3rd Edition
Covers PHP5, MySQL4 and Mac OS X

sitepoint

Build Your Own Database Driven Website Using PHP & MySQL

By Kevin Yank

A Practical Step-by-Step Guide

Build Your Own Database Driven Website Using PHP and MySQL, 3rd Edition (First 4 Chapters)

Thank you for downloading the first four chapters of Kevin Yank's *Build Your Own Database Driven Website Using PHP and MySQL, 3rd Edition*.

This excerpt encapsulates the Summary of Contents, Information about the Author and SitePoint, Table of Contents, Introduction, and the first four chapters of the book.

We hope you find this information useful in evaluating the book.

Summary of Contents of this Excerpt

Summary of Additional Book Contents

Build Your Own Database Driven Website Using PHP & MySQL

by Kevin Yank

Build Your Own Database Driven Website Using PHP & MySQL

by Kevin Yank

Editor: Georgina Laidlaw
Managing Editor: Simon Mackie
Index Editor: Bill Johncocks
Cover Design: Julian Carroll
Printing History:
First Edition: August 2001
Second Edition: February 2003
Third Edition: October 2004

Notice of Rights

Notice of Liability

Trademark Notice

Published by SitePoint Pty. Ltd.

424 Smith Street Collingwood
VIC Australia 3066.
Web: www.sitepoint.com
Email: business@sitepoint.com

ISBN 0–9752402–1–8
Printed and bound in the United States of America

About the Author

As Technical Director for SitePoint, Kevin Yank oversees all of its technical publications—books, articles, newsletters and blogs. He has written over 50 articles for SitePoint on technologies including PHP, XML, ASP.NET, Java, JavaScript and CSS. He writes *The SitePoint Tech Times*, SitePoint's biweekly technical newsletter for Web developers, which has over 75,000 readers worldwide.

When he's not discovering new technologies, editing books, or catching up on sleep, Kevin can be found helping other up-and-coming Web developers in the SitePoint Forums.

Kevin lives in Melbourne, Australia, with several potted plants. In his spare time he enjoys flying light aircraft and learning the fine art of improvised acting. Go you big red fire engine!

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for Web professionals. Visit http://www.sitepoint.com/ to access our books, newsletters, articles and community forums.

To my parents, Cheryl and Richard, for making all this possible.

Table of Contents

Preface

"Content is king." Cliché, yes; but it has never been more true. Once you've mastered HTML and learned a few neat tricks in JavaScript and Dynamic HTML, you can probably design a pretty impressive-looking Website. But your next task must be to fill that fancy page layout with some real information. Any site that successfully attracts repeat visitors has to have fresh and constantly updated content. In the world of traditional site building, that means HTML files—and lots of 'em.

The problem is that, more often than not, the people who provide the content for a site are not the same people who handle its design. Frequently, the content provider doesn't even *know* HTML. How, then, is the content to get from the provider onto the Website? Not every company can afford to staff a full-time Webmaster, and most Webmasters have better things to do than copying Word files into HTML templates, anyway.

Maintenance of a content-driven site can be a real pain, too. Many sites (perhaps yours?) feel locked into a dry, outdated design because rewriting those hundreds of HTML files to reflect a new look would take forever. Server-side includes (SSIs) can help alleviate the burden a little, but you still end up with hundreds of files that need to be maintained should you wish to make a fundamental change to your site.

The solution to these headaches is database-driven site design. By achieving complete separation between your site's design and the content you want to present, you can work with each without disturbing the other. Instead of writing an HTML file for every page of your site, you need only to write a page for each *kind* of information you want to be able to present. Instead of endlessly pasting new content into your tired page layouts, create a simple content management system that allows the writers to post new content themselves without a lick of HTML!

In this book, I'll provide you with a hands-on look at what's involved in building a database-driven Website. We'll use two tools for this, both of which may be new to you: the **PHP** scripting language and the **MySQL** relational database management system. If your Web host provides PHP and MySQL support, you're in great shape. If not, we'll be looking at the setup procedures under Linux, Windows, and Mac OS X, so don't sweat it.

Who Should Read This Book

This book is aimed at intermediate and advanced Web designers looking to make the leap into server-side programming. You'll be expected to be comfortable with simple HTML, as I'll make use of it without much in the way of explanation. No knowledge of JavaScript is assumed or required, but if you *do* know JavaScript, you'll find it will make learning PHP a breeze, since the languages are quite similar.

By the end of this book, you can expect to have a grasp of what's involved in setting up and building a database-driven Website. If you follow the examples, you'll also learn the basics of PHP (a server-side scripting language that gives you easy access to a database, and a lot more) and **Structured Query Language** (**SQL**—the standard language for interacting with relational databases) as supported by **MySQL**, one of the most popular free database engines available today. Most importantly, you'll come away with everything you need to get started on your very own database-driven site!

What's In This Book

This book comprises the following 12 chapters. Read them in order from beginning to end to gain a complete understanding of the subject, or skip around if you need a refresher on a particular topic.

Chapter 1: ***Installation***

Before you can start building your database-driven Web presence, you must first ensure that you have the right tools for the job. In this first chapter, I'll tell you where to obtain the two essential components you'll need: the PHP scripting language and the MySQL database management system. I'll step you through the setup procedures on Windows, Linux, and Mac OS X, and show you how to test that PHP is operational on your Web server.

Chapter 2: ***Getting Started with MySQL***

Although I'm sure you'll be anxious to get started building dynamic Web pages, I'll begin with an introduction to databases in general, and the MySQL relational database management system in particular. If you've never worked with a relational database before, this should definitely be an enlightening chapter that will whet your appetite for things to come! In the process, we'll build up a simple database to be used in later chapters.

Chapter 3: ***Getting Started with PHP***

Here's where the fun really starts. In this chapter, I'll introduce you to the PHP scripting language, which can easily be used to build dynamic Web pages that present up-to-the-moment information to your visitors. Readers with previous programming experience will probably be able to get away with a quick skim of this chapter, as I explain the essentials of the language from the ground up. This is a must-read chapter for beginners, however, as the rest of this book relies heavily on the basic concepts presented here.

Chapter 4: ***Publishing MySQL Data on the Web***

In this chapter we bring together PHP and MySQL, which you'll have seen separately in the previous chapters, to create some of your first database-driven Web pages. We'll explore the basic techniques of using PHP to retrieve information from a database and display it on the Web in real time. I'll also show you how to use PHP to create Web-based forms for adding new entries to, and modifying existing information in, a MySQL database on-the-fly.

Chapter 5: ***Relational Database Design***

Although we'll have worked with a very simple sample database in the previous chapters, most database-driven Websites require the storage of more complex forms of data than we'll have dealt with so far. Far too many database-driven Website designs are abandoned midstream, or are forced to start again from the beginning, because of mistakes made early on, during the design of the database structure. In this critical chapter, I'll teach the essential principles of good database design, emphasizing the importance of data normalization. If you don't know what that means, then this is definitely an important chapter for you to read!

Chapter 6: ***A Content Management System***

In many ways the climax of the book, this chapter is the big payoff for all you frustrated site builders who are tired of updating hundreds of pages whenever you need to make a change to a site's design. I'll walk you through the code for a basic content management system that allows you to manage a database of jokes, their categories, and their authors. A system like this can be used to manage simple content on your Website; just a few modifications, and you'll have a Web administration system that will have your content providers submitting content for publication on your site in no time—all without having to know a shred of HTML!

Chapter 7: ***Content Formatting and Submission***

Just because you're implementing a nice, easy tool to allow site administrators to add content to your site without their knowing HTML, doesn't mean you

have to restrict that content to plain, unformatted text. In this chapter, I'll show you some neat tweaks you can make to the page that displays the contents of your database—tweaks that allow it to incorporate simple formatting such as bold or italicized text, among other things. I'll also show you a simple way safely to make a content submission form directly available to your content providers, so that they can submit new content directly into your system for publication, pending an administrator's approval.

Chapter 8: *MySQL Administration*

While MySQL is a good, simple database solution for those who don't need many frills, it does have some complexities of its own that you'll need to understand if you're going to rely on a MySQL database to store your content. In this section, I'll teach you how to perform backups of, and manage access to, your MySQL database. In addition to a couple of inside tricks (like what to do if you forget your MySQL password), I'll explain how to repair a MySQL database that has become damaged in a server crash.

Chapter 9: *Advanced SQL Queries*

In Chapter 5 we saw what was involved in modelling complex relationships between pieces of information in a relational database like MySQL. Although the theory was quite sound, putting these concepts into practice requires that you learn a few more tricks of Structured Query Language. In this chapter, I'll cover some of the more advanced features of this language to get you juggling complex data like a pro.

Chapter 10: *Binary Data*

Some of the most interesting applications of database-driven Web design include some juggling of binary files. Online file storage services like the now-defunct *iDrive* are prime examples, but even a system as simple as a personal photo gallery can benefit from storing binary files (e.g. pictures) in a database for retrieval and management on the fly. In this chapter, I'll demonstrate how to speed up your Website by creating static copies of dynamic pages as regular intervals—using PHP, of course! With these basic file-juggling skills in hand, we'll go on to develop a simple online file storage and viewing system and learn the ins and outs of working with binary data in MySQL.

Chapter 11: *Cookies and Sessions in PHP*

One of the most hyped new features in PHP 4.0 was built-in support for sessions. But what are sessions? How are they related to cookies, a long-suffering technology for preserving stored data on the Web? What makes persistent data so important in current ecommerce systems and other Web applications? This chapter answers all those questions by explaining how PHP

supports both cookies and sessions, and exploring the link between the two. At the end of this chapter, we'll develop a simple shopping cart system to demonstrate their use.

Chapter 12: *Structured PHP Programming*

Techniques to better structure your code are useful in all but the simplest of PHP projects. The PHP language offers many facilities to help you do this, and in this chapter, I'll explore some of the simple techniques that exist to keep your code manageable and maintainable. You'll learn to use include files to avoid having to write the same code more than once when it's needed by many pages of your site; I'll show you how to write your own functions to extend the built-in capabilities of PHP and to streamline the code that appears within your Web pages; we'll also dabble in the art of defining constants that control aspects of your Web applications' functionality. We'll then put all these pieces together to build an access control system for your Website. Its sophisticated structure will ensure that it can be used and reused on just about any site you decide to build.

The Book's Website

Located at http://www.sitepoint.com/books/phpmysql1/, the Website supporting this book will give you access to the following facilities:

The Code Archive

As you progress through the text, you'll note a number of references to the code archive. This is a downloadable ZIP archive that contains complete code for all the examples presented in this book.

Updates and Errata

No book is perfect, and even though this is a third edition, I expect that watchful readers will be able to spot at least one or two mistakes before its end. Also, PHP and MySQL (and even the Web in general) are moving targets, constantly undergoing changes with each new release. The Errata page on the book's Website will always have the latest information about known typographical and code errors, and necessary updates for changes to PHP and MySQL.

The SitePoint Forums

While I've made every attempt to anticipate any questions you may have, and answer them in this book, there is no way that *any* book could cover everything there is to know about PHP and MySQL. If you have a question about anything in this book, the best place to go for a quick answer is http://www.sitepoint.com/forums/. Not only will you find a vibrant and knowledgeable PHP community, but you'll occasionally even find me, the author, there in my spare hours.

The SitePoint Newsletters

In addition to books like this one, I write a free, biweekly (that's every two weeks) email newsletter called *The SitePoint Tech Times*. In it, I write about the latest news, product releases, trends, tips, and techniques for all technical aspects of Web development. If nothing else, you'll get useful PHP articles and tips, but if you're interested in learning other languages, you'll find it especially useful.

SitePoint also publishes a number of other newsletters. The long-running *SitePoint Tribune* is a biweekly digest of the business and moneymaking aspects of the Web. Whether you're a freelance developer looking for tips to score that dream contract, or a marketing major striving to keep abreast of changes to the major search engines, this is the newsletter for you. *The SitePoint Design View* is a monthly compilation of the best in Web design. From new CSS layout methods to subtle PhotoShop techniques, SitePoint's chief designer shares his years of experience in its pages.

Browse the archives or sign up to any of SitePoint's free newsletters at http://www.sitepoint.com/newsletter/.

Your Feedback

If you can't find your answer through the forums, or you wish to contact me for any other reason, the best place to write is `<books@sitepoint.com>`. We have a well-manned email support system set up to track your inquiries, and if our support staff is unable to answer your question, they send it straight to me. Suggestions for improvement as well as notices of any mistakes you may find are especially welcome.

And now, without further ado, let's get started!

1 Installation

Over the course of this book, it will be my job to guide you as you take your first steps beyond the HTML world of client-side site design. Together, we'll explore what it takes to develop the kind of large, content-driven sites that are so successful today, but which can be a real headache to maintain if they aren't built right.

Before we get started, you need to gather together the tools you'll need for the job. In this first chapter, I'll guide you as you download and set up the two software packages you'll need: PHP and MySQL.

PHP is a server-side scripting language. You can think of it as a "plug-in" for your Web server that will allow it to do more than just send plain Web pages when browsers request them. With PHP installed, your Web server will be able to read a new kind of file (called a **PHP script**) that can do things like retrieve up-to-the-minute information from a database and insert it into a Web page before sending it to the browser that requested it. PHP is completely free to download and use.

To retrieve information from a database, you first need to *have* a database. That's where **MySQL** comes in. MySQL is a relational database management system, or RDBMS. We'll get into the exact role it plays and how it works later, but basically it's a software package that is very good at the organization and management of large amounts of information. MySQL also makes that information really easy to access with server-side scripting languages like PHP. MySQL is released

under the GNU General Public License (GPL), and is thus free for most uses on all of the platforms it supports. This includes most Unix-based platforms, like Linux and even Mac OS X, as well as Windows.

If you're lucky, your current Web host may already have installed MySQL and PHP on your Web server. If that's the case, much of this chapter will not apply to you, and you can skip straight to the section called "If Your Web Host Provides PHP and MySQL" to make sure your setup is shipshape.

Everything we'll discuss in this book may be carried out on a Windows- or Unix-based[1] server. The installation procedure will differ in accordance with the type of server you have at your disposal. The next few sections deal with installation on a Windows-based Web server, installation under Linux, and installation on Mac OS X. Unless you're especially curious, you need only read the section that applies to you.

Windows Installation

Installing MySQL

As I mentioned above, MySQL may be downloaded free of charge. Simply proceed to http://dev.mysql.com/downloads/ and choose the recommended stable release (as of this writing, it is MySQL 4.0). On the MySQL 4.0 download page, under the heading Windows downloads, select and download the release that includes the installer. After downloading the file (it's about 21MB as of this writing), unzip it and run the `setup.exe` program contained therein.

Once installed, MySQL is ready to roll (barring a couple of configuration tasks that we'll look at shortly), except for one minor issue that only affects you if you're running Windows NT, 2000, XP, or Server 2003. If you use any of those operating systems, you need to create a file called `my.cnf` in the root of your `C:` drive to indicate where you have installed MySQL.

To create this file, simply open Notepad and type these three lines:

```
[mysqld]
basedir = c:/mysql/
datadir = c:/mysql/data/
```

[1] From this point forward, I'll refer to all Unix-style platforms supported by PHP and MySQL, such as Linux, FreeBSD, and Mac OS X, with the collective name 'Linux'.

If you installed MySQL into a directory other than `C:\mysql`, replace both occurrences of `c:/mysql` in the above with the path to which you installed. Notice the use of forward slashes (`/`) instead of the usual backslashes (`\`) in the paths. For instance, on my system I edited the file to read as follows:

```
[mysqld]
basedir = d:/Program Files/MySQL/
datadir = d:/Program Files/MySQL/data/
```

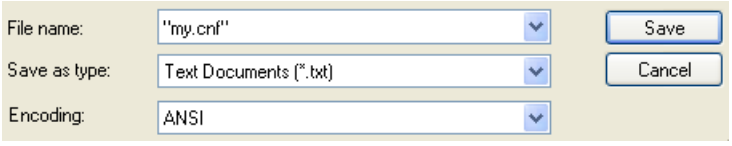
Save the file as `my.cnf` in the root directory of `C:` drive.

Notepad and File Name Extensions

Notepad is designed to edit text files, which normally have a file name extension of `.txt`. When you try to save a file with a different extension (e.g. `my.cnf`), Notepad will normally add a `.txt` extension to the end of the file name (`my.cnf.txt`) so that Windows will treat it as a text file.

To prevent this, simply put double quotes around the file name as you enter it in the Save As dialog box, as shown in Figure 1.1.

Figure 1.1. Save the File As `.cnf` in Notepad



If you don't like the idea of a MySQL configuration file sitting in the root of your `C:` drive, instead, you can name it `my.ini` and put it in your Windows directory (e.g. `C:\WINDOWS` or `C:\WINNT` if Windows is installed on drive `C:`).

MySQL will now run on your Windows NT, 2000, XP, or Server 2003 system! If you're using Windows 95, 98, or ME, this step is not necessary—MySQL will run just fine as installed.

Working with `.cnf` files in Windows

It just so happens that files ending in `.cnf` have a special meaning to Windows, so, even if you have Windows configured to show file extensions, the `my.cnf` file you created will still appear as simply `my` with a special icon. Windows actually expects these files to contain SpeedDial links for Microsoft NetMeeting.

Assuming you don't use NetMeeting (or at least, that you don't use its SpeedDial facility) you can remove this file type from your system, enabling you to work with these files normally:

1. Open the Windows Registry Editor (in Windows NT, 2000, XP, or Server 2003, click Start, Run..., and then type `regedt32.exe` to launch the editor; in Windows 9x/ME run `regedit.exe` instead).
2. Navigate to the `HKEY_LOCAL_MACHINE\SOFTWARE\Classes` branch of the registry, where you'll find a list of all the registered file types on the system.
3. Select the `.cnf` key and choose Edit, Delete from the menu to remove it.
4. Log out and log back in, or restart Windows for the change to take effect.

If you prefer not to mess with the file types on your system, you should still be able to open the file in Notepad to edit it as needed.

Just like your Web server, MySQL is a program that should be run in the background so that it may respond to requests for information at any time. The server program may be found in the `bin` subfolder of the folder into which you installed MySQL. However, to complicate matters, several versions of the MySQL server are available:

`mysqld.exe` This is the basic version of MySQL if you run Windows 95, 98, or ME. It includes support for all the advanced features, and includes debug code to provide additional information in the case of a crash (if your system is set up to debug programs). As a result of this code, however, the server might run a little slow, and generally I've found that MySQL is so stable that crashes aren't really a concern.

`mysqld-opt.exe` This version of the server lacks a few of the advanced features of the basic server, and does not include the debug code. It's optimized to run quickly on today's processors. For beginners, the advanced features are

not a big concern. You certainly won't be using them while you complete the tasks in this book. This is the version of choice for beginners running Windows 95, 98, or ME.

`mysqld-nt.exe` This version of the server is compiled and optimized like `mysqld-opt`, but is designed to run under Windows NT, 2000, XP, or Server 2003 as a service. If you're using any of those operating systems, this is probably the server for you.

`mysqld-max.exe` This version is like `mysqld-opt.exe`, but contains advanced features that support transactions. You won't need these features in this book.

`mysqld-max-nt.exe` This version's similar to `mysqld-nt.exe`, in that it will run as a Windows service, but it has the same advanced features as `mysqld-max.exe`.

All these versions were installed for you in the `bin` directory. If you're running on Win9x/ME, I recommend you stick with `mysql-opt` for now—move to `mysqld-max` if you ever need the advanced features. On WinNT/2000/XP/2003, `mysqld-nt` is my recommendation. Upgrade to `mysqld-max-nt` when you need more advanced features.

Starting MySQL is also a little different under WinNT/2000/XP/2003, but this time let's begin with the procedure for Win9x/ME. Open an MS-DOS Command Prompt,[2] proceed to the MySQL `bin` directory, and run your chosen server program:

```
C:\mysql\bin>mysqld-opt
```

Don't be surprised when you receive another command prompt. This command launches the server program so that it runs in the background, even after you close the command prompt. If you press Ctrl-Alt-Del to pull up the task list, you should see the MySQL server listed as one of the tasks that's active on your system.

[2] If you're unfamiliar with the workings of the Command Prompt, check out my article Kev's Command Prompt Cheat Sheet [http://www.sitepoint.com/article/846] to get familiar with how it works before you proceed further.

To ensure that the server is started whenever Windows starts, you might want to create a shortcut to the program and put it in your Startup folder. This is just like creating a shortcut to any other program on your system.

On WinNT/2000/XP/2003, you must install MySQL as a system service. Fortunately, this is very easy to do. Simply open a Command Prompt (under Accessories in the Start Menu) and run your chosen server program with the **--install** option:

```
C:\mysql\bin>mysqld-nt --install
Service successfully installed.
```

This will install MySQL as a service that will be started the next time you reboot Windows. To start MySQL manually without having to reboot, just type this command (which can be run from any directory):

```
C:\>net start mysql
The MySQL service is starting.
The MySQL service was started successfully.
```

To verify that the MySQL server is running properly, press **Ctrl-Alt-Del** and open the Task List. If all is well, the server program should be listed on the *Processes* tab.

Installing PHP

The next step is to install PHP. At the time of this writing, PHP 5.0 has just been released, with numerous improvements over the previous version; however, PHP 4.3 has become well-established as the version of choice due to its track record of stability and performance. The procedures for installing these two versions are nearly identical. Although I'll focus primarily on installing PHP 5.0 in these pages, I'll note any significant differences if you happen to be working with PHP 4.3. All of the code in this book will work with both versions of PHP.

Download PHP for free from http://www.php.net/downloads.php. You'll want the PHP 5.x zip package under Windows Binaries; avoid the installer version if you can.

PHP was designed to run as a plug-in for existing Web server software such as Internet Information Services, Apache, Sambar or OmniHTTPD. To test dynamic Web pages with PHP, you'll need to equip your own computer with Web server software, so that PHP has something to plug into.

If you have Windows 2000, XP Professional[3], or Server 2003, then install IIS (if it's not already on your system): open Control Panel > Add/Remove Programs > Add/Remove Windows Components, and select Internet Information Services (IIS) from the list of components. If you're not lucky enough to have IIS at your disposal,[4] you can use a free, third-party Web server like Apache instead. I'll give instructions for both options in detail.

First, *whether or not you have IIS*, complete these steps:

1. Unzip the file you downloaded from the PHP Website into a directory of your choice. I recommend `C:\PHP` and will refer to this directory from this point onward, but feel free to choose another directory if you like.

2. Find the file called `php5ts.dll` in the PHP folder and copy it to the `system32` subfolder of your Windows folder (e.g. `C:\WINDOWS\system32`).

PHP 4.3

The file is called `php4ts.dll` for PHP 4.3.

3. Find the file called `php.ini-dist` in the PHP folder and copy it to your Windows folder. Once it's there, rename it `php.ini`.

4. Open the `php.ini` file in your favorite text editor (use WordPad if Notepad doesn't display the file properly). It's a large file with a lot of confusing options, but look for a line that begins with `extension_dir`, and set it so that it points to the `ext` subfolder of your PHP folder:

```
extension_dir = "C:\PHP\ext"
```

   A little further down, you'll see a bunch of lines beginning with `;extension=`. These are optional extensions, disabled by default. We want to enable the MySQL extension so that PHP can communicate with MySQL. To do this, remove the semicolon from the start of the `php_mysql.dll` line:

```
extension=php_mysql.dll
```

[3]Windows XP Home Edition does not come with IIS.
[4]A feature-limited edition of IIS called "Personal Web Server" (PWS) was distributed on the Windows 98 Second Edition CD, and was available for earlier editions of Windows as well. While, technically, PHP can run on PWS, this Web server is somewhat unstable and has a great many known security holes. For these reasons, I highly recommend using Apache if an up-to-date version of IIS is not available for your Windows operating system.

Even further down, look for a line that starts with `session.save_path` and set it to your Windows `TEMP` folder:

```
session.save_path = "C:\WINDOWS\Temp"
```

Save the changes you made and close your text editor.

Now, if *you have IIS*, follow these instructions:

1. In the Windows Control Panel, open Administrative Tools > Internet Information Services.

2. In the tree view, expand the entry labelled local computer, then under Web Sites look for Default Web Site (unless you have virtual hosts set up, in which case, choose the site to which you want to add PHP support). Right-click on the site and choose Properties.

3. Click the ISAPI Filters tab, and click Add…. In the Filter Name field, type `PHP`, and in the Executable field, browse for the file called `php5isapi.dll` in the PHP folder. Click OK.

PHP 4.3

For PHP 4.3, the file is called `php4isapi.dll`, and is located in the `sapi` subfolder of your PHP folder.

Can't click OK?

In older versions of Windows, the OK button may remain disabled even after you have used the Browse… button to fill in the Executable field. Simply make a small change to the value of the field using the keyboard and then reverse it to enable the button.

4. Click the Home Directory tab, and click the Configuration… button. On the Mappings tab, click Add. Again choose your `php5isapi.dll` file as the executable (note that the file type filter in the dialog is set to show `.exe` files only by default) and type `.php` in the extension box (including the `.`). Leave everything else unchanged and click OK. If you want your Web server to treat other file extensions as PHP files (`.php3`, `.php4`, and `.phtml` are common choices), repeat this step for each extension. Click OK to close the Application Configuration window.

5. Click the Documents tab, and click the Add... button. Type `index.php` as the Default Document Name and click OK. This will ensure that a file called `index.php` will be displayed as the default document in a given folder on your site. You may also want to add entries for `index.php3` and `index.phtml`.

6. Click OK to close the Web Site Properties window. Close the Internet Information Services window.

7. Again, in the Control Panel under Administrative Tools, open Services. Look for the World Wide Web Publishing service near the bottom of the list. Right-click on it and choose Restart to restart IIS with the new configuration options. Close the Services window.

8. You're done! PHP is installed!

If *you don't have IIS*, you'll first need to install some other Web server. For our purposes, I'll assume you have downloaded and installed Apache server from http://httpd.apache.org/; however, PHP can also be installed on Sambar Server[5], OmniHTTPD[6], and others. I recommend Apache 1.3 for now, but if you want to use Apache 2.0, be sure to read the following sidebar.

[5] http://www.sambar.com/
[6] http://www.omnicron.ca/httpd/

PHP and Apache 2.0 in Windows

As of this writing, the PHP team continues to insist that support for PHP on Apache 2.0 is *experimental only*. There are a number of bugs that arise within PHP when it is run on an Apache 2.0 server and, on Windows especially, installation can be problematic. That said, many people (myself included!) are running PHP on Apache 2.0 quite successfully, and the bugs that do exist probably won't affect you if you're just setting up a low-traffic testing server.

The instructions below apply to both Apache 1.3 and Apache 2.0; however, it is possible that after configuring Apache 2.0 to use PHP, the server will fail to start. It is also possible that it will start, but that it will fail to process PHP scripts. In both cases, an error message should appear when you start Apache and/or in the Apache error log file.

This problem is caused by the fact that Apache 2.0 is a server still very much under development. With each minor release they put out, they tend to break compatibility with all server plug-in modules (such as PHP) that were compiled to work with the previous version. On Linux, this isn't such a big deal because people tend to compile PHP for themselves, so they simply recompile PHP at the same time they're compiling the new release of Apache and PHP adapts accordingly. Unfortunately, on Windows, where people are used to simply downloading precompiled files, the situation is different.

The `php4apache2.dll` file that is distributed with PHP will only work on versions of Apache 2.0 up to the one that was current at the time that version of PHP was released. So if you run into problems, the version of PHP you're using is probably older than the version of Apache you're using. This problem can often be fixed by downloading the very latest version of PHP; however, every time a new release of Apache 2.0 comes out, the current release of PHP will be incompatible until they get around to updating it.

Should you ever install a later version of Apache and break compatibility with the latest PHP build, you should be able to download a 'work-in-progress' version of PHP and grab only the files you need (those responsible for the PHP-Apache interface). Information about doing this can be found in the PHP bug database[7].

Once you've downloaded and installed Apache according to the instructions included with it, open http://localhost/ in your Web browser, to make sure it works properly. If you don't see a Web page explaining that Apache was successfully installed, then either you haven't yet run Apache, or your installation is faulty. Check the documentation and make sure Apache is running properly before you install PHP.

If you've made sure Apache is up and running, you can add PHP support:

[7] http://bugs.php.net/bug.php?id=17826

1. On your Start Menu, choose Programs > Apache HTTP Server > Configure Apache Server > Edit the Apache httpd.conf Configuration File. This will open the `httpd.conf` file (choose Notepad if you don't have a text editor configured to edit `.conf` files).

2. All of the options in this long and intimidating configuration file should have been set up correctly by the Apache install program. All you need to do is add the following lines to the very bottom of the file:

```
LoadModule php5_module c:/php/php5apache.dll
AddModule mod_php5.c
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

   Make sure the `LoadModule` line points to the appropriate file in the PHP installation directory on your system, and note the use of forward slashes (`/`) instead of backslashes (`\`).

Apache 2.0

If you're using Apache 2.0 or later, the `LoadModule` line needs to point to `php5apache2.dll` instead of `php5apache.dll`, and you must remove the `AddModule` line entirely.

PHP 4.3

For PHP 4.3, the file in the `LoadModule` line is called `php4apache.dll` (`php4apache2.dll` for Apache 2.0) and is located in the `sapi` subfolder of your PHP folder.

3. Next, look for the line that begins with `DirectoryIndex`. This line tells Apache which file names to use when it looks for the default page for a given directory. You'll see the usual `index.html` and so forth, but you need to add `index.php` to that list if it's not there already:

```
DirectoryIndex index.html ... index.php
```

4. Save your changes and close Notepad.

5. Restart Apache by restarting the Apache service in Control Panel > Administrative Tools > Services. If all is well, Apache will start up again without complaint.

6. You're done! PHP is installed!

With MySQL and PHP installed, you're ready to proceed to the section called "Post-Installation Setup Tasks".

Linux Installation

This section covers the procedure for installing PHP and MySQL under most current distributions of Linux. These instructions were tested under Fedora Core 2; however, they should work on other distributions such as Debian, SUSE, and Mandrake without much trouble. The steps involved will be very similar, if not identical.

As a user of one of the handful of Linux distributions available, you may be tempted to download and install **packaged distributions** of PHP and MySQL. Debian users will be used to installing software using the `apt-get` utility, while distributions like Fedora Core tend to rely on RPM packages. These prepackaged versions of software are really easy to install; unfortunately, they also limit the software configuration options available to you. If you already have MySQL and PHP installed in packaged form, feel free to proceed with those versions, and skip forward to the section called "Post-Installation Setup Tasks". If you encounter any problems, you can always return here to uninstall the packaged versions and reinstall PHP and MySQL by hand.

This section will assume that you have the Apache Web server installed on your machine already. If you don't, chances are that your distribution offers an easy way to install it (I have no objection to your using the packaged distributions of Apache). I recommend Apache 1.3 over Apache 2.0, as support for Apache 2.0 in PHP is still experimental, but I'll provide instructions for both versions here.

Building Apache yourself

If you want to compile and install Apache by hand, the necessary downloads and ample installation instructions may be found at the Apache Website[9]. To support the PHP installation instructions provided below, you will have to build Apache with shared module support. When you configure your copy of Apache prior to compiling it, make sure you include the `--enable-module=so` option.

[9] http://httpd.apache.org/

Removing Packaged Software

Since many Linux distributions will automatically install PHP and MySQL for you, your first step should be to remove any old packaged versions of PHP and MySQL from your system. If one exists, use your distribution's graphical software manager to remove all packages with `php` or `mysql` in their names.

If your distribution doesn't have a graphical software manager, or if you didn't install a graphical user interface for your server, you can remove these packages from the command prompt. You'll need to be logged in as the `root` user to issue the commands to do this. Note that in the following commands, `shell#` represents the shell prompt, and shouldn't be typed in.

In Fedora Core, RedHat, or Mandrake, you can use the `rpm` command-line utility:

```
shell#rpm -e mysql
shell#rpm -e php
```

In Debian, you can use `apt-get` to remove the relevant packages:

```
shell#apt-get remove mysql-server
shell#apt-get remove mysql-client
shell#apt-get remove php4
shell#apt-get remove php5
```

If any of these commands tell you that the package in question is not installed, don't worry about it unless you know for a fact that it is. In such cases, it will be necessary for you to remove the offending item by hand. Seek help from an experienced user if you don't know how.

If the command(s) for removing PHP completed successfully (i.e. no error message was displayed), then you have just removed PHP from your Web server, and you should check that you haven't broken it in the process. To make sure Apache is still in working order, you should restart it without the PHP plug-in:

```
shell#apachectl graceful
```

If Apache fails to start up, you'll need to have a look through its configuration file, which is usually called `httpd.conf` and may be found in `/etc/apache` or `/etc/httpd`. Look for leftover commands that may be trying to load the PHP plug-in that you have just removed from the system. The Apache error log files may be of assistance in tracking these down if you can't find them. When you're finished, try restarting Apache again.

With everything neat and tidy, you're ready to download and install MySQL and PHP.

Installing MySQL

MySQL is freely available for Linux from http://dev.mysql.com/downloads/. Download the recommended stable release (4.0 as of this writing). You should grab the Standard version under Linux (x86, libc6) in the Linux downloads section.

Once you've downloaded the program (it was about 15MB as of this writing), you should make sure you're logged in as root before proceeding with the installation, unless you want to install MySQL only in your own home directory. To begin, move to `/usr/local` (unless you want to install MySQL elsewhere for some reason) and unpack the downloaded file to create the MySQL directory (replace *`version`* with the full version of your MySQL download to match the downloaded file name on your system):

```
shell#cd /usr/local
shell#tar xfz mysql-version.tar.gz
```

Next, create a symbolic link to the `mysql-`*`version`* directory with the name `mysql` to make accessing the directory easier, then enter the directory:

```
shell#ln -s mysql-version mysql
shell#cd mysql
```

While you can run the server as the root user, or even as yourself (if, for example, you installed the server in your own home directory), the best idea is to set up on the system a special user whose sole purpose is to run the MySQL server. This will remove any possibility of someone using the MySQL server as a way to break into the rest of your system. To create a special MySQL user, you'll need to log in as root and type the following commands:

```
shell#groupadd mysql
shell#useradd -g mysql mysql
```

MySQL is now installed, but before it can do anything useful, its database files need to be installed, too. In the new `mysql` directory, type the following command:

```
shell#scripts/mysql_install_db --user=mysql
```

By default, MySQL stores all database information in the `data` subdirectory of the directory to which it was installed. We want to ensure that nobody can access

that directory except our new MySQL user. Assuming you installed MySQL to the `/usr/local/mysql` directory, you can use these commands:

```
shell#cd /usr/local/mysql
shell#chown -R root .
shell#chown -R mysql data
shell#chgrp -R mysql .
```

Now everything's set for you to launch the MySQL server for the first time. From the MySQL directory, type the following command:

```
shell#bin/mysqld_safe --user=mysql &
```

safe_mysqld

Prior to MySQL 4.0, the `mysqld_safe` script was called `safe_mysqld`. If you happen to be installing an old version of MySQL, you'll have to use that file name instead.

If you see the message `mysql daemon ended`, then the MySQL server was prevented from starting. The error message should have been written to a file called *`hostname`*`.err` (where *`hostname`* is your machine's host name) in MySQL's `data` directory. You'll usually find that this happens because another MySQL server is already running on your computer.

If the MySQL server was launched without complaint, the server will run (just like your Web or FTP server) until your computer is shut down. To test that the server is running properly, type the following command:

```
shell#bin/mysqladmin -u root status
```

A little blurb with some statistics about the MySQL server should be displayed. If you receive an error message, something has gone wrong. Again, check the *`hostname`*`.err` file to see if the MySQL server output an error message while starting up. If you retrace your steps to make sure you followed the process described above, and this doesn't solve the problem, a post to the SitePoint Forums[11] will help you pin it down in no time.

If you want your MySQL server to run automatically whenever the system is running (just like your Web server probably does), you'll have to set it up to do so. In the `support-files` subdirectory of the MySQL directory, you'll find a

[11] http://www.sitepoint.com/forums/

script called `mysql.server` that can be added to your system startup routines to do this. Let me show you how.

First of all, assuming you've set up a special MySQL user to run the MySQL server, you'll need to tell the MySQL server to start as that user by default. To do this, create in your system's `/etc` directory a file called `my.cnf` that contains these two lines:

```
[mysqld]
user=mysql
```

Now, when you run `safe_mysqld` or `mysql.server` to start the MySQL server, it will launch as user `mysql` automatically. You can test this by stopping MySQL, then running `mysql.server` with the `start` argument:

```
shell#bin/mysqladmin -u root shutdown
shell#support-files/mysql.server start
```

Request the server's status using `mysqladmin` as before, to make sure it's running correctly.

All that's left to do is to set up your system to run `mysql.server` automatically at startup (to launch the server) and at shutdown (to terminate the server). This is a highly operating system-dependant task. If you're not sure how to do it, you'd be best to ask someone who is. The following commands, however, will do the trick for most versions of Linux:

```
shell#cp /usr/local/mysql/support-files/mysql.server /etc/init.d/
shell#cd /etc/rc2.d
shell#ln -s ../init.d/mysql.server S99mysql
shell#cd /etc/rc3.d
shell#ln -s ../init.d/mysql.server S99mysql
shell#cd /etc/rc5.d
shell#ln -s ../init.d/mysql.server S99mysql
shell#cd /etc/rc0.d
shell#ln -s ../init.d/mysql.server K01mysql
```

That's it! To test that this works, reboot your system and request the status of the server as before.

One final thing you might like to do for the sake of convenience is to place the MySQL client programs, which you'll use to administer your MySQL server later on, in the system path. To this end, you can place symbolic links to `mysql`, `mysqladmin`, and `mysqldump` in your `/usr/local/bin` directory:


```
shell#ln -s /usr/local/mysql/bin/mysql /usr/local/bin/mysql
shell#ln -s /usr/local/mysql/bin/mysqladmin
/usr/local/bin/mysqladmin
shell#ln -s /usr/local/mysql/bin/mysqldump
/usr/local/bin/mysqldump
```

Installing PHP

As mentioned above, PHP is not really a program in and of itself. Instead, it's a plug-in module for your Web server (probably Apache). There are actually three ways to install the PHP plug-in for Apache:

- As a CGI program that Apache runs every time it needs to process a PHP-enhanced Web page
- As an Apache module compiled right into the Apache program
- As an Apache module loaded by Apache each time it starts up

The first option is the easiest to install and set up, but it requires Apache to launch PHP as a program on your computer every time a PHP page is requested. This activity can really slow down the response time of your Web server, especially if more than one request needs to be processed at a time.

The second and third options are almost identical in terms of performance, but since you're likely to have Apache installed already, you'd probably prefer to avoid having to download, recompile, and reinstall it from scratch. For this reason, we'll use the third option.

To start, download the PHP Complete Source Code package from http://www.php.net/downloads.php. At the time of this writing, PHP 4 has become well-established as the version of choice; however, the newly released PHP 5 is gaining ground quickly. I'll be covering the installation of PHP 5.0 here, but the same steps should work just as well with PHP 4.

The file you downloaded should be called `php-`*`version`*`.tar.gz`. To begin, we'll extract the files it contains (the `shell%` prompt is included to represent that you can run these steps without being logged in as `root`):

```
shell%tar xfz php-version.tar.gz
shell%cd php-version
```

To install PHP as a loadable Apache module, you'll need the Apache `apxs` program. This comes with most versions of Apache (both versions 1.3 and 2.0), but if you're using the copy that was installed with your distribution of Linux, you may need to install the "Apache development" package to access Apache `apxs`. You should be able to install this package by the means provided by your software distribution. For example, on Debian Linux, you can use `apt-get` to install it as follows (you'll have to log in as `root` first):

```
shell#apt-get install apache-dev
```

By default, Fedora Core, RedHat, and Mandrake will install the program as `/usr/sbin/apxs`, so if you see this file, you know it's installed. If you've installed Apache by hand, it will probably be `/usr/local/apache/bin/apxs`.

For the rest of the install procedure, you'll need to be logged in as the root user so you can make changes to the Apache configuration files.

The next step is to configure the PHP installation program by telling it which options you want to enable, and where it should find the programs it needs to know about (such as Apache and MySQL). Unless you know exactly what you're doing, simply type the command like this (all on one line):

```
shell#./configure --prefix=/usr/local/php
  --with-apxs=/usr/sbin/apxs
  --with-mysql=/usr/local/mysql
  --enable-magic-quotes
```

Replace `/usr/sbin/apxs` and `/usr/local/mysql` with the location of your `apxs` program and the base directory of your MySQL installation, respectively.

Apache 2.0

If you're using Apache 2.0 or later, you need to type `--with-apxs2=...` instead of `--with-apxs=...` to enable support for Apache 2.0. As of this writing, this support is still experimental and is not recommended for production sites. As a result of the ongoing work on this front, you may need to download the latest pre-release (unstable) version of PHP to get it working with the latest release of Apache 2.0, but it's worth trying the stable release version first.

For full instructions on how to download the latest pre-release version of PHP, see http://www.php.net/anoncvs.php.

Again, check for any error messages and install any files it identifies as missing. On Mandrake 8.0, for example, it complained that the `lex` command wasn't

found. I searched for "lex" in the Mandrake package list and it came up with `flex`, which it described as a program for matching patterns of text used in many programs' build processes. Once that was installed, the configuration process went without a hitch. After you watch several screens of tests scroll by, you'll be returned to the command prompt. The following two commands will compile and then install PHP. Take a coffee break: this will take some time.

```
shell#make
shell#make install
```

Upon completion of `make install`, PHP is installed in `/usr/local/php` (unless you specified a different directory with the `--prefix` option of the `configure` script above), with one important exception—its configuration file, `php.ini`. PHP comes with two sample `php.ini` files called `php.ini-dist` and `php.ini-recommended`. Copy these files from your installation work directory to the `/usr/local/php/lib` directory, then make a copy of the `php.ini-dist` file and call it `php.ini`:

```
shell#cp php.ini* /usr/local/php/lib/
shell#cd /usr/local/php/lib
shell#cp php.ini-dist php.ini
```

You may now delete the directory from which you compiled PHP—it's no longer needed.

We'll worry about fine-tuning `php.ini` shortly. For now, we need to tweak Apache's configuration to make it more PHP-friendly. Open your Apache `httpd.conf` configuration file (usually under `/etc/apache/` or `/etc/httpd/` if you're using your Linux distribution's copy of Apache) in your favorite text editor.

Next, look for the line that begins with `DirectoryIndex`. In certain distributions, this may be in a separate file called `commonhttpd.conf`. This line tells Apache which file names to use when it looks for the default page for a given directory. You'll see the usual `index.html`, but you need to add `index.php` to the list if it's not there already:

```
DirectoryIndex index.html index.php
```

Finally, go right to the bottom of the file (again, this should go in `commonhttpd.conf` if you have such a file) and add these lines to tell Apache which file extensions should be seen as PHP files:

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
```

That should do it! Save your changes and restart your Apache server. If all things go according to plan, Apache should start up without any error messages. If you run into any trouble, the helpful folks in the SitePoint Forums[14] (myself included) will be happy to help.

Mac OS X Installation

As of version 10.2 (Jaguar), Mac OS X distinguishes itself by being the only consumer OS to install both Apache and PHP as components of every standard installation. That said, the version of PHP provided is a little out-of-date, and you'll need to install the MySQL database as well.

In this section, I'll briefly cover what's involved in setting up up-to-date versions of PHP and MySQL on Mac OS X. Before doing that, however, I'll ask you to make sure that the Apache Web server built into your Mac OS X installation is enabled.

1. Click to pull down the Apple menu.
2. Choose System Preferences from the menu.
3. Select Sharing from the System Preferences panel.
4. If the Sharing preference panel says Web Sharing Off, click the Start button to launch the Apache Web server.
5. Exit the System Preferences program.

With this procedure complete, Apache will automatically be run at startup on your system from now on. You're now ready to enhance this server by installing PHP and MySQL!

Installing MySQL

Apple maintains a fairly comprehensive guide to installing MySQL on Mac OS X on its Mac OS X Internet Developer site[15] if you want to get your hands dirty and compile MySQL yourself. It is much easier, however, to obtain the precompiled binary version directly from the MySQL Website, and follow the installation instructions in the MySQL manual. In this section, I'll attempt to

[14] http://www.sitepoint.com/forums/
[15] http://developer.apple.com/internet/macosx/osdb.html

boil down this information to the essentials to help you get started as quickly as possible.

First of all, if you happen to be running Mac OS X Server, MySQL is already installed for you. You can run `Applications/Utilities/MySQL Manager` to access it. More likely, however, you are using the client version of Mac OS X.

To install MySQL on the client version of Mac OS X, begin by going to http://dev.mysql.com/downloads/ and selecting the latest production release of MySQL (4.0 as of this writing). Scroll down to the Mac OS X downloads section, then select and download the Installer package version for your operating system. You'll have a choice of the Standard, Max, and Debug releases; choose the Standard release unless you have a special reason for choosing one of the others.

Once you've downloaded the `mysql-standard-`*`version`*`-apple-darwin`*`ver-sion`*`-powerpc.dmg` file, double-click it to mount the disk image if your browser hasn't already done this for you. Inside it, you'll find the installer in `.pkg` format, as well as a `MySQLStartupItem.pkg` file. Double-click the installer, which will guide you through the installation of MySQL.

Once MySQL is installed, you can launch the MySQL server by opening a Terminal window and typing this command:

```
shell%sudo /usr/local/mysql/bin/mysqld_safe
```

Enter the administrator password if prompted. Once MySQL is running, you can switch it to background execution by typing **Ctrl-Z** to suspend it, and typing this command:

```
shell%bg
```

You can then close the Terminal window and MySQL will continue to run as a server on your system.

Presumably, you'll want your system automatically to launch the MySQL server at startup so that you don't have to repeat the above process whenever you restart your system. To do this, simply double-click the `MySQLStartupItem.pkg` file and follow the instructions.

When you're done, you can safely drag the mounted drive for the MySQL installation package to the trash, then delete the `.dmg` file.

Installing PHP

As with MySQL, a Mac OS X version of PHP is not available from the official Website, but from a third party. Again, Apple also maintains a Web page detailing the installation procedure[17], although in this case it is somewhat out of date. A better source of information is http://www.entropy.ch/software/macosx/php/, where you can download an installer package in the form of a disk image.

The latest version of PHP available for Mac OS X 10.2 is PHP 4.3.4. More recent versions of PHP (up to 5.0.1 as of this writing) are available for Mac OS X 10.3 or later only. Select the version that is right for your system and download it.

If your browser doesn't do it for you, mount the disk image by double-clicking the `Entropy-PHP-`*`version`*`.dmg` file, then double-click the installer `.pkg` file it contains. Simply follow the instructions, and PHP will be installed on your server. That's all there is to it!

Mac OS X and Linux

Because Mac OS X is based on the BSD operating system, much of its internals work just like any other Unix-like OS (e.g. Linux). From this point forward, owners of Mac OS X servers can follow the instructions provided for Unix/Linux systems unless otherwise indicated. No separate instructions are provided for Mac OS X unless they differ from those for other Unix-like systems.

Post-Installation Setup Tasks

No matter which operating system you're running, once PHP is installed and the MySQL server is in operation, the very first thing you need to do is assign a **root password** for MySQL. MySQL allows authorized users only to view and manipulate the information stored in its databases, so you'll need to tell MySQL who is an authorized user, and who isn't. When MySQL is first installed, it's configured with a user named `root` that has access to do pretty much anything without even entering a password. Your first task should be to assign a password to the `root` user so that unauthorized users can't tamper with your databases.

[17] http://developer.apple.com/internet/macosx/php.html

Why should I bother?

It's important to realize that MySQL, just like a Web server or an FTP server, can be accessed from any computer on the same network. If you're working on a computer connected to the Internet, then, depending on your security measures, that means anyone in the world could try to connect to your MySQL server! The need to pick a hard-to-guess password should be immediately obvious!

To set a root password for MySQL, open a command prompt (or Terminal window) and type the following command in the `bin` directory of your MySQL installation:

```
mysql -u root mysql
```

This command connects you to your newly-installed MySQL server as the `root` user, and chooses the `mysql` database. After a few lines of introductory text, you should see the MySQL command prompt (`mysql>`). To assign a password to the `root` user, type the following two commands (pressing **Enter** after each one):

```
mysql>UPDATE mysql.user SET Password=PASSWORD("new password")
    ->WHERE User="root";
Query OK, 2 rows affected (0.12 sec)
Rows matched: 2  Changed: 2  Warnings: 0
mysql>FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.24 sec)
```

Be sure to replace *`new password`* with the password you want to assign to your `root` user.

With that done, disconnect from MySQL with the `quit` command:

```
mysql>quit
Bye
```

Now, to try out your new password, request that the MySQL server tell you its current status at the system command prompt:

```
mysqladmin -u root -p status
```

Enter your new password when prompted. You should see a brief message that provides information about the server and its current status. The `-u root` argument tells the program that you want to be identified as the MySQL user called `root`. The `-p` argument tells the program to prompt you for your password before

it tries to connect. The `status` argument just tells it that you're interested in viewing the system status.

If at any time you want to shut down the MySQL server, you can use the command below. Notice the same `-u root` and `-p` arguments as before:

```
mysqladmin -u root -p shutdown
```

With your MySQL database system safe from intrusion, all that's left is to configure PHP. To do this, we'll use a text file called `php.ini`. If you installed PHP under Windows, you should already have copied `php.ini` into your Windows directory. If you installed PHP under Linux using the instructions above, you should already have copied `php.ini` into the PHP `lib` folder (`/usr/local/php/lib`), or wherever you chose to put it. The Mac OS X installation program will have placed the file in `/usr/local/php/lib` for you automatically.

Open `php.ini` in your favorite text editor and have a glance through it. Most of the settings are fairly well explained, and most of the default settings are fine for our purposes. Just check to make sure that your settings match these:

```
register_globals = Off
magic_quotes_gpc = On[5]
extension_dir = the directory where you installed PHP[6]
```

Save the changes to `php.ini`, and then restart your Web server. To restart Apache under Linux (or Mac OS X), log in as `root` and type this command:

```
shell#apachectl graceful
```

You're done! Now, you just need to test to make sure everything's working (see the section called "Your First PHP Script").

[5]PHP experts may tell you that you'll achieve better performance with it set to `Off`, but that setting exposes you to hackers attempting SQL injection attacks on your Website if you are not very careful to write scripts that protect themselves from such malicious behavior. Until you fully understand PHP and the types of security issues that scripts must combat, leave this setting `On`.

[6]Usually `c:\php` on Windows, and `/usr/local/php` on Linux.

If Your Web Host Provides PHP and MySQL

If the host that provides you with Web space has already installed and set up MySQL and PHP for you, and you just want to learn how to use them, there really isn't a lot you need to do. Now would be a good time to get in touch with your host and request any information you may need to access these services.

Specifically, you'll need a user name and password to access the MySQL server they've set up for you. They'll probably also have provided an empty database for your use, which prevents you from interfering with the databases of other users who share the same MySQL server, and you'll want to know the name of your database.

There are two ways you can access the MySQL server directly. Firstly, you can use telnet or secure shell (SSH) to log in to the host. You can then use the MySQL client programs (`mysql`, `mysqladmin`, `mysqldump`) installed there to interact with the MySQL server directly. The second method is to install those client programs onto your own computer, and have them connect to your host's MySQL server. Your Web host may support one, both, or neither of these methods, so you'll need to ask.

If your host allows you to log in by telnet or SSH to do your work, you'll need a user name and password for the login, in addition to those you'll use to access the MySQL server (they can be different). Be sure to ask for both sets of information.

If they support direct access to the MySQL server, you'll want to download a program that lets you connect to, and interact with, the server. This book assumes you've downloaded from http://www.mysql.com/ a binary distribution of MySQL that includes the three client programs (`mysql`, `mysqladmin`, and `mysqldump`). Free packages are available for Windows, Linux and other operating systems. Installation basically consists of finding the three programs and putting them in a convenient place. The rest of the package, which includes the MySQL server, can be freely discarded. If you prefer a more graphical interface, download something like MySQL Control Center[20]. I'd recommend getting comfortable with the basic client programs first, though, as the commands you use with them

[20] http://www.mysql.com/products/mysqlcc/

will be similar to those you'll include in your PHP scripts to access MySQL databases.

Many less expensive Web hosts support neither telnet/SSH access, nor direct access to their MySQL servers. Instead, they normally provide a management console that allows you to browse and edit your database through your Web browser (though some actually expect you to install one yourself, which I'll cover briefly in Chapter 2). Although this is a fairly convenient and not overly restrictive solution, it doesn't help you learn. Instead, I'd recommend you install a MySQL server on your own system for experimentation, especially in the next chapter. Once you're comfortable working with your learning server, you can start using the server provided by your Web host with the Web-based management console. See the previous sections for instructions on installing MySQL under Windows, Linux, and Mac OS X.

Your First PHP Script

It would be unfair of me to help you get everything installed and not even give you a taste of what a PHP-driven Web page looks like until Chapter 3, so here's a little something to whet your appetite.

Open your favorite text or HTML editor and create a new file called `today.php`. Windows users should note that, to save a file with a `.php` extension in Notepad, you'll need to either select *All Files* as the file type, or surround the file name with quotes in the Save As dialogue; otherwise, Notepad will helpfully save the file as `today.php.txt`, which won't work (see the note earlier in this chapter for more information). Mac OS users are advised not to use TextEdit to edit `.php` files, as it saves them in Rich Text Format with an invisible `.rtf` file name extension. Learn to use the `vi` editor in a Terminal window or obtain an editor that can save `.php` files as plain text.

Whichever editor you use, type this into the file:

File: **today.php**

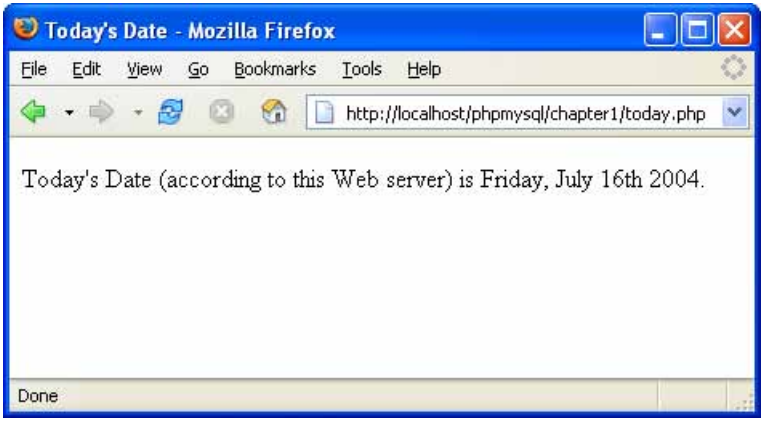
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Today's Date</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
```

```
<body>
<p>Today's Date (according to this Web server) is
<?php

echo date('l, F dS Y.');

?></p>
</body>
</html>
```

If you prefer, you can download this file, which, along with the rest of the code in this book, is contained in the code archive. See the Preface for details on how to download the archive.

Save the file, and place it on your Website as you would any regular HTML file, then view it in your browser. Note that if you view the file on your own machine, you *cannot* use the File > Open... feature of your browser, because your Web server must intervene to interpret the PHP code in the file. Instead, you must move the file into the **root document folder** of your Web server software (e.g. `C:\inetpub\wwwroot\` in IIS, or `C:\Program Files\Apache Group\Apache\htdocs\` in Apache for Windows), then load it into your browser by typing http://localhost/today.php. This process allows the Web server to run the PHP code in the file and replace it with the date before it's sent to the Web browser. Figure 1.2 shows what the output should look like.

Figure 1.2. See your first PHP script in action!



Pretty neat, huh? If you use the View Source feature in your browser, all you'll see is a regular HTML file with the date in it. The PHP code (everything between

<?php and ?> in the code above) was interpreted by the Web server and converted to normal text before it was sent to your browser. The beauty of PHP, and other server-side scripting languages, is that the Web browser doesn't have to know anything about it — the Web server does all the work!

Don't worry too much about the exact code I used in this example. Before too long you'll know it like the back of your hand.

If you don't see the date, then something is wrong with the PHP support on your Web server. Use View Source in your browser to look at the code of the page. You'll probably see the PHP code there in the page. Since the browser doesn't understand PHP, it just sees <?php … ?> as one long, invalid HTML tag, which it ignores. Make sure that PHP support has been properly installed on your Web server, either in accordance with the instructions provided in previous sections of this chapter, or by your Web host.

Summary

You should now have everything you need to install MySQL and PHP on your Web Server. If the little example above didn't work (for example, if the raw PHP code appeared instead of the date), something went wrong with your setup procedure. Drop by the SitePoint Forums[22] and we'll be glad to help you figure out the problem!

In Chapter 2, you'll learn the basics of relational databases and get started working with MySQL. If you've never even touched a database before, I promise you it'll be a real eye-opener!

[22] http://www.sitepoint.com/forums/

2 Getting Started with MySQL

In Chapter 1, we installed and set up two software programs: PHP and MySQL. In this chapter, we'll learn how to work with MySQL databases using Structured Query Language (SQL).

An Introduction to Databases

As I've already explained, PHP is a server-side scripting language that lets you insert into your Web pages instructions that your Web server software (be it Apache, IIS, or whatever) will execute before it sends those pages to browsers that request them. In a brief example, I showed how it was possible to insert the current date into a Web page every time it was requested.

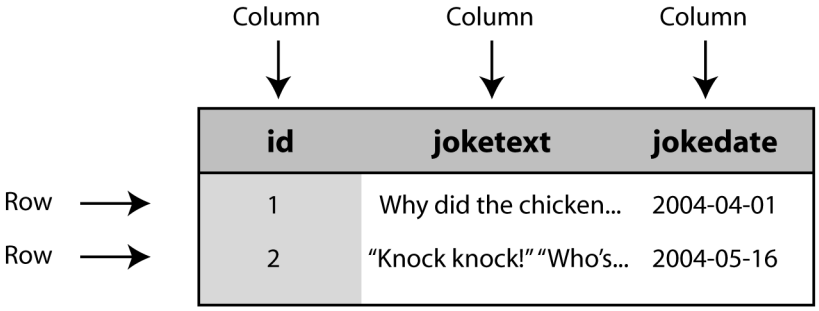
Now, that's all well and good, but things really get interesting when a database is added to the mix. A database server (in our case, MySQL) is a program that can store large amounts of information in an organized format that's easily accessible through scripting languages like PHP. For example, you could tell PHP to look in the database for a list of jokes that you'd like to appear on your Website.

In this example, the jokes would be stored entirely in the database. The advantages of this approach would be twofold. First, instead of having to write an HTML file for each of your jokes, you could write a single PHP file that was designed to

fetch any joke from the database and display it. Second, adding a joke to your Website would be a simple matter of inserting the joke into the database. The PHP code would take care of the rest, automatically displaying the new joke along with the others when it fetched the list from the database.

Let's run with this example as we look at how data is stored in a database. A database is composed of one or more **tables**, each of which contains a list of *things*. For our joke database, we'd probably start with a table called `joke` that would contain a list of jokes. Each table in a database has one or more **columns**, or **fields**. Each column holds a certain piece of information about each item in the table. In our example, our `joke` table might have one column for the text of the jokes, and another for the dates on which the jokes were added to the database. Each joke stored in this way would then be said to be a **row** in the table. These rows and columns form a table that looks like Figure 2.1.

Figure 2.1. The structure of a typical database table includes rows and columns.



Notice that, in addition to columns for the joke text (`joketext`) and the date of the joke (`jokedate`), I included a column named `id`. As a matter of good design, a database table should always provide a means by which we can identify each of its rows uniquely. Since it's possible that a single joke could be entered more than once on the same date, the `joketext` and `jokedate` columns can't be relied upon to tell all the jokes apart. The function of the `id` column, therefore, is to assign a unique number to each joke so that we have an easy way to refer to them and to keep track of which joke is which. Such database design issues will be covered in greater depth in Chapter 5.

So, to review, the above is a three-column table with two rows, or entries. Each row in the table contains three fields, one for each column in the table: the joke's

ID, its text, and the date of the joke. With this basic terminology under our belts, we're ready to get started with MySQL.

Logging On to MySQL

The standard interface for working with MySQL databases is to connect to the MySQL server software (which you set up in Chapter 1) and type commands one at a time. To make this connection to the server, you'll need the MySQL client program. If you installed the MySQL server software yourself, either under Windows or some brand of UNIX, this program will have been installed in the same location as the server program. Under Linux, for example, the program is called `mysql` and is located by default in the `/usr/local/mysql/bin` directory. Under Windows, the program is called `mysql.exe` and is located by default in the `C:\mysql\bin` directory.

If you didn't set up the MySQL server yourself (if, for example, you're working on your Web host's MySQL server), there are two ways to connect to the MySQL server. The first is to use Telnet or a Secure Shell (SSH) connection to log into your Web host's server, then run `mysql` from there. The second is to download the MySQL client software from http://www.mysql.com/ (available free for Windows and Linux), install it on your own computer, and use it to connect to the MySQL server over the Internet. Both methods work well, and your Web host may support one, the other, or both—you'll need to ask.

No shell? No direct connection? No problem!

Many Web hosts do not allow direct access to their MySQL servers over the Internet for security reasons. If your host has adopted this policy (you'll have to ask them if you're not sure), installing the MySQL client software on your own computer won't do you any good. Instead, you'll need to install a Web-based MySQL administration script onto your site. phpMyAdmin[2] is the most popular script available; indeed, many Web hosts will configure your account with a copy of phpMyAdmin.

While Web-based MySQL administration systems provide a convenient, graphical interface for working with your MySQL databases, it is still important to learn the basics of MySQL's command-line interface. The commands you use in this interface are the very same commands you'll have to include in your PHP code later in this book. I therefore recommend going back to Chapter 1 and installing MySQL on your own computer so you can complete

[2] http://www.phpmyadmin.net/

the exercises in this chapter before you get comfortable with your Web-based administration interface.

Whichever method and operating system you use, you'll end up at a command prompt, ready to run the MySQL client program and connect to your MySQL server. Here's what you should type:

```
mysql -h hostname –u username -p
```

You need to replace *hostname* with the host name or IP address of the computer on which the MySQL server is running. If the client program is run on the same computer as the server, you would use `-h localhost` or `–h 127.0.0.1`, but in this special case you can actually leave off this part of the command entirely. *username* should be your MySQL user name. If you installed the MySQL server yourself, this will just be `root`. If you're using your Web host's MySQL server, this should be the MySQL user name the host assigned you.

The `-p` argument tells the program to prompt you for your password, which it should do as soon as you enter the command above. If you set up the MySQL server yourself, this password is the root password you chose in Chapter 1. If you're using your Web host's MySQL server, this should be the MySQL password the host gave you.

If you typed everything correctly, the MySQL client program will introduce itself and dump you on the MySQL command prompt:

```
mysql>
```

The MySQL server can actually keep track of more than one database. This allows a Web host to set up a single MySQL server for use by several of its subscribers, for example. So, your next step should be to choose a database with which to work. First, let's retrieve a list of databases on the current server. Type this command (don't forget the semicolon!) and press Enter.

```
mysql>SHOW DATABASES;
```

MySQL will show you a list of the databases on the server. If you're working on a brand new server (i.e. if you installed the server yourself in Chapter 1), the list should look like this:

```
+----------+
| Database |
+----------+
| mysql    |
| test     |
```



```
+----------+
2 rows in set (0.11 sec)
```

The MySQL server uses the first database, named `mysql`, to keep track of users, their passwords, and what they're allowed to do. We'll steer clear of this database for now, though we will revisit it in Chapter 8, when we discuss MySQL Administration. The second database, named `test`, is a sample database. You can actually get rid of this database. I won't be referring to it in this book, and we'll create our own example database momentarily. Deleting something in MySQL is called "dropping" it, and the command for doing so is appropriately named:

```
mysql>DROP DATABASE test;
```

If you type this command and press Enter, MySQL will obediently delete the database, displaying "Query OK" in confirmation. Notice that you're not prompted with any kind of "Are you sure?" message. You have to be very careful to type your commands correctly in MySQL because, as this example shows, you can obliterate your entire database—along with all the information it contains—with a single command!

Before we go any further, let's learn a couple of things about the MySQL command prompt. As you may have noticed, all commands in MySQL are terminated by a semicolon (`;`). If you forget the semicolon, MySQL will think you haven't finished typing your command, and will let you continue to type on another line:

```
mysql>SHOW
    ->DATABASES;
```

MySQL shows that it's waiting for you to type more of your command by changing the prompt from `mysql>` to `->`. This handy functionality allows you to spread long commands over several lines.

If you get halfway through a command and realize that you made a mistake early on, you may want to cancel the current command entirely and start over from scratch. To do this, type `\c` and press Enter:

```
mysql>DROP DATABASE\c
mysql>
```

MySQL will ignore completely the command you had begun to type and will return to the prompt to await another command.

Finally, if at any time you want to exit the MySQL client program, just type `quit` or `exit` (either will work). This is the only command that doesn't need a semicolon, but you can use one if you want to.

```
mysql>quit
Bye
```

So, What's SQL?

The set of commands we'll use to direct MySQL throughout the rest of this book is part of a standard called **Structured Query Language**, or **SQL** (pronounced either "sequel" or "ess-cue-ell"—take your pick). Commands in SQL are also referred to as **queries** (I'll use these two terms interchangeably).

SQL is the standard language for interacting with most databases, so, even if you move from MySQL to a database like Microsoft SQL Server in the future, you'll find that most of the commands are identical. It's important that you understand the distinction between SQL and MySQL. MySQL is the database server software that you're using. SQL is the language that you use to interact with that database.

Creating a Database

Those who are working on their Web host's MySQL server are likely already to have been assigned a database with which to work. Sit tight; we'll get back to you in a moment. If you're running a MySQL server that you installed yourself, however, you'll need to create your own database. It's just as easy to create a database as it is to delete one:

```
mysql>CREATE DATABASE ijdb;
```

I chose to name the database `ijdb`, for Internet Joke Database, because that fits with the example we're using. Feel free to give the database any name you like, though. Those of you working on your Web host's MySQL server will probably have no choice in what to name your database, as it will probably already have been created for you.

Now that we have a database, we need to tell MySQL that we want to use it. Again, the command isn't difficult to remember:

```
mysql>USE ijdb;
```

You're now ready to use your database. Since a database is empty until you add some tables to it, our first order of business will be to create a table that will hold our jokes.

Creating a Table

The SQL commands we've encountered so far have been reasonably simple, but as tables are so flexible, it takes a more complicated command to create them. The basic form of the command is as follows:

```
mysql>CREATE TABLE table_name (
    ->  column_1_name column_1_type column_1_details,
    ->  column_2_name column_2_type column_2_details,
    ->  ...
    ->);
```

Let's return to our example `joke` table. Recall that it had three columns: `id` (a number), `joketext` (the text of the joke), and `jokedate` (the date on which the joke was entered). The command to create this table is as follows:

```
mysql>CREATE TABLE joke (
    ->  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->  joketext TEXT,
    ->  jokedate DATE NOT NULL
    ->);
```

It looks pretty scary, huh? Let's break it down:

- The first line is fairly simple; it says that we want to create a new table named `joke`.
- The second line says that we want a column called `id` that will contain an integer (`INT`), that is, a whole number. The rest of this line deals with special details for the column. First, this column is not allowed to be left blank (`NOT NULL`). Next, if we don't specify any value in particular when we add a new entry to the table, we want MySQL to pick a value that is one more than the highest value in the table so far (`AUTO_INCREMENT`). Finally, this column is to act as a unique identifier for the entries in the table, so all values in this column must be unique (`PRIMARY KEY`).
- The third line is super-simple; it says that we want a column called `joketext`, which will contain text (`TEXT`).

- The fourth line defines our last column, called `jokedate`, which will contain data of type `DATE`, and which cannot be left blank (`NOT NULL`).

Note that, while you're free to type your SQL commands in upper– or lowercase, a MySQL server running on a UNIX-based system will be case-sensitive when it comes to database and table names, as these correspond to directories and files in the MySQL data directory. Otherwise, MySQL is completely case-insensitive, but for one exception: table, column, and other names must be spelled exactly the same when they're used more than once in the same command.

Note also that we assigned a specific type of data to each column we created. `id` will contain integers, `joketext` will contain text, and `jokedate` will contain dates. MySQL requires you to specify in advance a data type for each column. Not only does this help keep your data organized, but it allows you to compare the values within a column in powerful ways, as we'll see later. For a complete list of supported MySQL data types, see Appendix C.

Now, if you typed the above command correctly, MySQL will respond with `Query OK`, and your first table will be created. If you made a typing mistake, MySQL will tell you there was a problem with the query you typed, and will try to indicate where it had trouble understanding what you meant.

For such a complicated command, `Query OK` is a pretty boring response. Let's have a look at your new table to make sure it was created properly. Type the following command:

```
mysql>SHOW TABLES;
```

The response should look like this:

```
+----------------+
| Tables in ijdb |
+----------------+
| joke           |
+----------------+
1 row in set
```

This is a list of all the tables in our database (which I named `ijdb` above). The list contains only one table: the `joke` table we just created. So far, everything seems fine. Let's take a closer look at the `joke` table itself:

```
mysql>DESCRIBE joke;
+----------+---------+------+-----+------------+----------------+
| Field    | Type    | Null | Key | Default    | Extra          |
```



```
+----------+---------+------+-----+------------+----------------+
| id       | int(11) |      | PRI | NULL       | auto_increment |
| joketext | text    | YES  |     | NULL       |                |
| jokedate | date    |      |     | 0000-00-00 |                |
+----------+---------+------+-----+------------+----------------+
3 rows in set
```

As you can see, there are three columns (or fields) in this table, which appear as the three rows in this table of results. The details are somewhat cryptic, but if you look at them closely, you should be able to figure out what they mean. Don't worry about it too much, though. We've got better things to do, like adding some jokes to our table!

We need to look at just one more thing before we get to that, though: deleting a table. This task is as frighteningly easy as deleting a database. In fact, the command is almost identical:

```
mysql>DROP TABLE tableName;
```

Inserting Data into a Table

Our database is created and our table is built; all that's left is to put some actual jokes into the database. The command that inserts data into a database is called, appropriately enough, `INSERT`. This command takes two basic forms:

```
mysql>INSERT INTO table_name SET
    -> columnName1 = value1,
    -> columnName2 = value2,
    -> ...
    ->;
```

```
mysql>INSERT INTO table_name
    -> (columnName1, columnName2, ...)
    -> VALUES (value1, value2, ...);
```

So, to add a joke to our table, we can use either of these commands:

```
mysql>INSERT INTO joke SET
    ->joketext = "Why did the chicken cross the road? To get to
    "> the other side!",
    ->jokedate = "2004-04-01";
```

```
mysql>INSERT INTO joke
    ->(joketext, jokedate) VALUES (
    ->"Why did the chicken cross the road? To get to the other
```

```
    "> side!",
    ->"2004-04-01"
    ->);
```

Note that in the second form of the `INSERT` command, the order in which you list the columns must match the order in which you list the values. Otherwise, the order of the columns doesn't matter, as long as you provide values for all required fields. Now that you know how to add entries to a table, let's see how we can view those entries.

Viewing Stored Data

The command we use to view data stored in database tables, `SELECT`, is the most complicated command in the SQL language. The reason for this complexity is that the chief strength of a database is its flexibility in data retrieval and presentation. At this early point in our experience with databases we need only fairly simple lists of results, so we'll just consider the simpler forms of the `SELECT` command here. This command will list everything that's stored in the `joke` table:

```
mysql>SELECT * FROM joke;
```

Read aloud, this command says "select everything from `joke`." If you try this command, your results will resemble the following:

```
+----+-------------------------------------------------
------------+------------+
| id | joketext
            | jokedate   |
+----+-------------------------------------------------
------------+------------+
|  1 | Why did the chicken cross the road? To get to the
other side! | 2004-04-01 |
+----+-------------------------------------------------
------------+------------+
1 row in set (0.05 sec)
```

The results look a little disorganized because the text in the `joketext` column is so long that the table can't fit on the screen properly. For this reason, you might want to tell MySQL to leave out the `joketext` column. The command for doing this is as follows:

```
mysql>SELECT id, jokedate FROM joke;
```

This time, instead of telling it to "select everything," we told it precisely which columns we wanted to see. The results look like this:

```
+----+------------+
| id | jokedate   |
+----+------------+
| 1  | 2004-04-01 |
+----+------------+
1 row in set (0.00 sec)
```

Not bad, but we'd like to see at least *some* of the joke text, wouldn't we? As well as being able to name specific columns that we want the `SELECT` command to show us, we can use functions to modify each column's display. One function, called `LEFT`, lets us tell MySQL to display a column's contents up to a specified maximum number of characters. For example, let's say we wanted to see only the first 20 characters of the `joketext` column. Here's the command we'd use:

```
mysql>SELECT ID, LEFT(joketext, 20), jokedate FROM joke;
+----+----------------------+------------+
| id | LEFT(joketext, 20)   | jokedate   |
+----+----------------------+------------+
| 1  | Why did the chicken  | 2004-04-01 |
+----+----------------------+------------+
1 row in set (0.05 sec)
```

See how that worked? Another useful function is `COUNT`, which lets us count the number of results returned. If, for example, we wanted to find out how many jokes were stored in our table, we could use the following command:

```
mysql>SELECT COUNT(*) FROM joke;
+----------+
| COUNT(*) |
+----------+
| 1        |
+----------+
1 row in set (0.06 sec)
```

As you can see, we have just one joke in our table and, so far, all the examples have fetched all the entries in our table. However, we can limit our results to include only those database entries that have the specific attributes we want. We set these restrictions by adding what's called a **WHERE clause** to the `SELECT` command. Consider this example:

```
mysql>SELECT COUNT(*) FROM joke WHERE jokedate >= "2004-01-01";
```

This query will count the number of jokes that have dates greater than or equal to January 1, 2004. In the case of dates, "greater than or equal to" means "on or after." Another variation on this theme lets you search for entries that contain a certain piece of text. Check out this query:

```
mysql>SELECT joketext FROM joke WHERE joketext LIKE "%chicken%";
```

The above query displays the text of all jokes that contain the word "chicken" in their `joketext` column. The `LIKE` keyword tells MySQL that the named column must match the given pattern. In this case, the pattern we've used is `"%chicken%"`. The `%` signs indicate that the word "chicken" may be preceded and/or followed by any string of text.

Additional conditions may also be combined in the `WHERE` clause to further restrict results. For example, to display knock-knock jokes from April 2004 only, we could use the following query:

```
mysql>SELECT joketext FROM joke WHERE
    ->joketext LIKE "%knock%" AND
    ->jokedate >= "2004-04-01" AND
    ->jokedate < "2004-05-01";
```

Enter a few more jokes into the table and experiment with `SELECT` statements. A good familiarity with the `SELECT` statement will come in handy later in this book.

You can do a lot with the `SELECT` statement. We'll look at some of its more advanced features later, when we need them.

Modifying Stored Data

Having entered your data into a database table, you might like to change it. Whether you want to correct a spelling mistake, or change the date attached to a joke, such alterations are made using the `UPDATE` command. This command contains elements of the `INSERT` command that set column values, and elements of the `SELECT` command that pick out entries for modification. The general form of the `UPDATE` command is as follows:

```
mysql>UPDATE table_name SET
    ->  col_name = new_value, ...
    ->WHERE conditions;
```

So, for example, if we wanted to change the date on the joke we entered above, we'd use the following command:

```
mysql>UPDATE joke SET jokedate="1994-04-01" WHERE id=1;
```

Here's where that `id` column comes in handy: it allows us to easily single out a joke for changes. The `WHERE` clause used here works just as it did in the `SELECT` command. This next command, for example, changes the date of all entries that contain the word "chicken:"

```
mysql>UPDATE joke SET jokedate="1994-04-01"
    ->WHERE joketext LIKE "%chicken%";
```

Deleting Stored Data

The deletion of entries in SQL is dangerously easy, which, if you haven't noticed yet, is a recurring theme. Here's the command syntax:

```
mysql>DELETE FROM table_name WHERE conditions;
```

To delete all chicken jokes from your table, you'd use the following query:

```
mysql>DELETE FROM joke WHERE joketext LIKE "%chicken%";
```

One thing to note is that the `WHERE` clause is actually optional. You should be very careful, however, if you leave it out, as the `DELETE` command will then apply to all entries in the table. This command will empty the `joke` table in one fell swoop:

```
mysql>DELETE FROM joke;
```

Scary, huh?

Summary

There's a lot more to the MySQL database system and the SQL language than the few basic commands we've discussed here, but these commands are by far the most commonly used. To date, we've only worked with a single table, but to realize the true power of a relational database, we'll also need to learn how to use multiple tables together to represent potentially complex relationships between database entities.

We'll cover all this and more in Chapter 5, where we'll discuss database design principles and look at some more advanced examples. For now, though, we've accomplished our objective, and you can comfortably interact with MySQL using the command line interface. In Chapter 3, the fun continues as we delve into the PHP server-side scripting language, and use it to create dynamic Web pages. If you like, you can practice with MySQL a little before you move on by creating a decent-sized `joke` table. This knowledge will come in handy in Chapter 4.

3

Getting Started with PHP

In Chapter 2, we learned how to use the MySQL database engine to store a list of jokes in a simple database (composed of a single table named `joke`). To do so, we used the MySQL command-line client to enter SQL commands (queries). In this chapter, we'll introduce the PHP server-side scripting language. In addition to the basic features we'll explore here, this language has full support for communication with MySQL databases.

Introducing PHP

As we've discussed previously, PHP is a server-side scripting language. This concept is not obvious, especially if you're used to designing pages with just HTML and JavaScript. A server-side scripting language is similar to JavaScript in that it allows you to embed little programs (scripts) into the HTML of a Web page. When executed, such scripts allow you to control what appears in the browser window more flexibly than straight HTML.

The key difference between JavaScript and PHP is simple. JavaScript is interpreted by the Web browser once the Web page that contains the script has been downloaded. Conversely, server-side scripting languages such as PHP are interpreted by the Web server before the page is even sent to the browser. And, once it's interpreted, the results of the script replace the PHP code in the Web page

itself—all the browser sees is a standard HTML file. The script is processed entirely by the server, hence the designation: server-side scripting language.

Let's look back at the `today.php` example presented in Chapter 1:

File: **today.php**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Today's Date</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>
<p>Today's Date (according to this Web server) is
<?php

echo date('l, F dS Y.');

?></p>
</body>
</html>
```

Most of this is plain HTML; however, the line between `<?php` and `?>` is written in PHP. `<?php` means "begin PHP code," and `?>` means "end PHP code." The Web server is asked to interpret everything between these two delimiters, and to convert it to regular HTML code before it sends the Web page to the requesting browser. The browser is presented with something like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Today's Date</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>
<p>Today's Date (according to this Web server) is
Sunday, May 16th 2004.</p>
</body>
</html>
```

Notice that all signs of the PHP code have disappeared. In its place, the output of the script has appeared, and it looks just like standard HTML. This example demonstrates several advantages of server-side scripting:

No browser compatibility issues
: PHP scripts are interpreted by the Web server alone, so you don't have to worry about whether the language you're using is supported by visitors' browsers.

Access to server-side resources
: In the above example, we placed the date, according to the Web server, into the Web page. If we had inserted the date using JavaScript, we would only be able to display the date according to the computer on which the Web browser was running. Now, while this isn't an especially impressive example of the exploitation of server-side resources, we could just as easily have inserted some other information that would be available only to a script running on the Web server. An example might be information stored in a MySQL database that runs on the Web server computer.

Reduced load on the client
: JavaScript can slow significantly the display of a Web page on slower computers, as the browser must run the script before it can display the Web page. With server-side scripting, this burden is passed to the Web server machine.

Basic Syntax and Commands

PHP syntax will be very familiar to anyone with an understanding of C, C++, C#, Java, JavaScript, Perl, or any other C-derived language. A PHP script consists of a series of commands, or **statements**. Each statement is an instruction that must be followed the Web server before it can proceed to the next. PHP statements, like those in the above-mentioned languages, are always terminated by a semicolon (`;`).

This is a typical PHP statement:

```
echo 'This is a <b>test</b>!';
```

This is an `echo` statement, which is used to send output to the browser. An `echo` statement simply takes the text it's given, and places it into the page's HTML code at the current location.

In this case, we have supplied a string of text to be output: `'This is a <b>test</b>!'`. Notice that the string of text contains HTML tags (`<b>` and `</b>`), which is perfectly acceptable. So, if we take this statement and put it into a complete PHP script (`echo.php` in the code archive), here's the code we get:

File: **echo.php**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Simple PHP Example</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>
<p><?php echo 'This is a <b>test</b>!'; ?></p>
</body>
</html>
```

If you place this file on your Web server, a browser that views the page will see this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Simple PHP Example</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>
<p>This is a <b>test</b>!</p>
</body>
</html>
```

Our `today.php` example contained a slightly more complex `echo` statement:

File: **today.php (excerpt)**

```
echo date('l, F dS Y.');
```

Instead of giving `echo` a simple string of text to output, this statement invokes a **built-in function** called `date` and passes *it* a string of text: `'l, F ds Y.'`. Built-in functions can be thought of as things that PHP knows how to do without our needing to spell out the details. PHP has many built-in functions that let us do everything from sending email, to working with information stored in various

types of databases. In this case, the `date` function produces a text representation of the current date, using the string it is given to determine the format.

You may wonder why we need to surround the string of text with both parentheses (`()`) and single quotes (`''`). Quotes are used to mark the beginning and end of strings of text in PHP, so their presence is fully justified. The parentheses serve two purposes. First, they indicate that `date` is a function that you want to call. Second, they mark the beginning and end of a list of **parameters** that you wish to provide, in order to tell the function what to do. In the case of the `date` function, you need to provide a string of text that describes the format in which you want the date to appear.[1] Later on, we'll look at functions that take more than one parameter, and we'll separate those parameters with commas. We'll also consider functions that take no parameters at all. These functions will still need the parentheses, though we won't type anything between them.

Variables and Operators

Variables in PHP are identical to variables in most other programming languages. For the uninitiated, a variable can be thought of as a name that's given to an imaginary box into which any value may be placed. The following statement creates a variable called `$testvariable` (all variable names in PHP begin with a dollar sign) and assigns it a value of `3`:

```
$testvariable = 3;
```

PHP is a **loosely typed** language. This means that a single variable may contain any type of data, be it a number, a string of text, or some other kind of value, and may change types over its lifetime. So the following statement, if it appears after the statement above, assigns a new value to our existing `$testvariable`. In the process, the variable changes type: where it used to contain a number, it now contains a string of text:

```
$testvariable = "Three";
```

The equals sign we used in the last two statements is called the **assignment operator**, as it is used to assign values to variables. Other operators may be used to perform various mathematical operations on values:

```
$testvariable = 1 + 1;  // Assigns a value of 2
$testvariable = 1 - 1;  // Assigns a value of 0
```

[1] A full reference is available in the online documentation for the `date` function [http://www.php.net/date].

```
$testvariable = 2 * 2;  // Assigns a value of 4
$testvariable = 2 / 2;  // Assigns a value of 1
```

Each of the lines above ends with a **comment**. Comments are a way to describe what your code is doing—they insert explanatory text into your code, and tell the PHP interpreter to ignore it. Comments begin with `//` and they finish at the end of the same line. You might be familiar with the `/* */` style of comment used in other languages—these work in PHP as well. I'll be using comments throughout the rest of this book to help explain the code I present.

Now, let's get back to the four statements above. The operators we used are called the **arithmetic operators**, and allow you to add, subtract, multiply, and divide numbers. Among others, there is an operator that sticks strings of text together, called the **concatenation operator**:

```
$testvariable = "Hi " . "there!";
                         // Assigns a value of "Hi there!"
```

Variables may be used almost anywhere that you use an actual value. Consider these examples:

```
$var1 = 'PHP';          // Assigns a value of 'PHP' to $var1
$var2 = 5;              // Assigns a value of 5 to $var2
$var3 = $var2 + 1;      // Assigns a value of 6 to $var3
$var2 = $var1;          // Assigns a value of 'PHP' to $var2
echo $var1;             // Outputs 'PHP'
echo $var2;             // Outputs 'PHP'
echo $var3;             // Outputs '6'
echo $var1 . ' rules!'; // Outputs 'PHP rules!'
echo "$var1 rules!";    // Outputs 'PHP rules!'
echo '$var1 rules!';    // Outputs '$var1 rules!'
```

Notice the last two lines in particular. You can include the name of a variable right inside a text string, and have the value inserted in its place if you surround the string with double quotes instead of single quotes. This process of converting variable names to their values is known as **variable interpolation**. However, as the last line demonstrates, a string surrounded with single quotes will not interpolate the variable names it contains.

Arrays

An **array** is a special kind of variable that contains multiple values. If you think of a variable as a box that contains a value, then an array can be thought of as a

box with compartments, where each compartment is able to store an individual value.

The simplest way to create an array in PHP is to use the built-in `array` function:

```
$myarray = array('one', 2, '3');
```

This code creates an array called `$myarray` that contains three values: `'one'`, `2`, and `'three'`. Just like an ordinary variable, each space in an array can contain any type of value. In this case, the first and third spaces contain strings, while the second contains a number.

To get at a value stored in an array, you need to know its **index**. Typically, arrays use numbers, starting with zero, as indices to point to the values they contain. That is, the first value (or element) of an array has index 0, the second has index 1, the third has index 2, and so on. In general, therefore, the index of the *n*th element of an array is *n*–1. Once you know the index of the value you're interested in, you can get that value by placing that index in square brackets after the array variable name:

```
echo $myarray[0];       // Outputs 'one'
echo $myarray[1];       // Outputs '2'
echo $myarray[2];       // Outputs '3'
```

You can also use the index in square brackets to create new elements, or assign new values to existing array elements:

```
$myarray[1] = 'two';    // Assign a new value
$myarray[3] = 'four';   // Create a new element
```

You can add elements to the end of an array using the assignment operator as usual, but leaving empty the square brackets that follow the variable name:

```
$myarray[] = 'the fifth element';
echo $myarray[4];       // Outputs 'the fifth element'
```

Array indices don't always have to be numbers; that's just the most common choice. You can also use strings as indices to create what is called an **associative array**. This type of array is called associative because it associates values with meaningful indices. In this example, we associate a date with each of three names:

```
$birthdays['Kevin'] = '1978-04-12';
$birthdays['Stephanie'] = '1980-05-16';
$birthdays['David'] = '1983-09-09';
```

The `array` function also lets you create associative arrays, if you prefer that method. Here's how we'd use it to create the `$birthdays` array:

```
$birthdays = array('Kevin' => '1978-04-12', 'Stephanie' =>
    '1980-05-16', 'David' => '1983-09-09');
```

Now, if we want to know Kevin's birthday, we look it up using the name as the index:

```
echo 'My birthday is: ' . $birthdays['Kevin'];
```

This type of array is especially important when it comes to user interaction in PHP, as we'll see in the next section. I'll demonstrate other uses of arrays throughout this book.

User Interaction and Forms

The ability to interact with users who view a Web page is essential for many applications of PHP. Veterans of JavaScript tend to think in terms of event handlers, which let you react directly to the actions of the user—for example, the movement of the cursor over a link on the page. Server-side scripting languages such as PHP have a more limited scope when it comes to user interaction. As PHP code is activated when a page is requested from the server, user interaction can occur only in a back-and-forth fashion: the user sends requests to the server, and the server replies with dynamically generated pages.

The key to creating interactivity with PHP is to understand the techniques we can use to send information about a user's interaction along with his or her request for a new Web page. PHP makes this fairly easy, as we'll now see.

The simplest method we can use to send information along with a page request uses the **URL query string**. If you've ever seen a URL in which a question mark followed the file name, you've witnessed this technique in use. Let's look at an easy example. Create a regular HTML file called `welcome1.html` (no `.php` file extension is required, since there will be no PHP code in this file) and insert this link:

File: **`welcome.html` (excerpt)**

```
<a href="welcome1.php?name=Kevin">Hi, I'm Kevin!</a>
```

This is a link to a file called `welcome1.php`, but as well as linking to the file, we're also passing a variable along with the page request. The variable is passed as part of the query string, which is the portion of the URL that follows the question

mark. The variable is called `name` and its value is `Kevin`. To restate, we have created a link that loads `welcome1.php`, and informs the PHP code contained in the file that `name` equals `Kevin`.

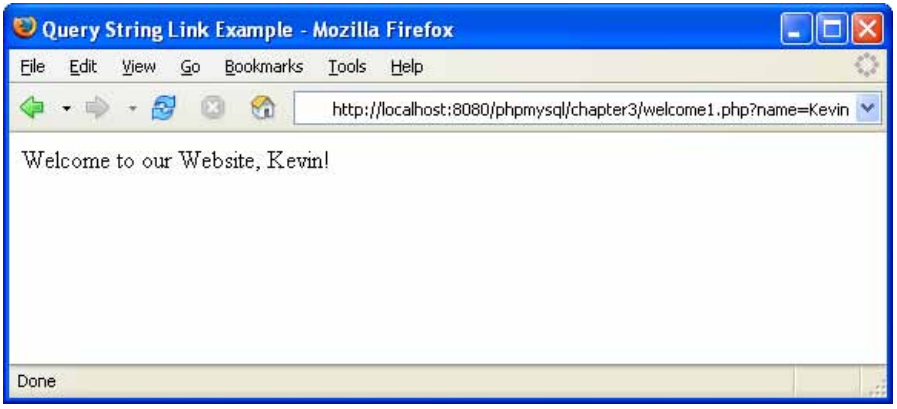
To really understand the results of this process, we need to look at `welcome1.php`. Create it as a new HTML file, but, this time, note the `.php` extension—this tells the Web server that it can expect to interpret some PHP code in the file. In the body of this new file, type the following:

File: **`welcome1.php` (excerpt)**

```
<?php
$name = $_GET['name'];
echo "Welcome to our Website, $name!";
?>
```

Now, if you use the link in the first file to load this second file, you'll see that the page says "Welcome to our Website, Kevin!" This is illustrated in Figure 3.1.

Figure 3.1. Greet users with a personalized welcome message.



PHP creates automatically an array variable called `$_GET` that contains any values passed in the query string. `$_GET` is an associative array, so the value of the `name` variable passed in the query string can be accessed as `$_GET['name']`. Our script assigns this value to an ordinary PHP variable (`$name`), then displays it as part of a text string using an `echo` statement.

`register_globals` before PHP 4.2

In versions of PHP prior to 4.2, the `register_globals` setting in `php.ini` was set to `On` by default. This setting tells PHP to create ordinary variables for all the values supplied in the request automatically. In the previous example, the `$name = $_GET['name'];` line would be completely unnecessary if the `register_globals` setting were set to `On`, since PHP would do it automatically. Although the convenience of this feature was one aspect that helped to make PHP such a popular language in the first place, novice developers could easily leave security holes in sensitive scripts with it enabled.

For a full discussion of the issues surrounding `register_globals`, see my article *Write Secure Scripts with PHP 4.2!*[2] at sitepoint.com.

You can pass more than one value in the query string. Let's look at a slightly more complex version of the same example. Change the link in the HTML file to read as follows:

File: **`welcome2.html` (excerpt)**

```
<a href="welcome2.php?firstname=Kevin&amp;lastname=Yank">Hi,
I'm Kevin Yank!</a>
```

This time, we'll pass two variables: `firstname` and `lastname`. The variables are separated in the query string by an ampersand (`&`, which is encoded as `&` as it is a special character in HTML). You can pass even more variables by separating each *`name=value`* pair from the next with an ampersand.

As before, we can use the two variable values in our `welcome2.php` file:

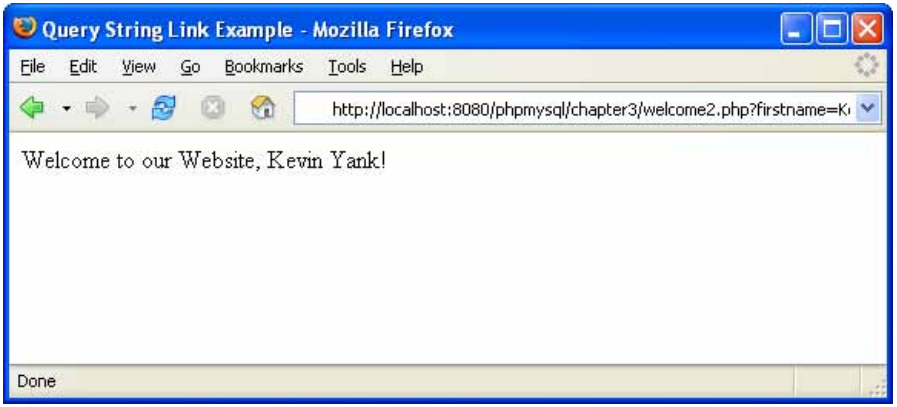
File: **`welcome2.php` (excerpt)**

```
<?php
$firstname = $_GET['firstname'];
$lastname = $_GET['lastname'];
echo "Welcome to my Website, $firstname $lastname!";
?>
```

The result is shown in Figure 3.2.

[2] http://www.sitepoint.com/article.php/758

Figure 3.2. Create an even more personalized welcome message.



This is all well and good, but we still have yet to achieve our goal of true user interaction, where the user can enter arbitrary information and have it processed by PHP. To continue with our example of a personalized welcome message, we'd like to allow the user to type his or her name and have it appear in the message. To allow the user to type in a value, we'll need to use an HTML form.

Here's the code:

File: **welcome3.html (excerpt)**

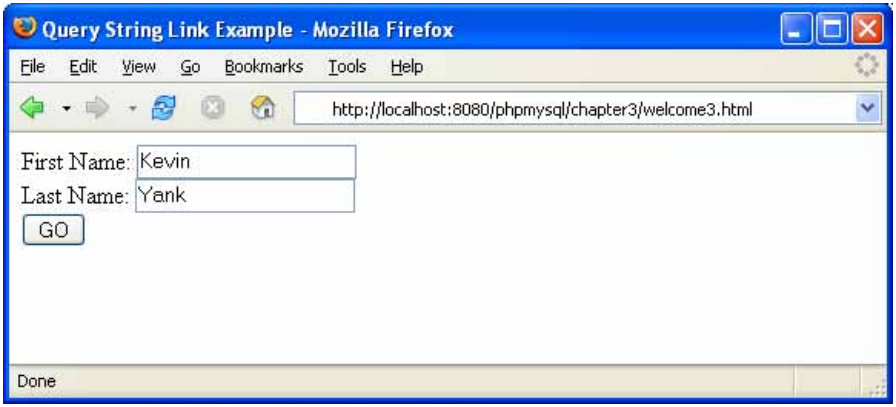
```
<form action="welcome3.php" method="get">
<label>First Name: <input type="text" name="firstname" />
  </label><br />
<label>Last Name: <input type="text" name="lastname" />
  </label><br />
<input type="submit" value="GO" />
</form>
```

Self-closing tags

Don't be alarmed at the slashes that appear in some of these tags (e.g. `<br />`). The XHTML standard for coding Web pages calls for slashes to be used in any tag that does not have a closing tag, which includes `input` and `br` tags, among others. Current browsers do not require you to use the slashes, of course, but for the sake of standards-compliance, the HTML code in this book will observe this recommendation.

The form this code produces is shown in Figure 3.3.

Figure 3.3. Make your own welcome message.



This form has the exact same effect as the second link we looked at (with `firstname=Kevin&lastname=Yank` in the query string), except that you can now enter whatever names you like. When you click the submit button (which is labelled GO), the browser will load `welcome3.php` and add the variables and their values to the query string for you automatically. It retrieves the names of the variables from the `name` attributes of the `input type="text"` tags, and obtains the values from the information the user typed into the text fields.

The `method` attribute of the `form` tag is used to tell the browser how to send the variables and their values along with the request. A value of `get` (as used above) causes them to be passed in the query string (and appear in PHP's `$_GET` array), but there is an alternative. It's not always desirable—or even technically feasible—to have the values appear in the query string. What if we included a `textarea` tag in the form, to let the user enter a large amount of text? A URL whose query string contained several paragraphs of text would be ridiculously long, and would possibly exceed the maximum length for a URL in today's browsers. The alternative is for the browser to pass the information invisibly, behind the scenes. The code for this looks exactly the same, but where we set the form method to `get` in the last example, here we set it to `post`:

File: **`welcome4.html` (excerpt)**

```
<form action="welcome4.php" method="post">
<label>First Name:
  <input type="text" name="firstname" /></label><br />
<label>Last Name:
  <input type="text" name="lastname" /></label><br />
```

```
<input type="submit" value="GO" />
</form>
```

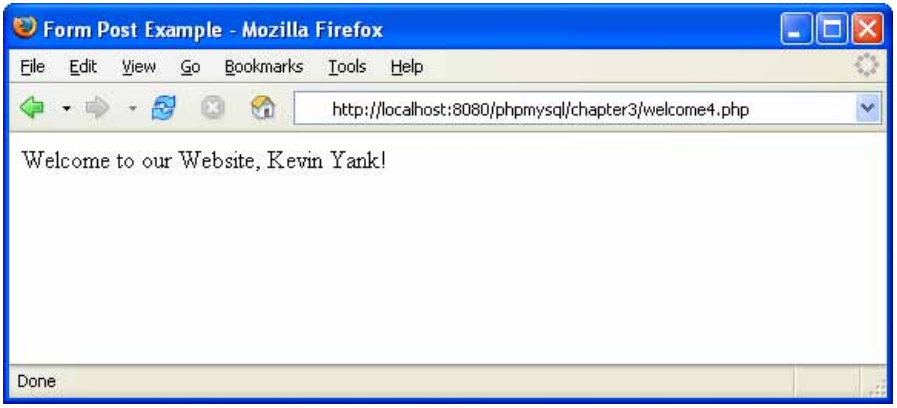
As we're no longer sending the variables as part of the query string, they no longer appear in PHP's `$_GET` array. Instead, they are placed in another array reserved especially for 'posted' form variables: `$_POST`. We must therefore modify `welcome4.php` to retrieve the values from this new array:

File: **welcome4.php (excerpt)**

```
<?php
$firstname = $_POST['firstname'];
$lastname = $_POST['lastname'];
echo "Welcome to my Website, $firstname $lastname!";
?>
```

Figure 3.4 shows what the resulting page looks like once this new form is submitted.

Figure 3.4. This personalized welcome is achieved without a query string.



The form is functionally identical to the previous one; the only difference is that the URL of the page that's loaded when the user clicks the GO button will not have a query string. On the one hand, this lets you include large values, or sensitive values (like passwords), in the data that's submitted by the form, without their appearing in the query string. On the other hand, if the user bookmarks the page that results from the form's submission, that bookmark will be useless, as it doesn't contain the submitted values. This, incidentally, is the main reason that search engines use the query string to submit search terms. If you bookmark

a search results page on Google[3] or AltaVista[4], you can use that bookmark to perform the same search again later, because the search terms are contained in the URL.

Sometimes, you want access to a variable without having to worry about whether it was sent as part of the query string or a form post. In cases like these, the special `$_REQUEST` array comes in handy. It contains all the variables that appear in both `$_GET` and `$_POST`. With this variable, we can modify our form processing script one more time so that it can receive the first and last names of the user from either source:

File: **welcome5.php (excerpt)**

```
<?php
$firstname = $_REQUEST['firstname'];
$lastname = $_REQUEST['lastname'];
echo "Welcome to my Website, $firstname $lastname!";
?>
```

That covers the basics of using forms to produce rudimentary user interaction with PHP. I'll cover more advanced issues and techniques in later examples.

Control Structures

All the examples of PHP code we've seen so far have been either simple, one-statement scripts that output a string of text to the Web page, or series of statements that were to be executed one after the other in order. If you've ever written programs in other languages (JavaScript, C, or BASIC) you already know that practical programs are rarely so simple.

PHP, just like any other programming language, provides facilities that allow us to affect the **flow of control** in a script. That is, the language contains special statements that permit you to deviate from the one-after-another execution order that has dominated our examples so far. Such statements are called **control structures**. Don't get it? Don't worry! A few examples will illustrate perfectly.

The most basic, and most often-used, control structure is the **if-else statement**. Here's what it looks like:

```
if (condition) {
  // Statement(s) to be executed if
```

[3] http://www.google.com/
[4] http://www.altavista.com/


```
  // condition is true.
} else {
  // (Optional) Statement(s) to be
  // executed if condition is false.
}
```

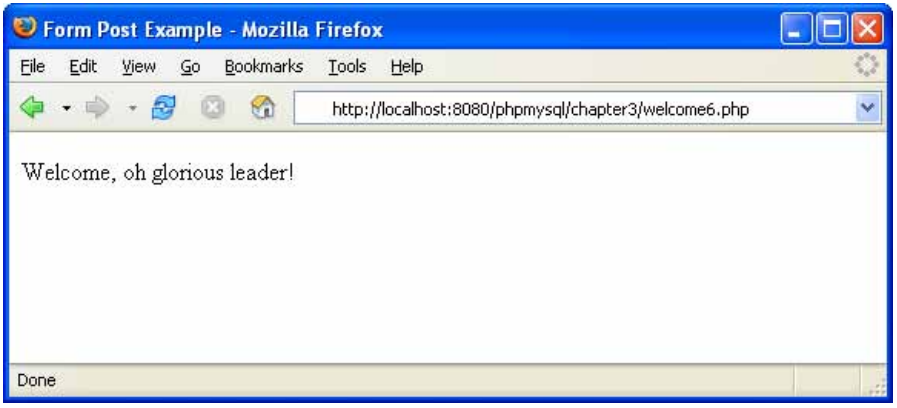
This control structure lets us tell PHP to execute one set of statements or another, depending on whether some condition is true or false. If you'll indulge my vanity for a moment, here's an example that shows a twist on the personalized welcome page example we created earlier:

File: **welcome6.php (excerpt)**

```
$name = $_REQUEST['name'];
if ($name == 'Kevin') {
  echo 'Welcome, oh glorious leader!';
} else {
  echo "Welcome to our Website, $name!";
}
```

Now, if the `name` variable passed to the page has a value of `Kevin`, a special message will be displayed. Otherwise, the normal message will be displayed and will contain the name that the user entered. The result in the former case is shown in Figure 3.5.

Figure 3.5. It's good to be the king.



As indicated in the code structure above, the **else clause** (that part of the `if-else` statement that says what to do if the condition is false) is optional. Let's say you wanted to display the special message above only if the appropriate name

was entered; otherwise, you didn't want to display any message. Here's how the code would look:

```
$name = $_REQUEST['name'];
if ($name == 'Kevin') {
  echo 'Welcome, oh glorious leader!';
}
```

The == used in the condition above is the PHP **equal-to operator** that's used to compare two values to see whether they're equal.

Double Trouble

Remember to type the double-equals, because if you were to use a single equals sign you'd be using the assignment operator discussed above. So, instead of comparing the variable to the designated value, instead, you'd assign a new value to the variable—an operation that evaluates as true as long as the new value isn't zero, false, or an empty string. This would not only cause the condition always to be true, but would also change the value in the variable you're checking, which could cause all sorts of problems later in the script.

Conditions can be more complex than a single comparison for equality. Recall that our form examples above would receive a first and last name. If we wanted to display a special message only for a particular person, we'd have to check the values of *both* names:

File: **welcome7.php (excerpt)**

```
$firstname = $_REQUEST['firstname'];
$lastname = $_REQUEST['lastname'];
if ($firstname == 'Kevin' and $lastname == 'Yank') {
  echo 'Welcome, oh glorious leader!';
} else {
  echo "Welcome to my Website, $firstname $lastname!";
}
```

This condition will be true if and only if `$firstname` has a value of `Kevin` and `$lastname` has a value of `Yank`. The word `and` in the above condition makes the whole condition true only if both of the comparisons evaluate to true. Another such operator is `or`, which makes the whole condition true if one or both of two simple conditions are true. If you're more familiar with the JavaScript or C forms of these operators (`&&` and `||` for `and` and `or` respectively), that's fine—they work in PHP as well.

Figure 3.6 shows that getting only one of the names right in this example doesn't cut the mustard.

Figure 3.6. Frankly, my dear...

We'll look at more complicated conditions as the need arises. For the time being, a general familiarity with the `if-else` statement is sufficient.

Another often-used PHP control structure is the **while loop**. Where the `if-else` statement allowed us to choose whether or not to execute a set of statements depending on some condition, the `while` loop allows us to use a condition to determine how many times we'll execute a set of statements repeatedly. Here's what a `while` loop looks like:

```
while (condition) {
  // statement(s) to execute over
  // and over as long as condition
  // remains true
}
```

The `while` loop works very similarly to an `if-else` statement without an `else` clause. The difference arises when the condition is true and the statement(s) are executed. Instead of continuing the execution with the statement that follows the closing brace (`}`), the condition is checked again. If the condition is still true, then the statement(s) are executed a second time, and a third, and will continue to be executed as long as the condition remains true. The first time the condition evaluates false (whether it's the first time it's checked, or the one-hundred-and-first), execution jumps immediately to the statement that follows the `while` loop, after the closing brace.

Loops like these come in handy whenever you're working with long lists of things (such as jokes stored in a database… hint, hint!), but for now we'll illustrate with a trivial example: counting to ten.

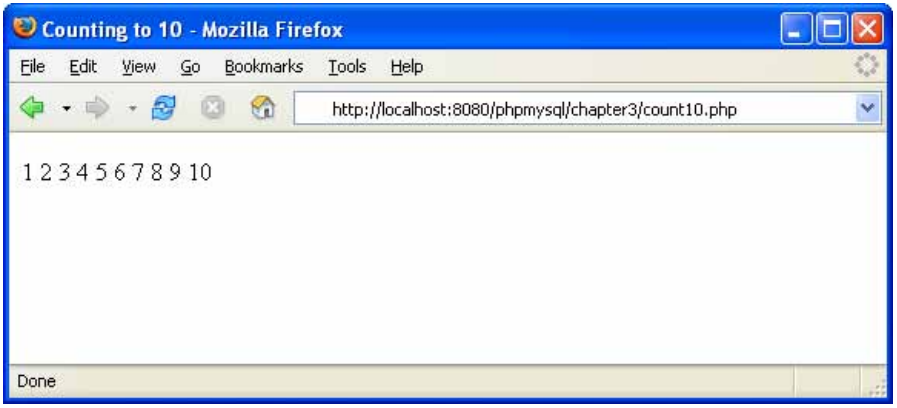
File: **count10.php (excerpt)**

```
$count = 1;
while ($count <= 10) {
  echo "$count ";
  ++$count;
}
```

It looks a bit frightening, I know, but let me talk you through it line by line. The first line creates a variable called `$count` and assigns it a value of `1`. The second line is the start of a `while` loop, the condition for which is that the value of `$count` is less than or equal (`<=`) to 10. The third and fourth lines make up the body of the `while` loop, and will be executed over and over, as long as that condition holds true. The third line simply outputs the value of `$count`, followed by a space. The fourth line adds one to the value of `$count` (`++$count` is a short cut for `$count = $count + 1`—both will work).

So here's what happens when this piece of code is executed. The first time the condition is checked, the value of `$count` is 1, so the condition is definitely true. The value of `$count` (1) is output, and `$count` is given a new value of 2. The condition is still true the second time it is checked, so the value (2) is output and a new value (3) is assigned. This process continues, outputting the values 3, 4, 5, 6, 7, 8, 9, and 10. Finally, `$count` is given a value of 11, and the condition is false, which ends the loop. The net result of the code is shown in Figure 3.7.

Figure 3.7. PHP demonstrates kindergarten-level math skills.



The condition in this example used a new operator: `<=` (**less than or equal**). Other numerical comparison operators of this type include `>=` (**greater than or equal**), `<` (**less than**), `>` (**greater than**), and `!=` (**not equal**). That last one also works when comparing text strings, by the way.

Another type of loop that is designed specifically to handle examples like that above, in which we're counting through a series of values until some condition is met, is called a **for loop**. Here's what it looks like:

```
for (initialize; condition; update) {
  // statement(s) to execute over
  // and over as long as condition
  // remains true after each update
}
```

The *`initialize`* statement is executed once at the start of the loop; the *`condition`* statement is checked each time through the loop, before the statements in the body are executed; the *`update`* statement is executed each time through the loop, but after the statements in the body.

Here's what the "counting to 10" example looks like when implemented with a `for` loop:

File: **`count10-for.php` (excerpt)**

```
for ($count = 1; $count <= 10; ++$count) {
  echo "$count ";
}
```

As you can see, the statements that initialize and increment the `$count` variable join the condition on the first line of the `for` loop. Although, at first glance, the code seems a little more difficult to read, putting all the code that deals with controlling the loop in the same place actually makes it easier to understand once you're used to the syntax. Many of the examples in this book will use `for` loops, so you'll have plenty of opportunity to practice reading them.

Multipurpose Pages

Let's say you wanted to construct your site so that it showed the visitor's name at the top of every page. With our custom welcome message example above, we're halfway there already. Here are the problems we'll need to overcome to extend the example:

- We need the name on every page of the site, not just one.

- We have no control over which page of our site users will view first.

The first problem isn't too hard to overcome. Once we have the user's name in a variable on one page, we can pass it with any request to another page by adding the name to the query string of all links:[2]

```
<a href="newpage.php?name=<?php echo urlencode($_GET['name']);?>">
A link</a>
```

Notice that we've embedded PHP code right in the middle of an HTML tag. This is perfectly legal, and will work just fine.

You're familiar with `echo` statements, but the `urlencode` function is probably new to you. This function takes special characters in the string (for example, spaces) and converts them into the special codes they need to be in order to appear in the query string. For example, if the `$name` variable had a value of `'Kevin Yank'`, then, as spaces are not allowed in the query string, the output of `urlencode` (and thus, the string output by `echo`) would be `'Kevin+Yank'`. PHP would then convert it back automatically when it created the `$_GET` variable in `newpage.php`.

Okay, so the user's name will be passed with every link in our site. Now all we need is to get that name in the first place. In our welcome message example, we had a special HTML page containing a form that prompted the user for his or her name. The problem with this (identified by the second point above) is that we couldn't—nor would we wish to—force the user to enter our Website by that page every time he or she visited our site.

The solution is to have every page of our site check to see if a name has been specified, and prompt the user for a name if necessary.[3] This means that every page of our site will either display its content, or prompt the user to enter a name, depending on whether the `$name` variable is found to have a value. If you think this is beginning to sound like a good place for an `if-else` statement, you're a quick study!

We'll refer to pages that can decide whether to display one thing or another as **multipurpose pages**. The code of a multipurpose page looks something like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

[2] If this sounds like a lot of work to you, it is. Don't worry; we'll learn much more practical methods for sharing variables between pages in Chapter 11.

[3] Again, if you're dreading the thought of adding PHP code to prompt the user for a name to every page of your site, don't fret; we'll cover a more practical way to do this later.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Multipurpose Page Outline</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>

<?php if (condition) { ?>

<!-- HTML content to display if condition is true -->

<?php } else { ?>

<!-- HTML content to display if condition is false -->

<?php } ?>

</body>
</html>
```

This code may confuse you at first, but, in fact, this is just a normal `if-else` statement with HTML code sections that depend on the condition, instead of PHP statements. This example illustrates one of the big selling points of PHP: that you can switch in and out of "PHP mode" whenever you like. If you think of `<?php` as the command to switch into "PHP mode", and `?>` as the command to go back into "normal HTML mode," the above example should make perfect sense.

There's an alternate form of the `if-else` statement that can make your code more readable in situations like this. Here's the outline for a multipurpose page using the alternate `if-else` form:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Multipurpose Page Outline</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>

<?php if (condition): ?>
```

```
<!-- HTML content to display if condition is true -->

<?php else: ?>

<!-- HTML content to display if condition is false -->

<?php endif; ?>

</body>
</html>
```

Okay, now that we have all the tools we need in hand, let's look at a sample page of our site:

File: **samplepage.php**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Sample Page</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>

<?php if (!isset($_GET['name'])): ?>

  <!-- No name has been provided, so we
       prompt the user for one.         -->

  <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="get">
  <label>Please enter your name:
    <input type="text" name="name" /></label>
  <input type="submit" value="GO" />
  </form>

<?php else: ?>

  <p>Your name: <?php echo $_GET['name']; ?></p>

  <p>This paragraph contains a
    <a href="newpage.php?name=<?php echo urlencode($_GET['name']);
    ?>">link</a> that passes the name variable on to the next
    document.</p>

<?php endif; ?>
```

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

There are two new tricks in the above code, but overall you should be fairly comfortable with the way it works. First of all, we're using a new function called `isset` in the condition. This function returns (outputs) a value of true if the variable it is given has been assigned a value (i.e. if a name has been provided in this example), and false if the variable does not exist (i.e. if a name has not yet been provided). The exclamation mark (also known as the **negation operator**, or the **not operator**), which appears before the name of the function, reverses the returned value from true to false, or vice-versa. Thus, the form is displayed when the `$_GET['name']` variable is not set.

The second new trick is the use of the variable `$_SERVER['PHP_SELF']` to specify the `action` attribute of the `<form>` tag. Like `$_GET`, `$_POST`, and `$_REQUEST`, `$_SERVER` is an array variable that is automatically created by PHP. `$_SERVER` contains a whole bunch of information supplied by your Web server. In particular, `$_SERVER['PHP_SELF']` will always be set to the URL of the current page. This gives us an easy way to create a form that, when submitted, will load the very same page, but this time with the `$name` variable specified.

Figure 3.8. Kicking butt and taking names.

If we structure all the pages on our site in this way, visitors will be prompted for their name by the first page they attempt to view, whichever page this happens to be, as shown in Figure 3.8. Once they enter their names and click GO, they'll be presented with the exact page they requested. As shown in the status bar of

Figure 3.9, the entered name is then passed in the query string of every link from that point onward, ensuring that the user is prompted only once.

Figure 3.9. We know who you are.

Summary

In this chapter, we've seen the PHP server-side scripting language in action as we've explored all the basic language features: statements, variables, operators, and control structures. The sample applications we've seen have been reasonably simple, but don't let that dissuade you. The real power of PHP is in its hundreds of built-in functions that let you access data in a MySQL database, send email, dynamically generate images, and even create Adobe Acrobat PDF files on the fly.

In Chapter 4, we'll delve into the MySQL functions in PHP, and see how to publish the joke database we created in Chapter 2 to the Web. This chapter will set the scene for the ultimate goal of this book—creating a complete content management system for your Website in PHP and MySQL.

4 Publishing MySQL Data on the Web

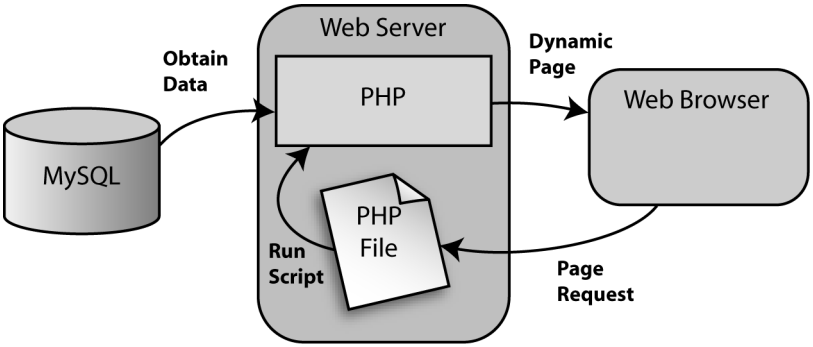
This is it—the stuff you signed up for! In this chapter, you'll learn how to take information stored in a database and display it on a Web page for all to see. So far, you've installed and learned the basics of MySQL, a relational database engine, and PHP, a server-side scripting language. Now you'll see how to use these two new tools together to create a true database-driven Website!

A Look Back at First Principles

Before we leap forward, it's worth a brief look back to remind you of our ultimate goal. We have two powerful tools at our disposal: the PHP scripting language, and the MySQL database engine. It's important to understand how these will fit together.

The whole idea of a database-driven Website is to allow the content of the site to reside in a database, and for that content to be pulled from the database dynamically to create Web pages for people to view with a regular Web browser. So, on one end of the system you have a visitor to your site who uses a Web browser to request a page, and expects to receive a standard HTML document. On the other end you have the content of your site, which sits in one or more tables in a MySQL database that understands only how to respond to SQL queries (commands).

Figure 4.1. PHP retrieves MySQL data to produce Web pages.



As shown in Figure 4.1, the PHP scripting language is the go-between that speaks both languages. It processes the page request and fetches the data from the MySQL database, then spits it out dynamically as the nicely-formatted HTML page that the browser expects. With PHP, you can write the presentation aspects of your site (the fancy graphics and page layouts) as "templates" in regular HTML. At the points at which content belongs in those templates, you use some PHP code to connect to the MySQL database and—using SQL queries just like those you used to create a table of jokes in Chapter 2—retrieve and display some content in its place.

Just so it's clear and fresh in your mind, this is what will happen when someone visits a page on your database-driven Website:

1. The visitor's Web browser requests the Web page using a standard URL.

2. The Web server software (Apache, IIS, or whatever) recognizes that the requested file is a PHP script, so the server interprets the file using its PHP plug-in before responding to the page request.

3. Certain PHP commands (which you have yet to learn) connect to the MySQL database and request the content that belongs in the Web page.

4. The MySQL database responds by sending the requested content to the PHP script.

5. The PHP script stores the content into one or more PHP variables, then uses the now-familiar `echo` statement to output the content as part of the Web page.

6. The PHP plug-in finishes up by handing a copy of the HTML it has created to the Web server.

7. The Web server sends the HTML to the Web browser as it would a plain HTML file, except that instead of coming directly from an HTML file, the page is the output provided by the PHP plug-in.

Connecting to MySQL with PHP

Before you can get content out of your MySQL database for inclusion in a Web page, you must know how to establish a connection to MySQL from inside a PHP script. Back in Chapter 2, you used a program called `mysql` that allowed you to make such a connection from the command prompt. PHP has no need of any special program, however; support for connecting to MySQL is built right into the language. The built-in function `mysql_connect` establishes the connection:

```
mysql_connect(address, username, password)
```

Here, *`address`* is the IP address or host name of the computer on which the MySQL server software is running (`'localhost'` if it's running on the same computer as the Web server software), and *`username`* and *`password`* are the same MySQL user name and password you used to connect to the MySQL server in Chapter 2.

You may remember that functions in PHP usually return (output) a value when they're called. Don't worry if this doesn't ring any bells for you—it's a detail that I glossed over when I first discussed functions in Chapter 3. In addition to doing something useful when they are called, most functions output a value; that value may be stored in a variable for later use. The `mysql_connect` function shown above, for example, returns a number that identifies the connection that has been established. Since we intend to make use of the connection, we should hold onto this value. Here's an example of how we might connect to our MySQL server.

```
$dbcnx = mysql_connect('localhost', 'root', 'mypasswd');
```

As described above, the values of the three function parameters may differ for your MySQL server. What's important to see here is that the value returned by `mysql_connect` (which we'll call a **connection identifier**) is stored in a variable named `$dbcnx`.

As the MySQL server is a completely separate piece of software, we must consider the possibility that the server may be unavailable or inaccessible due to a network

outage, or because the user name/password combination you provided is not accepted by the server. In such cases, the `mysql_connect` function doesn't return a connection identifier, as no connection is established; instead, it returns false. This allows us to react to such failures using an `if` statement:

```
$dbcnx = @mysql_connect('localhost', 'root', 'mypasswd');
if (!$dbcnx) {
  echo '<p>Unable to connect to the ' .
      'database server at this time.</p>' );
  exit();
}
```

There are three new tricks in the above code fragment. First, we have placed an `@` symbol in front of the `mysql_connect` function. Many functions, including `mysql_connect`, automatically display ugly error messages when they fail. Placing the `@` symbol (also known as the **error suppression operator**) in front of the function name tells the function to fail silently, and allows us to display our own, friendlier error message.

Next, we put an exclamation mark (`!`) in front of the `$dbcnx` variable in the condition of the `if` statement. The exclamation mark is the PHP **negation operator**, which basically flips a false value to true, or a true value to false. Thus, if the connection fails and `mysql_connect` returns false, `!$dbcnx` will evaluate to true, and cause the statements in the body of our `if` statement to be executed. Alternatively, if a connection was made, the connection identifier stored in `$dbcnx` will evaluate to true (any number other than zero is considered "true" in PHP), so `!$dbcnx` will evaluate to false, and the statements in the `if` statement will not be executed.

The last new trick is the `exit` function, which is the first example that we've encountered of a function that can be called with no parameters. When called this way, all this function does is cause PHP to stop reading the page at this point. This is a good response to a failed database connection because in most cases the page will be unable to display any useful information without that connection.

As in Chapter 2, once a connection is established, the next step is to select the database with which you want to work. Let's say we want to work with the joke database we created in Chapter 2. The database we created was called `ijdb`. Selecting that database in PHP is just a matter of another function call:

```
mysql_select_db('ijdb', $dbcnx);
```

Notice we use the `$dbcnx` variable that contains the database connection identifier to tell the function which database connection to use. This parameter is ac-

tually optional. When it's omitted, the function will automatically use the link identifier for the last connection opened. This function returns true when it's successful and false if an error occurs. Once again, it's prudent to use an `if` statement to handle errors:

```
if (!@mysql_select_db('ijdb')) {
  exit('<p>Unable to locate the joke ' .
       'database at this time.</p>');
}
```

Note that this time, instead of assigning the result of the function to a variable and then checking if the variable is true or false, I have simply used the function call itself as the condition. This may look a little strange, but it's a very commonly used shortcut. To check whether the condition is true or false, PHP executes the function and then checks its return value—exactly what we need to happen.

Another short cut I've used here is to call `exit` with a string parameter. When called with a parameter, `exit` works just like an `echo` statement, except that the script exits after the string is output. So, calling `exit` this way is equivalent to an `echo` statement followed by a call to `exit` with no parameters, which is what we used for `mysql_connect` above.

With a connection established and a database selected, we're ready to begin using the data stored in the database.

Sending SQL Queries with PHP

In Chapter 2, we connected to the MySQL database server using a program called `mysql` that allowed us to type SQL queries (commands) and view the results of those queries immediately. In PHP, a similar mechanism exists: the `mysql_query` function.

```
mysql_query(query[, connection_id])
```

Here *`query`* is a string that contains the SQL command we want to execute. As with `mysql_select_db`, the connection identifier parameter is optional.

What this function returns will depend on the type of query being sent. For most SQL commands, `mysql_query` returns either true or false to indicate success or failure respectively. Consider the following example, which attempts to create the `joke` table we created in Chapter 2:

```
$sql = 'CREATE TABLE joke (
      id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      joketext TEXT,
      jokedate DATE NOT NULL
    )';
if (@mysql_query($sql)) {
  echo '<p>joke table successfully created!</p>';
} else {
  exit('<p>Error creating joke table: ' .
      mysql_error() . '</p>');
}
```

Again, we use the `@` trick to suppress any error messages produced by `mysql_query`, and instead print out a friendlier error message of our own. The `mysql_error` function used here returns a string of text that describes the last error message that was sent by the MySQL server.

For `DELETE`, `INSERT`, and `UPDATE` queries (which serve to modify stored data), MySQL also keeps track of the number of table rows (entries) that were affected by the query. Consider the SQL command below, which we used in Chapter 2 to set the dates of all jokes that contained the word "chicken":

```
$sql = "UPDATE joke SET jokedate='1994-04-01'
    WHERE joketext LIKE '%chicken%'";
```

When we execute this query, we can use the `mysql_affected_rows` function to view the number of rows that were affected by this update:

```
if (@mysql_query($sql)) {
  echo '<p>Update affected ' . mysql_affected_rows() .
      ' rows.</p>';
} else {
  exit('<p>Error performing update: ' . mysql_error() .
      '</p>');
}
```

`SELECT` queries are treated a little differently, as they can retrieve a lot of data, and PHP must provide ways to handle that information.

Handling `SELECT` Result Sets

For most SQL queries, the `mysql_query` function returns either true (success) or false (failure). For `SELECT` queries, this just isn't enough. You'll recall that `SELECT` queries are used to view stored data in the database. In addition to indicating

whether the query succeeded or failed, PHP must also receive the results of the query. Thus, when it processes a `SELECT` query, `mysql_query` returns a number that identifies a **result set**, which contains a list of all the rows (entries) returned from the query. False is still returned if the query fails for any reason.

```
$result = @mysql_query('SELECT JokeText FROM Jokes');
if (!$result) {
  exit('<p>Error performing query: ' . mysql_error() .
      '</p>');
}
```

Provided that no error was encountered in processing the query, the above code will place a number into the variable `$result`. This number corresponds to a result set that contains the text of all the jokes stored in the `joke` table. As there's no practical limit on the number of jokes in the database, that result set can be pretty big.

We mentioned before that the `while` loop is a useful control structure for dealing with large amounts of data. Here's an outline of the code that will process the rows in a result set one at a time:

```
while ($row = mysql_fetch_array($result)) {
  // process the row...
}
```

The condition for the `while` loop probably doesn't resemble the conditions you're used to, so let me explain how it works. Consider the condition as a statement all by itself:

```
$row = mysql_fetch_array($result);
```

The `mysql_fetch_array` function accepts a result set number as a parameter (stored in the `$result` variable in this case), and returns the next row in the result set as an array (see Chapter 3 for a discussion of arrays). When there are no more rows in the result set, `mysql_fetch_array` instead returns false.

Now, the above statement assigns a value to the `$row` variable, but, at the same time, the whole statement itself takes on that same value. This is what lets you use the statement as a condition in the `while` loop. Since a `while` loop will keep looping until its condition evaluates to false, this loop will occur as many times as there are rows in the result set, with `$row` taking on the value of the next row each time the loop executes. All that's left to figure out is how to get the values out of the `$row` variable each time the loop runs.

Rows of a result set returned by `mysql_fetch_array` are represented as associative arrays. The indices are named after the table columns in the result set. If `$row` is a row in our result set, then `$row['joketext']` is the value in the `joketext` column of that row. So here's what our `while` loop should look like if we want to print the text of all the jokes in our database:

```
while ($row = mysql_fetch_array($result)) {
  echo '<p>' . $row['joketext'] . '</p>';
}
```

To summarize, here's the complete code of a PHP Web page that will connect to our database, fetch the text of all the jokes in the database, and display them in HTML paragraphs:

File: **jokelist.php**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Our List of Jokes</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>
<?php

// Connect to the database server
$dbcnx = @mysql_connect('localhost', 'root', 'mypasswd');
if (!$dbcnx) {
  exit('<p>Unable to connect to the ' .
      'database server at this time.</p>');
}

// Select the jokes database
if (!@mysql_select_db('ijdb')) {
  exit('<p>Unable to locate the joke ' .
      'database at this time.</p>');
}

?>
<p>Here are all the jokes in our database:</p>
<blockquote>
<?php

// Request the text of all the jokes
```

```
$result = @mysql_query('SELECT joketext FROM joke');
if (!$result) {
  exit('<p>Error performing query: ' . mysql_error() . '</p>');
}

// Display the text of each joke in a paragraph
while ($row = mysql_fetch_array($result)) {
  echo '<p>' . $row['joketext'] . '</p>';
}

?>
</blockquote>
</body>
</html>
```

Figure 4.2 shows what this page looks like once you've added a couple of jokes to the database.

Figure 4.2. All my best material—in one place!

Inserting Data into the Database

In this section, we'll see how we can use the tools at our disposal to allow site visitors to add their own jokes to the database. If you enjoy a challenge, you might want to try to figure this out on your own before you read any further. There is *little* new material in this section, but it's mostly just a sample application that incorporates everything we've learned so far.

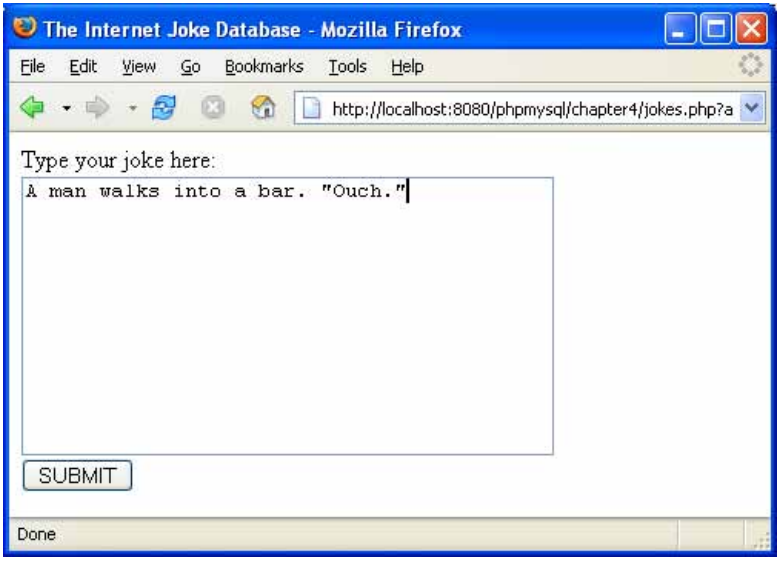
If you want to let visitors to your site type in new jokes, you'll obviously need a form. Here's the code for a form that will fit the bill:

File: **jokes.php (excerpt)**

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
<label>Type your joke here:<br />
<textarea name="joketext" rows="10" cols="40">
</textarea></label><br />
<input type="submit" value="SUBMIT" />
</form>
```

Figure 4.3 shows what this form looks like in a browser.

Figure 4.3. Another nugget of comic genius is added to the database.



As we've seen before, when submitted, this form will load the very same page (because we used the `$_SERVER['PHP_SELF']` variable for the form's `action` attribute) with one difference: a variable will be attached to the request. The variable, `joketext`, will contain the text of the joke as typed into the text area, and will appear in the `$_POST` and `$_REQUEST` arrays created by PHP.

To insert the submitted joke into the database, we use `mysql_query` to run an `INSERT` query, using the value stored in `$_POST['joketext']` to fill in the `joketext` column in the query:

File: **jokes.php (excerpt)**

```
if (isset($_POST['joketext'])) {
  $joketext = $_POST['joketext'];
  $sql = "INSERT INTO joke SET
      joketext='$joketext',
      jokedate=CURDATE()";
  if (@mysql_query($sql)) {
    echo '<p>Your joke has been added.</p>';
  } else {
    echo '<p>Error adding submitted joke: ' .
        mysql_error() . '</p>';
  }
}
```

The one new trick in this example is shown in bold. The MySQL function `CURDATE()` is used here to assign the current date as the value of the `jokedate` column. MySQL actually has dozens of these functions, but we'll introduce them only as required. For a complete MySQL function reference, refer to Appendix B.

We now have the code that will allow a user to type a joke and add it to our database. All that remains is to slot it into our existing joke viewing page in a useful fashion. As most users will only want to view jokes, we don't want to mar our page with a big, ugly form unless the user expresses an interest in adding a new joke. For this reason, our application is well suited for implementation as a multipurpose page. Here's the full code:

File: **jokes.php**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>The Internet Joke Database</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>

<?php if (isset($_GET['addjoke'])): // User wants to add a joke
?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
```

```
<label>Type your joke here:<br />
<textarea name="joketext" rows="10" cols="40">
</textarea></label><br />
<input type="submit" value="SUBMIT" />
</form>

<?php else: // Default page display

  // Connect to the database server
  $dbcnx = @mysql_connect('localhost', 'root', 'mypasswd');
  if (!$dbcnx) {
    exit('<p>Unable to connect to the ' .
        'database server at this time.</p>');
  }

  // Select the jokes database
  if (!@mysql_select_db('ijdb')) {
    exit('<p>Unable to locate the joke ' .
        'database at this time.</p>');
  }

  // If a joke has been submitted,
  // add it to the database.
  if (isset($_POST['joketext'])) {
    $joketext = $_POST['joketext'];
    $sql = "INSERT INTO joke SET
        joketext='$joketext',
        jokedate=CURDATE()";
    if (@mysql_query($sql)) {
      echo '<p>Your joke has been added.</p>';
    } else {
      echo '<p>Error adding submitted joke: ' .
          mysql_error() . '</p>';
    }
  }

  echo '<p>Here are all the jokes in our database:</p>';

  // Request the text of all the jokes
  $result = @mysql_query('SELECT joketext FROM joke');
  if (!$result) {
    exit('<p>Error performing query: ' .
        mysql_error() . '</p>');
  }

  // Display the text of each joke in a paragraph
```

```
  while ($row = mysql_fetch_array($result)) {
    echo '<p>' . $row['joketext'] . '</p>';
  }

  // When clicked, this link will load this page
  // with the joke submission form displayed.
  echo '<p><a href="' . $_SERVER['PHP_SELF'] .
      '?addjoke=1">Add a Joke!</a></p>';

endif;
?>
</body>
</html>
```

Load this up and add a new joke or two to the database via your browser. The resulting page should look like Figure 4.4.

Figure 4.4. Look, Ma! No SQL!

There we go! With a single file that contains a little PHP code, we're able to view existing jokes in, and add new jokes to, our MySQL database.

A Challenge

As "homework", see if you can figure out how to place next to each joke on the page a link labelled Delete this joke that, when clicked, will remove that joke from the database and display the updated joke list. Here are a few hints to get you started:

- You'll still be able to do it all in a single multipurpose page.
- You'll need to use the SQL `DELETE` command, which we learned about in Chapter 2.
- This is the tough one: to delete a particular joke, you'll need to be able to identify it uniquely. The `id` column in the `joke` table was designed to serve this purpose. You're going to have to pass the ID of the joke to be deleted with the request to delete a joke. The query string of the Delete this joke link is a perfect place to put this value.

If you think you have the answer, or if you'd just like to see the solution, turn the page. Good luck!

Summary

In this chapter, you learned some new PHP functions that allow you to interface with a MySQL database server. Using these functions, you built your first database-driven Website, which published the `ijdb` database online, and allowed visitors to add jokes to it.

In Chapter 5, we go back to the MySQL command line. We'll learn how to use relational database principles and advanced SQL queries to represent more complex types of information, and give our visitors credit for the jokes they add!

"Homework" Solution

Here's the solution to the "homework" challenge posed above. These changes were required to insert a Delete this joke link next to each joke:

- Previously, we passed an `addjoke` variable with our Add a Joke! link at the bottom of the page to signal that our script should display the joke entry form, instead of the usual list of jokes. In a similar fashion, we pass a `deletejoke`

variable with our Delete this joke link to indicate our desire to have a joke deleted.

- For each joke, we fetch the `id` column from the database, along with the `joketext` column, so that we know which ID is associated with each joke in the database.
- We set the value of the `$_GET['deletejoke']` variable to the ID of the joke that we're deleting. To do this, we insert the ID value fetched from the database into the HTML code for the Delete this joke link of each joke.
- Using an `if` statement, we watch to see if `$_GET['deletejoke']` is set to a particular value (through the `isset` function) when the page loads. If it is, we use the value to which it is set (the ID of the joke to be deleted) in an SQL `DELETE` statement that deletes the joke in question.

Here's the complete code. If you have any questions, don't hesitate to post them in the SitePoint Forums[1]!

File: **challenge.php**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>The Internet Joke Database</title>
<meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1" />
</head>
<body>
<?php if (isset($_GET['addjoke'])): // User wants to add a joke
?>

<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
<label>Type your joke here:<br />
<textarea name="joketext" rows="10" cols="40">
</textarea></label><br />
<input type="submit" value="SUBMIT" />
</form>

<?php else: // Default page display

  // Connect to the database server
  $dbcnx = @mysql_connect('localhost', 'root', 'mypasswd');
```

[1] http://www.sitepoint.com/forums/

```
if (!$dbcnx) {
  exit('<p>Unable to connect to the ' .
      'database server at this time.</p>');
}

// Select the jokes database
if (!@mysql_select_db('ijdb')) {
  exit('<p>Unable to locate the joke ' .
      'database at this time.</p>');
}

// If a joke has been submitted,
// add it to the database.
if (isset($_POST['joketext'])) {
  $joketext = $_POST['joketext'];
  $sql = "INSERT INTO joke SET
      joketext='$joketext',
      jokedate=CURDATE()";
  if (@mysql_query($sql)) {
    echo '<p>Your joke has been added.</p>';
  } else {
    echo '<p>Error adding submitted joke: ' .
        mysql_error() . '</p>';
  }
}

// If a joke has been deleted,
// remove it from the database.
if (isset($_GET['deletejoke'])) {
  $jokeid = $_GET['deletejoke'];
  $sql = "DELETE FROM joke
      WHERE id=$jokeid";
  if (@mysql_query($sql)) {
    echo '<p>The joke has been deleted.</p>';
  } else {
    echo '<p>Error deleting joke: ' .
        mysql_error() . '</p>';
  }
}

echo '<p> Here are all the jokes in our database: </p>';

// Request the ID and text of all the jokes
$result = @mysql_query('SELECT id, joketext FROM joke');
if (!$result) {
  exit('<p>Error performing query: ' .
```

```
        mysql_error() . '</p>');
  }

  // Display the text of each joke in a paragraph
  // with a "Delete this joke" link next to each.
  while ($row = mysql_fetch_array($result)) {
    $jokeid = $row['id'];
    $joketext = $row['joketext'];
    echo '<p>' . $joketext .
        ' <a href="' . $_SERVER['PHP_SELF'] .
        '?deletejoke=' . $jokeid . '">' .
        'Delete this joke</a></p>';
  }

  // When clicked, this link will load this page
  // with the joke submission form displayed.
  echo '<p><a href="' . $_SERVER['PHP_SELF'] .
      '?addjoke=1">Add a Joke!</a></p>';

endif;
?>
</body>
</html>
```

What's Next?

If you've enjoyed the sample chapters from *Build Your Own Database Driven Website Using PHP & MySQL, 3rd Edition*, why not order yourself a copy?

You'll learn how to speed up site re-designs drastically, how to install and administer PHP & MySQL on Windows, Linux or Mac OS X, and apply working code examples from the book to your Website instantly. You'll also gain access to our downloadable code archive—no retyping necessary!

In the next 8 chapters, you'll learn how to

- Build a working Content Management System (CMS)
- Build an ecommerce shopping cart
- Automatically send email in response to user requests
- Build a Web-based file repository or photo gallery
- Utilize sessions and cookies to track site visitors
- Structure your code to ease maintenance and reuse
- And a whole lot more…

You can't afford to be without this practical, step-by-step book!

"I like the book, and am going to recommend it to the students in my New Media class at Penn State. It is not often that I will stay up most of Saturday night reading a book on programming – but this book is both useful and clear."

— Gerry Santoro, PhD

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z