldset>
    </form>
```

2. In the `includes/classes/category.php`, add the `getCat_DropDown()` method to create the `<select>` drop-down. A category id is passed in, which is used to assign which category shows as selected. If no category id is passed then the first option is set as selected. The existing `getCategories()` method is used to get a list of categories. The HTML is

collected in an array called $html. When that array is passed back, it is *imploded* with \n. Imploding takes the array and turns it into a single string variable, putting a \n between each element. The \n is a newline character, which creates a new line for each section when you view the source of the web page. Without the newline character, the entire drop-down would appear as a single line in the source code. It does not affect what shows when you view the web page.

```php
public static function getCat_DropDown($selected = '') {
  // set up first option for selection if none selected
  $option_selected = '';
  if (!$selected) {
    $option_selected = ' selected="selected"';
  }

  // Get the categories
  $items = self::getCategories();

  $html  = array();

  $html[] = '<label for="cat_id">Choose Lot Category</label><br />';
  $html[] = '<select name="cat_id" id="cat_id">';

  foreach ($items as $i=>$item) {
    // If the selected parameter equals the current category id
    // then flag as selected
    if ((int) $selected == (int) $item->getCat_id()) {
      $option_selected = ' selected="selected"';
    }
    // set up the option line
    $html[]  =  '<option value="' . $item->getCat_id()
      . '"' . $option_selected . '>'
      . $item->getCat_name() . '</option>';
    // clear out the selected option flag
    $option_selected = '';
  }

  $html[] = '</select>';
  return implode("\n", $html);

}
```

3. In the includes/init.php file add a case block in the switch statement for lot.maint to process the lot maintenance form. This calls the maintLot() function.

```php
case 'lot.maint' :
// process the maint
$results = maintLot();
$message .= $results[1];
// If there is redirect information
// redirect to that page
if ($results[0] == 'lotmaint') {
  // pass on new messages
  if ($results[1]) {
```

```
                $_SESSION['message'] = $results[1];
            }
            $cat_id_in = (int) $_GET['cat_id'];
            header("Location: index.php?content=lotmaint&cat_id=$cat_id_in ↵
    &lot_id=$results[2]");
            exit;
        }
        break;
```

4.   Add the function `maintLot()` to `includes/functions.php`. Check that the token is
     good and then initialize an array with the information from the form. Create a `Lot` object
     with the data. If there is a lot id then this is an existing lot, so update the table with the
     `maintRecord()` method. Otherwise, create a new row with `AddRecord()`.

```php
function maintLot() {
  $results = '';
  if (isset($_POST['save']) AND $_POST['save'] == 'Save') {
    // check the token
    $badToken = true;
    if (!isset($_POST['token'])
       || !isset($_SESSION['token'])
       || empty($_POST['token'])
       || $_POST['token'] !== $_SESSION['token']) {
      $results = array('','Sorry, go back and try again.
        There was a security issue.');
      $badToken = true;
    } else {
      $badToken = false;
      unset($_SESSION['token']);
      // Put the sanitized variables in an associative array
      // Use the FILTER_FLAG_NO_ENCODE_QUOTES
      // to allow quotes in the description
      $item  = array ('lot_id' => (int) $_POST['lot_id'],
        'lot_name' => filter_input(INPUT_POST,'lot_name',
          FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES),
        'lot_description' => filter_input(INPUT_POST,'lot_description',
          FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES),
        'lot_image' => filter_input(INPUT_POST,'lot_image',
          FILTER_SANITIZE_STRING),
        'lot_number' => (int) $_POST['lot_number'],
        'lot_price' => filter_input(INPUT_POST,'lot_price',
          FILTER_SANITIZE_NUMBER_FLOAT,FILTER_FLAG_ALLOW_FRACTION),
        'cat_id' => (int) $_POST['cat_id']
      );

      // Set up a Lot object based on the posts
      $lot = new Lot($item);
      if ($lot->getLot_id()) {
        $results = $lot->editRecord();
      } else {
        $results = $lot->addRecord();
      }
    }
```

```
      }
    return $results;
  }
```

5. In `includes/classes/lot.php` add the `addRecord()` method to add rows to the `lots` table. When preparing numeric data for insertion in the database, force the variables to the proper case rather than using the `Database::prep()` method.

```
  /**
   * Add item
   * @return array
   */
  public function addRecord() {

    // Verify the fields
    if ($this->_verifyInput()) {

      // Get the Database connection
      $connection = Database::getConnection();

      // Prepare the data
      $query = "INSERT INTO
        lots(lot_name, lot_description, lot_image, lot_number, lot_price,
cat_id)
        VALUES ('" . Database::prep($this->lot_name) . "',
         '" . Database::prep($this->lot_description) ."',
         '" . Database::prep($this->lot_image) . "',
         '" . (int) $this->lot_number . "',
         '" . (float) $this->lot_price . "',
         '" . (int) $this->cat_id . "'
       )";

      // Run the MySQL statement
      if ($connection->query($query)) {
        $return = array('', 'Lot Record successfully added.');

        // add success message
        return $return;
      } else {
        // send fail message and return to categorymaint
      $return = array('lotmaint', 'No Lot Record Added. Unable to create
record.');
        return $return;
      }
    } else {
      // send fail message and return to categorymaint
      $return = array('lotmaint',
        'No Lot Record Added. Missing required information.');
      return $return;
    }

  }
```

6. Add the `_verifyInput()` method to verify that a category name was entered:

```
protected function _verifyInput() {
  $error = false;
  if (!trim($this->lot_name)) {
    $error = true;
  }
  if ($error) {
    return false;
  } else {
    return true;
  }
}
```

7. Add the `editRecord()` method to update records using prepared statements:

```
/**
 * Edit existing item
 * @return array
 */
public function editRecord() {

  // Verify the fields
  if ($this->_verifyInput()) {

    // Get the Database connection
    $connection = Database::getConnection();

    // Prepare the data
    // Set up the prepared statement
    $query = 'UPDATE `lots`
      SET lot_name=?, lot_description=?, lot_image=?, lot_number=?,
      lot_price=?, cat_id=?
      WHERE lot_id=?';
    $statement = $connection->prepare($query);
    // bind the parameters
    $statement->bind_param('sssidii',
      $this->lot_name, $this->lot_description, $this->lot_image,
      $this->lot_number, $this->lot_price, $this->cat_id, $this->lot_id);
    // Run the MySQL statement
    if ($statement) {
      $statement->execute();
      $statement->close();
      // add success message
      $return = array('', 'Lot Record successfully added.');
      // add success message
      return $return;
    } else {
      $return = array('lotmaint',
        'No Lot Record Added. Unable to create record.',
        '');
      return $return;
    }

  } else {
```

```
                // send fail message and return to categorymaint
                $return = array('lotmaint',
                  'No Lot Record Added. Missing required information.',
                  (int) $this->lot_id);
                return $return;
            }
        }
```

8. Test adding in lot rows. When you know you can successfully add lots, you can either add all the lots here or import the `insertlots.sql` file in phpMyAdmin as a shortcut.

9. Delete the `contents/gents.php`, `contents/sporting.php`, and `contents/women.php` files.

Now, add the Delete page and delete processing for the `lots` table.

1. Create `contents/lotdelete.php`. This page is similar to the `contents/lotmaint.php` file, but it displays the data from the table without allowing changes. Rather than submitting a Save button, you submit a Delete button.

```php
<?php
/**
 * lotdelete.php
 *
 * Delete for the Lots
 *
 * @version  1.2 2011-02-03
 * @package  Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license  GNU General Public License
 * @since    Since Release 1.0
 */

// Save the category so you return to the right lots page
$cat_id_in = (int) $_GET['cat_id'];
// Get the lot id. If it doesn't exist or is 0, then this is a new lot
$id = (int) $_GET['lot_id'];
  // Get the existing information for an existing item
  $item = Lot::getLot($id);
  // get the Category name for the lot
  $cat_name = Category::getCategory($item->getCat_id())->getCat_name();
?>
<h1>Lot Delete</h1>

<form action="index.php?content=lots&cat_id=<?php echo $cat_id_in; ↵
?>&sidebar=catnav"
  method="post" name="maint" id="maint">

    <fieldset class="maintform">
      <legend><?php echo 'ID: '. $id ?></legend>
      <ul>
        <li><strong>Lot Name: </strong>
          <?php echo htmlspecialchars($item->getLot_name()); ?></li>
        <li><strong>Lot Description: </strong><br />
          <?php echo htmlspecialchars($item->getLot_description()); ?></li>
        <li><strong>Lot Image File: </strong>
```

```
          <?php echo htmlspecialchars($item->getLot_image()); ?></li>
        <li><strong>Lot Number: </strong>
          <?php echo (int) $item->getLot_number(); ?></li>
        <li><strong>Lot Price: </strong>
          <?php echo number_format($item->getLot_price(),2); ?></li>
        <li><strong>Category: </strong>
          <?php echo htmlspecialchars($cat_name); ?></li>
      </ul>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="cat_id_in" id="cat_id_in"
      value="<?php echo $cat_id_in; ?>" />
    <input type="hidden" name="lot_id" id="lot_id"
      value="<?php echo $item->getLot_id(); ?>" />
    <input type="hidden" name="task" id="task" value="lot.delete" />
    <input type='hidden' name='token' value='<?php echo $token; ?>'/>
    <input type="submit" name="delete" value="Delete" />
    <a class="cancel"
    href="index.php?content=lots&cat_id=<?php echo $cat_id_in; ?> ↵
&sidebar=catnav">Cancel</a>
    </fieldset>
</form>
```

2. In the `includes/init.php` file, add the `lot.delete` case to process the delete form, which calls the `deleteLot()` function. The catalog id is also used in the URL to select the appropriate pages.

```
    case 'lot.delete' :
    // process the delete
    $results = deleteLot();
    $message .= $results[1];
    // If there is redirect information
    // redirect to that page
    if ($results[0] == 'lotdelete') {
      // pass on new messages
      if ($results[1]) {
        $_SESSION['message'] = $results[1];
      }
      $cat_id_in = (int) $_GET['cat_id'];
      header("Location:
index.php?content=lotdelete&cat_id=$cat_id_in&lot_id=$results[2]");
      exit;
    }
    break;
```

3. In the `includes/functions.php` file, add the `deleteCategory()` function, which calls the `deleteRecord()` method in the `Lot` class.

```
    function deleteLot() {
      $results = '';
      if (isset($_POST['delete']) AND $_POST['delete'] == 'Delete') {
```

```
        // check the token
        $badToken = true;
        if (!isset($_POST['token'])
        || !isset($_SESSION['token'])
        || empty($_POST['token'])
        || $_POST['token'] !== $_SESSION['token']) {
          $results = array('','Sorry, go back and try again.
            There was a security issue.');
          $badToken = true;
        } else {
          $badToken = false;
          unset($_SESSION['token']);

          // Delete the Lot from the table
          $results = Lot::deleteRecord((int) $_POST['lot_id']);
        }
      }
    }
    return $results;
  }
```

4. In the `includes/classes/lot.php` file, add the `deleteRecord()` method, which deletes the row from the `lots` table:

```
    public static function deleteRecord($id) {
        // Get the Database connection
        $connection = Database::getConnection();
        // Set up query
        $query = 'DELETE FROM `lots` WHERE lot_id="'. (int) $id.'"';
        // Run the query
        if ($result = $connection->query($query)) {
          $return = array('', 'Lot Record successfully deleted.', '');
          return $return;
        } else {
          $return = array('lotdelete', 'Unable to delete Lot.', (int) $id);
          return $return;
        }
    }
```

5. Check all your changes and verify that they look similar to Figure 26-18, Figure 26-19, and Figure 26-20. Test that you can delete items.

FIGURE 26-18

FIGURE 26-19

FIGURE 26-20

> Watch the video for Lesson 26 on the DVD or watch online at `www.wrox.com/go/24phpmysql`.

27

Preventing Database Security Issues

In this lesson, you learn the general security guidelines to use when using MySQL. Some of these guidelines are general ones that have been mentioned before in other lessons and some are particular to using a database and MySQL. They are gathered together here so that you can easily refer to them. As you are learning a new skill it can be exhilarating to just get things to work, and it's easy to ignore security issues. That can result in a painful lesson in the current climate.

Security steps must be taken to make MySQL itself more secure against attacks. These are related to your server setup and are not covered in this book. The XAMPP setup used throughout this book is for local development and is not secure for Internet access. However, the practices in this lesson are designed to make your code secure when used online.

UNDERSTANDING SECURITY ISSUES

There is no such thing as making your code completely secure against attacks. You can, however, reduce what harm can be done and make it less likely that you will be successfully hacked. Issues to be aware of are unauthorized access to your database files, unauthorized ability to change the database structure, unauthorized ability to see or change data, and SQL injection.

Unauthorized access to your database files is mostly dependent on your server setup. This is related to who has access to the MySQL files and what the permissions are on those files. MySQL is an application and as such its files are just as vulnerable to deletion and corruption as any other files on your system. If an unauthorized person can delete your whole MySQL setup, then you have a problem.

You don't want unauthorized MySQL users to have the ability to change the structure of your database. They could delete tables, for instance, if given a chance. Some hacking attacks need to exploit more than one vulnerability to be successful. When an attack succeeds in running

unplanned commands, the damage can be limited if the hackers don't have the authorization to use commands such as DROP TABLE or DROP DATABASE.

You also don't want unauthorized changing or displaying of data. If the application needs to SELECT, INSERT, UPDATE, and DELETE data, you want to make sure that that application user can do so only in a controlled manner. You need to check the data, whether it's coming from a form or a URL, to be sure that it does not contain unexpected code that also performs commands.

And that brings you to *SQL injections*. SQL injections are when additional SQL (in this case, MySQL) code is contained in seemingly innocent form fields or added to URLs. Imagine this command:

```
DELETE FROM authors WHERE id=4;
```

That command deletes a single row from the authors table. However, if you add a second condition to the WHERE clause that is always true, it deletes all the rows:

```
DELETE FROM authors WHERE id=4 OR 1=1;
```
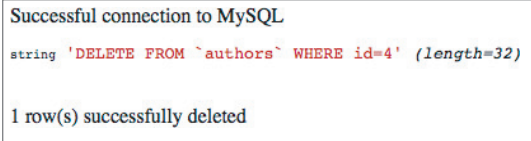
The following innocuous-looking code deletes a single row based on a value. In this example, that value comes from $id = 4, but imagine that the value is coming from a POST or GET variable from a form. Notice how the $id is handled when assembling the command in $query. The results are shown in Figure 27-1.

Successful connection to MySQL

string 'DELETE FROM `authors` WHERE id=4' *(length=32)*

1 row(s) successfully deleted

FIGURE 27-1

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  // assign an id value for the test
  $id = 4;

  // Set up the query
  $query = 'DELETE FROM 'authors' WHERE id='.  $id;
  var_dump($query); echo '<br />';
  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "No rows deleted<br />";
  } else {
    echo $connection->affected_rows . " row(s) successfully deleted<br />";
  }
}
```

Now, assume that instead of a 4, you are passed 4 OR 1=1 from the form as a GET or POST variable. All the rows are deleted as shown in Figure 27-2.

```php
<?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
} else {
  echo 'Successful connection to MySQL <br />';

  // assign an id value for the test
  $id = '4 OR 1=1';

  // Set up the query
  $query = 'DELETE FROM 'authors' WHERE id='.  $id;
  var_dump($query); echo '<br />';
  // Run the query and display appropriate message
  if (!$result = $connection->query($query)) {
    echo "No rows deleted<br />";
  } else {
    echo $connection->affected_rows . " row(s) successfully deleted<br />";
  }
}
```

Successful connection to MySQL

string 'DELETE FROM `authors` WHERE id=4 OR 1=1' *(length=39)*

4 row(s) successfully deleted

FIGURE 27-2

SQL injection is a serious issue, but if you write your code using best practices and filter and prepare it properly, you prevent most SQL injections.

USING BEST PRACTICES

MySQL users and passwords are under the Privileges tab in phpMyAdmin for your local MySQL installation. The global privileges are displayed if you are at the root level. There are also specific privileges at the database and table levels. If your MySQL is online, you may not have access to the Privilege tab in phpMyAdmin. Users and passwords for MySQL may be handled by a separate program.

Following are best practices for setting up MySQL users and passwords:

➤ The only users with access to the user table in MySQL should be the root accounts.

➤ Root users must have a password. Don't make it a trivial password.

➤ All users should have a password.

➤ Only grant as many privileges as necessary. If the application is only updating data in the database and not making structural changes, then don't allow the user profile used by the application any structural privileges. Structural privileges are the ones that allow you to create, alter, and destroy tables and databases. Don't allow the user profile administration rights, which would allow the user to create other users and grant privileges. Don't confuse the MySQL user profile the application is running under with operating system users or users you might create for an application.

➤ Don't grant privileges to all hosts (`%`).

➤ To see what privileges a user has use the SHOW GRANTS command, the Privileges tab in phpMyAdmin, or a special program on cpanel or another hosting administration software.

➤ To remove privileges from a user, use the SHOW GRANTS command, the Privileges tab in phpMyAdmin, or a special program on cpanel or another hosting administration software.

When you are writing your application, keep these best practices in mind:

➤ Use a separate username for your application and give it only the privileges it has to have.

➤ Use the standard precautions in creating your passwords:

> ➤ No words from dictionaries.
>
> ➤ Don't use all numbers or all letters.
>
> ➤ Include some capitalization and special characters.
>
> ➤ Instead of using passwords, use pass phrases without spaces but with capitalization and special characters to make very strong passwords that aren't difficult to remember. "I go to the store" could look like: iGo2thest0re.
>
> ➤ Using the first letters of a phrase is also a good way to get seemingly random letters that are easy to remember.

➤ Don't trust data entered by users.

> ➤ This is where the SQL injections can happen.
>
> ➤ Embedded code can be entered in forms or URLs.
>
> ➤ Protect string data values with filters. See Lessons 6 and 11 for more information on filtering.
>
> ➤ Make sure numeric values are only numeric (preventing the OR 1=1 injection). With integers, cast to `(int)` in PHP or use quote marks around the numeric variables even though they aren't required.
>
> ➤ Escape and/or encode all data appropriately for where it is going. This helps prevent hacking by preventing suspect control characters from acting like control characters. Because you are dealing with different types of software languages (that is, MySQL on one hand and HTML on the other), different characters act as control characters

and need to be dealt with differently. Do your escaping and encoding just before using the data so you have control over how the prepared data is used:

➤ When passing data into a database: Use `mysqli::real_escape_string` to escape the quotes. Remember to check to see if magic quotes is already on. (See Lesson 22 for information on magic quotes.) Prepared statements do it for you, as does the PDO connection.

➤ When passing data to the browser for display: Use `htmlspecialchars()` to encode the special HTML characters into appropriate HTML entities. For example, < is changed to `<`.

➤ If you create user logins and passwords for your application (these are separate from the MySQL users/passwords), don't store the passwords in plain text in the database. Use an encryption code such as `SHA1()` or `SHA2()` to convert the password and store that instead. MySQL has a `PASSWORD()` function, but that is for MySQL passwords and shouldn't be used for application passwords.

There are a couple tests you can try to check out your programs. Don't do the tests in place of following guidelines.

➤ Test what happens if you enter single and double quotes in web forms.

➤ Add non-numeric processing data in numeric form fields.

FILTERING DATA

You have several ways to filter and sanitize data, which you have learned through the course of this book. In Lesson 6 you learned about the `filter_var()` function. In Lesson 11 you learned how to filter `$_GET` and `$_POST` variables to make them safe. To keep your database secure, you need to use this knowledge to filter or sanitize any data going into your database.

Let's return to the example earlier in the lesson on SQL injection. The id in the example is an integer data type, so you can cast the variable to an integer and surround it in quotes to protect against the injection. When you change the `$query` assignment to the following code, you get the desired result in Figure 27-3. This code takes a bad input and sanitizes it to protect the database from SQL injection.

Successful connection to MySQL
string 'DELETE FROM `authors` WHERE id="4"' *(length=34)*
1 row(s) successfully deleted

FIGURE 27-3

```php
    <?php
define("MYSQLUSER", "php24sql");
define("MYSQLPASS", "hJQV8RTe5t");
define("HOSTNAME", "localhost");
define("MYSQLDB", "test");

// Make connection to database
$connection = @new mysqli(HOSTNAME, MYSQLUSER, MYSQLPASS, MYSQLDB);
if ($connection->connect_error) {
  die('Connect Error: ' . $connection->connect_error);
```

```
  } else {
    echo 'Successful connection to MySQL <br />';

    // assign an id value for the test
    $id = '4 OR 1=1';
    if (get_magic_quotes_gpc()) {
      // If magic quotes is active, remove the slashes
      $id = stripslashes($id);
    }
    // Escape special characters to avoid SQL injections
    $id = $connection->real_escape_string($id);

    // Set up the query
    $query = 'DELETE FROM 'authors' WHERE id="' . (int) $id . '"';
    var_dump($query); echo '<br />';
    // Run the query and display appropriate message
    if (!$result = $connection->query($query)) {
      echo "No rows deleted<br />";
    } else {
      echo $connection->affected_rows . " row(s) successfully deleted<br />";
    }
  }
}
```

Those are a lot of quotes to keep straight. You've been seeing this in the earlier lessons, but just in case you have trouble following it, here is how it is deciphered. The final value that is passed to MySQL is a valid MySQL statement that looks like this:

```
DELETE FROM 'authors' WHERE id="4"
```

In PHP, because you are assigning this MySQL statement to a variable (`$query`), you have to enclose strings in either double or single quotes. You have double quotes around the 4, so you can enclose the statement in single quotes. Note that at this point you could have swapped the single and double quotes. So when the whole line is a string, the assignment looks like this:

```
$query = 'DELETE FROM 'authors' WHERE id="4"';
```

You want to replace the 4 with the variable `$id`. To do that, you need to break the statement into three parts (the part before the 4, the 4, and the part after the 4), make sure each part is enclosed in single quotes and concatenate them. So here you see that the crazy looking `'"'` is a double quote enclosed by single quotes. The assignment looks like this:

```
$query = 'DELETE FROM 'authors' WHERE id="' . '4' . '"';
```

Now you replace the `'4'` with the `$id` variable. This is a variable, not a string, so there are no quotes around it. Don't get the PHP variable itself confused with the best practice of enclosing the resulting integer value in the MySQL statement with quotes. The assignment looks like this:

```
$query = 'DELETE FROM 'authors' WHERE id="' . $id . '"';
```

Cast the `$id` variable to an integer so add `(int)` in front of the variable to get the final assignment:

```
$query = 'DELETE FROM 'authors' WHERE id="'. (int) $id . '"';
```

This is a bit of overkill for an explanation, but sometimes getting the quotes and concatenations correct can be very frustrating and it is helpful to be able to build up to the final result.

An alternative method for numeric variables is to use the `is_numeric()` function and process the request only if the variable is actually numeric. If the value is coming from a form that you are validating, you could add this test in your form validation process along with the missing fields. You will still want to put quotes around the resulting number in the MySQL statement.

For string variables, either use prepared statements, PDO, or always run variables through the `mysqli::real_escape_string()` method or `mysqli_real_escape_string()` function you learned in Lesson 22.

TRY IT

It this Try It, you test the Case Study for vulnerabilities. You've been following the best practice guidelines as you have been creating the Case Study, but sometimes the tests can point out areas where you missed something.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson27 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 26. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Use the two tests listed under Best Practices:

➤   Test what happens if you enter single and double quotes in web forms.

➤   Add non-numeric processing data in numeric form fields.

Step-by-Step

Go through the following steps and correct the code if any errors are found.

1.   In the website, create a new contact with the name **Mitchell "Bud" O'Reilly**.

2.   Check that the double and single quotes appear as expected in the About Us page.

3. Go to the Edit page for this new contact. The first name is missing "Bud" as shown in Figure 27-4.

FIGURE 27-4

4. Go to `contents/contactmaint.php`. The data is contained in the value attributes for the `<input>` tags. The variable is enclosed by double quotes and is not escaped. Therefore when the browser encounters the first double quote around Bud, it thinks it is at the end of the value attribute. Enclose all the output variables with `htmlspecialchars()`. The result of refreshing the page is shown in Figure 27-5.

```
<li><label for="first_name" class="required">First Name</label><br />
  <input type="text" name="first_name" id="first_name" class="required"
  value="<?php echo htmlspecialchars($item->getFirst_name()); ?>" /></li>
<li><label for="last_name" class="required">Last Name</label><br />
  <input type="text" name="last_name" id="last_name" class="required"
  value="<?php echo htmlspecialchars($item->getLast_name()); ?>" /></li>
<li><label for="position">Position</label><br />
  <input type="text" name="position" id="position" class="required"
  value="<?php echo htmlspecialchars($item->getPosition()); ?>" /></li>
<li><label for="email" >Email</label><br />
  <input type="text" name="email" id="email"
  value="<?php echo htmlspecialchars($item->getEmail()); ?>" /></li>
<li><label for="phone" >Phone</label><br />
  <input type="text" name="phone" id="phone"
  value="<?php echo htmlspecialchars($item->getPhone()); ?>" /></li>
```

FIGURE 27-5

5. Make the same changes in `contents/categorymaint.php`:

```
<li><label for="cat_name" class="required">Category</label><br />
  <input type="text" name="cat_name" id="cat_name" class="required"
  value="<?php echo htmlspecialchars($item->getCat_name()); ?>" /></li>
<li><label for="cat_description">Description</label><br />
  <textarea rows="5" cols="60" name="cat_description"
  id="cat_description"><?php echo
htmlspecialchars($item->getCat_description()); ?></textarea></li>
  <li><label for="cat_image" >Image</label><br />
    <input type="text" name="cat_image" id="cat_image"
    value="<?php echo htmlspecialchars($item->getCat_image()); ?>" /></li>
```

6. In `contents/lotmaint.php` make the same changes to the text fields. For the lot number, cast the variable to an integer. For the price, cast the variable to a float.

```
<li><label for="lot_name" class="required">Lot Name</label><br />
  <input type="text" name="lot_name" id="lot_name" class="required"
  value="<?php echo htmlspecialchars($item->getLot_name()); ?>" /></li>
<li><label for="lot_description">Lot Description</label><br />
  <textarea rows="5" cols="60" name="lot_description"
  id="lot_description"><?php echo
  htmlspecialchars($item->getLot_description()); ?></textarea></li>
<li><label for="lot_image" >Lot Image File</label><br />
```

```
        <input type="text" name="lot_image" id="lot_image"
        value="<?php echo htmlspecialchars($item->getLot_image()); ?>" /></li>
    <li><label for="lot_number">Lot Number</label><br />
        <input type="text" name="lot_number" id="lot_number"
          value="<?php echo (int) $item->getLot_number(); ?>" /></li>
    <li><label for="lot_price" >Lot Price</label><br />
        <input type="text" name="lot_price" id="lot_price"
        value="<?php echo (float) $item->getLot_price(); ?>" /></li>
    <li><?php echo $cat_dropdown; ?></li>
```

7. Add a lot, typing a non-numeric value in the lot number and price. Make sure there are no errors. Look at the row in phpMyAdmin and verify that only numbers were placed in the table.

Watch the video for Lesson 27 on the DVD or watch online at www.wrox.com/go/24phpmysql.

SECTION VI
Putting It All Together

In this section you take what you have learned and use it to add advanced features to your site. You learn how to create user logins so that you can decide who can do what on your site. You create a table that contains content pages that you can update through your site and then have them display on your website. And, finally, you learn how to make the menuing system dynamic so that you can update it through your site rather than hardcoding in the PHP programs. This enables people who are creating text content to also create a menu item to access that content.

The examples are simple so that the concepts are easy to see.

28

Creating User Logins

In this lesson you learn how to restrict parts of your website to certain people. You learn what access control systems are and to use them to control who sees what on your site. You learn when and how to protect passwords and how to use cookies and sessions to remember who is logged in. Finally, you learn how to use that information to restrict and grant access to different parts of your site.

UNDERSTANDING ACCESS CONTROL

Access Control Lists, also known as ACLs, are the lists that are used to control who can see, add, change, or delete different elements of a system; in other words, controlling access. ACLs can be as simple as making sure someone is logged in. They can be as complex as listing what different people or groups have the ability to create, read, update, or delete specific files, tables, fields, or windows.

You can create a simple system in which you have only one type of user and all you need to know is whether she is signed on with just a table of users with usernames and passwords. When the user logs in, you check the username and password against a table to verify that the user exists and that the username and password are correct. A more complex system would have different levels of users. Some users can see but not touch. Others could see, touch, and add. Some could delete but not change.

A true ACL comes in when each of the items or groups of items (often called assets) can be addressed individually. So, for example, a user with a given access level can edit this item, but not this other type of item. You can end up with a matrix of permission levels, looking at what level is required on the asset before certain actions are allowed, looking at what level the user is, looking at whether the item or user is in a group, and what permissions might be inherited from the group.

In this lesson you keep to a simple system where the only things checked are whether the user is logged in and what his access level is. There are two hardcoded access levels: Registered and Admin.

PROTECTING PASSWORDS

Anytime you put passwords in a database, you should scramble or *hash* the password so if the database is compromised, the passwords can't be easily read. Hashing is a one-way process. You take a string of characters and pass it through a function that turns it into what looks like gibberish. This gibberish is not translated back. When someone logs in, you run the password through the same function and compare that result to the password stored in the database.

The functions used to scramble passwords are called hash functions. You used the `sha1()` hash function when creating tokens for validating the origin of forms. The cryptography field has developed many of these hash functions over the course of time. As problems develop with each one, newer, stronger ones are developed. Successive versions of PHP include some of the popular newer hashes as they are developed. MD5 was the hash of choice for many years but flaws have been discovered and exploited with it. SHA1 was the next favorite and is still used, though flaws have also been discovered in it. Both of these have individual functions in PHP. The following code shows a password encrypted with `md5()` and `sha1()`. See Figure 28-1 for the results.

md5: 9cf88e1df09629a23fcdabe2b02ab06a
sha1: d6702d6317a20db403175d12936cdcac71155d35

FIGURE 28-1

```php
<?php
$password = 'SomePassword!';
echo 'md5: ' . md5($password) . '<br />';
echo 'sha1: ' . sha1($password) . '<br />';
```

Newer versions of PHP have the `hash()` function, which enables you to specify the type of hash to be used. Using a generic function with the hash engine as a parameter makes it easier for PHP to remain current. To see a list of the available hash algorithms, use the `hash_algos()` function as shown in Figure 28-2.

```php
<?php
echo '<pre>';
print_r(hash_algos());
echo '</pre>';
```

It is more complex to figure out what the current recommended hashes are in PHP. Currently, two of the suggested hash algorithms are `sha512` and `whirlpool`. Note that these both create hashes that are 128 bytes long instead of the 32 and 40 of `md5()` and `sha1()`. See Figure 28-3.

```php
<?php
$password = 'SomePassword!';
echo 'md5: ' . hash('md5', $password) . '<br />';
echo 'sha1: ' . hash('sha1', $password) . '<br />';
echo 'sha512: ' . hash('sha512', $password) . '<br />';
echo 'whirlpool: ' . hash('whirlpool', $password) . '<br />';
```

In Lesson 17, you learned about salting tokens to make the hashed value more difficult to match for the hacker. The same is true for password hashes. You should always add a salt to the password. In Lesson 17 you used the same salt all the time. For passwords, it is best to use a different salt for each user. You need this salt when the user signs on. Some people store that salt in the database as well, and others use an existing field in the database that won't change, such as the id. The problem with storing the salt in the database is that if a hacker gets access to the database, he has the salt. The `hash_hmac()` function uses a password with a salt plus a site key that is not stored in the database. Therefore a hacker has to get access to your files and your database to be able to compromise

the passwords. The following example simulates a system using a password with a random salt that consists of the id of the user row appended to a static salt along with a site key that is a constant not stored in the database. See Figure 28-4.

```php
<?php
$password = 'SomePassword!';
$id = 42;
$salt = '!#$%jEkcy2884';
$sitekey ='d0d48339c3b82db413b3be8fbc5d7ea1c1fd3e2792605d3cbfda1HEM54!!';

echo 'sha512: ' . hash_hmac('sha512', $password . $id. $salt, $sitekey) . '<br />';
```

```
Array
(
    [0] => md2
    [1] => md4
    [2] => md5
    [3] => sha1
    [4] => sha224
    [5] => sha256
    [6] => sha384
    [7] => sha512
    [8] => ripemd128
    [9] => ripemd160
    [10] => ripemd256
    [11] => ripemd320
    [12] => whirlpool
    [13] => tiger128,3
    [14] => tiger160,3
    [15] => tiger192,3
    [16] => tiger128,4
    [17] => tiger160,4
    [18] => tiger192,4
    [19] => snefru
    [20] => snefru256
    [21] => gost
    [22] => adler32
    [23] => crc32
    [24] => crc32b
    [25] => salsa10
    [26] => salsa20
    [27] => haval128,3
    [28] => haval160,3
    [29] => haval192,3
    [30] => haval224,3
    [31] => haval256,3
    [32] => haval128,4
    [33] => haval160,4
    [34] => haval192,4
    [35] => haval224,4
    [36] => haval256,4
    [37] => haval128,5
    [38] => haval160,5
    [39] => haval192,5
    [40] => haval224,5
    [41] => haval256,5
)
```

FIGURE 28-2

```
md5: 9cf88e1df09629a23fcdabe2b02ab06a
sha1: d6702d6317a20db403175d12936cdcac71155d35
sha512: 64dd467d9f0e3b6c2fd8559009f178bad927eff588678c54453479d882400803394acae4ce73e1fc12b4cfcde1ef9ca931997b4a525926e495af57f0ffdbc895
whirlpool: 5b07fa8b6dde507cac330662d0d48339c3b82db413b3be8fbc5d7ea1c1fd3e2792605d3cbfda1a200c72d1fceb84a19c53a8848947571d0cfb4b2a038e7b69b2
```

FIGURE 28-3

```
sha512: 904e33b5a13e3c65ee197394130179ef0222c2b64f4350ed13660b8b4c0728248dc28908fc390b12e6e5cee6290c45de1cd0a5b38e96f2c8d56abca5c03f93f2
```

FIGURE 28-4

To protect the passwords you need to use a two-part login where you request both the username and the password. If you only have the password then either there's a chance that the password is unique, or you have to inform the user that the password is already taken, in which case she knows the password of another user.

As an added precaution, don't retrieve the password hash field when you query the database. Use the calculated hash in the WHERE clause of the SELECT statement but leave it out of the list of fields to retrieve. This means that you can't use an asterisk (*) and retrieve all fields.

USING COOKIES AND SESSIONS

HTTP, the protocol that runs the Web, is *stateless*. Stateless means that the system doesn't know the status of what has happened before. When the server processes requests from your website, it knows nothing more than what you've given it in that one command. It doesn't know what this user has done in the past or that he has successfully logged in.

Cookies and sessions are used to overcome this short-term memory problem. You've used cookies and sessions when working with forms. As you learned there, cookies are stored on a user's computer in plain text. Sessions, on the other hand, are stored on the server with only the id to the user session being stored in the cookie.

Security information, such as the fact that the user is logged in and what his access levels are, should be kept in a session rather than a cookie. To use a session you must always start the session before you do anything else. Start a session with `session_start();`. If you are using this throughout your program and you have an initialization program, put the command in there.

After you have started a session with `session_start()`, the session is accessed like an associative array using `$_SESSION` just as `$_POST`, `$_GET`, and `$_COOKIE` are accessed. For example, if the user data is an associative array, this code stores the user's id and access level in the session:

```
$_SESSION['user_id'] = $result['id'];
$_SESSION['access'] = $result['access'];
```

If the session is used only for logged-in users, when the user logs out, you clear the session variables, expire the cookie that points to the session, and destroy the session itself:

```
<?php
$_SESSION = array();
if (isset($_COOKIE[session_name()])) {
  setcookie(session_name(), '', time()-360000);
}
session_destroy();
```

If you are also using the session for users who are not logged in, you unset the session variables that pertain to the logged-in user:

```
<?php
unset($_SESSION['user_id']);
unset($_SESSION['access']);
```

PUTTING LOGINS TO WORK

Now that you have the access information in a session you can use that to allow access to specific pages or even specific pieces of information. You frequently check whether someone is logged in and what her access is, so the first thing to do is to put those checks in a function or add it as a method to a class. In these examples you use a function, though in the Case Study you use a static method in the `Contact` class.

Continuing with the examples you used earlier to check whether a user is logged in, create the following function:

```php
<?php
function isLoggedIn() {
  if (isset($_SESSION['user_id'])) {
    return true;
  } else {
    return false;
  }
}
```

Using this function assumes that you have called a `session_start()` and that you put this information in the session when the user logged in.

When you have a page that you want only a logged-in user to see, check at the beginning of the page to see if she is logged in. Display the page only if the function returns true. In this example, if the user is not logged in, she sees a message that says the page is restricted. Otherwise, if the user is logged in, she sees the private page:

```html
<html>
<title>Private Page</title>
<body>
<?php if (!isLoggedIn()) : ?>
  <p>Sorry, this page is restricted. </p>
<?php else : ?>
  <h1>Welcome</h1>
  <p>You are looking at a private page.</p>
<?php endif; ?>
</body>
</html>
```

You can use this same logic to restrict access to a part of the page as shown in the following example:

```html
<html>
<title>Private Information</title>
<body>
  <h1>Welcome</h1>
  <p>You are looking at the public part of this page.</p>

<?php if (isLoggedIn()) : ?>
  <p>And here you see private information</p>
<?php endif; ?>
```

```
    <p>And here is more public information.</p>

</body>
</html>
```

These examples only care if the user was logged in. The next level of ACL is when you have different authorizations for different people or groups. The logic is similar to checking if the user is logged in, except here you check to see that she has the appropriate access level. This function returns the actual access level if the user is logged in and false if the user is not logged in:

```
function accessLevel() {
  if (isset($_SESSION['access'])) {
    return $_SESSION['access'];
  } else {
    return false;
  }
}
```

You can then use this access level to determine what can be displayed. This page shows one link for administrators, a different link for registered users, and a third for everyone else:

```
<html>
<title>Private Information</title>
<body>
  <h1>Welcome</h1>
  <p>You are looking at the public part of this page.</p>

<?php if (accessLevel() == 'Admin') : ?>
  <a href="index.php?content=adminstuff">Admin functions</a>
<?php elseif (accessLevel() == 'Registered') : ?>
  <a href="index.php?content=registeredstuff">Registered functions</a>
<?php else : ?>
  <a href="index.php?content=publicstuff">Public functions</a>
<?php endif; ?>

</body>
</html>
```

You also can assign access levels to your assets, such as tables, fields, or rows, and then match those access levels with the access levels of the users.

TRY IT

Available for
download on
Wrox.com

In this Try It, you add a simple ACL system to the Case Study. You use this to restrict add, edit, and delete functions to administrators. The ACL system has two types of logged-in users: administrators and registered users.

To keep things simple, you expand the contacts to include the user information so you add user-name, password, and access level to the `contacts` table and corresponding maintenance pages and processing. In an actual system these might be two different tables.

Users need a way to sign on, so you create a login page and a logout page. Display the link to the login page to users who are not logged in and the logout link to those who are logged in.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson28 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 27. Alternatively, you can download the files from the book's website at www.wrox.com. The password for the contacts in the database is 12345678.

Hints

There are many different ways to hash a password. The one used in this section adds three salts, each of a different kind. The first salt is a salt that is different for each user and is stored in the database. The id field is used for that. The second salt is an arbitrary list of characters hardcoded wherever hashing is needed. The third salt is a salt that is specific to the site. This is shown as a hardcoded constant. In a real program you might have that in a separate configuration file. The hashing used is sha512, which takes up 128 bytes.

Step-by-Step

Add username, password, and access level to the contacts table.

1. Add the following fields to the contacts table. You can use phpMyAdmin to add the fields to the table either with the GUI interface, in the SQL table with the following code, or by importing the addacl.sql file from the downloaded code.

```
ALTER TABLE `contacts`
   ADD `user_name` varchar(15) NOT NULL,
   ADD `password` varchar(128) NOT NULL,
   ADD `access` varchar(10) NOT NULL;
```

2. Update the content/contactmaint.php file with the new fields. You use a drop-down for the access levels, so set that up at the very end of the first PHP block. You create this drop-down later in the Try It. You want to fill in the password only if you want to change it, so don't display the value for the password. The type should be password so that the text is not displayed when typed in. Add the attribute autocomplete="off". This prevents the

password from being automatically filled in by the browser. This attribute does not validate but is needed. Add a second input for passwords to get a confirming password.

```
// set up the access dropdown,
// setting up the selected option for existing records
  $access_dropdown = $item->getAccess_DropDown();
?>
<h1>Contact Maintenance</h1>

...

<li><label for="user_name" class="required">User Name</label><br />
  <input type="text" name="user_name" id="user_name"
  value="<?php echo htmlspecialchars($item->getUser_name()); ?>" />
</li>
<li><label for="password1" >New Password</label><br />
  <input type="password" name="password1" id="password1" autocomplete="off"  />
</li>
<li><label for="password2" >Confirm Password</label><br />
  <input type="password" name="password2" id="password2" autocomplete="off" />
</li>
<li><?php echo $access_dropdown; ?></li>
```

3. In the `includes/classes/contact.php` file, add the additional fields as properties:

```
/**
 * User name
 * @var string
 */
protected $user_name;

/**
 * Password
 * @var string
 */
protected $password;

/**
 * Access level
 * @var string
 */
protected $access;
```

4. In the same file, add the getters for the username and the access level. Do not create one for the password.

```
/**
 * Return User Name
 * @return string
 */
public function getUser_name() {
  return $this->user_name;
}

/**
 * Return Access
```

```
 * @return string
 */
public function getAccess() {
  return $this->access;
}
```

5. Still in the same file, create a public method that creates the HTML for a drop-down showing the options `Registered` and `Admin`:

```
public function getAccess_DropDown() {
  // set up first option for selection if none selected
  $option_selected = '';
  if (!$this->access) {
    $option_selected = ' selected="selected"';
  }

  // Get the categories
  $items = array('Registered', 'Admin');

  $html  = array();

  $html[] = '<label for="access">Choose Access</label><br />';
  $html[] = '<select name="access" id="access">';

  foreach ($items as $i=>$item) {
    // If the selected parameter equals the current category id
    // then flag as selected
    if ($this->access == $item) {
      $option_selected = ' selected="selected"';
    }
    // set up the option line
    $html[]  = '<option value="' . $item . '"' . $option_selected . '>' . ↵
$item . '</option>';
    // clear out the selected option flag
    $option_selected = '';
  }

  $html[] = '</select>';
  return implode("\n", $html);

}
```

6. In the `includes/init.php` file, add a constant that defines a site key. This site key is used to hash the password. The value in the site key is purely arbitrary. If you later change it, none of your passwords would work.

```
define('SITE_KEY',
  'd0d48339c3b82db413b3be8fbc5d7ea1c1fd3e2792605d3cbfda1HEM54!!');
```

7. In the `includes/function.php` file, find the `maintContact()` function. Add the username and access level to the `$item` array. You do not add the passwords here.

```
'phone'     => filter_input(INPUT_POST,'phone', FILTER_SANITIZE_STRING),
'user_name' => filter_input(INPUT_POST,'user_name', FILTER_SANITIZE_STRING),
'access'    => filter_input(INPUT_POST,'access', FILTER_SANITIZE_STRING)
 );
```

8. Back in the `includes/classes/contact.php` file, find the `_verifyInput()` function. Verify that a username exists and is at least six characters long. Run the `getContactIdByUser()` method. This method returns the id for the row with the given username. If an id is returned, it means the username is already taken. You create the method in the next step. If a password is entered, verify the length and confirm that it and the confirming password are the same. If they do not match, set an error.

```php
if (!trim($this->user_name)) {
    $error = true;
} elseif (strlen(trim($this->user_name)) < 6) {
    $error = true;
} elseif (self::getContactIdByUser(trim($this->user_name))) {
    $error = true;
}

$password1 = trim($_POST['password1']);
if ($password1) {
    if ($password1 != trim($_POST['password2'])) {
        $error = true;
    } elseif (strlen($password1) < 6) {
        $error = true;
    }
}
```

9. Create a static method that takes a username and looks up the id in the `contacts` table:

```php
public static function getContactIdByUser($user_name) {
    // Get the database connection
    $connection = Database::getConnection();
    // set up the query
    $id = '';
    $query = 'SELECT id FROM `contacts`
      WHERE user_name="'. Database::prep($user_name) .'"
      LIMIT 1';
    // Run the MySQL command
    $result_obj = '';
    // Run the MySQL command
    $result_obj = $connection->query($query);
      while($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
        $id = $result['id'];
      }
    // if user name not found, return false
    if (!$id) { // if user name not found, return with error message
      return false;
    } else {
      return $id;
    }
}
```

10. Update the `addRecord()` method by adding the processing for the new fields. Create the new row as before. This enables you to get the newly generated id with the `mysql_insert_id()` function so you can use that to help create the hash for the password. The `!hi#HUde9` is an

arbitrary salt. By hardcoding the salt here, it requires that hackers have access to your files to locate the information. You are also using the constant SITE_KEY, which you defined earlier.

Update the new row with the username, password hash, and the access level. The following code trims the password so that whitespaces are automatically used. Some people like to put deliberate whitespace at the beginning or end of the password. If you want to accommodate them, leave off the trim() around the $_POST['password1'].

```php
    public function addRecord() {

      // Verify the fields
      if ($this->_verifyInput()) {
        // prepare for the encrypted password
          $password = trim($_POST['password1']);

        // Get the Database connection
        $connection = Database::getConnection();

        // Prepare the data
        $query = "INSERT INTO contacts(first_name, last_name, position, email,
  phone)
        VALUES ('" . Database::prep($this->first_name) . "',
         '" . Database::prep($this->last_name) . "',
         '" . Database::prep($this->position) . "',
         '" . Database::prep($this->email) . "',
         '" . Database::prep($this->phone) . "')";
        // Run the MySQL statement
        if ($connection->query($query)) { // this inserts the row
          // update with the user name and password now that you know the id
          $query = "UPDATE contacts
          SET user_name = '" . Database::prep($this->user_name) . "',
          password = '" . hash_hmac('sha512',
            $password . '!hi#HUde9' . mysql_insert_id(),
            SITE_KEY) ."',
          access = '" . Database::prep($this->access) . "'";
          if ($connection->query($query)) { // this updates the row
            $return = array('', 'Contact Record successfully added.', '');
            // add success message
            return $return;
          } else {
            // send fail message
              $return = array('', 'User name/password not added to contact.',
  '');

              return $return;
          }

        } else {
        // send fail message and return to contactmaint
        $return = array('contactmaint',
          'No Contact Record Added. Unable to create record.',
          '0');
        return $return;
        }
      } else {
        // send fail message and return to contactmaint
```

```
        $return = array('contactmaint', 'No Contact Record Added.
          Missing required information or problem with user name or password.',
    '0');
        return $return;
      }

    }
```

11. Next, update the `editRecord()` method. If a password was entered, create the hash password and update all the fields. If no password was entered, just add the username and access fields to the existing fields to be updated. The following code shows the affected part of the method:

```
// Update with a password changed
if (trim($_POST['password1'])) {
  // prepare the encrypted password
  $hash_password = hash_hmac('sha512',
    trim($_POST['password1']) . '!hi#HUde9' . $this->id,
    SITE_KEY);
  // Set up the prepared statement
  $query = 'UPDATE `contacts` SET first_name=?, last_name=?,
    position=?, email=?, phone=?,
    user_name=?, password=?, access=?
    WHERE id=?';
  $statement = $connection->prepare($query);
  // bind the parameters
  $statement->bind_param('sssssssi',$this->first_name,
    $this->last_name,
    $this->position, $this->email, $this->phone,
    $this->user_name, $hash_password, $this->access,
    $this->id);
} else {
  // update without a password changed
  // Set up the prepared statement
  $query = 'UPDATE `contacts` SET first_name=?, last_name=?,
    position=?, email=?, phone=?,
    user_name=?, access=?
    WHERE id=?';
  $statement = $connection->prepare($query);
  // bind the parameters
  $statement->bind_param('sssssssi',$this->first_name, $this->last_name,
    $this->position, $this->email, $this->phone,
    $this->user_name, $this->access,
    $this->id);
}
```

12. Change the `getContact()` method to get specific fields instead of all fields with the *. This way you don't bring in the hash password.

```
$query = 'SELECT `id`,`first_name`,`last_name`,`position`, `email`, `phone`,
  `user_name`, `access`
  FROM `contacts` WHERE id="'. (int) $id.'"';
```

13. Change the `getcontacts()` method to get specific fields instead of all fields with the *.

```
$query = 'SELECT `id`,`first_name`,`last_name`,`position`, `email`, `phone`,
  `user_name`, `access`
  FROM `contacts` ORDER BY first_name, last_name';
```

14. Now add usernames and passwords to the existing contacts as demonstrated in Figure 28-5. You add the Logout menu item later in this Try It.

FIGURE 28-5

15. Flag `user_name` as unique in the database so that there are no duplicates in the table.

```
ALTER TABLE `contacts` ADD UNIQUE (`user_name`);
```

The next step is to create a login page so that users can log in.

1. Create a file called `content/login.php` that looks like Figure 28-6.

```php
<?php
?>
<h1>Login</h1>

<form action="index.php" method="post" name="maint" id="maint">
```

```
<fieldset class="maintform">
  <legend>Login</legend>
  <ul>
    <li><label for="user_name" class="required">User Name</label><br />
      <input type="text" name="user_name" id="user_name" class="required"
/></li>
      <li><label for="password" class="required">Password</label><br />
        <input type="password" name="password" id="password" class="required"
/>
    </li>
    </ul>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="task" id="task" value="login" />
    <input type='hidden' name='token' value='<?php echo $token; ?>'/>
    <input type="submit" name="login" value="Login" />
    <a class="cancel" href="index.php">Cancel, return to Home Page</a>
  </fieldset>
</form>
```

Login

FIGURE 28-6

2. Add the login task to the includes/init.php file. This calls the userLogin() function.

```
case 'login' :
// process the login
$results = userLogin();
$message .= $results[1];
// If there is redirect information
// redirect to that page
// pass on new messages
  if ($results[0] == 'login') {
  // pass on new messages
  if ($results[1]) {
```

```
            $_SESSION['message'] = $results[1];
        }
        header("Location: index.php?content=login");
        exit;
    }
    break;
```

3. Add the `userLogin()` function to the `includes/functions.php` file. This function checks the token as usual and loads the POST variables into an array. That array is then passed to the static method `logIn()` in the `Contact` class and the results passed back to the calling function in `init.php`.

```
function userLogin() {
    $results = '';
    if (isset($_POST['login']) AND $_POST['login'] == 'Login') {
        // check the token
        $badToken = true;
        if (!isset($_POST['token'])
        || !isset($_SESSION['token'])
        || empty($_POST['token'])
        || $_POST['token'] !== $_SESSION['token']) {
            $results = array('',
                'Sorry, go back and try again. There was a security issue.');
            $badToken = true;
        } else {
            $badToken = false;
            unset($_SESSION['token']);

            $item  = array (
                'user_name' => filter_input(INPUT_POST,'user_name', FILTER_SANITIZE_
STRING),
                'password' => filter_input(INPUT_POST,'password')
                );

            // login
            $results = Contact::logIn($item);
        }
    }
    return $results;
}
```

4. Add the `logIn()` method to the `includes/classes/contact.php` file. First you check to see that the user entered both a username and a password. Use the username to get the id from the database. If there is no matching user, send the error message that, the username or password is incorrect. Never tell the user what the exact problem is because that gives hackers too much information.

Next, set up the hash for the password. Select the user based on matching both the username and hash password. In this instance you are pulling in only those fields that you need, rather

than using an * to get all fields. It is better not to pull in the hash password. Post all of those fields to SESSION variables and return a success message.

```php
public static function logIn($item) {
  if (!$item['user_name'] || !$item['password']) {
    return array('login','Sorry, invalid User Name and/or Password.');
  }
  // Get the database connection
  $connection = Database::getConnection();

  // get the id for the user
  $id = self::getContactIdByUser($item['user_name']);

  if (!$id) { // if user name not found, return with error message
    return array('login','Sorry, invalid User Name and/or Password.','');
  }

  $hash_password = hash_hmac('sha512', $item['password'] . '!hi#HUde9' . ↵
(int) $id, SITE_KEY);

  // Set up the query
  $query = 'SELECT id, first_name, last_name, user_name, access
    FROM `contacts`
    WHERE user_name="'. $item['user_name'] .'"
    AND password = "' . $hash_password . '"
    LIMIT 1';
  // Run the MySQL command
  $result_obj = '';
  // Run the MySQL command
  $result_obj = $connection->query($query);
  try {
    while ($result = $result_obj->fetch_array(MYSQLI_ASSOC)) {
      // pass back the results
      $_SESSION['user_id'] = $result['id'];
      $_SESSION['first_name'] = $result['first_name'];
      $_SESSION['last_name'] = $result['last_name'];
      $_SESSION['user_name'] = $result['user_name'];
      $_SESSION['access'] = $result['access'];
      return array('',"Welcome, {$_SESSION['first_name']}", '');
    }
    return array('login','Sorry, invalid User Name and/or Password.','');
  }
  catch(Exception $e)
    {
      return false;
    }
}
```

5. Add the Login menu item to the main menu in `index.php`:

```html
<li><a href="index.php?content=home">Home</a></li>
<li><a href="index.php?content=login">Login</a></li>
```

> ⊗ *If you need to reset your session during your testing, an easy way is to completely close down your browser.*

The next step is to create a logout page so that users can log out.

1. Create a file called `content/logout.php`. Assuming that George is logged in, your results look like Figure 28-7.

```php
<?php
/**
 * logout.php
 *
 * Logout
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
?>
<h1>Logout</h1>

<form action="index.php" method="post" name="maint" id="maint">

  <fieldset class="maintform">
    <legend>Logout</legend>
    <p>Are you sure you want to logout, <?php echo $_SESSION['first_name'];
?>?</p>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="task" id="task" value="logout" />
    <input type='hidden' name='token' value='<?php echo $token; ?>'/>
    <input type="submit" name="logout" value="Logout" />
    <a class="cancel" href="index.php">Cancel, return to Home Page</a>
  </fieldset>
</form>
```


FIGURE 28-7

2. Add the `logout` task to the `includes/init.php` file. This calls the `userLogout()` function.

```php
case 'logout' :
// process the login
$results = userLogout();
$message .= $results[1];
// If there is redirect information
// redirect to that page
// pass on new messages
```

```
if ($results[0] == 'logout') {
    // pass on new messages
    if ($results[1]) {
        $_SESSION['message'] = $results[1];
    }
    header("Location: index.php?content=logout]");
    exit;
}
break;
```

3. Add the `userLogout()` function to the `includes/functions.php` file. This function checks the token as usual. The static method `logOut()` in the `Contact` class is called and the results are passed back to the calling function in `init.php`.

```
function userLogout() {
    $results = '';
    if (isset($_POST['logout']) AND $_POST['logout'] == 'Logout') {
        // check the token
        $badToken = true;
        if (!isset($_POST['token'])
        || !isset($_SESSION['token'])
        || empty($_POST['token'])
        || $_POST['token'] !== $_SESSION['token']) {
            $results = array('',
                'Sorry, go back and try again. There was a security issue.', '');
            $badToken = true;
        } else {
            // logout
            $badToken = false;
            unset($_SESSION['token']);

            Contact::logout();
            $results = array('',"You have successfully logged out",'');;
        }
    }
    return $results;
}
```

4. Add the `logOut()` method to the `includes/classes/contact.php` file. To log the user out, you unset the `SESSION` variables.

```
public static function logout() {
    unset($_SESSION['user_id']);
    unset($_SESSION['first_name']);
    unset($_SESSION['last_name']);
    unset($_SESSION['user_name']);
    unset($_SESSION['access']);
}
```

5. Add the Logout menu item to the main menu in `index.php`:

```
<li><a href="index.php?content=home">Home</a></li>
<li><a href="index.php?content=logout">Logout</a></li>
<li><a href="index.php?content=login">Login</a></li>
```

Finally, you add the access level checks to your program.

1.   Create a static method called `isLoggedIn()` in the `Contact` class that returns true if the user is logged in. To see if a user is logged in, check to see if the `SESSION` variable `user_id` is set. In the `includes/classes/contact.php` file add this code:

```php
public static function isLoggedIn() {
  if (isset($_SESSION['user_id'])) {
    return true;
  } else {
    return false;
  }
}
```

2.   Create a static method called `accessLevel()` in the `Contact` class that returns the access level of the user. This is stored in the `SESSION` variable `access`.

```php
public static function accessLevel() {
  if (isset($_SESSION['access'])) {
    return $_SESSION['access'];
  } else {
    return false;
  }
}
```

3.   Some pages have both public and private information. Only Admin users are allowed to add, edit, or delete items. Add access level checks to those buttons in the `about.php`, `categories.php`, and `lots.php` files in the `content` folder. Here is the example for the `about.php` file:

```php
// Get the contact information
$items = Contact::getContacts();
$accessLevel = Contact::accessLevel();
?>
<h1>About Us
  <?php if ($accessLevel == 'Admin') : ?>
    <a class="button" href="index.php?content=contactmaint&id=0">Add</a>
  <?php endif; ?>
</h1>

...

<h2><?php echo htmlspecialchars($item->name()); ?>
<?php if ($accessLevel == 'Admin') : ?>
  <a class="button"
    href="index.php?content=contactdelete&id=<?php echo $item->getId();
?>">Delete
  </a>
  <a class="button"
    href="index.php?content=contactmaint&id=<?php echo $item->getId();
?>">Edit</a>
<?php endif; ?>
</h2>
```

4.   Some pages should only be seen by Admin users. They don't have access through the program, but could still get to those pages by directly entering a URL. Check for the access level at the beginning of the content and skip the content if the user is not an Admin user.

The following files in the content folder should have this check: categorydelete.php, categorymaint.php, contactdelete.php, contactmaint.php, lotdelete.php, and lotmaint.php. Here is the example for the categorydelete.php file:

```
$accessLevel = Contact::accessLevel();
if ($accessLevel != 'Admin') :
  echo 'Sorry, no access allowed to this page';
else :

$id = (int) $_GET['cat_id'];
// Get the existing information for an existing item
$item = Category::getCategory($id);
?>
<h1>Category Maintenance</h1>

...

</form>
<?php endif;
```

5. Change the menu so that Login shows for those users who are not logged in and Logout shows for those users who are logged in:

```
require_once 'includes/init.php';
$logged_in = Contact::isLoggedIn();
?>

...

<li><a href="index.php?content=home">Home</a></li>
<?php if ($logged_in) : ?>
  <li><a href="index.php?content=logout">Logout</a></li>
<?php else : ?>
  <li><a href="index.php?content=login">Login</a></li>
<?php endif; ?>
```

6. Check that you can log in and out. See that the Add, Edit, and Delete buttons show only if you are logged in as an Admin user. See that the menu shows Login if you are not logged in and Logout if you are logged in.

Watch the video for Lesson 28 on the DVD or watch online at www.wrox.com/go/24phpmysql*.*

29

Turn the Case Study into a Content Management System

In this lesson you add the ability to create additional pages of information for the Case Study site. These could be news articles, a page listing the privacy policy, or a blog article. Instead of creating a new `.php` content file for each one, you create a single `.php` file that displays page text that you have in the database.

DESIGNING AND CREATING THE TABLE

The first step is to create the table that will contain the page data. What is the information you need in this file? There is the obvious information such as

➤   Title

➤   Text

and other information you might want to keep:

➤   User who created the article

➤   Date article was created

➤   User who last modified the article

➤   Last date the article was modified

You also need a primary key so that you can select which article you want to display. Remember that the primary key cannot change and should be unique. Therefore you do not want to use the title as the key. Use an auto-increment integer key as you have in the other file.

This example is for a freeform page, but depending on your application, you could add additional fields that you could then place in particular places on the page. For instance, if you want all the articles to start with a picture and a short introduction and be followed by a list of references, you could add fields for each of those in your table.

You might also want to put your articles into different categories the way that the lots are in the Case Study. For that, you need an article categories table and a key in the articles table that links to it. You don't use article categories for this example.

CREATING THE CLASS

After you have created the articles table, you need to create a class that describes it and contains the different actions you need to perform on the articles. This class is similar to the others that you have created, but this is a good place to list what you need in the class. In different projects that you do, you have different methods in your classes, but these are basic methods that most classes based on tables need to have.

Properties

The properties in the class match the fields in the table. They should be protected so that only this class or classes that extend this class can see them directly.

Methods

Your class needs methods to create the object and to handle the CRUD (creating, reading, updating, and deleting) of the articles.

➤ `__construct()`: This method is called when you create an object from this class. You have been using a method that takes an associative array where the indexes are equal to the properties and automatically updates the properties at instantiation. You can perform different actions in this method. In this example, you have the user who created the article and the user who last modified the article. You store the id, not the name of the user, from the contact table in the article table. If you call `Contact::getContact($this->created_by);` in this method you can assign it to a new property you create, which now contains all the information about the person who wrote the article.

➤ `getProperty_name():` This is a getter method, where `Property_name` is the name of the property with the first letter capitalized. You need getter methods for each of the properties that are protected or private so that parts of the program outside the class are able to see what the values are.

➤ `_verifyInput()`: This method is used to verify the input used to add or update the table.

➤ `addRecord()`: This method is used to add a row to the table.

➤ `editRecord()`: This method is used to make changes to a row in the table.

➤   `deleteRecord()`: This method is used to remove a row from the table.

➤   `getArticle()`: This method is used to get the information from one row in the table, based on the id passed to the method.

➤   `getArticles()`: This method is used to get a list of all the rows in the table.

When you have this much code that is similar, it makes sense to reuse the same code instead of copying and pasting it into new classes. It cuts down on errors, saves you time, makes the code easier to read, and makes it easier to make changes to all the classes. For instance, instead of having four `__construct()` methods in four different classes, you could create a class that has a `__construct()` method in it and then you can extend that class.

The following class called `Table` has a property `$id` and two methods. One is the `__construct()` method and the other is a getter method for the `$id` property.

```php
<?php
class Table
{
  protected $id;

  public function __construct($input = false) {
    if (is_array($input)) {
      foreach ($input as $key => $val) {
      // Note the $key instead of key.
      // This will give the value in $key instead of 'key' itself
      $this->$key = $val;
      }
    }
  }

  public function getId() {
    return $this->id;
  }

}
```

Any class that extends the `Table` class automatically includes a property `$id` and the methods `__construct()` and `getId()`. If the child class has no property or method with the same name, it automatically uses the parent class or property version. If the child class has a property or method with the same name, then the child's property or class is used. You can still access the parent, however, by using parent as the class name. This code uses the child construct that in turn calls the parent's `__construct()` and then adds more code:

```php
class Article extends Table
{

  public function __construct($input = false) {
    parent::__construct($input);
    $this->author = Contact::getContact($this->created_by);
  }
}
```

You can expand this idea by moving some of the other methods that are similar across the classes into the `Table` class and then extending the `Contact`, `Category`, and `Lot` classes from the `Table` class.

CREATING THE MAINTENANCE PAGES

After you have the table and class created, you need to create the pages to maintain the table. To maintain the articles you need a page that displays a list of the articles, a page to add or change articles, and a page to delete the articles. You can use the contacts or categories pages as a model for this.

The user who creates the article, the date the article was created, the user who last modified the article, and the last date modified are already known to the system. Those fields can be automatically updated and so do not need to be on the form.

For security and validation, the title field should be run through the `htmlspecialchars()` function. This turns control characters into the HTML entities so that they display as characters rather than performing any actions. For instance, an `&` would be turned into `&`.

The text field needs special consideration. If you treat it the standard way, by encoding the HTML entities, you are not able to use HTML in the text field. This is fine if you only want paragraphs, but if you want to be able to enter HTML commands you need to come up with a different solution. The `strip_tags()` function strips away HTML tags except for the ones that you specify. This lets you control what HTML you allow.

The `strip_tags()` function does not make the field entirely secure. You could still get things put into the attributes such as with JavaScript `mouseovers`. In this case, the form input is coming from within the company so it is acceptable. If this were coming from public input, you would need to program something more secure.

CREATING THE DISPLAY PAGE

To display the articles all you need to do is to create a file that looks up the article based on the id and display the fields. However, there is one additional change you need to make to the display of the text field. Figure 29-1 displays an article on the maintenance screen. Figure 29-2 displays what it looks like on the display screen.

As you can see, you have lost all the paragraphs. The text has run into a single paragraph. If you were to use `<p>` tags to surround the paragraphs you would get your paragraphs back, but there's an easier way to get them back. Control characters that signify a new line create those paragraphs. The `<textarea>` tag reads those and starts a new paragraph. Just displaying the text field does not activate those control characters. There is a function that turns those newline characters

into `<br />` tags. It is the "new line to `<br>`" function, which is written as `nl2br()`. Adding that function to the display of the text brings back the paragraphs. The results of the following code are shown in Figure 29-3.

```php
<?php echo strip_tags(nl2br($item->getText()),
    "<p><br><h2><h3><h4><strong><em><ul><ol><li><a>"); ?>
```
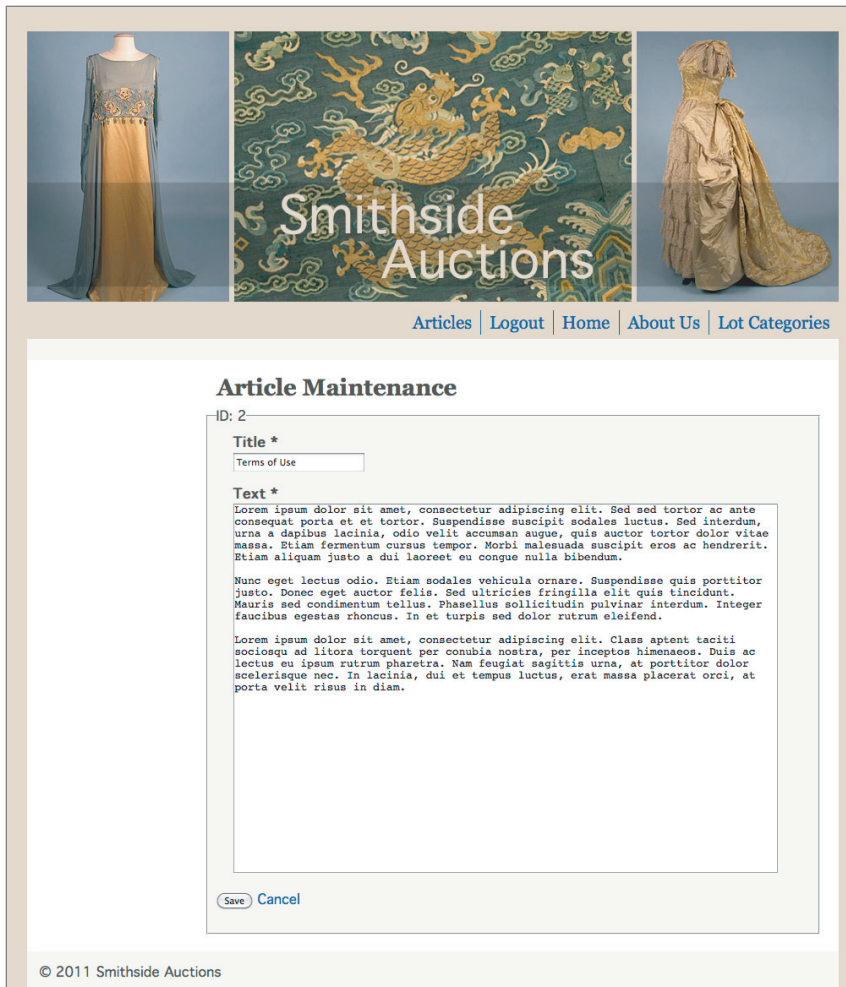

FIGURE 29-1
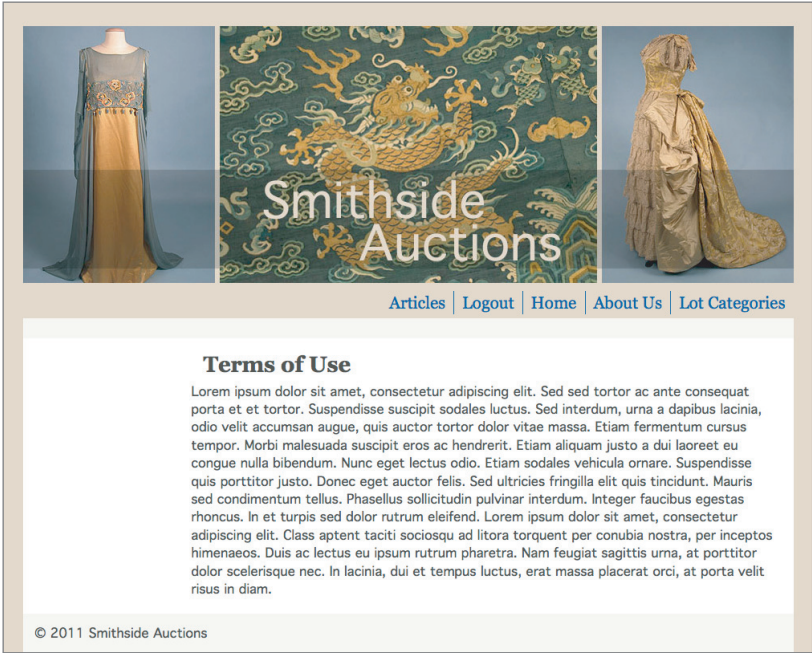
FIGURE 29-2
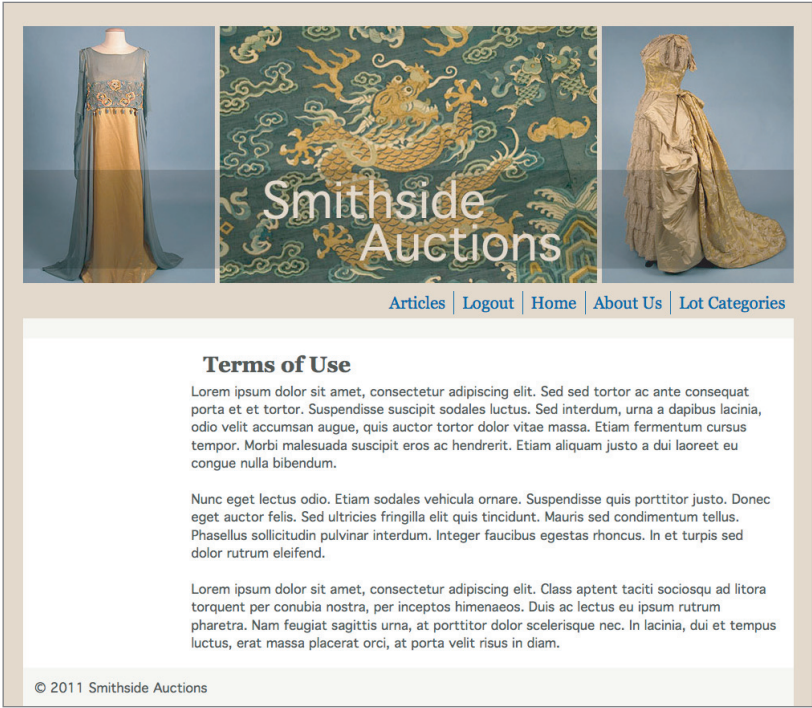
FIGURE 29-3

TRY IT

In this Try It, you add a new table to the Case Study database called `articles`. You set up mainte-
nance pages for this new table, including all the necessary processing for adding, editing, and
deleting articles. You create a new content file called `articledisplay.php` that you use to display
the articles as a web page.

> *You can download the code and resources for this Try It from the book's web
> page at* www.wrox.com. *You can find them in the Lesson29 folder in the down-
> load. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a pack-
age of software that installs the web server, PHP, and MySQL for you. You can find instructions for
downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for down-
loading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's
Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 28.
Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

Keep the end the `<textarea>` tag on the same line as the beginning of the text to go inside the
`textarea`. If you don't, you get whitespace at the beginning of the input box.

`CURRENT_TIMESTAMP` is a MySQL constant that contains the current time. You can use that to time-
stamp when an article is created and when it is modified.

The user must be logged in before he has access to the maintenance screen, so the program has
access to his user id in the `SESSION`. Access the `SESSION` variable to get the id to update the Article
row.

Step-by-Step

Create the `articles` table.

1. Create the `articles` table with the following fields. You can use phpMyAdmin to add the
fields to the table either with the GUI interface, in the SQL table with the following code, or
by importing the `addarticlestable.sql` file from the downloaded code.

```
DROP TABLE IF EXISTS `articles`;
CREATE TABLE `articles` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
```

```
    `title` varchar(100) NOT NULL,
    `text` text NOT NULL,
    `created_by` int(11) NOT NULL,
    `date_created` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
    `modified_by` int(11) NOT NULL,
    `date_modified` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
    PRIMARY KEY (`id`)
) ENGINE=MyISAM;
```

2. Create the `Table` class in `includes/classes/table.php`:

```php
<?php
/**
 * table.php
 *
 * Table class file
 *
 * @version  1.2 2011-02-03
 * @package  Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license  GNU General Public License
 * @since    Since Release 1.0
 */
/**
 * Table class
 *
 * @package  Smithside Auctions
 */
class Table
{
  /**
   * id
   * @var int
   */
  protected $id;

  /**
   * Initialize the class with data from database
   * @param array
   */
  public function __construct($input = false) {
    if (is_array($input)) {
      foreach ($input as $key => $val) {
      // Note the $key instead of key.
      // This will give the value in $key instead of 'key' itself
      $this->$key = $val;
      }
    }
  }

  /**
   * Return id
   * @return int
```

```
 */
public function getId() {
return $this->id;
}

}
```

3. Create the `Article` class, which extends the `Table` class in `includes/classes/article.php`. You don't need the property `$id` or the method `getId()` because those are in the `Table` class. Define a `__construct()` method that calls the parent method and also creates an object of the contact who created the article. Add the `addRecord()`, `editRecord()`, and `deleteRecord()` methods, as well as the `getArticle()` and `getArticles()` methods.

```php
<?php
/**
 * article.php
 *
 * Article class file
 *
 * @version  1.2 2011-02-03
 * @package  Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license  GNU General Public License
 * @since    Since Release 1.0
 */
/**
 * Article class
 *
 * @package  Smithside Auctions
 */
class Article extends Table
{

  /**
   * title
   * @var string
   */
  protected $title;

  /**
   * Text
   * @var String
   */
  protected $text;

  /**
   * Contact who created the article
   * @var int
   */
  protected $created_by;

  /**
```

```
 * Date & time created
 * @var datetime
 */
protected $date_created;

/**
 * Contact who modified the article
 * @var int
 */
protected $modified_by;

/**
 * Date & time modified
 * @var datetime
 */
protected $date_modified;

/**
 * Author Information
 * @var Contact
 */
protected $author;

/**
 * Initialize the Article with data from database
 * @param array
 */
public function __construct($input = false) {
  parent::__construct($input);
  $this->author = Contact::getContact($this->created_by);
}

/**
 * Return Title
 * @return string
 */
public function getTitle() {
return $this->title;
}

/**
 * Return Text
 * @return string
 */
public function getText() {
return $this->text;
}

/**
 * Return Created by
 * @return int
 */
```

```php
public function getCreated_by() {
return $this->created_by;
}

/**
 * Return Created date/time
 * @return datetime
 */
public function getDate_created() {
return $this->date_created;
}

/**
 * Return Modified by
 * @return int
 */
public function getModifiedBy() {
return $this->modified_by;
}

/**
 * Return Modified date/time
 * @return datetime
 */
public function getModified_by() {
return $this->modified_by;
}

/**
 * Verify the Input
 * @return boolean
 */
protected function _verifyInput() {
  $error = false;
  if (!trim($this->title)) {
    $error = true;
  }
  if (!trim($this->text)) {
    $error = true;
  }

  if ($error) {
    return false;
  } else {
    return true;
  }
}

/**
 * Add a Row to the table
 * @return array (redirect content,message,id)
 */
```

```php
public function addRecord() {

  // Verify the fields
  if ($this->_verifyInput()) {
    // prepare the encrypted password

    // Get the Database connection
    $connection = Database::getConnection();

    // Prepare the data
    $query = "INSERT INTO articles(title, text, created_by,
      date_created, modified_by, date_modified)
    VALUES ('" . Database::prep($this->title) . "',
     '" . Database::prep($this->text) . "',
     '" . (int) $_SESSION['user_id'] . "',
     CURRENT_TIMESTAMP,
     '" . (int) $_SESSION['user_id'] . "',
     CURRENT_TIMESTAMP)";
    // Run the MySQL statement
    if ($connection->query($query)) {
    $return = array('', 'Article successfully added.', '');

    // add success message
    return $return;
    } else {
    // send fail message and return to contactmaint
    $return = array('articlemaint', 'No Article Added. Unable to
      create record.', '0');
    return $return;
    }
  } else {
    // send fail message and return to contactmaint
    $return = array('articlemaint', 'No Article Added. Missing
      required information.', '0');
    return $return;
  }

}

/**
 * Update a Row in the table
 * @return array (redirect content,message,id)
 */
public function editRecord() {
  // Verify the fields
  if ($this->_verifyInput()) {

    // Get the Database connection
    $connection = Database::getConnection();

    // Prepare the data
    $query = "UPDATE `articles`
```

```
            SET `title` = '" . Database::prep($this->title) . "',
                `text` = '" . Database::prep($this->text) . "',
                `modified_by` = '" . (int) $_SESSION['user_id'] . "',
                `date_modified` = CURRENT_TIMESTAMP
                WHERE id='" . (int) $this->id . "'";
            // Run the MySQL statement
            if ($connection->query($query)) {
            $return = array('', 'Article successfully updated.', '');

            // add success message
            return $return;
            } else {
            // send fail message
            $return = array('articlemaint',
              'Article not updated. Unable to update record.',
              (int) $this->id);
            return $return;
            }
        } else {
            // send fail message and return to maint
            $return = array('articlemaint',
              'Article not updated. Missing required information.',
              '0');
            return $return;
        }

    }

    /**
     * Delete a Row from the table
     * @param  int
     * @return array (redirect content,message,id)
     */
    public static function deleteRecord($id) {
        // Get the Database connection
        $connection = Database::getConnection();
        // Set up query
        $query = 'DELETE FROM `articles` WHERE id="'. (int) $id.'"';
        // Run the query
        if ($result = $connection->query($query)) {
          $return = array('', 'Article successfully deleted.', '');
          return $return;
        } else {
          $return = array('articledelete', 'Unable to delete Article.', (int)
$id);
          return $return;
        }
    }

    /**
     * Get an Article
     * @param  int
```

```php
 * @return Article
 */
public static function getArticle($id) {
  // Get the database connection
  $connection = Database::getConnection();
  // Set up the query
  $query = 'SELECT * FROM `articles` WHERE id="'. (int) $id.'"';
  // Run the MySQL command
  $result_obj = '';
    try {
      $result_obj = $connection->query($query);
      if (!$result_obj) {
        throw new Exception($connection->error);
      } else {
        $item = $result_obj->fetch_object('Article');
        if (!$item) {
          throw new Exception($connection->error);
        } else {
          // pass back the results
          return($item);
        }
      }
    }
    catch(Exception $e) {
      echo $e->getMessage();
    }
}

/**
 * Get an array of Articles
 * @return array (Article)
 */
public static function getArticles() {
  // clear the results
  $items = '';
  // Get the connection
  $connection = Database::getConnection();
  // Set up query
  $query = 'SELECT * FROM `articles` ORDER BY title';
  // Run the query
  $result_obj = '';
  $result_obj = $connection->query($query);
  // Loop through the results,
  // passing them to a new version of this class,
  // and making a regular array of the objects
  try {
    while($result = $result_obj->fetch_object('Article')) {
      $items[]= $result;
    }
    // pass back the results
    return($items);
```

```
      }

      catch(Exception $e) {
        return false;
      }
    }

  }
```

4. Add `Articles` to the menu in `index.php`. Make it so that only Admin users can get to the link. This links to a list of articles so you can add, edit, and delete them.

```
<li><a href="index.php?content=home">Home</a></li>
<?php if ($logged_in) : ?>
  <li><a href="index.php?content=logout">Logout</a></li>
  <?php if ($accessLevel == 'Admin') : ?>
    <li><a href="index.php?content=articles">Articles</a></li>
  <?php endif; ?>
<?php else : ?>
  <li><a href="index.php?content=login">Login</a></li>
<?php endif; ?>
```

5. Create `content/articles.php` to display a list of the articles as shown in Figure 29-4:

```
<?php
/**
 * articles.php
 *
 * Content for Articles
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
$accessLevel = Contact::accessLevel();
if ($accessLevel != 'Admin') :
  echo 'Sorry, no access allowed to this page';
else :

// Get the article information
$items = Article::getArticles();
if (empty($items)) {
  $items = array();
}
?>
<h1>Articles
    <a class="button" href="index.php?content=articlemaint&id=0">Add</a>
</h1>

<ul class="ulfancy">
```

```php
        <?php foreach ($items as $i=>$item) : ?>
          <li class="row<?php echo $i % 2; ?>">
            <h2><?php echo htmlspecialchars($item->getTitle()); ?>
              <a class="button"
                href="index.php?content=articledelete&id=<?php echo $item->getId();
?>">
                  Delete
              </a>
              <a class="button"
                href="index.php?content=articlemaint&id=<?php echo $item->getId();
?>">
                  Edit
              </a>
            </h2>
          </li>
        <?php endforeach; ?>
      </ul>

      <?php endif; ?>
```
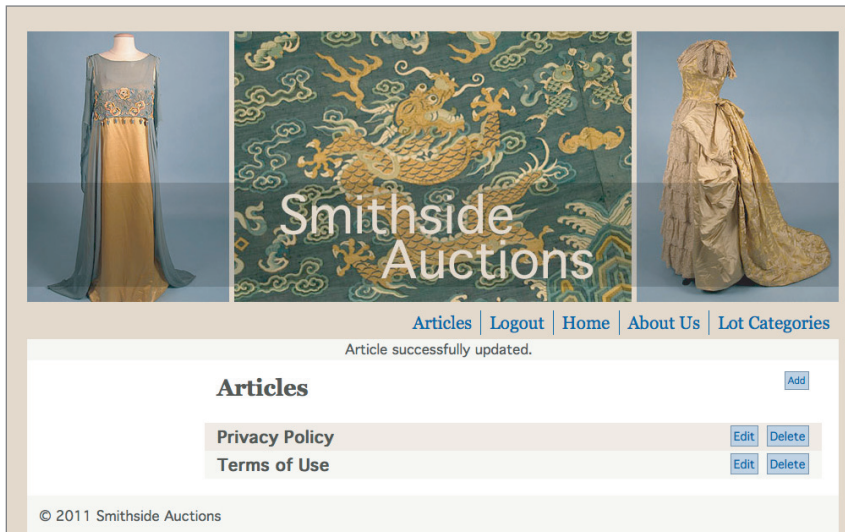

FIGURE 29-4

6. Create `content/articlemaint.php` to add and edit the articles as shown in Figure 29-5:

```php
<?php
/**
 * articlemaint.php
 *
 * Maintain the Articles table
 *
 * @version    1.2 2011-02-03
```

```php
 * @package   Smithside Auctions
 * @copyright Copyright (c) 2011 Smithside Auctions
 * @license   GNU General Public License
 * @since     Since Release 1.0
 */
$accessLevel = Contact::accessLevel();
if ($accessLevel != 'Admin') :
  echo 'Sorry, no access allowed to this page';
else :

$id = (int) $_GET['id'];
// Is this an existing item or a new one?
if ($id) {
  // Get the existing information for an existing item
  $item = Article::getArticle($id);
} else {
  // Set up for a new item
  $item = new Article;
}
?>
<h1>Article Maintenance</h1>

<form action="index.php?content=articles" method="post" name="maint"
id="maint">

  <fieldset class="maintform">
    <legend><?php echo ($id) ? 'id: '. $id : 'Add an Article' ?></legend>
    <ul>
      <li><label for="title" class="required">Title</label><br />
        <input type="text" name="title" id="title" class="required"
        value="<?php echo htmlspecialchars($item->getTitle()); ?>" /></li>
        <li><label for="text" class="required">Text</label><br />
          <textarea rows="30" cols="80" name="text"
            id="text" class="required"><?php echo strip_tags($item-
>getText(),
            "<p><br><h2><h3><h4><strong><em><ul><ol><li><a>"); ?>
          </textarea></li>
    </ul>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="id" id="id" value="<?php echo $item->getId();
?>" />
    <input type="hidden" name="task" id="task" value="article.maint" />
    <input type='hidden' name='token' value='<?php echo $token; ?>'/>
    <input type="submit" name="save" value="Save" />
    <a class="cancel" href="index.php?content=articles">Cancel</a>
  </fieldset>
```

```
</form>
<?php endif;
```


FIGURE 29-5

7. Create `content/articledelete.php` to delete articles as shown in Figure 29-6:

```php
<?php
/**
 * articledelete.php
 *
```

```
 * Delete the Articles
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */
$accessLevel = Contact::accessLevel();
if ($accessLevel != 'Admin') :
  echo 'Sorry, no access allowed to this page';
else :

$id = (int) $_GET['id'];
// Get the existing information for an existing item
$item = Article::getArticle($id);

?>
<h1>Article Delete</h1>

<form action="index.php?content=articles" method="post" name="maint"
id="maint">

  <fieldset class="maintform">
    <legend><?php echo 'Id: '. $id ?></legend>
    <ul>
      <li><strong>Title:</strong>
        <?php echo htmlspecialchars($item->getTitle()); ?></li>
      <li><strong>Text:</strong>
        <?php echo strip_tags(nl2br($item->getText()),
          "<p><br><h2><h3><h4><strong><em><ul><ol><li><a>"); ?>
    </ul>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="id" id="id" value="<?php echo $item->getId();
?>" />
    <input type="hidden" name="task" id="task" value="article.delete" />
    <input type='hidden' name='token' value='<?php echo $token; ?>'/>
    <input type="submit" name="delete" value="Delete" />
    <a class="cancel" href="index.php?content=articles">Cancel</a>
  </fieldset>
</form>
<?php endif;
```

FIGURE 29-6

8. Add the `article.maint` and `article.delete` tasks to `includes/init.php`:

```
case 'article.maint' :
// process the maint
$results = maintArticle();
$message .= $results[1];
// If there is redirect information
// redirect to that page
if ($results[0] == 'articlemaint') {
  // pass on new messages
  if ($results[1]) {
    $_SESSION['message'] = $results[1];
  }
  header("Location: index.php?content=articlemaint&id=$results[2]");
  exit;
}
break;
```

```
             case 'article.delete' :
             // process the delete
             $results = deleteArticle();
             $message .= $results[1];
             // If there is redirect information
             // redirect to that page
             if ($results[0] == 'articledelete') {
               // pass on new messages
               if ($results[1]) {
                 $_SESSION['message'] = $results[1];
               }
               header("Location: index.php?content=articledelete&id=$results[2]");
               exit;
             }
             break;
```

9. Add the `maintArticle()` and `deleteArticle()` functions to `includes/functions.php`:

```
        function maintArticle() {
          $results = '';
          if (isset($_POST['save']) AND $_POST['save'] == 'Save') {
            // check the token
            $badToken = true;
            if (!isset($_POST['token'])
            || !isset($_SESSION['token'])
            || empty($_POST['token'])
            || $_POST['token'] !== $_SESSION['token']) {
              $results = array('',
                'Sorry, go back and try again. There was a security issue.');
              $badToken = true;
            } else {
              $badToken = false;
              unset($_SESSION['token']);
              // Put the sanitized variables in an associative array
              // Use the FILTER_FLAG_NO_ENCODE_QUOTES to allow names like O'Connor
              $item  = array (
                'id' => (int) $_POST['id'],
                'title' => filter_input(INPUT_POST,'title',
                  FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES),
                'text'  => strip_tags($_POST['text'],
                  "<p><br><h2><h3><h4><strong><em><ul><ol><li><a>")
              );

              // Set up a Article object based on the posts
              $article = new Article($item);
              if ($article->getId()) {
                $results = $article->editRecord();
              } else {
                 $results = $article->addRecord();
              }
            }
          }
          return $results;
        }
```

```php
function deleteArticle() {
  $results = '';
  if (isset($_POST['delete']) AND $_POST['delete'] == 'Delete') {
    // check the token
    $badToken = true;
    if (!isset($_POST['token'])
    || !isset($_SESSION['token'])
    || empty($_POST['token'])
    || $_POST['token'] !== $_SESSION['token']) {
      $results = array('',
        'Sorry, go back and try again. There was a security issue.');
      $badToken = true;
    } else {
      $badToken = false;
      unset($_SESSION['token']);

      // Delete the Article from the table
      $results = Article::deleteRecord((int) $_POST['id']);
    }
  }
  return $results;
}
```

10. Create `content/articledisplay.php` as shown in Figure 29-7. The `id` of the article is in the URL. Use that to get the article. If an article is found, display it. Put the `title` between `<h1>` tags and the `text` in a `<div>`. Strip all the tags except the acceptable ones and change newline codes into `<br />` tags.

```php
<?php
/**
 * articledisplay.php
 *
 * Display the Article
 *
 * @version    1.2 2011-02-03
 * @package    Smithside Auctions
 * @copyright  Copyright (c) 2011 Smithside Auctions
 * @license    GNU General Public License
 * @since      Since Release 1.0
 */

$id = (int) $_GET['id'];
// Get the existing information for an existing item
$item = Article::getArticle($id);
if ($item) :
?>
<h1><?php echo htmlspecialchars($item->getTitle()); ?></h1>

<div>
 <?php echo strip_tags(nl2br($item->getText()),
    "<p><br><h2><h3><h4><strong><em><ul><ol><li><a>"); ?>
</div>
<?php endif;?>
```
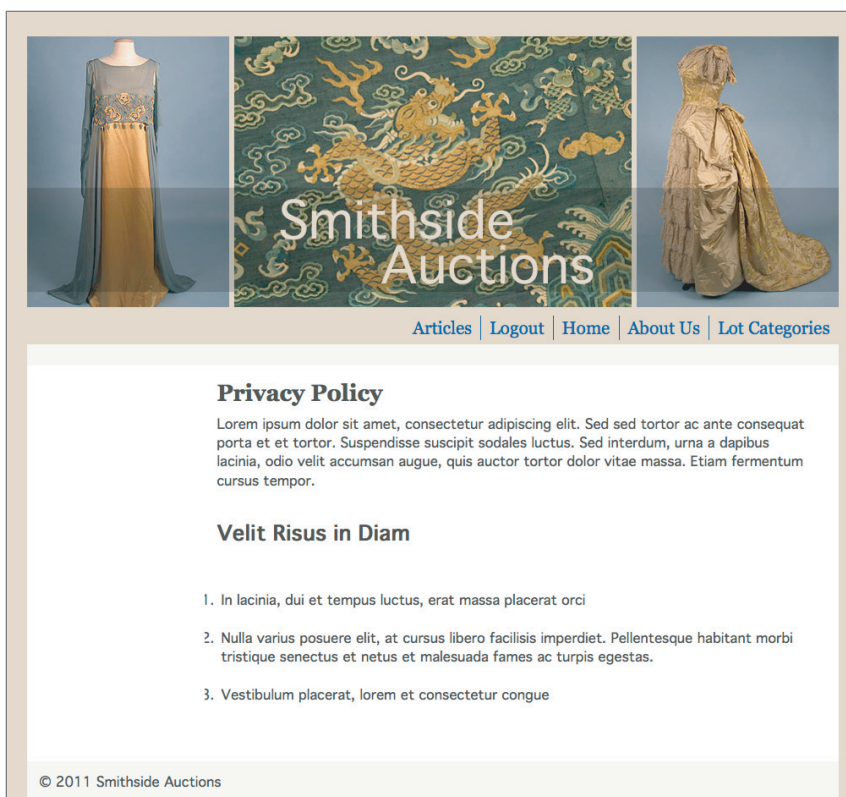
FIGURE 29-7

11. In the next lesson you add these article pages to a menu. For now, test the programs by typing in the URL manually. Assuming your article id is 1, add this to the end of your domain in the browser address bar:

```
/index.php?content=articledisplay&id=1
```

Watch the video for Lesson 29 on the DVD or watch online at `www.wrox.com/ go/24phpmysql.`

30

Creating a Dynamic Menu

In this lesson, you learn how to create a menu that draws its information from the database rather than from hardcoded HTML. In the previous lesson, you created new articles through the website, but then you had to go into the `index.php` file to add the code to display the pages. By using a database, you are able to create menu links easily through the website as well. You create a table for the menu links, set up the maintenance for that table, and write the code to assemble and display the menu.

SETTING UP THE MENU TABLE

The first step is to create the table that will contain the menu data. Here is some information you could have in a menu table, depending on the sophistication of the menuing system. The first two items are the minimum needed for a menu:

➤ **Menu Title:** This is the text that is displayed in the menu.

➤ **Menu Link:** This is the link for the menu item.

➤ **Order By:** If you want to be able to decide in what order the links should appear, you need a field to sort the fields in a specific order.

➤ **Access Level:** If you want to be able to show only authorized links, you need a field that gives the authorization. Depending on the complexity of the ACL, this could either be the access level itself or a key to another table that contains the access level information.

➤ **Menu:** If you are creating more than one menu, you need to specify which menu the link belongs to. This should be a link to another table, which contains the information about the whole menu.

➤ **Parent:** If you are creating a menu that has multiple levels, you need to know the parent of this link.

➤ **Link Type:** If you assign different types to the rows, you can make more of the link creation automatic. If there is no link type, then all the links need to be created the same way. Take, for instance, these following three link types:

➤ **External:** The link is whatever the user types in.

➤ **Internal:** Prefix `index.php?content=` to whatever the user types in.

➤ **Article:** Display a drop-down with all the articles and have the user select the right article. The value on the drop-down contains the `id`. When the menu is created, prefix `index.php?content=articles&id=` to the `id`.

In this instance, the menu you create for the Case Study is very simple. All the links are in a single menu, the access levels are specified in the `menus` table, and the links are only one level deep. All the links are automatically prefixed with `index.php?` when the menu is created.

The next step is to add the maintenance for the `menus` table to the website. You do this the same way that you did for the other tables. You need a content page that displays the menu links and from there you get to the pages to add, edit, and delete the links.

ADDING THE MENU TO THE WEBSITE

After you have the links listed in a table, you need to re-create the HTML for a menu from the rows in the table. You do this with the same type of processing that you used for creating drop-downs for selecting categories. There is an additional complication here, though, which is that you aren't necessarily going to display all the menu items. Depending on the authority the user has and the access level required before viewing the menu items, different menu items are displayed for different types of users.

You can make this selection either within the MySQL SELECT statements or you can select all the rows and then filter what you need to when you create the HTML. If you have a lot of menu items, it might make sense to read only those items that you know you need. However, most menus don't have many rows so the performance hit is negligible. If you already have a method set up to get a list of the items, you can reuse that to get the information:

```
$items = self::getMenus();
```

If you have to filter the rows, now you need to get the level information for the user. The methods here look up the information from the current user and pass it back to the variable. `$logged_in` is Boolean to say if the user is currently logged in. The second one gets the actual authorized access level.

```
$logged_in = Contact::isLoggedIn();
$accessLevel = Contact::accessLevel();
```

Use an array to collect the HTML statements as you create them. Putting each line of HTML in an array element means that it is easy to implode the array with a newline control character (`\n`). If you do this instead of concatenating each new line, when you look at the source code, each line appears on its own line instead of being in one long row.

You initialize the array to start and add in any header information:

```
$html = array();
$html[] = '<h3 class="element-invisible">Menu</h3>';
$html[] = '<ul class="mainnav">';
```

You loop through each row and select those that pass the authorization tests. The authorizations tests can be as simple as an if statement using series of and and or statements. As each row passes authentication, add it as an <a> tag in an unordered list.

```
foreach ($items as $item) {
  if (!($item->level)
    OR ($item->level == "Public")
    OR ($item->level == "Admin" AND $accessLevel == "Admin")
    OR ($item->level == "Registered" AND ($accessLevel == "Registered" OR ↵
$accessLevel == "Admin"))
    OR ($item->level == "LoggedIn" AND $logged_in)
    OR ($item->level == "LoggedOut" AND !$logged_in)) {
      $html[] = '<li><a href="index.php?' . $item->link. '">' . $item->title. ↵
'</a></li>';
  }
}
```

When the list is complete, add any closing tags and implode it with a newline and return the variable. See the created HTML in Figure 30-1.

```
$html[] = '</ul>';
$return = implode("\n", $html);
return $return;
```

```
    <h3 class="element-invisible">Menu</h3>
<ul class="mainnav">
<li><a href="index.php?content=categories">Lot Categories</a></li>
<li><a href="index.php?content=about">About Us</a></li>
<li><a href="index.php?content=home">Home</a></li>
<li><a href="index.php?content=login">Login</a></li>
<li><a href="index.php?content=articledisplay&id=1">Terms of Use</a></li>
<li><a href="index.php?content=articledisplay&id=2">Privacy Policy</a></li>
</ul>
```

FIGURE 30-1

TRY IT

In this Try It, you create a dynamic menu that reads its links from a table in the database. The class for this table resembles those you have built for the other tables for the Case Study. You can extend the Table class so that you don't need to redefine what is already in that class. You need the standard processing methods to add rows, change rows, delete rows, get a list of the menu links, and retrieve a single menu link. You then create a method that creates the HTML for the menu and replaces the static links in the index.php file with a call to the method.

The code examples that you can download are appropriately commented, but the large docblock comments are not displayed in this Try It.

> *You can download the code and resources for this Try It from the book's web page at* www.wrox.com. *You can find them in the Lesson30 folder in the download. You will find code for both before and after completing the exercises.*

Lesson Requirements

Your computer needs to be able to run as a web server with PHP and MySQL. XAMPP is a package of software that installs the web server, PHP, and MySQL for you. You can find instructions for downloading and installing XAMPP in Lesson 1.

You need a text editor that can produce plain-text files. You can find instructions for downloading and installing Eclipse PDT in Lesson 1. Other text editors that you can use are Adobe's Dreamweaver in code mode, Notepad, TextWrangler, or NetBeans.

If you are following along with the Case Study, you need your files from the end of Lesson 29. Alternatively, you can download the files from the book's website at www.wrox.com.

Hints

The menu CRUD can be created the same way you did for the other tables in the database.

When creating a drop-down of valid level choices for menu links, you can follow the example of the drop-down of valid categories.

Step-by-Step

Create the `menus` table and add the standard processing for creating, updating, and deleting the data.

1. Create the `menus` table with `id` (`auto_increment`, primary key), `title`, `link`, `level`, and `orderby`. You use the `level` field to determine who can see that link and the `orderby` to place the menu links in a specific order. You can use phpMyAdmin to add the fields to the table either with the GUI interface, in the SQL table with the following code, or by importing the `addmenustable.sql` file from the downloaded code.

```
DROP TABLE IF EXISTS `menus`;
CREATE TABLE `menus` (
  `id` int(11) NOT NULL UNSIGNED AUTO_INCREMENT,
  `title` varchar(100) NOT NULL,
  `link` varchar(255) NOT NULL,
  `level` varchar(10) NOT NULL DEFAULT 'Public',
  `orderby` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM;
```

2. Create the class `Menu` in `includes/classes/menu.php`. The `Menu` class extends the `Table` class. Define the properties based on the table fields and add the getter methods. You

don't need to add the `$id` property because it is in the `Table` class. Make the properties protected.

```php
<?php
class Menu extends Table
{
  protected $title;
  protected $link;
  protected $level;
  protected $orderby;
}
```

3. In steps 3 through 10, you add the methods in the `Menu` class. Put these methods immediately following the list of properties. Normally you would start with a `__construct()` function to create the object, but you don't need one here because you already have a `__construct()` function coming in from the `Table` class. Because your properties are protected, you need to create getter functions so that you can access the properties in other parts of the program. Having your properties protected and using getter functions means that you have the option of adding processing whenever the properties are accessed. Because you want the rest of the program to use these methods, make them `public`. Notice that there is no `getId()` method in the following code because that method is coming in from the `Table` class.

```php
public function getTitle() {
  return $this->title;
}
public function getLink() {
  return $this->link;
}
public function getLevel() {
  return $this->level;
}
public function getOrderby() {
  return $this->orderby;
}
```

4. Add the `_verifyInput()` method as shown in the following code. This method performs error checking for new and changing menu fields. You only use this method within this class, so make it protected.

```php
protected function _verifyInput() {
  $error = false;
  if (!trim($this->title)) {
    $error = true;
  }
  if (!trim($this->link)) {
    $error = true;
  }

  if ($error) {
    return false;
  } else {
    return true;
```

```
        }
      }
```

5. You need to be able to add rows. The `addRecord()` method that follows adds new rows to the `menus` table. This method should be `public` because it needs to be called from outside the class. You add the code to call this class in step 16.

```php
public function addRecord() {

  // Verify the fields
  if ($this->_verifyInput()) {

    // Get the Database connection
    $connection = Database::getConnection();

    // Prepare the data
    $query = "INSERT INTO menus (title, link, level, orderby)
    VALUES ('" . Database::prep($this->title) . "',
     '" . Database::prep($this->link) . "',
     '" . Database::prep($this->level) . "',
     '" . (int) $this->orderby .  "')";
    var_dump($query);
    // Run the MySQL statement
    if ($connection->query($query)) {
    $return = array('', 'Menu item successfully added.', '');

    // add success message
    return $return;
    } else {
    // send fail message and return to contactmaint
    $return = array('menumaint',
      'No Menu Item Added. Unable to create record.',
      '');
    return $return;
    }
  } else {
    // send fail message and return to maint
    $return = array('menumaint', 'No Menu Item Added.
      Missing required information.', '');
    return $return;
  }

}
```

6. You need to be able to update rows. The `editRecord()` method that follows changes existing rows in the `menus` table. This method should be `public` because it needs to be called from outside the class. You add the code to call this class in step 16.

```php
public function editRecord() {
  // Verify the fields
  if ($this->_verifyInput()) {

    // Get the Database connection
    $connection = Database::getConnection();
```

```
        // update without a password changed
        // Set up the prepared statement
        $query = 'UPDATE `menus` SET title=?, link=?, level=?, orderby=? WHERE
id=?';
        $statement = $connection->prepare($query);
        // bind the parameters
        $statement->bind_param('sssii',$this->title, $this->link,
          $this->level, $this->orderby, $this->id);

        if ($statement) {
          $statement->execute();
          $statement->close();
          // add success message
          $return = array('', 'Menu Item successfully updated.', '');
          return $return;
        } else {
          $return = array('menumaint', 'Menu Item not changed.
            Unable to change record.', (int) $this->id);
          return $return;
        }

      } else {
        // send fail message and return to categorymaint
        $return = array('menumaint', 'Menu Item not changed.
          Missing required information.', (int) $this->id);
        return $return;
      }

    }
```

7. You need to be able to delete rows. The `deleteRecord()` method deletes rows from the `menus` table. This method should be `public` because it needs to be called from outside the class. This method doesn't need an object to work because it isn't using any of the properties. The only information it needs from the menus is the `id`, which is passed to it in the parameters. Therefore, make it a static function as shown in the following code. You add the code to call this class in step 17.

```
    public static function deleteRecord($id) {
        // Get the Database connection
        $connection = Database::getConnection();
        // Set up query
        $query = 'DELETE FROM `menus` WHERE id="'. (int) $id.'"';
        // Run the query
        if ($result = $connection->query($query)) {
          $return = array('', 'Menu Item successfully deleted.', '');
          return $return;
        } else {
          $return = array('menudelete', 'Unable to delete Menu item.', (int)
$id);
          return $return;
        }
    }
```

8. You need to be able to list the menu rows. The `getMenus()` method that follows reads the rows from the `menus` table in order by the `orderby` field. It creates an object from each row and puts each object in an array that is returned. This method should be `public` because it needs to be called from outside the class. It creates a bunch of objects, but doesn't need to be an object itself, so make the method `static`.

```php
public static function getMenus() {
  // clear the results
  $items = '';
  // Get the connection
  $connection = Database::getConnection();
  // Set up query
  $query = 'SELECT * FROM `menus` ORDER BY `orderby` ASC';
  // Run the query
  $result_obj = '';
  $result_obj = $connection->query($query);
  // Loop through the results,
  // passing them to a new version of this class,
  // and making a regular array of the objects
  try {
    while($result = $result_obj->fetch_object('Menu')) {
      $items[]= $result;
    }
    // pass back the results
    return($items);
  }

  catch(Exception $e) {
    return false;
  }
}
```

9. You need to be able to get a single menu row from the menus table. The following method retrieves the row and creates an object from the data, which it returns. This method should be `public` because it needs to be called from outside the class. It creates an object, but doesn't need to be an object itself, so make the method `static`.

```php
public static function getMenu($id) {
  // Get the database connection
  $connection = Database::getConnection();
  // Set up the query
  $query = 'SELECT * FROM `menus` WHERE id="'. (int) $id.'"';
  // Run the MySQL command
  $result_obj = '';
    try {
      $result_obj = $connection->query($query);
      if (!$result_obj) {
        throw new Exception($connection->error);
      } else {
        $item = $result_obj->fetch_object('Menu');
```

```
                            if (!$item) {
                              throw new Exception($connection->error);
                            } else {
                              // pass back the results
                              return($item);
                            }
                          }
                        }
                      catch(Exception $e) {
                        echo $e->getMessage();
                      }
                  }
```

10. Finally, the last method in the Menu class is `getLevel_DropDown()`. This menu lists the level of authority required of the user before the title text appears in the menu. You use this when you add or edit menu link items. Because it needs to be called from outside the class, the method should be `public`. Unlike the previous three methods, this method works with the object so it cannot be `static`. These are the valid options to be listed:

➤ `Public`: Everyone can see the text. This is the default.

➤ `Registered`: This is the lowest level of logged-in users.

➤ `Admin`: This is the level for users who can perform admin work on the site.

➤ `LoggedIn`: This link is available for anyone who is logged in.

➤ `LoggedOut`: This link is available for anyone who is not logged in.

```
public function getLevel_DropDown() {
  // set up first option for selection if none selected
  $option_selected = '';
  if (!$this->level) {
    $option_selected = ' selected="selected"';
  }

  // Get the levels
  $items = array('Public', 'Registered', 'Admin', 'LoggedIn', 'LoggedOut');

  $html  = array();

  $html[] = '<label for="level">Choose Menu Level</label><br />';
  $html[] = '<select name="level" id="level">';

  foreach ($items as $i=>$item) {
    // If the selected parameter equals the current then flag as selected
    if ($this->level == $item) {
      $option_selected = ' selected="selected"';
    }
    // set up the option line
    $html[]  =  '<option value="' . $item . '"' . $option_selected . '>' . ↵
```

```
$item . '</option>';
      // clear out the selected option flag
      $option_selected = '';
    }

    $html[] = '</select>';
    return implode("\n", $html);

  }
```

11. Create `content/menus.php` to display a list of the menus such as are shown in Figure 30-2. This page should only be viewable by Admin-level users. Use the static `getMenus()` method to get the menu rows to be shown. Put the results in the `$items` variable and process that array with a `foreach` loop. Remember that the `getMenus()` method returns an array of objects, where each object is a row from the `menus` table. Before displaying information from the database, pass each variable through the `htmlspecialchars()` to encode the HTML entities, which helps prevent hacks and makes valid code.

```php
<?php
$accessLevel = Contact::accessLevel();
if ($accessLevel != 'Admin') :
  echo 'Sorry, no access allowed to this page';
else :
// Get the menu information
$items = Menu::getMenus();

?>
<h1>Menu List
    <a class="button" href="index.php?content=menumaint&id=0">Add</a>
</h1>

<ul class="ulfancy">
  <?php foreach ($items as $i=>$item) : ?>
    <li class="row<?php echo $i % 2; ?>">
      <h2><?php echo htmlspecialchars($item->getTitle()); ?>
        <a class="button"
          href="index.php?content=menudelete&id=<?php echo $item->getId();
?>">
          Delete
        </a>
        <a class="button"
          href="index.php?content=menumaint&id=<?php echo $item->getId();
?>">
          Edit
        </a>
      </h2>
      <p><?php echo htmlspecialchars($item->getLink()); ?></p>
    </li>
  <?php endforeach; ?>
</ul>
<?php endif; ?>
```
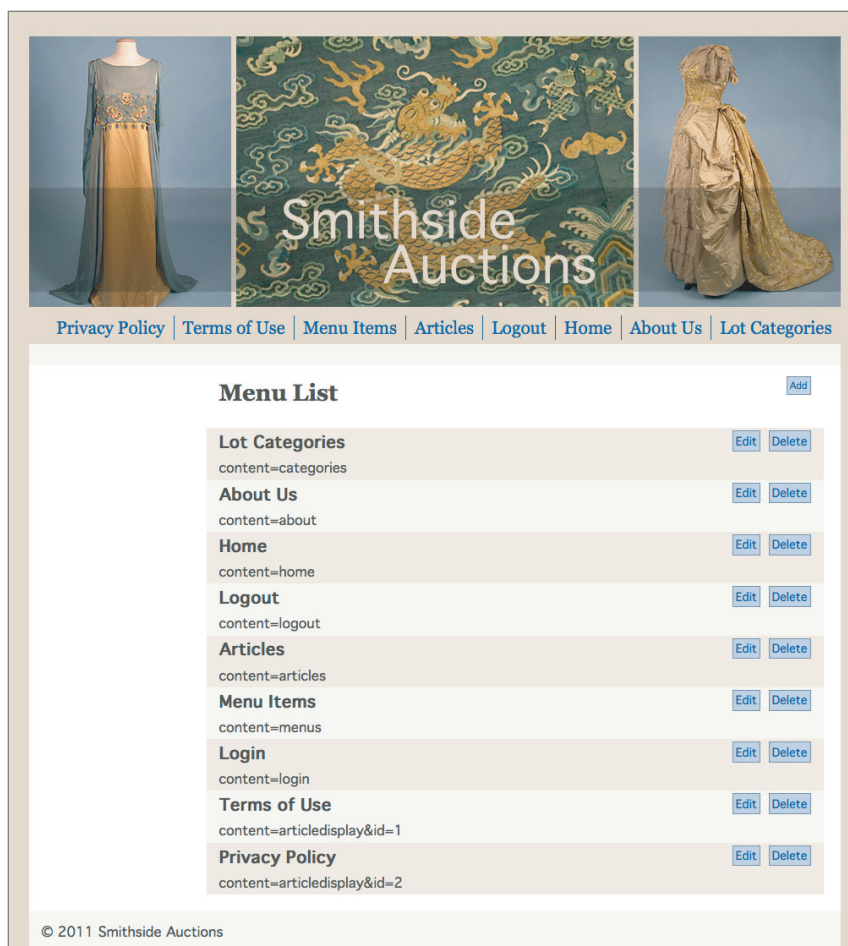
FIGURE 30-2

12. Create `content/menumaint.php` to add and edit the menu links as shown in Figure 30-3 and the following code. This page should only be viewable by Admin-level users. Pull the `id` for the menu row to be added or edited from the URL. Cast the `id` to an integer to prevent hacking. Pass that `id` to the static `getMenu()` method to get the menu row. Use the `getLevel_DropDown()` method to create a drop-down list of valid options with the level from the existing row (or the default for a new row) selected. Create a security token to pass through the SESSION and via a POST variable. These are used when you process the form to ensure that the information is coming from a real form.

```php
<?php
$accessLevel = Contact::accessLevel();
if ($accessLevel != 'Admin') :
```

```php
      echo 'Sorry, no access allowed to this page';
    else :

  $id = (int) $_GET['id'];
  // Is this an existing item or a new one?
  if ($id) {
    // Get the existing information for an existing item
    $item = Menu::getMenu($id);
  } else {
    // Set up for a new item
    $item = new Menu;
  }
    // set up the level dropdown, setting up the selected option for existing
records
    $level_dropdown = $item->getLevel_DropDown();
  ?>
  <h1>Menu Maintenance</h1>

  <form action="index.php?content=menus" method="post" name="maint" id="maint">

    <fieldset class="maintform">
      <legend><?php echo ($id) ? 'ID: '. $id : 'Add a Menu Item' ?></legend>
      <ul>
        <li><label for="title" class="required">Title</label><br />
          <input type="text" name="title" id="title" class="required"
          value="<?php echo htmlspecialchars($item->getTitle()); ?>" /></li>
        <li><label for="link" class="required">Link</label><br />
          <input type="text" name="link" id="link" class="required"
          value="<?php echo htmlspecialchars($item->getLink()); ?>" /></li>
        <li><label for="orderby">Order By</label><br />
          <input type="text" name="orderby" id="orderby" class="required"
          value="<?php echo (int) $item->getOrderby(); ?>" /></li>
        <li><?php echo $level_dropdown; ?></li>
      </ul>

      <?php
      // create token
      $salt = 'SomeSalt';
      $token = sha1(mt_rand(1,1000000) . $salt);
      $_SESSION['token'] = $token;
      ?>
      <input type="hidden" name="id" id="id" value="<?php echo $item->getId();
?>" />
      <input type="hidden" name="task" id="task" value="menu.maint" />
      <input type='hidden' name='token' value='<?php echo $token; ?>'/>
      <input type="submit" name="save" value="Save" />
      <a class="cancel" href="index.php?content=menus">Cancel</a>
    </fieldset>
  </form>
  <?php endif;;
```
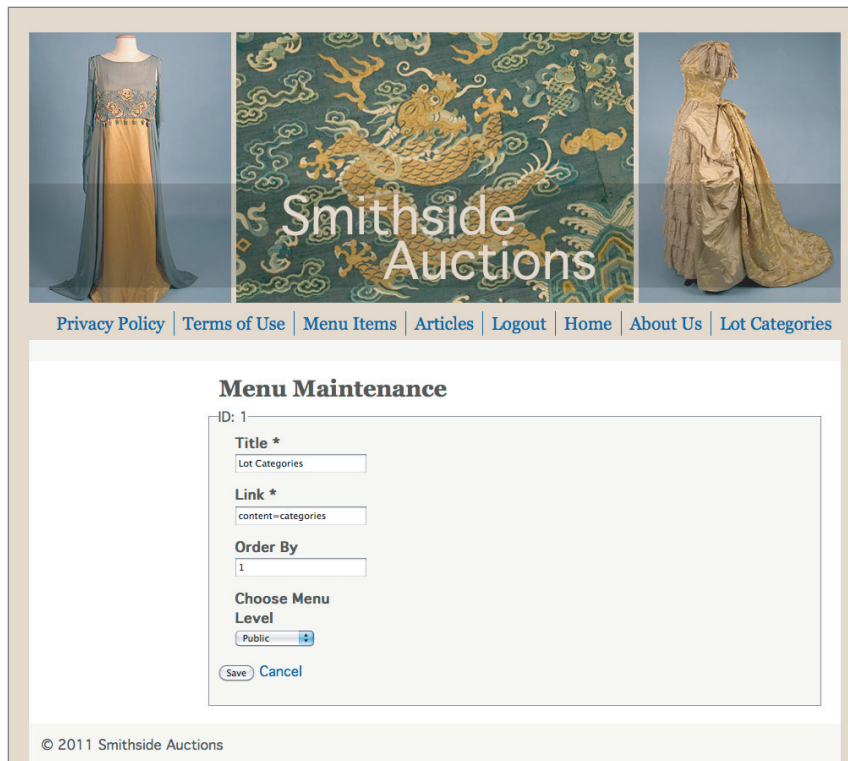
FIGURE 30-3

13. Create `content/menudelete.php` to delete menu links as shown in Figure 30-4 and the following code. This page should only be viewable by Admin-level users. Pull the `id` for the menu row to be deleted from the URL. Cast the `id` to an integer to prevent hacking. Pass that `id` to the static `getMenu()` method to get the menu row. Display the information from the row for the user to verify that this is the menu option she wants to delete.

```php
<?php
$accessLevel = Contact::accessLevel();
if ($accessLevel != 'Admin') :
  echo 'Sorry, no access allowed to this page';
else :

$id = (int) $_GET['id'];
// Get the existing information for an existing item
$item = Menu::getMenu($id);

?>
<h1>Menu Delete</h1>
```

```php
<form action="index.php?content=menus" method="post" name="maint" id="maint">

  <fieldset class="maintform">
    <legend><?php echo 'Id: '. $id ?></legend>
    <ul>
      <li><strong>Title:</strong>
        <?php echo htmlspecialchars($item->getTitle()); ?></li>
      <li><strong>Link:</strong>
        <?php echo htmlspecialchars($item->getLink()); ?></li>
    </ul>

    <?php
    // create token
    $salt = 'SomeSalt';
    $token = sha1(mt_rand(1,1000000) . $salt);
    $_SESSION['token'] = $token;
    ?>
    <input type="hidden" name="id" id="id" value="<?php echo $item->getId();
?>" />
    <input type="hidden" name="task" id="task" value="menu.delete" />
    <input type='hidden' name='token' value='<?php echo $token; ?>'/>
    <input type="submit" name="delete" value="Delete" />
    <a class="cancel" href="index.php?content=menus">Cancel</a>
  </fieldset>
</form>
<?php endif;
```
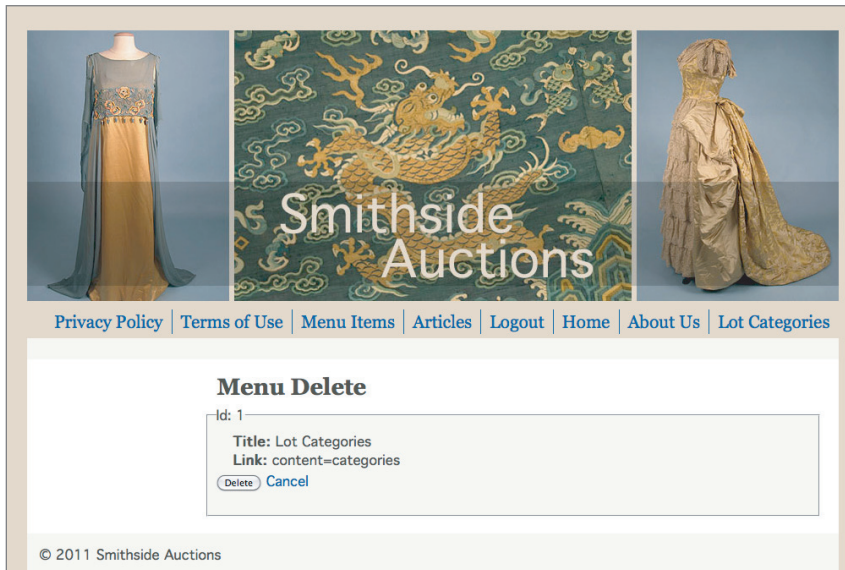

FIGURE 30-4

14. Add the `menu.maint` task to `includes/init.php`. The following code adds a `case` block to the existing `switch` statement to process this task. The task performs the function `maintMenu()` and, based on the results, either sends a success message or redirects back to the form page for corrections.

```
case 'menu.maint' :
// process the maint
$results = maintMenu();
$message .= $results[1];
// If there is redirect information
// redirect to that page
if ($results[0] == 'menumaint') {
  // pass on new messages
  if ($results[1]) {
    $_SESSION['message'] = $results[1];
  }
  header("Location: index.php?content=menumaint&id=$results[2]");
  exit;
}
break;
```

15. Add the `menu.delete` task to `includes/init.php`. The following code adds a `case` block to the existing `switch` statement to process this task. The task performs the function `deleteMenu()` and, based on the results, either sends a success message or redirects back to the form page for corrections.

```
case 'menu.delete' :
// process the delete
$results = deleteMenu();
$message .= $results[1];
// If there is redirect information
// redirect to that page
if ($results[0] == 'menudelete') {
  // pass on new messages
  if ($results[1]) {
    $_SESSION['message'] = $results[1];
  }
  header("Location: index.php?content=menudelete&id=$results[2]");
  exit;
}
break;
```

16. The following code adds the `maintMenu()` function to the `includes/functions.php` file. This function is called in the `menu.maint` task you created in step 14. It verifies that the appropriate POST values exist that the form would have submitted and verifies that the proper token exists and matches with the SESSION token. You then sanitize the information coming in through the POST variables and use them to create an array of the data. You then use that array to create an object out of the `Menu` class. If this is a new row, you call the `addRecord()` method, which verifies the data and adds a row to the database. If it is an

existing row, you call the `editRecord()` method, which verifies the data and updates the database.

```php
function maintMenu() {
  $results = '';
  if (isset($_POST['save']) AND $_POST['save'] == 'Save') {
    // check the token
    $badToken = true;
    if (!isset($_POST['token'])
    || !isset($_SESSION['token'])
    || empty($_POST['token'])
    || $_POST['token'] !== $_SESSION['token']) {
      $results = array('',
        'Sorry, go back and try again. There was a security issue.', '');
      $badToken = true;
    } else {
      $badToken = false;
      unset($_SESSION['token']);
      // Put the sanitized variables in an associative array
      // Use the FILTER_FLAG_NO_ENCODE_QUOTES to allow names like O'Connor
      $item  = array (
        'id' => (int) $_POST['id'],
        'title' => filter_input(INPUT_POST,'title',
          FILTER_SANITIZE_STRING,FILTER_FLAG_NO_ENCODE_QUOTES),
        'link'  => filter_input(INPUT_POST,'link', FILTER_SANITIZE_STRING),
        'orderby'   => (int) $_POST['orderby'],
        'level'       => filter_input(INPUT_POST,'level',
          FILTER_SANITIZE_STRING),
            );

      // Set up a Menu item object based on the posts
      $menu = new Menu($item);
      if ($menu->getId()) {
        $results = $menu->editRecord();
      } else {
        $results = $menu->addRecord();
      }
    }
  }
  return $results;
}
```

17. The following code adds the `deleteMenu()` function to the `includes/functions.php` file. This function is called in the `menu.delete` task you created in step 15. It verifies that the appropriate POST values that the form would have submitted exist and verifies that the proper token exists and matches with the SESSION token. Cast the `id` from the POST variable to an integer and pass it to the static method `deleteRecord()` to delete the row from the `menus` table.

```php
function deleteMenu() {
  $results = '';
  if (isset($_POST['delete']) AND $_POST['delete'] == 'Delete') {
```

```
              // check the token
              $badToken = true;
              if (!isset($_POST['token'])
              || !isset($_SESSION['token'])
              || empty($_POST['token'])
              || $_POST['token'] !== $_SESSION['token']) {
                $results = array('',
                  'Sorry, go back and try again. There was a security issue.', '');
                $badToken = true;
              } else {
                $badToken = false;
                unset($_SESSION['token']);

                // Delete the menu item from the table
                $results = Menu::deleteRecord((int) $_POST['id']);
              }
            }
          }
          return $results;
        }
```

18. Add the menu link temporarily to the `index.php` menu. If you want, you can skip this step and just add the `index.php?content=menus` manually in the address bar for the next step because you will be removing all these links in the `index.php` file in the step after that.

```
        <li><a href="index.php?content=menus">Menu Items</a></li>
```

19. Add the menu links for the main menu listed in the `index.php` file to the `menus` table using your new maintenance pages. The results are as shown in Figure 30-5. Now if someone creates an article page, she can also add it to the menu through the website.

id	title	link	level	orderby
1	Lot Categories	content=categories	Public	1
2	About Us	content=about	Public	2
3	Home	content=home	Public	3
4	Logout	content=logout	LoggedIn	4
5	Articles	content=articles	Admin	5
6	Menu Items	content=menus	Admin	6
7	Login	content=login	LoggedOut	7
8	Terms of Use	content=articledisplay&id=1	Public	8
9	Privacy Policy	content=articledisplay&id=2	Public	9

FIGURE 30-5

20. Remove the menu links from the `index.php` file and replace them with a call to the static `setMenu()` method in the `Menu` class. You create this method in the step 21.

```
        <div id="navigation">
          <?php echo Menu::setMenu(); ?>
          <div class="clearfloat"></div>
        </div><!-- end navigation -->
```

21. Create the static `setMenu()` method in the `Menu` class. This method calls the `getMenus()` method and then creates the HTML for an unordered list of the links read from the database, based on the authorization of the user and the access level requirements of the menu link. Because you use this method outside of the class, it needs to be `public`:

```php
public static function setMenu() {
  $items = self::getMenus();
  $logged_in = Contact::isLoggedIn();
  $accessLevel = Contact::accessLevel();

  $html = array();

  $html[] = '<h3 class="element-invisible">Menu</h3>';
  $html[] = '<ul class="mainnav">';

  foreach ($items as $item) {
    if (!($item->level)
      OR ($item->level == "Public")
      OR ($item->level == "Admin" AND $accessLevel == "Admin")
      OR ($item->level == "Registered" AND ($accessLevel == "Registered" OR ↵
$accessLevel == "Admin"))
      OR ($item->level == "LoggedIn" AND $logged_in)
      OR ($item->level == "LoggedOut" AND !$logged_in)) {
        $html[] = '<li><a href="index.php?' . $item->link. '">' . ↵
$item->title. '</a></li>';
    }
  }

  $html[] = '</ul>';
  $return = implode("\n", $html);
  return $return;
}
```

> *Watch the video for Lesson 30 on the DVD or watch online at* `www.wrox.com/go/24phpmysql`.

31

Next Steps

Over the course of this book you have taken a static website and, piece by piece, turned it into a dynamic website run from a database using PHP and MySQL. You learned the basics of programming and PHP: how to set up your computer so that it runs PHP, how to add PHP to your HTML page, and how to write PHP code. You learned what variables are, how to work with them, and how to debug your programs. You learned how to have your program make logical decisions and to loop through code, how to create functions and process forms, and how to work with objects and classes. You learned best practices and how to write secure code.

You learned how databases work and how to design one, how to use phpMyAdmin to work with MySQL, and different ways of connecting to MySQL through PHP. You learned how to create tables, enter data, select data, change data, and delete data. And finally, you learned how to combine all of these things into creating a mini content management system with a dynamic menu.

Now that you know the basics of programming in PHP and working with a MySQL database, there are different "next steps" that you can take. One is to look back at the code that you wrote for the Case Study and take it to the next level. As you worked, you probably noticed that you did a lot of copying and pasting. When you start seeing a lot of similar code, it means you have code you should be reusing instead of copying. It's time to *refactor*, which is rewriting your code to make it smaller, more secure, and more efficient. You started this process when you created the master parent `Table` class that you extended to create the `Article` and `Menu` classes. To finish that refactoring you redo the other classes to also extend the `Table` class and remove the duplicate code. You also add additional methods to the `Table` class by looking at what other methods, or parts of methods, are similar across the classes. You will find that there is a core of functionality you need that is the same from project to project.

Other programmers have found the need for that same "core of functionality," which has led to the development of *frameworks*. Frameworks are ordinary program files that have functions and master classes of commonly needed functionality that you can include in your program. A good next step is to become familiar with a framework. There is a learning curve to learn a specific framework, but once you do, you start each project with the base functionality of your

code in place, bug tested and, presumably, secure. CakePHP, CodeIgniter, Symfony, Yii, and Zend are examples of PHP frameworks that are available.

If you are building websites, popular open-source content management systems are available that are built on PHP and MySQL, such as Joomla!, Drupal, and WordPress, which have done the heavy lifting for you. For instance, Joomla is based on Joomla Platform, an object-oriented framework with classes you can extend. Each of these systems has its own way of doing things. If you use one of these CMSs, a good next step is to learn how to extend that CMS.

Seeing how others do things is a good way of learning, so examine how existing programs are written. However, you need to be cautious when studying or copying someone else's code. Coding styles in PHP have changed in the last decade and what was considered perfectly good code five years ago may not be how you should be coding today. Newer versions of PHP enable you to take advantage of programming best practices, including encapsulation and object-oriented programming, that older versions of PHP did not handle. In addition, the increasing availability of sophisticated hacking scripts means that you need to use programming techniques that are more secure than what used to be common practice.

In the same way, examining existing databases is a way to see how other programmers have organized their data into tables. An interesting website for seeing how databases are organized is `www.databaseanswers.org/data_models/index.htm`. This site shows examples of databases for various types of businesses and needs.

Try visiting `http://stackoverflow.com/` when you have questions that need to be answered. Use the search function to find out if an answer already exists for the question you have. I recommend that you include "php" or "mysql" as part of your search because this site covers many programming languages and database types. If you can't find your answer, you can post a specific question yourself.

Finally, to get more information on the PHP functions that you have learned and find out about additional functions you come across, use the online PHP Manual at `http://php.net/manual/en/index.php`. You learned about this manual in Lesson 3. This manual gives the syntax of the function, lists the parameters available, and what the function returns. It also contains comments from users that can be enlightening, though some of them are old and have been superseded by newer PHP versions.

For more information on MySQL, use the online MySQL Reference Manual at `http://dev.mysql.com/doc/refman/5.6/en/index.html`. There is a manual for each of the different versions, which you can easily switch to using the menu on the left of the page. If you are looking for a specific statement, go to Chapter 12, "SQL Statement Syntax," at `http://dev.mysql.com/doc/refman/5.6/en/sql-syntax.html`. Data definition statements are the statements that deal with the structure of the database, such as creating and altering tables, or listing what fields are in a particular table. Data manipulation statements are the ones that deal with the actual data such as the INSERT, UPDATE, and SELECT statements.

A

What's on the DVD?

This appendix provides you with information on the contents of the DVD that accompanies this book. For the latest and greatest information, please refer to the ReadMe file located at the root of the DVD. Here is what you will find in this appendix:

➤ System Requirements

➤ Using the DVD

➤ What's on the DVD

➤ Troubleshooting

SYSTEM REQUIREMENTS

Most reasonably up-to-date computers with a DVD drive should be able to play the screencasts that are included on the DVD.

USING THE DVD

To access the content from the DVD, follow these steps:

1. Insert the DVD into your computer's DVD-ROM drive. The license agreement appears.

> *The interface won't launch if you have autorun disabled. In that case, click Start ⇨ Run (for Windows 7, click Start ⇨ All Programs ⇨ Accessories ⇨ Run). In the dialog box that appears, type **D:\Start.exe.** (Replace D with the proper letter if your DVD drive uses a different letter. If you don't know the letter, check how your DVD drive is listed under My Computer.) Click OK.*

   2. Read through the license agreement, and then click the Accept button if you want to use the DVD.

   The DVD interface appears. Simply select the lesson number for the video you want to view.

WHAT'S ON THE DVD?

Each of this book's lessons contains a Try It section that enables you to practice the concepts covered by that lesson. The Try It includes a high-level overview, requirements, and step-by-step instructions explaining how to build the example program.

This DVD contains video screencasts showing my computer screen as I work through key pieces of the Try Its from each lesson. In the audio I explain what I'm doing step-by-step so you can see how the techniques described in the lesson translate into actions.

I don't always show how to build every last piece of a Try It's program. For example, if the requirements ask you to do the same thing multiple times, I may only do the first one and let you do the rest so you don't need waste time watching me do the same thing again and again.

I recommend using the following steps when reading a lesson:

   1. Read the lesson's text.

   2. Read the Try It's overview, requirements, and hints.

   3. Read the step-by-step instructions. If the code you write doesn't work, use the code provided to find your problem. Look for places where my solution differs from yours. In programming there's always more than one way to solve a problem, and it's good to know about several different approaches.

   4. Watch the screencast to see how I handle the key issues.

You can also download all of the book's examples and solutions to the Try Its at the book's websites.

TROUBLESHOOTING

If you have difficulty installing or using any of the materials on the companion DVD, try the following solutions:

➤ **Reboot if necessary.** As with many troubleshooting situations, it may make sense to reboot your machine to reset any faults in your environment.

➤ **Turn off any anti-virus software that you may have running.** Installers sometimes mimic virus activity and can make your computer incorrectly believe that it is being infected by a virus. (Be sure to turn the anti-virus software back on later.)

➤ **Close all running programs.** The more programs you're running, the less memory is available to other programs. Installers also typically update files and programs; if you keep other programs running, installation may not work properly.

➤ **Reference the ReadMe.** Please refer to the ReadMe file located at the root of the DVD for the latest product information at the time of publication.

CUSTOMER CARE

If you have trouble with the DVD, please call the Wiley Product Technical Support phone number at (800) 762-2974. Outside the United States, call 1(317) 572-3994. You can also contact Wiley Product Technical Support at `http://support.wiley.com`. John Wiley & Sons will provide technical support only for installation and other general quality control items. For technical support on the applications themselves, consult the program's vendor or author.

To place additional orders or to request information about other Wiley products, please call (877) 762-2974.

INDEX

SOFTWARE LICENSE AGREEMENT

Important - Read carefully before opening software package.

This is a legal agreement between you, the end user, and John Wiley & Sons, Inc. ("Wiley").

The enclosed Wiley software program and accompanying data (the "Software") is licensed by Wiley for use only on the terms set forth herein. Please read this license agreement. Registering the product indicates that you accept these terms. If you do not agree to these terms, return the full product (including documentation) with proof of purchase within 30 days for a full refund. In addition, if you are not satisfied with this product for any other reason, you may return the entire product (including documentation) with proof of purchase within 15 days for a full refund.

1. **License:** Wiley hereby grants you, and you accept, a non-exclusive and non-transferable license, to use the Software on the following terms and conditions only:

   (a) The Software is for your personal use only.

   (b) You may use the Software on a single terminal connected to a single computer (i.e., single CPU) and a laptop or other secondary machine for personal use.

   (c) A backup copy or copies of the Software may be made solely for your personal use. Except for such back up copy or copies, you may not copy, modify, distribute, transmit or otherwise reproduce the Software or related documentation, in whole or in part, or systematically store such material in any form or media in a retrieval system; or store such material in electronic format in electronic reading rooms; or transmit such material, directly or indirectly, for use in any service such as document delivery or list serve, or for use by any information brokerage or for systematic distribution of material, whether for a fee or free

   of charge. You agree to protect the Software and documentation from unauthorized use, reproduction, or distribution.

   (d) You agree not to remove or modify any copyright or proprietary notices, author attribution or disclaimer contained in the Software or documentation or on any screen display, and not to integrate material from therefrom with other material or otherwise create derivative works in any medium based on or including materials from the Software or documentation.

   (e) You agree not to translate, decompile, disassemble or otherwise reverse engineer the Software.

2. **Limited Warranty:**

   (a) Wiley warrants that this product is free of defects in materials and workmanship under normal use for a period of 60 days from the date of purchase as evidenced by a copy of your receipt. If during the 60-day period a defect occurs, you may return the product. Your sole and exclusive remedy in the event of a defect is expressly limited to the replacement of the defective product at no additional charge.

   (b) The limited warranty set forth above is in lieu of any and all other warranties, both express and implied, including but not limited to the implied warranties of merchantability or fitness for a particular purpose. The liability of Wiley pursuant to this limited warranty will be limited to replacement of the defective copies of the Software. Some states do not allow the exclusion of implied warranties, so the preceding exclusion may not apply to you.

   (c) Because software is inherently complex and may not be completely free of errors, you are advised to verify your work and to make backup

copies. In no event will Wiley, nor anyone else involved in creating, producing or delivering the Software, documentation or the materials contained therein, be liable to you for any direct, indirect, incidental, special, consequential or punitive damages arising out of the use or inability to use the Software, documentation or materials contained therein even if advised of the possibility of such damages, or for any claim by any other party. In no case will Wiley's liability exceed the amount paid by you for the Software. Some states do not allow the exclusion or limitation of liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

(d) Wiley reserves the right to make changes, additions, and improvements to the Software at any time without notice to any person or organization. No guarantee is made that future versions of the Software will be compatible with any other version.

3. **Term:** Your license to use the Software and documentation will automatically terminate if you fail to comply with the terms of this Agreement. If this license is terminated you agree to destroy all copies of the Software and documentation.

4. **Ownership:** You acknowledge that all rights (including without limitation, copyrights, patents and trade secrets) in the Software and documentation (including without limitation, the structure, sequence, organization, flow, logic, source code, object code and all means and forms of operation of the Software) are the sole and exclusive property of Wiley and/or its licensors, and are protected by the United Sates copyright laws, other applicable copyright laws, and international treaty provisions.

5. **Restricted Rights:** This Software and/or user documentation are provided with restricted and limited rights.

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the Rights in Technical Data and Computer Software clause in DAR 7-104.9(a), FAR 52.2227-14 (June 1987) Alternate III(g)(3)(June 1987), FAR 52.227-19 (June 1987), or DFARS 52.227-701 (c) (1)(ii)(June 1988), or their successors, as applicable. Contractor/manufacturer is John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030.

6. **Canadian Purchase:** If you purchased this product in Canada, you agree to the following: the parties hereto confirm that it is their wish that this Agreement, as well as all other documents relating hereto, including Notices, have been and will be drawn up in the English language only.

7. **Technical Support:** Wiley will respond to all technical support inquiries within 48 hours.

8. **General:** This Agreement represents the entire agreement between us and supersedes any proposals or prior Agreements, oral or written, and any other communication between us relating to the subject matter of this Agreement. This Agreement will be construed and interpreted pursuant to the laws of the State of New York, without regard to such State's conflict of law rules. Any legal action, suit or proceeding arising out of or relating to this Agreement or the breach thereof will be instituted in a court of competent jurisdiction in New York County in the State of New York and each party hereby consents and submits to the personal jurisdiction of such court, waives any objection to venue in such court and consents to the service of process by registered or certified mail, return receipt requested, at the last known address of such party. Should you have any questions concerning this Agreement or if you desire to contact Wiley for any reason, please write to: John Wiley & Sons, Inc., Customer Sales and Service, 10475 Crosspoint Blvd, Indianapolis, IN 46256.