Build Your Own Database-Driven Website Using PHP & MySQL

by Kevin Yank

ISBN:0957921810

SitePoint © 2003 (275 pages)

This book is a hands-on guide to learning all the tools, principles, and techniques needed to build a fully functional database-driven Web site using PHP and MySQL from scratch.

Table of Contents

Back Cover

PHP & MySQL are the most widely used open source database and scripting technologies on the Web today. As a Web developer you can demand a lot more $$$ for your time if you can master PHP & MySQL.

Build Your Own Database Driven Website Using PHP & MySQL is a practical hands-on guide to learning all the tools, principles and techniques needed to build a fully functional database driven Website using PHP & MySQL.

This book covers everything from installing PHP & MySQL under Windows, Linux, and Mac through to building a live Web-based content management system. While this is essentially a beginners book, it also covers more advanced topics such as the storage of binary data in MySQL, and cookies and sessions in PHP. It comes complete with a set of handy reference guides for PHP & MySQL which include:

- MySQL Syntax
- MySQL Functions
- MySQL Column Types, and
- PHP Functions for working with MySQL, and more.

Build Your Own Database Driven Website Using PHP & MySQL also includes download access to all the code samples used throughout the book so you can adapt them to your own custom Web solutions.

About the Author

Kevin Yank started developing Websites in 1995, long before graduating from McGill University with a Bachelor of Computer Engineering. Today, Kevin is the Technical Business Director for SitePoint, editor of the SitePoint Tech Times and a highly respected author.

Build Your Own Database Driven Website Using PHP & MySQL

Kevin Yank

About SitePoint

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

Julian Carroll

Copyright © 2003 SitePoint Pty. Ltd.

SitePoint Pty. Ltd.
Suite 6, 50 Regent Street,
Richmond,VICAustralia3121..
0-9579218-1-0

First Edition: August 2001

Second Edition: February 2003, June 2003

All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

The author and publisher have made every effort to ensure the accuracy of the information herein. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors and SitePoint Pty. Ltd., nor its dealers or distributors will be held liable for any damages to be caused either directly or indirectly by the instructions contained in this book, or by the software or hardware products described herein.

Rather than indicating every occurrence of a trademarked name as such, this book uses the names only in an editorial fashion and to the benefit of the trademark owner with no intention of infringement of the trademark.

About the Author

Kevin Yank is the Technical Business Director for SitePoint, author of numerous well received tutorials and articles, and editor of the *SitePoint Tech Times*, an extremely popular technically-oriented newsletter for Web developers.

Before graduating from McGill University in Montreal with a Bachelor of Computer Engineering, Kevin was not only a budding Web developer himself, but also an active advisor for the Sausage Software Web Development Forums, and writer of several practical guides on advanced HTML and JavaScript.

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

Second

Dedication

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

Introduction

“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 only need 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 or 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.

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 in no time!

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.

"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.

"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.

"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 be easily 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.

"Publishing MySQL Data on the Web"

In this chapter we bring together PHP and MySQL, which you'll have seen separately in the previous two 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.

"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!

"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, and with a few modifications you should be able to build 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!

"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 to safely 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.

"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.

"Advanced SQL"

In"Relational Database Design" 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.

"Advanced PHP"

PHP lets you do a lot more than just retrieve, display, insert, and update information stored in a MySQL database. In this chapter, I'll give you a peek at some other interesting things you can do with PHP, such as server-side includes, handling file uploads, and sending email. As we'll see, these features are really useful for improving the performance and security of your database-driven site, as well as sending feedback to your visitors.

"Storing Binary Data in MySQL"

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 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, we develop a simple online file storage and viewing system and learn the ins and outs of working with binary data in MySQL.

"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.

The Book's Web Site

Located at http://www.sitepoint.com/books/, 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 the book.

Updates and Errata

No book is perfect, and even though this is a second edition, I expect that watchful readers will be able to spot at least one or two mistakes before the end of this one. 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 that needs answering, the best place to go for a quick answer is http://www.sitepointforums.com/. Not only will you find a vibrant and knowledgeable PHP community there, but you'll occasionally even find me, the author, there in my spare hours.

The SitePoint Tech Times

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. Sign up to the Tech Times (and other SitePoint newsletters) at http://www.sitepoint.com/newsletter/.

Your Feedback

If you can't find your answer through the forums, or if 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 so, without further ado, let's get started!

Chapter 1: Installation

Welcome to the Show

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 build 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 done 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. Exactly what role it plays and how it works we'll get into 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 for you. If that's the case, much of this chapter will not apply to you, and you can skip straight to "If Your Web Host Provides PHP and MySQL" to make sure your setup is ship shape.

Everything we'll discuss in this book may be done 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.

[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 'Unix'.

Windows Installation

Installing MySQL

As I mentioned above, MySQL may be downloaded free of charge. Simply proceed to http://www.mysql.com/downloads/ and choose the recommended stable release (as of this writing, it is MySQL 3.23). On the MySQL 3.23 download page, under the heading of Windows downloads, click the Download link next to the latest version of MySQL (3.23.54 as of this writing). After downloading the file (it's about 13MB 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 .NET Server. If you use any of those operating systems, find a file called *my-small.cnf* in the directory to which you just installed MySQL. Copy it to the root of your *C:* drive and rename it to *my.cnf*. See the following sidebar if you have any trouble working with *.cnf* files on your Windows system.

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-small.cnf* file will still appear as simply *my-small* 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 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 WinNT/2000/XP/.NET, click Start, Run..., and then type *regedt32.exe* to launch it, in Win9x/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, however, you should still be able to open the files in Notepad to edit them and you can rename it to *my.cnf* by renaming the SpeedDial link icon to*my* (Windows will maintain the *.cnf* extension automatically).

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

```
#basedir = d:/mysql/
#datadir = d:/mysql/data/
```

Uncomment these lines by removing the # symbol at the start, and change the paths to point to your MySQL installation directory, using slashes (`/`) instead of backslashes (`\`). For instance, I changed the lines on my system to read as follows:

```
basedir = d:/Program Files/MySQL/
datadir = d:/Program Files/MySQL/data/
```

With that change made, save the file and close Notepad. MySQL will now run on your Windows NT/2000/XP system! If you're using Windows 95/98/ME, this step is not necessary—MySQL will run just fine as-installed.

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. To make things complicated, however, there are actually several versions of the MySQL server to choose from:

- *mysqld.exe* This is the basic version of MySQL if you run Windows 95, 98, or ME. It includes support for all 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 I've generally 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/.NET 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*, but contains advanced features that support transactions.
- *mysqld-max-nt.exe* This version's similar to *mysqld-nt*, but has advanced features that support transactions.

All these versions were installed for you in the *bin* directory. If you're running on Win98x/ME I recommend sticking with *mysql-opt* for now—move to *mysqld-max* if you ever need the advanced features. On Windows NT/2000/XP/.NET, *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/.NET, but this time let's start with the procedure for Win95/98/ME. Open an MS-DOS Command Prompt[2] and 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.

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

On WinNT/2000/XP/.NET, 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 manually start MySQL 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 4.x has become well-established as the version of choice; however, some old servers still use PHP 3.x (usually because nobody has bothered to update it). I'll cover the installation of PHP 4.3.0 here, so be aware that if you're still working with PHP 3.x there may be some differences.

Download PHP for free from http://www.php.net/downloads.php. You'll want the Windows Binaries package, and be sure to grab the version that includes both the CGI binary and the server API versions if you have a choice.

In addition to PHP itself, you will need a *Web server* such as Internet Information Services (IIS), Apache, Sambar or OmniHTTPD. PHP was designed to run as a plug-in for existing Web server software. 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 .NET Server, then install IIS (if it's not already on your system): open Control Panel, Add/Remove Programs, Add/Remove Windows Components, and select IIS from the list of components. If you're not lucky enough to have IIS at your disposal[4], you can instead use a free 3rd party Web server like Apache. I'll give instructions for both options in detail.

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

1. Unzip the file you downloaded into a directory of your choice. I recommend *C:\PHP* and will refer to this directory from here onward, but feel free to choose another directory if you like.
2. Find the file called *php4ts.dll* in the PHP folder and copy it to the *System32* subfolder of your Windows folder (e.g. *C:\Windows\System32*).
3. Find the file called *php.ini-dist* in the PHP folder and copy it to your Windows folder. Once there, rename it to *php.ini*.
4. Open the *php.ini* file in your favourite 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 your PHP folder:

   ```
   extension_dir = C:\PHP
   ```

   A little 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 you want to add PHP support to). 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 *php4isapi.dll* in the *sapi* subfolder of your PHP folder (e.g.

C:\PHP\sapi\php4isapi.dll). Click OK.

4. Click the Home Directory tab, and click the Configuration... button. On the *Mappings* tab click *Add*. Again choose your *php4isapi.dll* file as the executable 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, OmniHTTPD, 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.

PHP and Apache 2.x in Windows

As of this writing, the PHP team continues to insist that support for running 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 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 Unix, this isn't such a big deal because people tend to compile PHP for themselves, so they simply re-compile 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 pre-compiled 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 just the files you need (those responsible for the PHP-Apache interface). Information about doing this can be found in the PHP bug database.

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 run Apache yet, 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:

1. On your Start Menu, choose Programs, Apache HTTP Server, Configure Apache Server, Edit Configuration. This will open the *httpd.conf* file in Notepad.
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 three lines to the very bottom of the file:

   ```
   LoadModule php4_module c:/php/sapi/php4apache.dll
   AddType application/x-httpd-php .php .php3 .phtml
   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 slashes (`/`) instead of backslashes (`\`).
   Important If you're using Apache 2.0 or later, the `LoadModule` line needs to point to *php4apache2.dll* instead of *php4apache.dll*.
3. Next, look for the line that begins with `DirectoryIndex`. This line tells Apache what 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*,*index.php3*, and *index.phtml* to that list if they're not there already:

   ```
   DirectoryIndex index.html ... index.php index.php3 index.phtml
   ```
4. Save your changes and close Notepad.
5. Restart Apache by choosing Programs, Apache HTTP Server, Control Apache Server, Restart on the Start menu (or type *NET STOP Apache && NET START Apache* at the command prompt). 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 "Post-Installation Setup Tasks".
[2]If you're unfamiliar with the workings of the Command Prompt, check out my article Kev's Command Prompt Cheat Sheet to get familiar with how it works before you proceed further.

[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 PHP can technically 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.

Linux Installation

This section covers the procedure for installing PHP and MySQL under most current distributions of Linux. These instructions were tested under the latest version Debian Linux (3.0); however, they should work on other distributions such as RedHat 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 other distributions often 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, then feel free to proceed with those versions, and skip forward to "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.

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 from the command line. 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 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
```

In RedHat or Mandrake, you can use the *rpm* command-line utility:

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

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 last command runs successfully (i.e. no message is displayed), then you did indeed have an RPM version of PHP installed, and you'll need to do one more thing to get rid of it entirely. Open your Apache configuration file (usually */etc/httpd/conf/httpd.conf*) in your favourite text editor and look for the two lines shown here. They usually appear in separate sections of the file, so don't worry if they're not together. The path of the *libphp4.so* file may also be slightly different (e.g. *extramodules* instead of just *modules*). If you can't find them, don't worry - it just means that the package utility was smart enough to remove them for you.

```
LoadModule php4_module modules/libphp4.so
AddModule mod_php4.c
```

These lines are responsible for telling Apache to load PHP as a plug-in module. Since you just uninstalled that module, you'll need to get rid of these lines to make sure Apache keeps working properly. You can comment out these lines by adding a hash (#) at the beginning of both lines.

To make sure Apache is still in working order, you should now restart it without the PHP plug-in:

```
shell#apachectl graceful
```

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://www.mysql.com/. Download the latest stable release (listed as recommended on the download page); as of this writing this is MySQL 3.23.54a, which you'll find at http://www.mysql.com/downloads/mysql-3.23.html. You should grab the Linux (x86, libc6) version under Binary packages in the Linux downloads section.

With the program downloaded (it was about 9.4MB as of this writing), you should make sure you're logged in as root before proceeding with the installation, unless you only want to install MySQL 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
```

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

```
shell#scripts/mysql_install_db
```

With MySQL installed and ready to store information, all that's left is to get the server running on your computer. 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
```

By default, MySQL stores all database information in the *data* subdirectory of the directory to which it was installed[5]. We want to make it so that nobody can access that directory except our new MySQL user. Still assuming you installed MySQL to the */usr/local/mysql* directory, you can use these commands:

```
shell#cd /usr/local/mysql
shell#chown -R mysql data
shell#chgrp -R mysql .
shell#chmod -R go-rwx data
```

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/safe_mysqld --user=mysql &
```

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 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 script called *mysql.server* that can be added to your system start-up routines to do this.

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, and then running *mysql.server* with the `start` argument:

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

Dealing with '@HOSTNAME@: command not found'

In recent versions of MySQL as of this writing, *mysql.server* may spit out an error message along the lines of '`@HOSTNAME@: command not found`'. This error is the result of a bug in the binary distribution of MySQL for Linux, and can be easily remedied. Simply open *mysql.server* in your favourite text editor and find the single occurrence of the string `@HOSTNAME@` in the file. Replace it with `/bin/hostname`, to point to the program on your server that will output the machine's host name. Save that change, shutdown MySQL again, and try starting it using *mysql.server start*. This time, it should work.

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 start-up (to launch the server) and at shutdown (to terminate the server). This is a highly operating system-dependant task. If

you're not sure of how to do it, you'd be best to ask someone who knows. 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/init.d
shell#chmod 755 mysql.server
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 convenience's sake 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/. At the time of this writing, PHP 4.x has become well-established as the version of choice; however, some old servers still use PHP 3.x (usually because nobody has bothered to update it). I'll be covering the installation of PHP 4.3.0 here, so be aware that if you still work with PHP 3.x there may be some minor differences.

The file you downloaded should be called *php-version.tar.gz*. To begin, we'll extract the files it contains (the `shell%` prompt is 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, but if you're using the copy that was installed with your distribution of Linux, you may need to install the Apache development RPM package to access Apache *apxs*. You should be able to install this package by whatever means your software distribution provides. 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, RedHat and Mandrake will install the program as */usr/sbin/apxs*, so if you see this file, you know it's installed.

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 (like 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
--enable-magic-quotes
```

Important 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
```

As of this writing, the *make* command often ends with a warning message about the function `tempnam` being dangerous (the exact wording will vary with your configuration), and is often mistaken as a sign that the process has failed. Don't worry - the warning is normal, and you can safely proceed with *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 favourite 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 what 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*, *index.php3*, and *index.phtml* to that list if they're not there already:

```
DirectoryIndex index.html ... index.php index.php3 index.phtml
```

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 .php3 .phtml
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 (myself included) will be happy to help.

[5] Until recently, it used the *var* subdirectory.

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 be automatically run at start-up 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. In this section, I'll attempt to boil down this information to the essentials to permit you to 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 downloading the Mac OS X 'pkg' format installation package from http://www.entropy.ch/software/MacOSx/mysql/. As of this writing, that site is the official source of MySQL for Mac OS X; however, MySQL AB (the developers of MySQL) have announced that they plan to take over distribution of this version beginning in February 2003, very soon after this book goes to print. You may, therefore, need to visit the download section of http://www.mysql.com/ to obtain an up-to-date installation package (and possibly updated installation instructions) by the time you read this.

Download and unpack the *mysql-`version`.pkg.tar.gz* file to obtain the *mysql-`version`.pkg* installation file, then double-click it to install MySQL.

Now, unlike most *.pkg* installations, MySQL requires some further configuration before it's ready to run on your system. Complete the following steps:

1. If you're running a version of Mac OS X older than 10.2 (Jaguar), you need to create a special user on your system that can run the server securely (this is already done for you on Mac OS 10.2 or later). To do this, open a Terminal window and type the following commands (don't type `shell%` - that's just there to represent the prompt displayed by the terminal):

```
shell%sudo niutil -create / /groups/mysql
shell%sudo niutil -createprop / /groups/mysql gid 401
shell%sudo niutil -create / /users/mysql
shell%sudo niutil -createprop / /users/mysql gid 401
shell%sudo niutil -createprop / /users/mysql uid 401
```

This creates a new user called `mysql` as well as a new user group for that user, also called `mysql`. You'll need to provide the administrator password for the first of these commands. Once the user is created, assign it a password of your choice by typing this command:

```
shell%sudo passwd mysql
```

2. Next, you need to initialize MySQL's databases. In a Terminal window, type the following commands (and provide the administrator password if you are prompted):

```
shell%cd /usr/local/mysql
shell%sudo ./scripts/mysql_install_db
```

3. Finally, you must assign permissions to the *mysql* directory to prevent unauthorized access to it by anyone except the `mysql` user:

```
shell%sudo chown -R mysql /usr/local/mysql/*
```

4. With all the configuration done, you can launch the MySQL server with this command:

```
shell%sudo /usr/local/mysql/bin/safe_mysqld --user=mysql &
```

5. Presumably, you'll want your system to automatically launch the MySQL server at start-up. You can download, extract, and run *mysql-startupitem.pkg.tar.gz* from http://www.entropy.ch/software/MacOSx/mysql/ to make this happen - that's all there is to it!

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 (although in this case, it is somewhat out of date).

Download the latest version of *libphp4.so.tar.gz* from http://www.entropy.ch/software/macosx/php/. It may be named *libphp4.so-`version`.tar.gz*; if so, rename it to `libphp4.so.tar.gz` before proceeding with the following steps:

1. Double-click the downloaded file to extract *libphp4.so* onto your desktop.
2. Open a new Terminal window and type this command to move the file to the Apache configuration directory:

```
shell%sudo mv Desktop/libphp4.so /usr/libexec/httpd/
```

Provide the administrator password if you are prompted.

3. Go to the */etc/httpd* directory and run the Apache module configuration program (*apxs*) to install or upgrade to the new module with the following commands:

```
shell%cd /etc/httpd
shell%sudo apxs -e -a -n php4 libexec/httpd/libphp4.so
```

4. Add a line telling Apache which file extensions to treat as PHP scripts to the *httpd.conf* configuration file with the following command (which you must type all on one line):

```
shell%echo 'echo "AddType application/x-httpd-php .php .php3"
>> /etc/httpd/httpd.conf' | sudo sh -s'
```

5. Finally, restart Apache with the new PHP module in place:

```
shell%sudo apachectl graceful
```

Mac OS X and Unix

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 on in the book, 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 lets only authorized users 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 any task 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.

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 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, 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 three commands (pressing Enter after each one):

```
mysql>SET PASSWORD FOR root@localhost=PASSWORD("new password");
Query OK, 0 rows affected (0.00 sec)
mysql>SET PASSWORD FOR root@"%"=PASSWORD("new password");
Query OK, 0 rows affected (0.00 sec)
mysql>FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

Be sure to replace both instances of *`new password`* with the password you want to assign to your `root` user. The first command sets the password required when connecting from the machine on which the server is running; the second sets the password for all other connections.

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

```
mysql>quit
Bye
```

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

```
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.

No*php.ini* on Mac OS X?

Mac OS X distributions of PHP don't come with a `php.ini` file by default; you can usually just let it use its own default settings. If you're happy to do this, you can go ahead and skip the rest of this section. If not, you can pinch a copy of *php.ini-dist* from the Windows Binary distribution at http://www.php.net/, rename it to *php.ini*, and place it in */usr/local/lib* (which you may have to create).

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

```
register_globals = Off
magic_quotes_gpc = On
doc_root = the root document folder of your Web server[6]
extension_dir = the directory where you installed PHP[7]
```

Save the changes to *php.ini*, and then restart your Web server. To restart Apache under Linux, 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 "Your First PHP Script").

[6]The "root document folder" of a Web server is the folder on the server computer where you must place a file to make it available in the root of your Website. On IIS servers, this is usually *c:\inetpub\wwwroot*, unless you have specifically set it to something else. On Apache servers, this is often the *htdocs* folder in the Apache installation directory unless you set it to something else yourself. Many Unix distributions use other locations when installing their packaged version of Apache; examples include */var/www* and */home/httpd*.

[7]Usually*c:\php* on Windows, and `/usr/local/php` on Unix.

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 have provided an empty database for you to use as well, 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 the 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 remote 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 MySQLGUI. I'd really recommend getting comfortable with the basic client programs first, though, as the commands you use with them will be similar to those you'll include in your PHP scripts to access MySQL databases.

Many less expensive Web hosts these days 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 "Getting Started with MySQL"). 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 to experiment with, 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 "Getting Started with PHP", so here's a little something to whet your appetite.

Open up your favourite 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. 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:

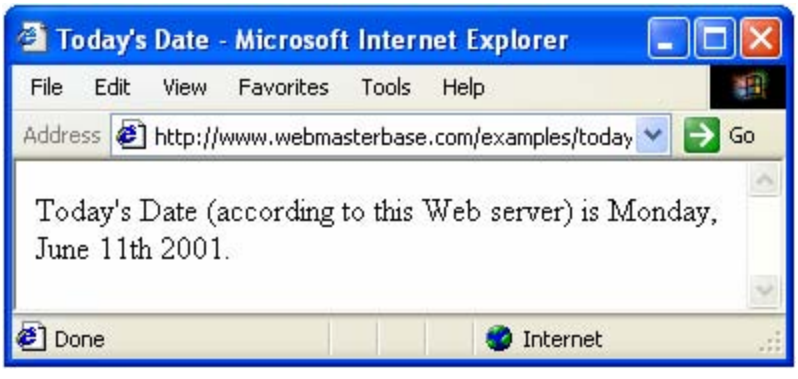
```
<html>
<head>
<title>Today's Date</title>
</head>
<body>
<p>Today's Date (according to this Web server) is
<?php

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

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

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

Save this material, 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:\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. "Output of today.php" shows what the output should look like.



Output of *today.php*

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) has been interpreted by the Web server and converted to normal text before it's 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!

And 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 in 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 get MySQL and PHP installed 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 and we'll be glad to help you figure out the problem!

In"Getting Started with MySQL", 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!

Chapter 2: Getting Started with MySQL

In"Installation", 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).

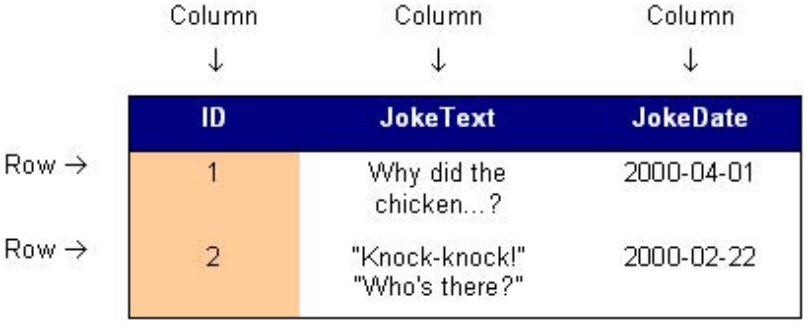
An Introduction to Databases

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 out of 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 Jokes 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 Jokes table might have columns for the text of the jokes, and the dates on which the jokes were added to the database. Each joke that we stored in this table would then be said to be a *row* in the table. These rows and columns form a table that looks like "Structure of a typical database tabletables structural overview".



Structure of a typical database table

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 way to identify uniquely each of its rows. 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 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 "Relational Database Design".

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 "Installation") 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 under some brand of UNIX, you already have this program 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'll be 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, and then run *mysql* from there. The second is to download and install the MySQL client software from http://www.mysql.com/ (available free for Windows and Linux) 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.

Warning 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 is the most popular one available; indeed, many Web hosts will configure your account with a copy of phpMyAdmin for you.

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 "Installation" and installing MySQL on your own computer so you can complete the exercises in this chapter before getting comfortable with your Web-based administration interface.

Whichever method and operating system you use, you'll end up at a command line, 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 can actually leave off the `-hhostname` part of the command instead of typing `-h localhost` or `–h 127.0.0.1`. `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 they 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 "Installation". If you're using your Web host's MySQL server, this should be the MySQL password they gave you.

If you typed everything properly, the MySQL client program will introduce itself and then dump you on the MySQL command line:

```
mysql>
```

Now, 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 this is a brand new server (i.e. if you installed this 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, called mysql, to keep track of users, their passwords, and what they're allowed to do. We'll steer clear of this database for the time being, and come back to it in "MySQL Administration" when we discuss MySQL Administration. The second database, called 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, saying "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 one single command!

Before we go any further, let's learn a couple of things about the MySQL command line. 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 you that it's waiting for you to type more of your command by changing the prompt from `mysql>` to `->`. For long commands, this can be handy, as it allows you to spread your commands out over several lines.

If you get halfway through a command and realize 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 completely ignore the command you had begun to type, and will go back to the prompt to wait for another command.

Finally, if at any time you want to exit the MySQL client program, just type `quit` or `exit` (either one 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 tell MySQL what to do for 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 called *queries* (I'll use these two terms interchangeably in this book).

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

```
mysql>CREATE DATABASE jokes;
```

I chose to name the database jokes, 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, since it will usually 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 too hard to remember:

```
mysql>USE jokes;
```

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_typecolumn_1_details,
    ->  column_2_name column_2_typecolumn_2_details,
    ->  ...
    ->);
```

Let's return to our example Jokes table. Recall that it had three columns: ID (a number), JokeText (the text of the joke), and JokeDate (the date the joke was entered). The command to create this table looks like this:

```
mysql>CREATE TABLE Jokes (
    ->  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 called Jokes.
- 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 this 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 this 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 lower case, 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 a data type for each column in advance. 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 "MySQL Column Types".

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 give you some indication of 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 jokes |
+-----------------+
| Jokes           |
+-----------------+
1 row in set
```

This is a list of all the tables in our database (which I named jokes above). The list contains only one table: the Jokes table we just created. So far everything looks good. Let's have a closer look at the Jokes table itself:

```
mysql>DESCRIBE Jokes;
+----------+---------+------+-----+------------+----------------+
| 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 we can see, there are three columns (or fields) in this table, which appear as the 3 rows in this table of results. The details are somewhat cryptic, but if you look at them closely for a while you should be able to figure out what most of them 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 our database. The command for inserting data into our database is called, appropriately enough, `INSERT`. There are two basic forms of this command:

```
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 choose from either of these commands:

```
mysql>INSERT INTO Jokes SET
    ->JokeText = "Why did the chicken cross the road? To get to
    "> the other side!",
    ->JokeDate = "2000-04-01";
```

```
mysql>INSERT INTO Jokes
   ->(JokeText, JokeDate) VALUES (
   ->"Why did the chicken cross the road? To get to the other
   "> side!",
   ->"2000-04-01"
   ->);
```

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 give 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 your 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. As, at this point in our experience with databases, we need only fairly simple lists of results, we'll just consider the simpler forms of the SELECT command. This command will list everything stored in the Jokes table:

```
mysql>SELECT * FROM Jokes;
```

Read aloud, this command says "select everything from Jokes". If you try this command, your results will resemble this:

```
+----+-------------------------------------------------
------------+------------+
| ID | JokeText
            | JokeDate   |
+----+-------------------------------------------------
------------+------------+
|  1 | Why did the chicken cross the road? To get to the
other side! | 2000-04-01 |
+----+-------------------------------------------------
------------+------------+
1 row in set (0.05 sec)
```

It looks a little disorganised because the text in the JokeText column is too long for the table to fit properly on the screen. 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 Jokes;
```

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  | 2000-04-01 |
+----+------------+
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? In addition to listing the columns that we want the SELECT command to show us, we can modify those columns with functions. One function, called LEFT, lets us tell MySQL to display up to a specified maximum number of characters when it displays a column. For example, let's say we wanted to see only the first 20 characters of the JokeText column:

```
mysql>SELECT ID, LEFT(JokeText,20), JokeDate FROM Jokes;
+----+----------------------+------------+
| ID | LEFT(JokeText,20)    | JokeDate   |
+----+----------------------+------------+
| 1  | Why did the chicken  | 2000-04-01 |
+----+----------------------+------------+
1 row in set (0.05 sec)
```

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

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

As you can see, we have just one joke in our table. So far, all our examples have fetched all the entries in the table. But if we add what's called a *WHERE clause* (for reasons that will become obvious in a moment) to a SELECT command, we can limit which entries are returned as results. Consider this example:

```
mysql>SELECT COUNT(*) FROM Jokes WHERE JokeDate >= "2000-01-01";
```

This query will count the number of jokes that have dates "greater than or equal to" January 1st, 2000. "Greater than or equal to", when dealing with dates, 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 Jokes WHERE JokeText LIKE "%chicken%";
```

This 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 here 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 2000 only, we could use the following query:

```
mysql>SELECT JokeText FROM Jokes WHERE
    ->JokeText LIKE "%knock%" AND
    ->JokeDate >= "2000-04-01" AND
    ->JokeDate < "2000-05-01";
```

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

There's a lot more you can do with the SELECT statement, but we'll save looking at some of its more advanced features for 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 of theSELECT command (that pick out entries to modify). The general form of the UPDATE command is as follows:

```
mysql>UPDATEtable_name SET
    ->  col_name = new_value,...
    ->WHEREconditions;
```

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

```
mysql>UPDATE Jokes SET JokeDate="1990-04-01" 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 here works just like it does in the SELECT command. This next command, for example, changes the date of all entries that contain the word "chicken":

```
mysql>UPDATE Jokes SET JokeDate="1990-04-01"
    ->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 conditons;
```

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

```
mysql>DELETE FROM Jokes 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 off, as the `DELETE` command will then apply to all entries in the table. This command will empty the Jokes table in one fell swoop:

```
mysql>DELETE FROM Jokes;
```

Scary, huh?

Summary

There's a lot more to the MySQL database system and the SQL language than the few basic commands we've looked at here, but these commands are by far the most commonly used. So far we've only worked with a single table. 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 "Relational Database Design", 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 "Getting Started with PHP", 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 Jokes table - this knowledge will come in handy in "Publishing MySQL Data on the Web"!

Chapter 3: Getting Started with PHP

In"Getting Started with MySQL", 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 Jokes). 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 many ways, as they both allow you to embed little programs (scripts) into the HTML of a Web page. When executed, such scripts allow you to control what will actually appear in the browser window with more flexibility than is possible using 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. Meanwhile, server-side scripting languages like 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, so 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 "Installation":

```
<html>
<head>
<title>Today's Date</title>
</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. The line between <?php and ?>, however, 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:

```
<html>
<head>
<title>Today's Date</title>
</head>
<body>
<p>Today's Date (according to this Web server) is
Wednesday, May 30th 2001.</p>
</body>
</html>
```

Notice that all signs of the PHP code have disappeared. In its place, the output of the script has appeared, and 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 and nothing else, so you don't have to worry about whether the language you're using will be supported by your 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++, Java, JavaScript, Perl, or any other C-derived language. A PHP script consists of a series of commands, or *statements*, each of which is an instruction that the Web server must follow 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 statement invokes a *built-in function* called `echo` and passes it a string of text: `This is a <b>test</b>!` Built-in functions can be thought of as things that PHP knows how to do without us having to spell out the details. PHP has a lot of built-in functions that let us do everything from sending email, to working with information that's stored in various types of databases. The `echo` function, however, simply takes the text that it's given, and places it into the HTML code of the page at the current location. Consider the following (*echo.php* in the code package):

```
<html>
<head>
<title> Simple PHP Example </title>
</head>
<body>
<p><?php echo('This is a <b>test</b>!'); ?></p>
</body>
</html>
```

If you paste this code into a file called *echo.php* and place it on your Web server, a browser that views the page will see this:

```
<html>
<head>
<title> Simple PHP Example </title>
</head>
<body>
<p>This is a <b>test</b>!</p>
</body>
</html>
```

Notice that the string of text contained HTML tags (`<b>` and `</b>`), which is perfectly acceptable.

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 a dual purpose. First, they indicate that `echo` 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 `echo` function, you need only provide the string of text that you want to appear on the page. 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, for which we'll 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 is a name 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
$testvariable = 2 * 2;   // Assigns a value of 4
$testvariable = 2 / 2;   // Assigns a value of 1
```

The lines above each end 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 `/* */` style comments in other languages-these work in PHP as well. I'll be using comments throughout the rest of this book to help explain what the code I present is doing.

Now, to 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 also 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. This process of converting variable names to their values is known in technical circles as *variable interpolation*. However, as the last line demonstrates, a string surrounded with single quotes will not interpolate variable names within the string.

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 with the built-in `array` function:

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

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 the index in square brackets following the array variable name:

```
echo($myarray[0]);       // Outputs "one"
echo($myarray[1]);       // Outputs "2"
echo($myarray[2]);       // Outputs "three"
```

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 by using the assignment operator as usual, except with empty square brackets following the variable name:

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

Array indices don't always have to be numbers; that is 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';
```

Now if we want to know Kevin's birthday, we just 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 also demonstrate other uses of arrays throughout this book.

User Interaction and Forms

For many applications of PHP, the ability to interact with users who view the Web page is essential. 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 mouse 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 with a question mark following 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:

```
<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:

```
<?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!"

PHP automatically creates an array variable called `$_GET`[1] that contains any values passed in the query string.`$_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`) and then displays it as part of a text string using the `echo` function.

`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 automatically ordinary variables for all the values supplied in the request. In the previous example, the `$name = $_GET['name'];` line is 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 of PHP that helped to make it 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!* 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 (this is *welcome2.html* in the code

archive):

```
<a href="welcome2.php?firstname=Kevin&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 (`&`). 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 *welcome.php* file (this is *welcome2.php* in the code archive):

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

This is all well and good, but we still have yet to achieve our goal of true user interaction, where the user can actually 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 actually 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 (*welcome3.html*):

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

> **Note** Don't be alarmed at the slashes that appear in some of these tags (e.g. `<br />`). The new XHTML standard for coding Web pages calls for these 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. Feel free to leave the slashes out if you prefer - I agree that they're not especially nice to look at.

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 enter whatever names you like. When you click the submit button (which has a label of "GO"), the browser will load *welcome3.php* and automatically add the variables and their values to the query string for you. It retrieves the names of the variables from the `name` attributes of the `input type="text"` tags, and it 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 that contained several paragraphs of text in the query string would be ridiculously long, and would exceed by far the maximum length of the 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` (*welcome4.html*):

```
<form action="welcome4.php" method="post">
First Name: <input type="text" name="firstname" /><br />
Last Name: <input type="text" name="lastname" /><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[2]. We must therefore modify *welcome3.php* to retrieve the values from this new array (*welcome4.php*):

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

This 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 like Google use the query string to submit search terms. If you bookmark a search results page on AltaVista, 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[3] array comes in handy. It contains all the variables that appear in both $_GET and $_POST. With this variable, we can modify *welcome4.php* one more time so that it can receive the first and last names of the user from either source (*welcome5.php*):

```
<?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.

[1]Prior to PHP 4.1, this variable was called $HTTP_GET_VARS. This variable name remains in current PHP versions for backwards compatibility. If your server has an older version of PHP installed, or if you're writing a script that must be compatible with older versions, you should use $HTTP_GET_VARS instead of $_GET.

[2]Prior to PHP 4.1, 'posted' form variables were available in the $HTTP_POST_VARS array. This array remains available in current versions of PHP for backwards compatibility.

[3]$_REQUEST is not available in versions of PHP prior to PHP 4.1.

Control Structures

All the examples of PHP code that we've seen so far have been either simple, one-statement scripts that output a string of text to the Web page, or have been series of statements that were to be executed one after the other in order. If you've ever written programs in any other languages (be they 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
  // 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 *welcome1.php* file we created earlier:

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

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, but 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.

Important 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 which, incidentally, evaluates as true). This would not only cause the condition always to be true, but might also change the value in the variable you're checking, which could cause all sorts of problems.

Conditions can be more complex than a single comparison for equality. Recall that we modified *welcome1.php* to take 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 (*welcome6.php*):

```
$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), they work in PHP as well.

We'll look at more complicated comparisons 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 repeatedly a set of statements. 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 next statement following 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. This script is available as *count10.php* in the code archive.

```
$count = 1;
while ($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 to output the string "`1 2 3 4 5 6 7 8 9 10 `".

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, where we are counting through a series of values until some condition is met, is called a *for loop*. Here's what they look like:

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

Here's what the above `while` loop example looks like when implemented as a `for` loop:

```
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 the code is a little harder to read at first glance, having everything to do 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 into what we need:

- We need the name on every page of the site, not just on 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[4]:

```
<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. A short cut exists for those times when you simply want to echo a PHP value in the middle of your HTML code. The short cut looks like this:

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

The tags <?=... ?> perform the same function as the much longer code <?php echo(...); ?>. This is a handy short cut that I'll use several times through the rest of this book.

You're familiar with the `echo` function, but `urlencode` is probably new to you. This function takes any 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 `$name` variable had a value of `"Kevin Yank"`, then, as spaces are not allowed in the query string, the output of `urlencode` (and thus the string output by `echo`) would be `"Kevin+Yank"`. PHP would then automatically convert it back when it created the`$_GET` variable in *newpage.php*.

Okay, so we've got the user's name being 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 with a form in it 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[5]. 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 this is beginning to sound to you 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:

```
<html>
<head>
<title> Multipurpose Page Outline </title>
</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:

```
<html>
<head>
<title> Multi-Purpose Page Outline </title>
</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 (*samplepage.php* in the code archive):

```
<html>
<head>
<title> Sample Page </title>
</head>
<body>

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

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

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

<?php else: ?>

  <p>Your name: <?=$_GET['name']?></p>

  <p>This paragraph contains a
```

```
    <a href="newpage.php?name=<?=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 given). 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.[6]

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. Once they enter their name and click "GO", they'll be presented with the exact page they requested. The name they entered is then passed in the query string of every link from that point onward, ensuring that they are prompted only once.
[4]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 "Cookies and Sessions in PHP".

[5]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.

[6]The`$_SERVER` array was introduced in PHP 4.1. In previous versions of PHP, these values were available in an array called `$HTTP_SERVER_VARS`. Also, when `register_globals` is set to `On` in the *php.ini* file (the default setting in PHP versions prior to 4.2), `$_SERVER['PHP_SELF']` was available simply as `$PHP_SELF`.

Summary

In this chapter, we've had a taste of the PHP server-side scripting language by exploring 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 the 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"Publishing MySQL Data on the Web", we'll delve into the MySQL functions in PHP, to show how to publish the joke database that we created in "Getting Started with MySQL" on 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.

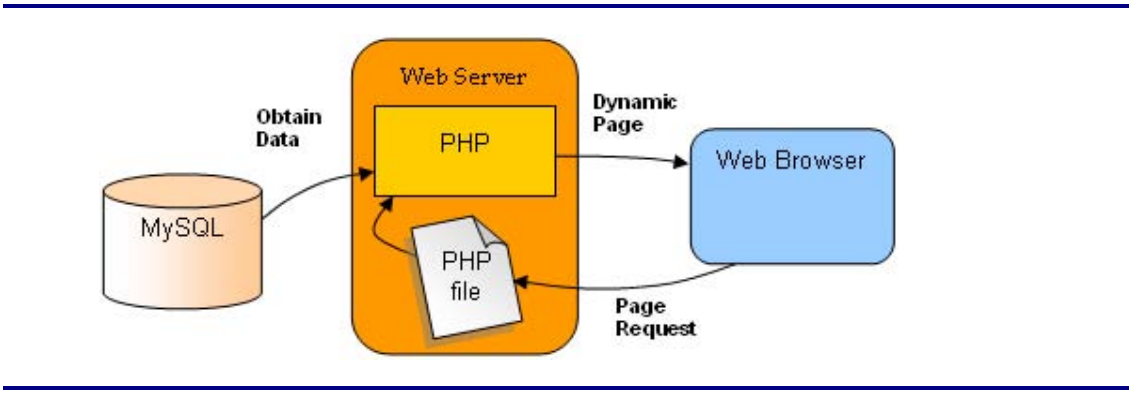
Chapter 4: Publishing MySQL Data on the Web

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 have 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, new tools at our disposal: the PHP scripting language, and the MySQL database engine. It's important to understand how these two 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 dynamically pulled from the database 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 load `http://www.yoursite.com/`, and expects to view a standard HTML Web page. 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).



PHP retrieves MySQL data to produce Web pages

As shown in "PHP retrieves MySQL data to produce Web pages", 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. Where the 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"Getting Started with MySQL"-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:

- The visitor's Web browser requests the Web page using a standard URL.
- The Web server software (Apache, IIS, or whatever) recognizes that the requested file is a PHP script, and so the server interprets the file using its PHP plug-in, before responding to the page request.
- Certain PHP commands (which you have yet to learn) connect to the MySQL database and request the content that belongs in the Web page.
- The MySQL database responds by sending the requested content to the PHP script.
- The PHP script stores the content into one or more PHP variables, and then uses the now-familiar

echo function to output the content as part of the Web page.

- The PHP plug-in finishes up by handing a copy of the HTML it has created to the Web server.
- 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 first know how to establish a connection to MySQL from inside a PHP script. Back in "Getting Started with MySQL", you used a program called *mysql* that allowed you to make such a connection. PHP has no need of any special program, however; support for connecting to MySQL is built right into the language. The following PHP function call 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"Getting Started with MySQL".

You may remember that functions in PHP usually return (output) a value when they are 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 addition to doing something useful when they are called, most functions output a value, and 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 is unavailable or inaccessible due to a network outage, or because the username/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, allowing us to display our own, friendlier error message.

Next, we put an exclamation point in front of the `$dbcnx` variable in the condition of the `if` statement. The exclamation point is the PHP *negation operator*, which basically flips a false value to true, or a true value to false. Thus, if the connection fails and `mysql_connect` returns false, `!$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 takes no parameters. 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 "Getting Started with MySQL", 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 "Getting Started with MySQL". The database we created was called jokes. Selecting that database in PHP is just a matter of another function call:

```
mysql_select_db('jokes', $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 actually 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('jokes') ) {
  die( '<p>Unable to locate the joke ' .
       'database at this time.</p>' );
}
```

Notice 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 short cut. To check if 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 the `die` function.`die` works just like `echo`, except that the script exits after it. So calling `die` is equivalent to a call to `echo` followed by a call to `exit`, which is what we used for `mysql_connect` above.

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

Sending SQL Queries with PHP

In"Getting Started with MySQL", 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 Jokes table we created in "Getting Started with MySQL":

```
$sql = 'CREATE TABLE Jokes (
          ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
          JokeText TEXT,
          JokeDate DATE NOT NULL
        )';
if ( @mysql_query($sql) ) {
  echo('<p>Jokes table successfully created!</p>');
} else {
  die('<p>Error creating Jokes 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 "Getting Started with MySQL" to set the dates of all jokes that contained the word "chicken":

```
$sql = "UPDATE Jokes SET JokeDate='1990-04-01'
         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 {
  die('<p>Error performing update: ' . mysql_error() .
      '</p>');
}
```

`SELECT` queries are treated a little differently, since 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. As a result, 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) {
  die('<p>Error performing query: ' . mysql_error() .
      '</p>');
}
```

Provided no error was encountered in processing the query, the above code will place a result set that contains the text of all the jokes stored in the Jokes table into the variable `$result`. 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 to 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 much 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 as a parameter (stored in the `$result` variable in this case), and returns the next row in the result set as an array (see "Getting Started with PHP" 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 is to figure out how to get the values out of the `$row` variable each time the loop runs.

Rows of a result set 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. The code of this example is available as *jokelist.php* in the code archive.

```
<html>
<head>
<title> Our List of Jokes </title>
</head>
```

```
<body>
<?php

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

  // Select the jokes database
  if (! @mysql_select_db('jokes') ) {
    die( '<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
  $result = @mysql_query('SELECT JokeText FROM Jokes');
  if (!$result) {
    die('<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>
```

Inserting Data into the Database

In this section, we'll see how we can use all the tools at our disposal to allow visitors to our site 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. It's mostly just a sample application of 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:

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

As we've seen before, this form, when submitted, will load the very same page (because we used the `$_SERVER['PHP_SELF']` variable for the form's `action` attribute), but with two variables attached to the request. The first, `joketext`, will contain the text of the joke as typed into the text area. The second, `submitjoke`, will always contain the value `"SUBMIT"`; the presence of this variable is a signal that a joke has been submitted. Both of these variables will appear in the `$_POST` and `$_REQUEST` arrays created by PHP.

To insert the submitted joke into the database, we just use `mysql_query` to run an `INSERT` query, using the`$joketext` variable for the value to be submitted:

```
if (isset($_POST['submitjoke'])) {
  $joketext = $_POST['joketext'];
  $sql = "INSERT INTO Jokes SET
          JokeText='$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 whole example is shown here 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 only introduce them as required. For a complete function reference, refer to "MySQL Functions".

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. Since most users will only want to view our 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 code (available as *jokes.php* in the code archive):

```
<html>
<head>
<title> The Internet Joke Database </title>
</head>
<body>
<?php
  if (isset($_GET['addjoke'])): // If the user wants to add a joke
```

```
?>

<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
<p>Type your joke here:<br />
<textarea name="joketext" rows="10" cols="40" wrap>
</textarea><br />
<input type="submit" name="submitjoke" value="SUBMIT" />
</p>
</form>
<?php
  else: // Default page display

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

    // Select the jokes database
    if (! @mysql_select_db('jokes') ) {
      die( '<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['submitjoke'])) {
      $joketext = $_POST['joketext'];
      $sql = "INSERT INTO Jokes SET
              JokeText='$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
    $result = @mysql_query('SELECT JokeText FROM Jokes');
    if (!$result) {
      die('<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>
```

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 put a link labelled "Delete this Joke" next to each joke on the page 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 "Getting Started with MySQL".
- 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 Jokes 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 jokes database online, and allowed visitors to add jokes of their own to it.

In"Relational Database Design", 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, which is also available as *challege.php* in the code archive. If you have any questions, don't hesitate to post them in the SitePoint Forums!

```
<html>
<head>
<title> The Internet Joke Database </title>
</head>
<body>
<?php
  if (isset($_GET['addjoke'])): // If the user wants to add a joke
?>

<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
<p>Type your joke here:<br />
<textarea name="joketext" rows="10" cols="40" wrap>
</textarea><br />
<input type="submit" name="submitjoke" value="SUBMIT" />
</p>
</form>
<?php
  else: // Default page display

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

    // Select the jokes database
    if (! @mysql_select_db('jokes') ) {
      die( '<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['submitjoke'])) {
      $joketext = $_POST['joketext'];
      $sql = "INSERT INTO Jokes SET
              JokeText='$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'];
      $sql = "DELETE FROM Jokes
              WHERE ID=$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
    $result = @mysql_query('SELECT ID, JokeText FROM Jokes');
    if (!$result) {
      die('<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>

Chapter 5: Relational Database Design

Since"Getting Started with MySQL" of this book, we've worked with a very simple database of jokes, which is composed of a single table named, appropriately enough, Jokes. While this database has served us well as an introduction to MySQL databases, there's more to relational database design than this simple example illustrates. In this chapter, we'll expand on our example, and learn a few new features of MySQL, in an effort to realize and appreciate what relational databases have to offer.

Be forewarned that many topics will be covered only in an informal, hands-on (i.e. non-rigorous) sort of way. As any computer science major will tell you, database design is a serious area of research, with tested and mathematically provable principles that, while useful, are beyond the scope of this text. If you want more information, stop by http://www.datamodel.org/ for a list of good books, as well as several useful resources on the subject. In particular, check out the 5 Rules of Normalization in the Data Modelling section of the site.

Giving Credit where Credit is Due

To start things off, let's recall the structure of our Jokes table. It contains three columns: ID, JokeText, and JokeDate. Together, these columns allow us to identify jokes (ID), and keep track of their text (JokeText) and the date they were entered (JokeDate). The SQL code that creates this table and inserts a couple of entries is provided as *jokes1.sql* in the code archive.

Now let's say we wanted to track another piece of information about our jokes: the names of the people who submitted them. It would seem natural to want to add a new column to our Jokes table for this. The SQLALTER command (which we haven't seen before) lets us do exactly what we need. Log into your MySQL server using the `mysql` command-line program as in "Getting Started with MySQL", select your database (jokes if you used the name suggested in that chapter) then type this command:

```
mysql>ALTER TABLE Jokes ADD COLUMN
    ->AuthorName VARCHAR(255);
```

This code adds a column called AuthorName to your table. The type declared is a variable-length character string of up to 255 characters, plenty of space for even very esoteric names. Let's also add a column for the author's email address:

```
mysql>ALTER TABLE Jokes ADD COLUMN
    ->AuthorEMail VARCHAR(255);
```

For more information about the ALTER command, see "MySQL Syntax". Just to make sure the two columns were added properly, we should ask MySQL to describe the table to us:

```
mysql>DESCRIBE Jokes;
+-------------+--------------+------+-----+------------+-- -
| Field       | Type         | Null | Key | Default    |
+-------------+--------------+------+-----+------------+-- -
| ID          | int(11)      |      | PRI | NULL       |
| JokeText    | text         | YES  |     | NULL       |
| JokeDate    | date         |      |     | 0000-00-00 |
| AuthorName  | varchar(255) | YES  |     | NULL       |
| AuthorEMail | varchar(255) | YES  |     | NULL       |
+-------------+--------------+------+-----+------------+-- -
5 rows in set (0.01 sec)
```

Looks good, right? Obviously, we would need to make changes to the HTML and PHP form code we created in "Publishing MySQL Data on the Web" that allows us to add new jokes to the database, but I'll leave the figuring out of those details to you, as an exercise. Using UPDATE queries, we could now add author details to all the jokes in the table. But before we get carried away with these additions, we need to stop and consider whether this new table design was the right choice here. In this case, it turns out that it wasn't.

Rule of Thumb: Keep Things Separate

As your knowledge of database-driven Websites continues to grow, you may decide that a personal joke list isn't enough. In fact, you might begin to receive more submitted jokes than you have original jokes of your own. Let's say you decide to launch a Website where people from all over the world can share jokes with each other. You've heard of the Internet Movie Database (IMDB)? You decide to open the Internet Joke Database (IJDB)! To add the author's name and email address to each joke certainly makes a lot of sense, but the way we did it above leads to several potential problems:

- What if a frequent contributor to your site named Joan Smith changed her email address? She might begin to submit new jokes using the new address, but all the old jokes would still have the old address attached to them. Looking at your database, you might simply think there were two different people named Joan Smith who submit jokes. If she were especially thoughtful, she might inform you of the change of address, and you might try to update all the old jokes with the new address, but if you missed just one joke, your database would still have incorrect information stored in it. Database design experts refer to this sort of problem as an *update anomaly*.

- It would be natural for you to rely on your database to provide a list of all the people who've ever submitted jokes to your site. In fact, you could easily obtain a mailing list by using the following query:

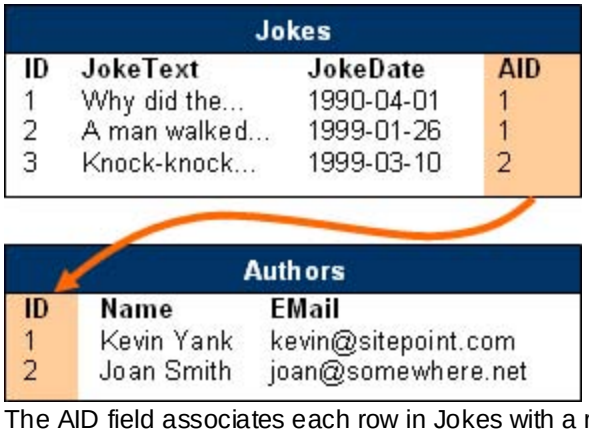
```
mysql>SELECT DISTINCT AuthorName, AuthorEMail
    ->FROM Jokes;
```

  The word `DISTINCT` in the above query tells MySQL not to output duplicate result rows. For example, if Joan Smith submitted 20 jokes to your site, her name and email address would appear 20 times in the list, instead of just once, if you failed to use the `DISTINCT` option.

  If for some reason you decided to remove all the jokes that a particular author had submitted to your site, you'd remove any record of this person from the database in the process, and you'd no longer be able to email him or her with information about your site! As your mailing list might be a major source of income for your site, you wouldn't want to go throwing away an author's email address just because you didn't like the jokes that person had submitted to your site. Database design experts call this a *delete anomaly*.

- You have no guarantee that Joan Smith would not enter her name as "Joan Smith" one day, as "J. Smith" the next, and as "Smith, Joan" on yet another occasion. This would make keeping track of a particular author exceedingly difficult, especially if Joan Smith had several email addresses she liked to use, too.

These problems-and more-can be dealt with very quickly. Instead of storing the information for the authors in the Jokes table, let's create an entirely new table for our list of authors. Since we used a column called ID in the Jokes table to identify each of our jokes with a unique number, we'll use an identically-named column in our new table to identify our authors. We can then use those "author ID's" in our Jokes table to associate authors with their jokes. The complete database layout is shown in "The AID field associates each row in Jokes with a row in Authors".



The AID field associates each row in Jokes with a row in Authors

What the above two tables show are three jokes and two authors. The AID column (short for "Author ID") of the Jokes table provides a relationship between the two tables, indicating that Kevin Yank submitted jokes 1 and 2 and Joan Smith submitted joke 3. Notice also that, since each author now only appears once in the database, and appears independently of the jokes he or she has submitted, we've avoided all the problems outlined above.

The most important characteristic of this database design, however, is that, since we're storing information about two types of "things" (jokes and authors), it's most appropriate to have two tables. This is a rule of thumb that you should always keep in mind when designing a database: *each type of entity (or "thing") that you want to be able to store information about should be given its own table.*

To set up the above database from scratch is fairly simple (involving just two CREATE TABLE queries), but since we'd like to make these changes in a non-destructive manner (i.e. without losing any of our precious knock-knock jokes), we'll use the ALTER command again. First, we get rid of the author-related columns in the Jokes table:

```
mysql>ALTER TABLE Jokes DROP COLUMN AuthorName;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>ALTER TABLE Jokes DROP COLUMN AuthorEMail;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Now we create our new table:

```
mysql>CREATE TABLE Authors (
    ->  ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->  Name VARCHAR(255),
    ->  EMail VARCHAR(255)
    ->);
```

Finally, we add the AID column to our Jokes table:

```
mysql>ALTER TABLE Jokes ADD COLUMN AID INT;
```

If you prefer, the CREATE TABLE commands that will create the two tables from scratch are provided in *2tables.sql* in the code archive. All that's left is to add some authors to the new table, and assign authors to all the existing jokes in the database by filling in the AID column[1]. Go ahead and do this now if you like.

This should give you some practice with `INSERT` and `UPDATE` queries.
[1]For now you'll have to do this manually. But don't worry, in "A Content Management System"we'll see how PHP can insert entries with the correct IDs automatically to reflect the relationships between them.

Dealing with Multiple Tables

With your data now separated into two tables, it may seem that you're complicating the process of data retrieval. Consider, for example, our original goal: to display a list of jokes with the name and email address of the author next to each joke. In the single-table solution, you could get all the information you needed to produce such a list using a single SELECT statement in your PHP code:

```
$jokelist = mysql_query(
  "SELECT JokeText, AuthorName, AuthorEMail FROM Jokes");

while ($joke = mysql_fetch_array($jokelist)) {
  $joketext = $joke["JokeText"];
  $name = $joke["AuthorName"];
  $email = $joke["AuthorEMail"];

  // Display the joke with author information
  echo( "<p>$joketext<br></br>" .
        "(by <a href='mailto:$email'>$name</a>)</p>" );
}
```

In the new system, this would, at first, no longer seem possible. As the details about the author of each joke aren't stored in the Jokes table, you might think that you'd have to fetch those details individually for each Joke you wanted to display. The code to do perform this task would look like:

```
// Get the list of jokes
$jokelist = mysql_query("SELECT JokeText, AID FROM Jokes");

while ($joke = mysql_fetch_array($jokelist)) {

  // Get the text and Author ID for the joke
  $joketext = $joke["JokeText"];
  $aid = $joke["AID"];

  // Get the author details for the joke
  $authordetails = mysql_query(
    "SELECT Name, EMail FROM Authors WHERE ID=$aid");
  $author = mysql_fetch_array($authordetails);
  $name = $author["Name"];
  $email = $author["EMail"];

  // Display the joke with author information
  echo( "<p>$joketext<br></br>" .
        "(by <a href='mailto:$email'>$name</a>)</p>" );
}
```

It's pretty messy, and it involves a query to the database for every single joke that's displayed, which could slow down the display of your page considerably. With all this taken into account, it would seem that the "old way" was actually the better solution, despite its weaknesses. Fortunately, relational databases like MySQL are designed to make working with data stored in multiple tables easy! Using a new form of the SELECT statement, called a *join*, you can have the best of both worlds. Joins allow you to treat related data in multiple tables as if they were stored in a single table. Here's what the syntax of a simple join looks like:

```
mysql>SELECTcolumns FROM tables
    ->WHEREcondition(s) for data to be related;
```

In your case, the columns you're interested in are JokeText in the Jokes table, and Name and EMail in the Authors table. The condition for an entry in the Jokes table to be related to an entry in the Authors table is that the value of the AID column in the Jokes table is equal to the value of the ID column in the Authors table. Here's an example of a join (the first two queries simply show you what's contained in the two tables - they aren't necessary):

```
mysql>SELECT LEFT(JokeText,20), AID FROM Jokes;
+----------------------+------+
| LEFT(JokeText,20)    | AID  |
+----------------------+------+
| Why did the chicken  |    1 |
| A man walked into a  |    1 |
| Knock knock. Who's t |    2 |
+----------------------+------+
3 rows in set (0.00 sec)

mysql>SELECT * FROM Authors;
+----+------------+---------------------+
| ID | Name       | EMail               |
+----+------------+---------------------+
|  1 | Kevin Yank | kyank@attglobal.net |
|  2 | Joan Smith | joan@somewhere.net  |
+----+------------+---------------------+
2 rows in set (0.00 sec)

mysql>SELECT LEFT(JokeText,20), Name, EMail
    ->FROM Jokes, Authors WHERE AID = Authors.ID;
+----------------------+------------+---------------------+
| LEFT(JokeText,20)    | Name       | EMail               |
+----------------------+------------+---------------------+
| Why did the chicken  | Kevin Yank | kyank@attglobal.net |
| A man walked into a  | Kevin Yank | kyank@attglobal.net |
| Knock knock. Who's t | Joan Smith | joan@somewhere.net  |
+----------------------+------------+---------------------+
3 rows in set (0.00 sec)
```

See? The results of the third SELECT, which is a join, group the values stored in the two tables into a single table of results, with related data correctly appearing together. Even though the data is stored in two tables, you can still get all the information you need to produce the joke list on your Web page with a single database query. Note in the query that, since there are columns named ID in both tables, you must specify the name of the table when you refer to the ID column in the Authors table (`Authors.ID`). If you don't specify the table name, MySQL won't know which ID you're referring to, and will produce this error:

```
mysql>SELECT LEFT(JokeText,20), Name, EMail
    ->FROM Jokes, Authors WHERE AID = ID;
ERROR 1052: Column: 'ID' in where clause is ambiguous
```

Now that you know how to access the data stored in your two tables efficiently, you can rewrite the code for your joke list to take advantage of joins. The following is reproduced with complete error checking (which has been omitted here for brevity) in *jokelist2.php* in the code archive.

```
$jokelist = mysql_query(
  'SELECT JokeText, Name, EMail
   FROM Jokes, Authors WHERE AID=Authors.ID');
```

```
while ($joke = mysql_fetch_array($jokelist)) {
  $joketext = $joke['JokeText'];
  $name = $joke['Name'];
  $email = $joke['EMail'];

  // Display the joke with author information
  echo( "<p>$joketext<br></br>" .
        "(by <a href='mailto:$email'>$name</a>)</p>" );
}
```

The more you work with databases, the more you'll come to realize just how powerful this simple ability to combine data contained in separate tables into a single table of results really is. Consider, for example, the following query, which displays a list of all jokes written by Joan Smith:

```
mysql>SELECT JokeText FROM Jokes, Authors WHERE
    ->Name="Joan Smith" AND AID=Authors.ID;
```

The results that are output from the above query come only from the Jokes table, but the query uses a join to let it search for jokes based on a value stored in the Authors table. There will be plenty more examples of clever queries like this throughout this book, but this example alone illustrates that the practical applications of joins are many and varied, and in almost all cases can save you a lot of work!

Simple Data Relationships

The best type of database layout for a given situation is usually dictated by the type of relationship that exists between the pieces of data that it needs to store. In this section, I'll examine the typical relationship types, and explain how best to represent them in a relational database.

In the case of a simple *one-to-one relationship*, a single table is all you'll need. An example of a one-to-one relationship that you've seen is the email address of each author in our joke database. Since there will be one email address for each author, and one author for each email address, there is no reason to split the addresses off into a separate table.

A*many-to-one relationship* is a little more complicated, but you've already seen one of these as well. Each joke in our database is associated with just one author, but many jokes may have been written by that one author. This joke-author relationship is many-to-one. I've already covered the problems that result from storing the information associated with a joke's author in the same table as the joke itself. In brief, it can result in many copies of the same data, which are difficult to keep synchronized, and which waste space. If we split the data into two tables, and use an ID column to link the two together, which will make joins possible as shown above, all these problems disappear.

You have yet to see a *one-to-many relationship*, but finding an example isn't difficult. In our database so far, we've assumed that each author has only one email address. While this may not always be the case, this is a reasonable limitation to impose since you only really need one email address to get in touch with an author. You simply trust that each author would enter his or her most-used email address—or at least one that is checked regularly—when adding him or herself to the database. If you did, however, want to support multiple email addresses, you'd be faced with a one-to-many relationship (one author may have many email addresses, but each email address belongs to exactly one author).

When someone inexperienced in database design approaches a one-to-many relationship like this one, his or her first approach is often to try to store multiple values in a single database field, as shown in "Never overload a table field to store multiple values, as is done here".

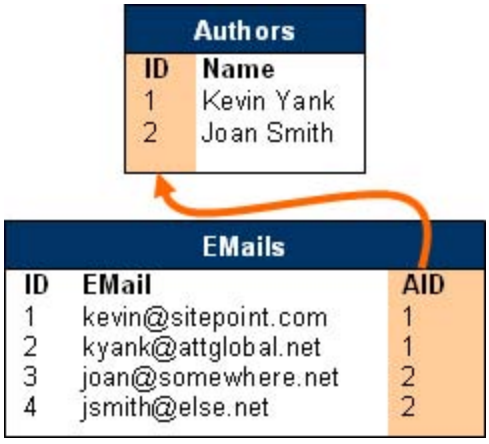
Authors

ID	Name	EMail
1	Kevin Yank	kevin@sitepoint.com, kyank@attglobal.net
2	Joan Smith	joan@somewhere.net, jsmith@else.net

Never overload a table field to store multiple values, as is done here

While this would work, to retrieve a single email address from the database, we'd need to break up the string by searching for commas (or whatever special character you chose to use as a separator)—a not-so-simple, and potentially time-consuming operation. Try to imagine the PHP code necessary to remove one particular email address from one particular author! In addition, you'd need to allow for much longer values in the EMail column, which could result in wasted disk space, because the majority of authors would have just one email address.

The solution for a one-to-many relationship such as this is very similar to the solution we saw for a many-to-one relationship above. As you might expect, the pattern is simply reversed. You just break the Authors table into two tables—Authors and EMails—and then associate the email addresses with their authors using an Author ID (AID) column in the EMails table (see "The AID field associates each row of Emails with one row of Authors").



The AID field associates each row of Emails with one row of Authors

Using a join, it's easy to list the email addresses associated with a particular author:

```
mysql>SELECT EMail FROM Authors, EMails WHERE
    ->Name="Kevin Yank" AND AID=Authors.ID;
+---------------------+
| EMail               |
+---------------------+
| kevin@sitepoint.com |
| kyank@attglobal.net |
+---------------------+
2 rows in set (0.00 sec)
```

Many-to-Many Relationships

Okay, you've now got a steadily-growing database of jokes published on your Website. It's growing so quickly, in fact, that the number of jokes has become unmanageable! People who visit your site are faced with a mammoth page that contains hundreds of jokes listed with no structure whatsoever. Something has to change.

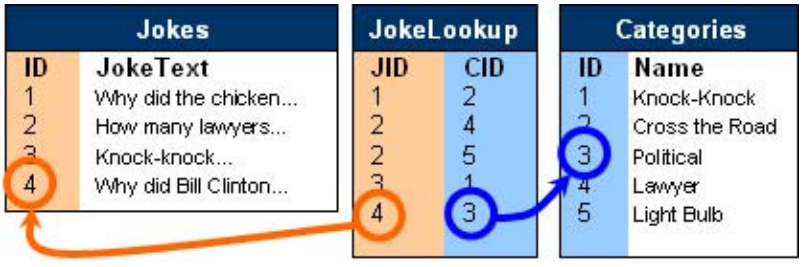
You decide to place your jokes into categories such as "Knock-Knock Jokes", "Crossing the Road Jokes", "Lawyer Jokes", and "Political Jokes". Remembering our rule of thumb from earlier, you identify joke categories as a different type of "thing", and create a new table for them:

```
mysql>CREATE TABLE Categories (
    ->  ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->  Name VARCHAR(255)
    ->);
Query OK, 0 rows affected (0.00 sec)
```

Now you come to the daunting task of assigning categories to your jokes. It occurs to you that a "political" joke might also be a "crossing the road" joke, and a "knock-knock" joke might also be a "lawyer" joke. A single joke might belong to many categories, and each category will contain many jokes. This is a *many-to-many* relationship.

Once again, many inexperienced developers begin to think of ways to store several values in a single column, because the obvious solution is to add a Categories column to the Jokes table and use it to list the ID's of those categories to which each joke belongs. A second rule of thumb would be useful here: *if you need to store multiple values in a single column, your design is probably flawed.*

The correct way to represent a many-to-many relationship is to use a *look-up table*. This is a table that contains no actual data, but which defines pairs of entries that are related. "The JokeLookup table associates pairs of rows from the Jokes and Categories tables" shows what the database design would look like for our joke categories.



The JokeLookup table associates pairs of rows from the Jokes and Categories tables

The JokeLookup table associates joke IDs (JID) with category IDs (CID). In this example, we can see that the joke that starts with "How many lawyers..." belongs to both the "Lawyer" and "Light Bulb" categories.

A look-up table is created in much the same way as is any other table. The difference lies in the choice of the primary key. Every table we've created so far has had a column named ID that was designated to be the`PRIMARY KEY` when the table was created. Designating a column as a primary key tells MySQL not to allow two entries to have the same value in that column. It also speeds up join operations based on that column.

In the case of a look-up table, there is no single column that we want to force to have unique values. Each joke ID may appear more than once, as a joke may belong to more than one category, and each category ID may appear more than once, as a category may contain many jokes. What we don't want to allow is the same*pair* of values to appear in the table twice. And since the sole purpose of this table is to facilitate

joins, the speed benefits offered by a primary key would come in very handy. For this reason, we usually create look-up tables with a multi-column primary key as follows:

```
mysql>CREATE TABLE JokeLookup (
    ->  JID INT NOT NULL,
    ->  CID INT NOT NULL,
    ->  PRIMARY KEY(JID,CID)
    ->);
```

This creates the table in which the JID and CID columns together form the primary key. This enforces the uniqueness that is appropriate to a look-up table, preventing a particular joke from being assigned to a particular category more than once, and speeds up joins that make use of this table.

With your look-up table in place and containing category assignments, you can use joins to create several interesting and very practical queries. This query lists all jokes in the "Knock-Knock" category:

```
mysql>SELECT JokeText
    ->FROM Jokes, Categories, JokeLookup
    ->WHERE Name="Knock-Knock" AND
    ->  CID=Categories.ID AND JID=Jokes.ID;
```

The following query lists the categories that contain jokes that begin with "How many lawyers...":

```
mysql>SELECT Categories.Name
    ->FROM Jokes, Categories, JokeLookup
    ->WHERE JokeText LIKE "How many lawyers%"
    ->  AND CID=Categories.ID AND JID=Jokes.ID;
```

And this query, which also makes use of our Authors table to form a join of four tables (!!!), lists the names of all authors who have written knock-knock jokes:

```
mysql>SELECT Authors.Name
    ->FROM Jokes, Authors, Categories, JokeLookup
    ->WHERE Categories.Name="Knock-Knock"
    ->  AND CID=Categories.ID AND JID=Jokes.ID
    ->  AND AID=Authors.ID;
```

Summary

In this chapter, I explained the fundamentals of good database design, and we learned how MySQL and, for that matter, all relational database management systems, provide support for the representation of different types of relationships between entities. From your meagre understanding of one-to-one relationships, you should now have expanded your knowledge to include many-to-one, one-to-many, and many-to-many relationships. And in the process, you learned a few new features of common SQL commands. In particular, you learned how to use a SELECT to join data spread between multiple tables into a single set of results. In "A Content Management System", you'll use all the knowledge you have gained so far, plus a few new tricks, to build a basic content management system in PHP. The aim of such a system is to provide a customized, secure, Web-based interface that manages the contents of the database, instead of requiring you to type everything in by hand on the MySQL command line.

Chapter 6: A Content Management System

Overview

So far, we've seen several examples of database-driven Web pages: pages that display information that's culled from a MySQL database when the page is requested. Until now, however, we haven't seen a solution that would be much more manageable than raw HTML files if it was scaled up to encompass a Website as large and complex as, say, sitepoint.com. Sure, our Internet Joke Database was nice, but when it came to managing categories and authors, we'd always have to return to the MySQL command line and try to remember complicated `SELECT` and `INSERT` statements, as well as table and column names, to accomplish the most menial of tasks.

To make the leap from a Web page that displays information stored in a database to a completely database-driven Website, we need to add a *content management system*. Such a system usually takes the form of a series of Web pages, access to which is restricted to users who are authorized to make changes to the Website. These pages provide a database administration interface, which allows a user to view and change the information that's stored in the database without bothering with the mundane details of SQL syntax.

The beginnings of a content management system were seen at the end of "Publishing MySQL Data on the Web", where we allowed site visitors to add jokes to, and (if you worked through the challenge) delete jokes from, the database using a Web-based form and a "delete this joke" link, respectively. While impressive, these are not features that you'd normally include in the interface presented to casual site visitors. For example, you don't want someone to be able to add offensive material to your Website without your knowledge. And you *definitely* don't want just anyone to be able to delete jokes from your site.

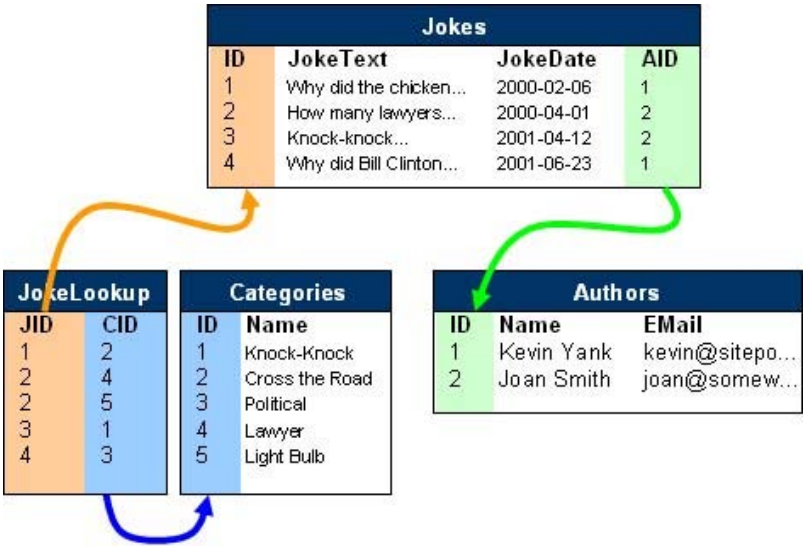
By relegating those "dangerous" features to the restricted-access site administration pages, you avoid the risk of exposing your data to the average user, and you maintain the power to manage the contents of your database without having to memorize SQL queries. In this chapter, we'll expand on the capabilities of our joke management system to take advantage of the enhancements we made to our database in "Relational Database Design". Specifically, we'll allow a site administrator to manage authors and categories, and assign these to appropriate jokes.

As we've seen, these administration pages must be protected by an appropriate access restriction scheme. One way to do this would be to place the relevant PHP files into a directory that was protected by an Apache-style *.htaccess* file that listed authorized users. Consult your Web server's documentation or ask your Web host for information on how to restrict access to Web pages.

Since we'll work with some fairly large PHP files in this part, it'll be necessary to gloss over some of the details, because of space constraints. The complete code of all the files discussed in this chapter, together with the SQL code you'll need to create the database tables from scratch, will form a complete content management system, and is provided in the code archive for this book.

The Front Page

At the end of "Relational Database Design", your database contained tables for three types of entities: jokes, authors, and joke categories. This database layout is represented in "The structure of the finished jokes database". Note that we're sticking with our original assumption that we'll have one email address per author. If you need to recreate this table structure, the SQL queries to do so may be found in the *joketables.sql* file in the code archive.



The structure of the finished jokes database

The front page of the content management system, therefore, will contain links to pages that manage each of these three things:

```
<!-- admin.html -->
<html>
<head>
<title>JMS</title>
</head>
<body>
<h1>Joke Management System</h1>
<ul>
  <li><a href="jokes.php">Manage Jokes</a></li>
  <li><a href="authors.php">Manage Authors</a></li>
  <li><a href="cats.php">Manage Joke Categories</a></li>
</ul>
</body>
</html>
```

Managing Authors

Let us begin with *authors.php*, the file that allows administrators to add new authors, and delete and edit existing ones. If you're comfortable with the idea of multipurpose pages, you may want to place the code for all of this into the single file, *authors.php*. Since the code for this file would be fairly long, I'll use separate files in my examples to break it up a little.

The first thing we'll present to an administrator who needs to manage authors is a list of all authors currently stored in the database. Code-wise, this is the same as listing the jokes in the database. Since we'll want to allow administrators to delete and edit existing authors, you should include links for these functions next to each author's name. Just like the "Delete this Joke" links in the challenge at the end of "Publishing MySQL Data on the Web", these links will have the ID of the author attached to them, so that the target document knows which author the user wishes to edit or delete. Finally, we shall provide a "Create New Author" link that leads to a form similar in operation to the "Add a Joke" link we created in "Publishing MySQL Data on the Web".

```
<!-- authors.php -->
<html>
<head>
<title> Manage Authors </title>
</head>
<body>
<h1>Manage Authors</h1>
<p align="center"><a href="newauthor.php">Create New Author</a>
</p>
<ul>
<?php

$cnx = mysql_connect('localhost','root','mypasswd');
mysql_select_db('jokes');

$authors = @mysql_query('SELECT ID, Name FROM Authors');
if (!$authors) {
  die('<p>Error retrieving authors from database!<br />'.
      'Error: ' . mysql_error() . '</p>');
}

while ($author = mysql_fetch_array($authors)) {
  $id   = $author['ID'];
  $name = htmlspecialchars($author['Name']);
  echo("<li>$name ".
       "[<a href='editauthor.php?id=$id'>Edit</a>|".
       "<a href='deleteauthor.php?id=$id'>Delete</a>]</li>");
}

?>
</ul>
<p align="center"><a href="admin.html">Return to Front Page</a>
</p>
</body>
</html>
```

The`htmlspecialchars` function used within the `while` loop in the code above may be a little worrisome to you. For the moment, you can simply ignore it. I'll explain exactly what it does in "Editing Authors" below.

Deleting Authors

deleteauthor.php will allow us to remove an author from the database given its ID. As we have seen before, this is frighteningly easy to do, but there is added complexity here. Remember that our Jokes table has an AID column that indicates the author responsible for a given joke. When we remove an author from the database, we must also get rid of any references to that author in other tables. If we didn't, then we might have jokes left in the database that were associated with a nonexistent author.

This is one weakness of MySQL-it won't automatically clean up orphaned data related to an entry that you delete. More advanced database management systems can actually keep track of which entries are related to which, and make sure that the set of relationships represented in the database (the *referential integrity* of the database) is always maintained. However, MySQL was designed to forego such niceties in order to achieve significant performance gains, so the task of cleaning up orphaned entries falls to our PHP script.

We have two possible ways to handle this situation:

- When we delete an author, also delete any jokes attributed to the author.
- When we delete an author, set the AID of any jokes attributed to the author to NULL, to indicate that they have no author.

Since most authors would not like us using their jokes without giving them credit, we'll opt for the first option. This also saves us from having to handle jokes with `NULL` values in their AID column when we display our library of jokes.

```
<!-- deleteauthor.php -->
<html>
<head>
<title> Delete Author </title>
</head>
<body>
<?php

$cnx = mysql_connect('localhost','root','mypasswd');
mysql_select_db('jokes');

// Delete all jokes belonging to the author
// along with the entry for the author.
$id = $_GET['id'];
$ok1 = @mysql_query("DELETE FROM Jokes WHERE AID='$id'");
$ok2 = @mysql_query("DELETE FROM Authors WHERE ID='$id'");
if ($ok1 and $ok2) {
  echo('<p>Author deleted successfully!</p>');
} else {
  echo('<p>Error deleting author from database!<br />'.
       'Error: ' . mysql_error() . '</p>');
}

?>
<p><a href="authors.php">Return to Authors list</a></p>
</body>
</html>
```

Adding Authors

Next comes *newauthor.php*, which allows administrators to add new authors to the database. Again, this is just like adding new jokes, which we tackled in "Publishing MySQL Data on the Web".

```
<!-- newauthor.php -->
<html>
<head>
<title> Add New Author </title>
</head>
<body>
<?php

if (isset($_POST['submit'])):
  // A new author has been entered
  // using the form below.

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

  $name = $_POST['name'];
  $email = $_POST['email'];
  $sql = "INSERT INTO Authors SET
          Name='$name',
          EMail='$email'";
  if (@mysql_query($sql)) {
    echo('<p>New author added</p>');
  } else {
    echo('<p>Error adding new author: ' .
         mysql_error() . '</p>');
  }

?>

<p><a href="<?=$_SERVER['PHP_SELF']?>">Add another Author</a></p>
<p><a href="authors.php">Return to Authors list</a></p>

<?php
  else: // Allow the user to enter a new author
?>

<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
<p>Enter the new author:<br />
Name: <input type="text" name="name" size="20" maxlength="255"
  /><br />
Email: <input type="text" name="email" size="20" maxlength="255"
  /><br />
<input type="submit" name="submit" value="SUBMIT" /></p>
</form>

<?php endif; ?>

</body>
</html>
```

Editing Authors

All that's left is *editauthor.php*, which must provide an interface for us to edit an existing author's details. This page will actually be very similar to *newauthor.php*, except the form fields will initially contain the values stored in the database, and an `UPDATE` query will be used instead of an `INSERT` query when the form is submitted.

One minor complication comes into play here. To initialize the form fields with the values stored in the database, the page will obviously use the `$id` variable passed from *authors.php* to retrieve the values and store them in PHP variables (say, `$name` and `$email`). The code for our form should then look like this:

```
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
<p>Edit the author:<br />
Name: <input type="text" name="name" value="<?=$name?>"
  size="20" maxlength="255" /><br />
EMail: <input type="text" name="email" value="<?=$email?>"
  size="20" maxlength="255" /><br />
<input type="hidden" name="id" value="<?=$id?>" />
<input type="submit" name="submit" value="SUBMIT" /></p>
</form>
```

As an aside, notice the hidden form field, which we use to pass along the author's ID with the updated values when the form is submitted.

But consider what would happen if the author's name were `"The Jokester"` (*with* the quotes). The input tag produced by the PHP script would look like this:

```
<input type="text" name="name" value=""The Jokester""
  size="20" maxlength="255" />
```

Obviously, this is invalid HTML. We need to replace the quotes in the name with their HTML *character entity* equivalents. Specifically, any double quotes in the name should be converted to the character code `&quot;` as follows:

```
<input type="text" name="name" value="&quot;The Jokester&quot;"
  size="20" maxlength="255" />
```

PHP provides a function called `htmlspecialchars` that automatically converts special HTML characters such as <,> and quotes (among others) like those above into their respective character codes. Consider the following basic example:

```
$text = htmlspecialchars('<HTML> can be dangerous!');
echo($text); // output: &lt;HTML&gt; can be dangerous!
```

To avoid problems with quotes and angled brackets in your text strings, you should use this function whenever you output a non-HTML text string, especially when you output variables retrieved from a database, which can have unpredictable values.

```
// Convert special characters for safe use
// as HTML attributes.
$name = htmlspecialchars($name);
$email = htmlspecialchars($email);
```

With this issue in mind, we can now create *editauthor.php*, the complete code for which is provided in the code archive.

Magic Quotes

While we're on the subject of troublesome special characters, there is another situation where particular characters in a string can cause problems. Consider the following SQL query:

```
mysql>INSERT INTO Authors SET
    ->Name='Molly O'Reilly',
    ->EMail='mollyo@hotmail.com';
```

Obviously, the apostrophe in the author's last name will cause problems here, as MySQL can no longer figure out where the author's name ends. The solution in this case would be to use another function provided by PHP: `addslashes`. This function, like `htmlspecialchars`, converts unsafe characters in a string so that they're safe. The difference is that `addslashes` is used to *escape* special characters by putting backslashes before them, as follows:

```
mysql>INSERT INTO Authors SET
    ->Name='Molly O\'Reilly',
    ->EMail='mollyo@hotmail.com';
```

A backslash tells MySQL to treat the next character (the apostrophe, in this case) as a character in the string, ignoring any special meaning it might normally have. Thus, the above code will correctly insert the name Molly O'Reilly into the Authors table.

So why haven't we worried about this problem before now? PHP has a nifty little feature called *Magic Quotes*, which is enabled by default with the following setting in your *php.ini* file:

```
magic_quotes_gpc = On
```

This setting basically tells PHP to use the `addslashes` function automatically upon any variables that are passed with the request for the page. The "`gpc`" stands for "get, post, cookies", which are the three methods by which information may be passed with a request for a Web page. As all the values we've inserted into our database up until now have been passed as part of a form submission, the Magic Quotes feature of PHP has automatically added slashes to them every time. Values retrieved from a MySQL database, however, do not benefit from the Magic Quotes feature, and so we must add slashes before we can use them in any situation where quotes, apostrophes, and other special characters may be a problem.

In some cases, you may not actually *want* to add backslashes to submitted values. For example, if you are just going to print out a value that was submitted with a form, then those backslashes could turn out to be quite an eyesore. To undo the work of either the `addslashes` function or the Magic Quotes feature, you can use yet another function called `stripslashes`.

Complete information about these functions may be found in the PHP online manual at http://www.php.net/manual. All of the scripts in this book are written with the default setting, `magic_quotes_gpc = On` in mind.

Managing Categories

When you compare the roles of authors and joke categories in the database, they are really very similar. They both reside in tables of their own, and they both serve to group jokes together in some way. As a result, categories can be handled with almost the exact same code as we've developed for authors, with one important exception.

When we delete a category, we can't simultaneously delete any jokes that belong to that category, because those jokes may also belong to other categories. We could check each joke to see if it belonged to any other categories, and only delete those that did not, but rather than engage in such a time-consuming process, let's allow for the possibility of including jokes in our database that don't belong to any category at all. These jokes would be invisible to our site visitors, but would remain in the database in case we wanted to assign them to a category later on.

Thus, to delete a category, we also need to delete any entries in the JokeLookup table that refer to that category:

```
<!-- deletecat.php -->

...

// Delete all joke look-up entries for the
// category along with the entry for the category.
$ok1 = @mysql_query("DELETE FROM JokeLookup WHERE CID='$id'");
$ok2 = @mysql_query("DELETE FROM Categories WHERE ID='$id'");

...
```

Other than this one detail, category management is functionally identical to author management. The code for *cats.php*,*newcat.php*,*deletecat.php*, and *editcat.php* is provided in the code archive if you need it.

Managing Jokes

Along with the addition, deletion, and modification of jokes in our database, we also need to be able to assign categories and authors to our jokes. Furthermore, we're likely to have many more jokes than authors or categories. As a result, to try to display a complete list of jokes, as we did for the authors and categories, could result in an unmanageably long list, and no easy way to spot the one joke we're after. So we need to create a more intelligent method of browsing our library of jokes.

Searching for Jokes

At different times we may know the category, author, or some of the text in a joke we wish to work with, so let's support all of these methods for the location of jokes in our database. When we're done, it should work like a simple *search engine*. The form that will prompt the administrator for information about the desired joke must present lists of categories and authors. The code for this is as follows:

```
<!-- jokes.php -->
<html>
<head>
<title> Manage Jokes </title>
</head>
<body>
<h1>Manage Jokes</h1>
<p><a href="newjoke.php">Create New Joke</a></p>
<?php

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

$authors = mysql_query('SELECT ID, Name FROM Authors');
$cats    = mysql_query('SELECT ID, Name FROM Categories');
?>

<form action="jokelist.php" method="post">
<p>View jokes satisfying the following criteria:<br />
By Author:
<select name="aid" size="1">
  <option selected value="">Any Author</option>
<?php
  while ($author = mysql_fetch_array($authors)) {
    $aid = $author['ID'];
    $aname = htmlspecialchars($author['Name']);
    echo("<option value='$aid'>$aname</option>\n");
  }
?>
</select><br />
By Category:
<select name="cid" size="1">
  <option selected value="">Any Category</option>
<?php
  while ($cat = mysql_fetch_array($cats)) {
    $cid = $cat['ID'];
    $cname = htmlspecialchars($cat['Name']);
    echo("<option value='$cid'>$cname</option>\n");
  }
?>
</select><br />
```

```
Containing Text: <input type="text" name="searchtext" /><br />
<input type="submit" name="submit" value="Search" />
</form>

<p align="center"><a href="admin.html">Return to Front Page</a>
</p>
</body>
</html>
```

Note that the `\n` at the end of the strings that are output by the `echo` function is the special code for a new line, which serves to make the HTML code output by this script more readable.[1] Also, note the use of `htmlspecialchars` to ensure that author and category names don't contain any troublesome characters when they're displayed.

It's up to *jokelist.php* to use the values submitted in the above form to build a list of jokes that satisfies the criteria specified. Obviously, this will be done with a `SELECT` query, but the exact nature of that query will depend on what was entered through the form we defined above. Because the building of this `SELECT` statement is a fairly complicated process, let's work through *jokelist.php* a little at a time.

First, we get the preliminaries out of the way:

```
<!-- jokelist.php -->
<html>
<head>
<title> Manage Jokes </title>
</head>
<body>
<h1>Manage Jokes</h1>
<p><a href="jokes.php">New Search</a></p>
<?php

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

Now, to start, we define a few strings that, when strung together, form the `SELECT` query we'd need if no constraints had been selected in the form:

```
// The basic SELECT statement
$select = 'SELECT DISTINCT ID, JokeText';
$from   = ' FROM Jokes';
$where  = ' WHERE 1=1';
```

The`WHERE` clause in the above code might be somewhat confusing. The idea here is for us to be able to build on this basic `SELECT` statement, depending on which constraints are selected in the form. These constraints will require us to add to the `FROM` and `WHERE` clauses (portions) of the `SELECT` statement. But if there were no constraints specified (i.e. the administrator wanted a list of all jokes in the database), there would be no need for a `WHERE` clause at all! Because it's difficult to add to a `WHERE` clause that doesn't exist, we needed to come up with a "do nothing" `WHERE` clause that will always be true. Thus, we have introduced the requirement that 1 equals 1, which fits the bill nicely[2].

Our next task is to check each of the possible constraints (author, category, and search text) that may have been set in the form, and adjust the SQL accordingly. First, we deal with the possibility that an author was specified. The "Any Author" option in the form was given a value of `""` (the empty string), so if the value of that form field (stored in `$_POST['aid']`) is not equal to `""`, then an author has been specified, and we must adjust our query:

```
$aid = $_POST['aid'];
if ($aid != '') { // An author is selected
```

```
  $where .= " AND AID='$aid'";
}
```

`.=`, the *string concatenation operator* is used to tack a new string onto the end of an existing one. In this case, we add to the `WHERE` clause the condition that the AID in the Jokes table must match the author ID selected in the form (`$aid`).

Next, we handle the specification of a joke category:

```
$cid = $_POST['cid'];
if ($cid != '') { // A category is selected
  $from  .= ', JokeLookup';
  $where .= " AND ID=JID AND CID='$cid'";
}
```

As the categories associated with a particular joke are stored in the JokeLookup table, we need to add this table to the query to create a join. To do this, we simply tack the name of the table onto the end of the `$from` variable. And to complete the join, we must also specify that the ID column (in the Jokes table) must match the JID column (in JokeLookup), so we add this condition to the `$where` variable. Finally, we require the CID column (in JokeLookup) to match the category ID selected in the form (`$cid`).

Handling search text is fairly simple, and uses the `LIKE` SQL operator that we learned way back in "Getting Started with MySQL":

```
$searchtext = $_POST['searchtext'];
if ($searchtext != '') { // Some search text was specified
  $where .= " AND JokeText LIKE '%$searchtext%'";
}
```

Now that we've built our SQL query, we can use it to retrieve and display our jokes, along with links that allow us to edit and delete them, just like we did for authors and joke categories. For readability, we display our jokes in an HTML table:

```
?>

<table border="1">
<tr><th>Joke Text</th><th>Options</th></tr>

<?php
$jokes = @mysql_query($select . $from . $where);
if (!$jokes) {
  echo('</table>');
  die('<p>Error retrieving jokes from database!<br />'.
      'Error: ' . mysql_error() . '</p>');
}

while ($joke = mysql_fetch_array($jokes)) {
  echo("<tr valign='top'>\n");
  $id       = $joke['ID'];
  $joketext = htmlspecialchars($joke['JokeText']);
  echo("<td>$joketext</td>\n");
  echo("<td>[<a href='editjoke.php?id=$id'>Edit</a>|".
       "<a href='deletejoke.php?id=$id'>Delete</a>]</td>\n");
  echo("</tr>\n");
}
?>

</table>
</body>
```

```
</html>
```

Adding Jokes

With*jokelist.php* out of the way, let's tackle *newjoke.php*, which is linked-to at the top of *jokes.php*. This page will be very similar to *newauthor.php* and *newcat.php*. However, in addition to specifying the joke text, this page must allow an administrator to assign an author and categories to a joke. These features make the code of this file worth some examination.

We know from viewing the code of *newauthor.php*, that the PHP code that processes the form submission comes before the form code itself. It doesn't have to, but this is the layout we've used so far. Let's begin by looking at the form code, so that the code for handling form submissions makes more sense.

First, we fetch lists of all the authors and categories in the database:

```
<?php
else: // Allow the user to enter a new joke

  $authors = mysql_query('SELECT ID, Name FROM Authors');
  $cats    = mysql_query('SELECT ID, Name FROM Categories');
?>
```

Next, we create our form. We begin with a standard text area for typing in the text of the joke:

```
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
<p>Enter the new joke:<br />
<textarea name="joketext" rows="15" cols="45" wrap>
</textarea></p>
```

We'll prompt the administrator to select an author from a drop-down list of those authors in the database:

```
<p>Author:
<select name="aid" size="1">
  <option selected value="">Select One</option>
  <option value="">---------</option>
<?php
  while ($author = mysql_fetch_array($authors)) {
    $aid = $author['ID'];
    $aname = htmlspecialchars($author['Name']);
    echo("<option value='$aid'>$aname</option>\n");
  }
?>
</select></p>
```

However, a drop-down list won't suffice for the selection of categories, because we want the administrator to be able to select multiple categories. Thus, we'll use a series of check boxes—one for each category. Since we have no way to know in advance the number of check boxes we'll need, the matter of naming them becomes an interesting challenge. What we'll actually do is use a *single* variable for all of the check boxes; thus, all the check boxes will have the same name. To be able to receive multiple values from a single variable name, we must make that variable an *array*. Recall from "Getting Started with PHP" that an array is a single variable with 'compartments', each of which can hold a value. To submit a form element as part of an array variable, we simply add a pair of square brackets to the end of the variable name (making it `cats[]` in this case).[3]

With all of our check boxes named the same, we'll need a way to identify which particular check boxes have been selected. To this end, we assign a different value to each check box—the ID of the corresponding category in the database. Thus, what gets submitted by the form is an array that contains all category IDs to which the new joke should be added.

```
<p>Place in categories:<br />
```

```
<?php
  while ($cat = mysql_fetch_array($cats)) {
    $cid = $cat['ID'];
    $cname = htmlspecialchars($cat['Name']);
    echo("<input type='checkbox' name='cats[]' value='$cid' />" .
         "$cname<br />\n");
  }
?>
</p>
```

And we finish off our form as usual:

```
<p><input type="submit" name="submit" value="SUBMIT" /></p>
</form>
<?php endif; ?>
```

Since we're submitting an array for the first time, the code that processes this form is not totally straightforward. It starts off pretty simply as we add the joke to the Jokes table. Since an author is required, we make sure that `$_POST['aid']` contains a value. This prevents the administrator from choosing the "Select One" option in the author select list, as that choice has a value of `""` (the empty string).

```
<?php
$dbcnx = mysql_connect('localhost', 'root', 'mypasswd');
mysql_select_db('jokes');

if (isset($_POST['submit'])):
  // A new joke has been entered
  // using the form.

  $aid = $_POST['aid'];
  $joketext = $_POST['joketext'];
  $cats = $_POST['cats'];

  if ($aid == '') {
    die('<p>You must choose an author ' .
        'for this joke. Click "Back" ' .
        'and try again.</p>');
  }

  $sql = "INSERT INTO Jokes SET
          JokeText='$joketext',
          JokeDate=CURDATE(),
          AID='$aid'";
  if (@mysql_query($sql)) {
    echo('<p>New joke added</p>');
  } else {
    echo('<p>Error adding new joke: ' .
         mysql_error() . '</p>');
  }

  $jid = mysql_insert_id();
```

The last line in the above code uses a function that we haven't seen before: `mysql_insert_id`. This function returns the number assigned to the last-inserted entry by the `AUTO_INCREMENT` feature in MySQL. In other words, it retrieves the ID of the newly inserted joke, which we'll need later.

The code that adds the entries to JokeLookup based on which check boxes were checked is not so simple. First of all, we've never seen how a check box passes its value to a PHP variable before. Also, we

need to deal with the fact that these particular check boxes will submit into an array variable.

A typical check box will pass its value to a PHP variable if it is checked, and will do nothing when it is unchecked. Check boxes without assigned values pass "on" as the value of their corresponding variables when they are checked. However, we've assigned values to our check boxes (the category IDs), so this is not an issue.

The fact that these check boxes submit into an array actually adds quite a measure of convenience to our code. In essence, what we'll receive from the submitted form is either

1. an array of category IDs to add the joke to, or
2. nothing at all (if none of the check boxes were checked).

First, let's handle the latter, special case, by creating an empty array when we find that the `$cats` variable is empty:

```
if ($cats == '') $cats = array();
```

The `array` function that appears here is used to create a new array in PHP. The parameters that are passed to it become the elements of the array. Because we're passing no parameters to it here, it'll simply create an empty array.

Now that we've guaranteed that the `$cats` variable contains an array, we can use a loop to consider each category ID in the array in turn, and to insert the appropriate entry into the database. Since this array isn't based on a database row, you might wonder how we can access the values in the array. After all, we've usually retrieved an array value using its database column name (e.g. `$cat['Name']`). In this case, our array was created simply by feeding a series of values into the same variable name. When this happens, PHP automatically assigns numerical indices to the values in the array.

For instance, the value of the first check box that was checked will be submitted first into the array and will be accessible as `$cat[0]`. That is, PHP assigns it an array index of 0. The second check box that is checked will have its value stored with an index of 1, accessible as `$cat[1]`. So if there are *n* check boxes checked, then the value of the last check box will be in `$cat[`*n*`-1]`. By counting up through the array indexes as we proceed through a loop in our code, we can process the elements of this array one at a time.

But wait... what is *n*? We have no way of knowing in advance how many check boxes will be checked, so how should the loop know when to stop counting? Well, there are two ways. The first is to use a PHP function called `count` that takes an array as a parameter and counts the number of elements in it. Here's what our `while` loop would look like if we use this method:

```
$i = 0; // First index
while ($i < count($cats)) { // While we're not at the end
  // process $cats[$i]
  $i = $i + 1;
}
```

As you can see, this loop uses a counter variable (`$i`), that is, a variable that counts the number of times the loop has executed. The first time through the loop, it will have a value of 0, then at the end of the loop we'll add 1 to it. Therefore, the second time through the loop it will have a value of 1, and so on. Within the loop, we can use this variable as the array index to pull a category ID out of the `$cats` array. The loop stops looping when `$i` reaches `count($cats)`, the number of elements in the `$cats` array. If `$cats` doesn't contain any elements (i.e. if no categories were selected), then `$i` will *start out* equal to `count($cats)`, and the contents of the loop won't be executed at all!

This all seems very slick, but there's actually a better way. Instead of using the `count` function, we can simply keep going until we reach a value of `$i` for which `$cat[$i]` is empty. When we do, we know we'll have reached the end of the list of category IDs:

```
$i = 0; // First index
while ($cats[$i] != "") { // While we're not at the end
  // process $cats[$i]
  $i = $i + 1;
}
```

This will run a little faster because we don't call a function each time through the loop. Plus, it's a teeny bit more clever, and we programmers have to have our fun when we can! Believe it or not, however, PHP spoils our fun by having a completely separate type of loop that's specialized for looping through arrays, called a *foreach loop*. Here's what the code looks like in this case:

```
foreach ($cats as $catID) {
  // Process $catID
}
```

This `foreach loop` will execute the code inside the loop once *for each* item in the `$cats` array (you see where the `foreach` loop gets its name), and will assign the item for each loop to the variable `$catID`. Since this code is indisputably tidier than the equivalent `while` loop, we'll settle on this as a solution. All that remains is to determine what to do for each selected category ID.

Before we became sidetracked by all these different types of loops, we were about to take our array of category IDs and use it to place our newly-inserted joke into its corresponding categories. A cursory examination of our database layout reveals that we just have to insert an entry into the JokeLookup table for each category of which that joke should be a member. Recall that each entry in the JokeLookup table consists of a joke ID (JID) and a category ID (CID), which together indicate that a particular joke belongs to a particular category. Here's the finished `foreach` loop:

```
  $numCats = 0;
  foreach ($cats as $catID) {
    $sql = "INSERT IGNORE INTO JokeLookup
            SET JID=$jid, CID=$catID";
    $ok = @mysql_query($sql);
    if ($ok) {
      $numCats = $numCats + 1;
    } else {
      echo("<p>Error inserting joke into category $catID: " .
           mysql_error() . '</p>');
    }
  }
?>

<p>Joke was added to <?=$numCats?> categories.</p>

<p><a href="<?=$_SERVER['PHP_SELF']?>">Add another Joke</a></p>
<p><a href="jokes.php">Return to Joke Search</a></p>
```

The word `IGNORE` in the `INSERT` query used here is a precaution only. Recall that when we defined the JokeLookup table we set the JID and CID columns to be the primary key for the table. If somehow the JID/CID pair that is inserted already exists in the table, an attempt to insert it again would normally cause an error. By adding `IGNORE` to the command, a re-insert of the same pair is simply ignored by MySQL and no error occurs. This situation should never actually happen, but it's better to be safe than sorry.

Editing and Deleting Jokes

The two files that remain, *editjoke.php* and *deletejoke.php*, mirror their author and category counterparts, with minor adjustments. *editjoke.php* must provide the same author select box and category check boxes as*newjoke.php*, except that this time they must be initialized to reflect those values stored in the database

for the particular joke we've selected. *deletejoke.php*, meanwhile, must not only delete the selected joke from the Jokes table, but must also remove any entries in the JokeLookup table for that joke. The code for both of these files is provided in the code archive, but we won't spend time examining its details, since these files are just an application of skills that should be fairly familiar to you by now.

[1] Other special character codes include `\r` (carriage return) and `\t` (tab). Like variables, these codes only work inside double-quoted strings.

[2] In fact, the "do nothing" `WHERE` clause could just be `' WHERE 1'`, since MySQL considers any positive number true. Feel free to change it if you don't find the idea confusing.

[3] Another way to submit an array is with a `<select multiple>` tag. Again, you would set the `name` attribute to end with square brackets. What will be submitted is an array of all the `<option>` values selected from the list by the user. Feel free to experiment with this approach by modifying *newjoke.php* to present the categories in a list.

Summary

There are a few minor tasks that our content management system is still not able to handle. For example, it's currently unable to provide a listing of all jokes that don't belong to *any* category-something that could come in very handy as the number of jokes in the database grows. You might also like to sort joke listings by various criteria. These particular capabilities require a few more advanced SQL tricks that we'll see in "Advanced SQL".

If we ignore these little details for the moment, you'll see that you now have a system that allows someone with no SQL or database knowledge to administer your database of jokes with ease! Together with a set of PHP-powered pages through which regular site visitors can view the jokes, this content management system allows us to set up a complete database-driven Website that can be maintained by someone with absolutely no database knowledge. And if you think that sounds like a valuable commodity to businesses looking to get on the Web today, you're right!

In fact, only one aspect of our site requires special knowledge (beyond the use of a Web browser) to use: content formatting. For example, it would not be unusual for someone to want to enter a joke that contained more than one paragraph of text. In our current system, this could be accomplished by entering the HTML code for the joke directly into the "Create New Joke" form. Why is this unacceptable?

As we stated way back in the introduction to this book, one of the most desirable features of a database-driven Website is that the people responsible for adding content to the site need not be familiar with HTML. If we require knowledge of HTML for something as simple as dividing a joke into paragraphs, we have failed to achieve our goal.

As a bonus in this chapter, you also learned a little more about arrays in PHP. You learned how a set of form elements can submit their values into a single array variable, and you learned how to process that array on the receiving end by looping through it with a `while` loop, a `for` loop, and a `foreach` loop.

In"Content Formatting and Submission", we'll see how we can make use of some features of PHP to provide a simpler means by which we can format content without requiring site administrators to know the ins and outs of HTML. We'll also bring back the "Submit Your Own Joke" link to our site, and discover how we can safely accept content submissions from casual site visitors.

Chapter 7: Content Formatting and Submission

Overview

We're almost there! We've designed a database to store jokes, organize them into categories, and track their authors. We've learned how to create a Web page that displays this library of jokes to site visitors. We've even developed a set of Web pages that a site administrator can use to manage the joke library without having to know anything about databases.

In so doing, we've built a site that frees the resident Webmaster from continually having to plug new content into tired HTML page templates, and from maintaining an unmanageable mass of HTML files. The HTML is now kept completely separate from the data it displays. If you want to redesign the site, you simply have to make the changes to the HTML contained in the PHP files that site visitors see. A change to one file (e.g. changing a font) is immediately reflected in the page layouts of all jokes, because all jokes are displayed using that single PHP file. Only one task still requires HTML to enter into the equation that manages the content of the Website: *content formatting*.

On any but the simplest of Websites, it will be necessary to allow content (in our case study, jokes) to be formatted. In a simple case, this might merely be the ability to break text into paragraphs. Often, however, content providers will expect facilities such as **boldface** or *italicized* text, hyperlinks, etc.

Our current database and site design already supports all these requirements. A site administrator can include in the text of a joke HTML tags that will have their usual effects on the joke text when it's inserted into a page that a browser requests. However, to eliminate HTML from the system entirely, we must provide some other way to format text.

In this chapter, we'll learn some new PHP functions that provide basic text formatting without the use of HTML. In so doing, we'll have completed a content management system that anyone with a Web browser can use. We'll then take full advantage of this ease of use, and allow site visitors to once again submit their own jokes-this time without the risk that users might fill our site with inappropriate material.

Out with the Old

Before we introduce a new method to format text, we should first disable the old one. A user with no knowledge of HTML might unknowingly include HTML syntax (however invalid) in a plain text document, and if this syntax is still accepted, it could produce unexpected results-or even mess up your finely tuned page layout. Consider the following sentence:

```
The gunman drew his weapon. <BANG!>
```

The user who entered this text into the database might be surprised to see the last word (`<BANG!>`) missing from the Web page that displayed this content. And while anyone with a basic knowledge of HTML would know that the Web browser discarded that segment of text as an invalid HTML tag, we're trying to cater to users with no knowledge of HTML whatsoever.

In"A Content Management System", we saw a PHP function that solved this problem quite neatly: `htmlspecialchars`. This function, when applied to the text of our joke before it was inserted into a Web page, would convert the string above into the following "HTML safe" version:

```
The gunman drew his weapon. &lt;BANG!&gt;
```

When this string was interpreted by the site visitor's Web browser, it would produce the desired result. As a first step, therefore, we must modify the PHP file on our Website that displays the text of jokes, so that it uses`htmlspecialchars` on all text before it outputs it to the Web. Since up until now I have not given the complete code for a page to display a joke, I'll be starting from scratch. The complete code for *joke.php* with this entire chapter taken into account is provided in the code archive, so don't feel that you have to follow along by typing out the code that I'll show you.

Here's the basic code for fetching a joke with a given ID out of the database and formatting it for display by converting it to an 'HTML Safe' version:

```
<!-- joke.php -->
...
// Get the joke text from the database
$id = $_GET['id'];
$joke = mysql_query("SELECT JokeText FROM Jokes
                     WHERE ID=$id");
$joke = mysql_fetch_array($joke);
$joketext = $joke['JokeText'];

// Filter out HTML code
$joketext = htmlspecialchars($joketext);

echo( $joketext );
...
```

We have now neutralized any HTML code that may appear in the site's content. With this clean slate, we are ready to implement a markup language of our own that will let administrators format content.

Regular Expressions

To implement our own markup language, we'll require a script to spot our custom tags in the text of jokes and replace them with their HTML equivalents, before it outputs the joke text to the user's browser. Anyone with experience in regular expressions will know that they're ideal for this sort of work.

A *regular expression* is a string of text that contains special codes, which allow it to be used with a few PHP functions to search and manipulate other strings of text. This, for example, is a regular expression that searches for the text "PHP" (without the quotes)[1]:

```
PHP
```

Not much to it, is there? To use a regular expression, you must be familiar with the regular expression functions available in PHP. `ereg` is the most basic, and can be used to determine whether a regular expression is *satisfied* by a particular text string. Consider the code:

```
$text = 'PHP rules!';

if (ereg('PHP', $text)) {
  echo( '$text contains the string "PHP".' );
} else {
  echo( '$text does not contain the string "PHP".' );
}
```

In this example, the regular expression is satisfied because the string stored in variable `$text` contains "PHP". The above code will thus output the following (note that the single quotes prevent PHP from filling in the value of the variable `$text`):

```
$text contains the string "PHP".
```

`eregi` is a function that behaves almost identically to `ereg`, except that it ignores the case of text when it looks for matches:

```
$text = "What is Php?";

if (eregi("PHP", $text)) {
  echo( '$text contains the string "PHP".' );
} else {
  echo( '$text does not contain the string "PHP".' );
}
```

Again, this outputs the same message, despite the fact that the string actually contains `Php`:

```
$text contains the string "PHP".
```

As was mentioned above, there are special codes that may be used in regular expressions. Some of these can be downright confusing and difficult to remember, so if you intend to make extensive use of them, a good reference might come in handy. A tutorial-style reference to standard regular expression syntax may be found at http://www.delorie.com/gnu/docs/rx/rx_toc.html. Let's work our way through a few examples to learn the basic regular expression syntax.

First of all, a caret (^) may be used to indicate the start of the string, while a dollar sign ($) is used to indicate its end:

```
PHP      Matches "PHP rules!" and "What is PHP?"
^PHP     Matches "PHP rules!" but not "What is PHP?"
PHP$     Matches "I love PHP" but not "What is PHP?"
^PHP$    Matches "PHP" but nothing else
```

Obviously, you may sometimes want to use ^,$, or other special characters to represent the corresponding character in the search string, rather than the special meaning implied by regular

expression syntax. To remove the special meaning of a character, prefix it with a backslash:

\$\$\$ Matches "Show me the $$$!" but not "$10"

Square brackets can be used to define a set of characters that may match. For example, the following regular expression will match any string that contains a digit from 1 to 5 inclusive:

[12345] Matches "1" and "39", but not "a" or "76"

Ranges of numbers and letters may also be specified.

[1-5] Same as previous
^[a-z]$ Matches any single lowercase letter
[0-9a-zA-Z] Matches any string with a letter or number

The characters ?,+, and * also have special meanings. Specifically, ? means "the preceding character is optional",+ means "one or more of the previous character", and * means "zero or more of the previous character".

bana?na Matches "banana" and "banna",
but not "banaana".
bana+na Matches "banana" and "banaana",
but not "banna".
bana*na Matches "banna", "banana", and "banaaana",
but not "bnana".
^[a-zA-Z]+$ Matches any string of one or more
letters and nothing else.

Parentheses may be used to group strings together to apply ?,+, or * to them as a whole.

ba(na)+na Matches "banana" and "bananananana",
but not "bana" or "banaana".

And finally, the a period (.) matches any character except a new line:

^.+$ Matches any string of one or more characters with no line breaks.

There are more special codes and syntax tricks for regular expressions, all of which should be covered in any reference, such as those mentioned above. For now, we have more than enough for our purposes. [1]This book covers PHP's support for POSIX Regular Expressions. A more complex, more powerful, but less standardized form of regular expressions called Perl Compatible Regular Expressions (PCRE) is also supported by PHP; however, I will not cover it in this book. For more information on PCRE, see http://www.php.net/pcre.

String Replacement with Regular Expressions

Using`ereg` or `eregi` with the regular expression syntax we've just learned, we can easily detect the presence of tags in a given text string. However, what we need to do is pinpoint those tags, and replace them with appropriate HTML tags. To achieve this, we need to look at a couple more regular expression functions offered by PHP: `ereg_replace` and `eregi_replace`.

`ereg_replace`, like `ereg`, accepts a regular expression and a string of text, and attempts to match the regular expression in the string. In addition, `ereg_replace` takes a second string of text, and replaces every match of the regular expression with that string.

The syntax for `ereg_replace` is as follows:

```
$newstr = ereg_replace(regexp,replacewith,oldstr);
```

Here,*`regexp`* is the regular expression, and *`replacewith`* is the string that will replace matches to *`regexp`* in *`oldstr`*. The function returns the new string that's the outcome of the replacement operation. In the above, this newly-generated string is stored in `$newstr`.

`eregi_replace`, as you might expect, is identical to `ereg_replace`, except that the case of letters is not considered when searching for matches.

We're now ready to build our custom markup language.

Boldface and Italic Text

Let's start by implementing tags that create **boldface** and *italic* text. Let's say we want `[B]` to begin bold text and `[EB]` to end bold text. Obviously, we must replace `[B]` with `<strong>` and `[EB]` with `</strong>` [2]. Achieving this is a simple application of `eregi_replace`[3]:

```
$joketext = eregi_replace('\[b]','<strong>',$joketext);
$joketext = eregi_replace('\[eb]','</strong>',$joketext);
```

Notice that, because `[` normally indicates the start of a set of acceptable characters in a regular expression, we put a backslash before it in order to remove its special meaning. Without a matching `[`, the `]` loses its special meaning, so it doesn't need a backslash, although you could put a backslash in front of it as well if you wanted to be thorough.

Also notice that, as we're using `eregi_replace`, which is case insensitive, both `[B]` and `[b]` will work as tags in our custom markup language.

Italic text can be done the same way:

```
$joketext = eregi_replace('\[i]','<em>',$joketext);
$joketext = eregi_replace('\[ei]','</em>',$joketext);
```

Paragraphs

While we could create tags for paragraphs just as we did for boldface and italicized text above, a simpler approach makes even more sense. As users will type the content into a form field that allows them to format text using the enter key, we shall take a single new line (`\n`) to indicate a line break (`<br />`) and a double new line (`\n\n`) to indicate a new paragraph (`</p><p>`). Of course, because Windows computers represent an end-of-line as a new line/carriage return pair (`\n\r`) and Macintosh computers represent it as a carriage return/new line pair (`\r\n`), we must strip out carriage returns first. The code for all this is as follows:

```
// Strip out carriage returns
$joketext = ereg_replace("\r",'',$joketext);
// Handle paragraphs
$joketext = ereg_replace("\n\n",'</p><p>',$joketext);
```

```
// Handle line breaks
$joketext = ereg_replace("\n",'<br />',$joketext);
```

That's it! The text will now appear in the paragraphs expected by the user, who hasn't had to learn any custom tags to format content into paragraphs.

Hyperlinks

While supporting the inclusion of hyperlinks in the text of jokes may seem unnecessary, this feature makes plenty of sense in other applications. Hyperlinks are a little more complicated than the simple conversion of a fixed code fragment into an HTML tag. We need to be able to output a URL, as well as the text that should appear as the link.

Another feature of `ereg_replace` and `eregi_replace` comes into play here. If you surround a portion of the regular expression with parentheses, you can *capture* the corresponding portion of the matched text, and use it in the replace string. To do this, you'll use the code `\\n`, where *n* is 1 for the first parenthesized portion of the regular expression, 2 for the second, up to 9 for the 9th. Consider this example:

```
$text = 'banana';
$text = eregi_replace('(.*)(nana)', '\\2\\1', $text);
echo($text); // outputs "nanaba"
```

In the above, `\\1` gets replaced with `ba` in the replace string, which corresponds to `(.*)` (zero or more non-new line characters) in the regular expression. `\\2` gets replaced with `nana`, which corresponds to `(nana)` in the regular expression.

We can use the same principle to create our hyperlinks. Let's begin with a simple form of link, where the text of the link is the same as the URL. We want to support this syntax:

```
Visit [L]http://www.php.net/[EL].
```

The corresponding HTML code, which we want to output, is as follows:

```
Visit <a href="http://www.php.net/">http://www.php.net/</a>.
```

First, we need a regular expression that will match links of this form. The regular expression is as follows:

```
\[L][-_./a-zA-Z0-9!&%#?+,'=:~]+\[EL]
```

Again, we've placed backslashes in front of the opening square brackets in `[L]` and `[EL]` to indicate that they are to be taken literally. We then use square brackets to list all the characters we wish to accept as part of the URL[4]. We place a + after the square brackets to indicate that the URL will be composed of one or more characters taken from this list.

To output our link, we'll need to capture the URL and output it both as the `href` attribute of the a tag, and as the text of the link. To capture the URL, we surround the corresponding portion of our regular expression with parentheses:

```
\[L]([-_./a-zA-Z0-9!&%#?+,'=:~]+)\[EL]
```

So we convert the link with the following code:

```
$joketext = ereg_replace(
  '\[L]([-_./a-zA-Z0-9!&%#?+,\'=:~]+)\[EL]',
  '<a href="\\1">\\1</a>', $joketext);
```

Note that we had to escape the quote (`'`) in the regular expression with a backslash (`\'`) to prevent PHP from thinking it indicated the end of the regular expression string.. Meanwhile, `\\1` in the replacement string gets replaced by the URL for the link, and the output is as expected!

We'd also like to support hyperlinks whose link text differs from the URL. Let's say the form of our link is as follows:

```
Check out [L=http://www.php.net/]PHP[EL].
```

Here's our regular expression (wrapped to fit on the page):

```
\[L=([-_./a-zA-Z0-9!&%#?+,'=:~]+)]
([-_./a-zA-Z0-9 !&%#?+$,'"=:;~]+)\[EL]
```

Quite a mess, isn't it? Squint at it for a little while, and you'll see it achieves exactly what we need it to, capturing both the URL (`\\1`) and the text (`\\2`) for the link. The PHP code that performs the substitution is as follows:

```
$joketext = ereg_replace(
  '\[L=([-_./a-zA-Z0-9!&%#?+,\'=:~]+)]'.
  '([-_./a-zA-Z0-9 !&%#?+$,\'"=:;~]+)\[EL]',
  '<a href="\\1">\\2</a>', $joketext);
```

Matching Tags

A nice side-effect of the regular expressions we developed to read hyperlinks is that they will only find matched pairs of `[L]` and `[EL]` tags. A `[L]` tag missing its `[EL]` or vice versa will not be detected, and will appear unchanged in the finished document, allowing the person updating the site to spot the error and fix it.

In contrast, the PHP code we developed for boldface and italic text in "Boldface and Italic Text" will convert unmatched`[B]` and `[I]` tags into unmatched HTML tags! This can easily lead to ugly situations like the entire text of a joke starting from an unmatched tag being displayed in bold-possibly even spilling into subsequent content on the page.

We can rewrite our code for bold/italic text in the same style as we used for hyperlinks to solve this problem by only processing matched pairs of tags:

```
$joketext = eregi_replace(
  '\[b]([-_./a-zA-Z0-9 !&%#?+$,\'"=:;~]+)\[eb]',
  '<strong>\\1</strong>',$joketext);
$joketext = eregi_replace(
  '\[i]([-_./a-zA-Z0-9 !&%#?+$,\'"=:;~]+)\[ei]',
  '<em>\\1</em>',$joketext);
```

If unmatched tags aren't much of a concern for you, however, you can actually simplify your code by not using regular expressions at all! PHP's `str_replace` function works a lot like `ereg_replace`, except that it only searches for strings-not patterns.

```
$newstr = str_replace(searchfor,replacewith,oldstr);
```

We can therefore rewrite our bold/italic code as follows:

```
$joketext = str_replace('[b]','<strong>',$joketext);
$joketext = str_replace('[eb]','</strong>',$joketext);

$joketext = str_replace('[i]','<em>',$joketext);
$joketext = str_replace('[ei]','</em>',$joketext);
```

One difference remains between this and our regular expression code. We used `eregi_replace` in our previous code to match both lowercase `[b]` and uppercase `[B]` tags, as that function was case-insensitive.`str_replace` is case sensitive, so we need to make a further modification to allow uppercase tags:

```
$joketext = str_replace(
  array('[b]','[B]'),'<strong>',$joketext);
$joketext = str_replace(
  array('[eb]','[EB]'),'</strong>',$joketext);
```

```
$joketext = str_replace(
  array('[i]','[I]'),'<em>',$joketext);
$joketext = str_replace(
  array('[ei]','[EI]'),'</em>',$joketext);
```

`str_replace` lets you give an array for the search string, so the above code will replace either `[b]` or `[B]` with `<strong>`, `[eb]` or `[EB]` with `</strong>`, and so on. For more information about the intricacies of str_replace, refer to the PHP manual.

While this code looks more complicated than the original version with `eregi_replace`, `str_replace` is a lot more efficient because it doesn't need to interpret your search string for regular expression codes. Whenever `str_replace` can do the job, you should always use it instead of `ereg_replace` or `eregi_replace`.

The *joke.php* file included in the code archive makes use of `str_replace`; feel free to replace it with the regular expression code above if you are worried about unmatched tags.

[2] You may be more accustomed to using `<b>` and `<i>` tags; however, I have chosen to respect the most recent HTML standards, which recommend replacing these with `<strong>` and `<em>`, respectively.

[3] Experienced developers may object to this use of regular expressions. Yes, regular expressions are not required for this simple example, and yes, a single regular expression for both tags would be more appropriate than two separate expressions. I'll address both of these issues later in this chapter.

[4] I have not included a space in the list of characters I want to allow in a link URL. Although Microsoft Internet Explorer supports such URLs, spaces in the path or file name portions of a URL should be replaced with the code `%20`, and spaces in the query string should be replaced by +. If you want to allow spaces in your URLs, feel free to add a space to the list of characters in square brackets.

Splitting Text into Pages

While no joke is likely to be so long that it will require more than one page, many content-driven sites (like sitepoint.com) provide lengthy content that is best presented when it's broken into pages. Yet another regular expression function in PHP makes this exceedingly easy to do.

`split` is a function that takes a regular expression and a string of text, and uses matches for the regular expression to break the text into an array. Consider the following example:

```
$regexp="[ \n\t\r]+"; // One or more white space chars
$text="This is a\ntest.";
$textarray=split($regexp,$text);
echo($textarray[0]."<br />"); // Outputs "This<br />"
echo($textarray[1]."<br />"); // Outputs "is<br />"
echo($textarray[2]."<br />"); // Outputs "a<br />"
echo($textarray[3]."<br />"); // Outputs "test.<br />"
```

As you might expect, there is also a `spliti` function that is case insensitive.

If we search for a `[PAGEBREAK]` tag instead of a white space character, and we display only the page in which we're interested (indicated by a `$page` variable passed with the page request, for example) instead of all of the resulting portions of the text, we can successfully divide our content into pages.[5]

```
// If no page specified, default to the
// first page ($page = 0)
if (!isset($_GET['page'])) $page = 0;
else $page = $_GET['page'];

// Split the text into an array of pages
$textarray=spliti("\[PAGEBREAK]",$text);

// Select the page we want
$pagetext=$textarray[$page];
```

Obviously, we'll want to provide a way for users to move between pages. Let's put a link to the previous page at the top of the current page, and a link to the next page at the bottom.

However, if this is the first page, clearly we won't need a link to the previous page. We know we're on the first page if `$page` equals zero.

Likewise, we don't need a link to the next page on the last page of content. To detect the last page, we need to use the `count` function that I introduced briefly in "A Content Management System". `count` takes an array, and returns the number of elements in that array. When `count` is passed our array of pages, it will tell us how many pages there are. If there are 10 pages, then `$textarray[9]` will contain the last page. Thus, we know we're on the last page if `$page` equals `count($textarray)` minus one.

The code for the links that will turn our pages looks like this:

```
$PHP_SELF = $_SERVER['PHP_SELF'];

if ($page != 0) {
  $prevpage = $page - 1;
  echo("<p><a href=\"$PHP_SELF?id=$id&page=$prevpage\">".
       'Previous Page</a></p>');
}

// Output page content here...

if ($page < count($textarray) - 1) {
```

```
  $nextpage = $page + 1;
  echo("<p><a href=\"$PHP_SELF?id=$id&page=$nextpage\">".
       'Next Page</a></p>');
}
```

[5]The real reason for using regular expressions here is to allow `[PAGEBREAK]` to be case insensitive; that is, we want `[pagebreak]` or even `[Pagebreak]` to work just as well. If you are happy with requiring the tag to be typed in uppercase, you can actually use PHP's `explode` function instead. It works just like `split`, but it searches for a specific string rather than a pattern defined by a regular expression. Unlike `str_replace` (see "Matching Tags"),`explode` cannot accept an array as its search argument. See the PHP manual for details.

Putting it all Together

The completed code that will output our joke text (with all special character, multi-page, and custom tag conversion in place) is as follows. This file, along with an updated joke listing script (*jokelist.php*), and a front page that lets our visitors choose a joke category to view (*index.php*), are provided in the code archive.

```
<!-- joke.php -->
...
// Get the joke text from the database
$id = $_GET['id'];
$joke = mysql_query("SELECT JokeText FROM Jokes
                     WHERE ID='$id'");
$joke = mysql_fetch_array($joke);
$joketext = $joke['JokeText'];

// Filter out HTML code
$joketext = htmlspecialchars($joketext);

// If no page specified, default to the
// first page ($page = 0)
if (!isset($_GET['page'])) $page = 0;
else $page = $_GET['page'];

// Split the text into an array of pages
$textarray=spliti('\[PAGEBREAK]',$joketext);

// Select the page we want
$joketext=$textarray[$page];

// Bold and italics
$joketext = str_replace(
  array('[b]','[B]'),'<strong>',$joketext);
$joketext = str_replace(
  array('[eb]','[EB]'),'</strong>',$joketext);
$joketext = str_replace(
  array('[i]','[I]'),'<em>',$joketext);
$joketext = str_replace(
  array('[ei]','[EI]'),'</em>',$joketext);

// Paragraphs and line breaks
$joketext = ereg_replace("\r",'',$joketext);
$joketext = ereg_replace("\n\n",'</p><p>',$joketext);
$joketext = ereg_replace("\n",'<br />',$joketext);

// Hyperlinks
$joketext = ereg_replace(
  '\[L]([-_./a-zA-Z0-9!&%#?+,\'=:~]+)\[EL]',
  '<a href="\\1">\\1</a>', $joketext);
$joketext = ereg_replace(
  '\[L=([-_./a-zA-Z0-9!&%#?+,\'=:~]+)]'.
  '([-_./a-zA-Z0-9 !&%#?+$,\'"=:;~]+)\[EL]',
  '<a href="\\1">\\2</a>', $joketext);

$PHP_SELF = $_SERVER['PHP_SELF'];
```

```
if ($page != 0) {
  $prevpage = $page - 1;
  echo("<p><a href=\"$PHP_SELF?id=$id&page=$prevpage\">".
       'Previous Page</a></p>');
}

echo( "<p>$joketext</p>" );

if ($page < count($textarray) - 1) {
  $nextpage = $page + 1;
  echo("<p><a href=\"$PHP_SELF?id=$id&page=$nextpage\">".
       'Next Page</a></p>');
}

...
```

Don't forget to provide documentation so that users of your joke submission form know what tags are available and what each of them does.

Automatic Content Submission

It seems a shame to have spent so much time and effort on a content management system that's so easy that anyone can use it, if the only people who are actually *allowed* to use it are the site administrators. Furthermore, while it's extremely convenient for an administrator not to have to edit HTML to make updates to the site's content, he or she must still transcribe submitted documents into the "Add New Joke" form, and convert any formatted text into the custom formatting language we developed above-a tedious and mind-numbing task to say the least.

What if we put the "Add New Joke" form in the hands of casual site visitors? If you recall, we actually did this in "Publishing MySQL Data on the Web" when we provided a form for users to submit their own jokes. At the time, this was simply a device that demonstrated how `INSERT` statements could be made from within PHP scripts. We did not include it in the code we developed from scratch in this chapter because of the inherent security risks involved. After all, who wants to open the content of his or her site for just anyone to tamper with?

But new joke submissions don't have to appear on the site immediately. What if we added a new column to the Jokes table called Visible that could take one of two values: `Y` and `N`. Newly submitted jokes could automatically be set to `Visible='N'`, and could be prevented from appearing on the site if we simply add`WHERE Visible='Y'` to any query of the Jokes table for which the results are intended for public access. Jokes with `Visible='N'` would wait in the database for review by a content manager, who could edit each joke before making it visible, or deleting it out of hand.

To create a column that contains one of two values, of which one is the default, we'll need a new MySQL column type called `ENUM`:

```
mysql>ALTER TABLE Jokes ADD COLUMN
    ->Visible ENUM('N','Y') NOT NULL;
```

Since we declared this column as required (`NOT NULL`), the first value listed in the parentheses (`'N'` in this case) is the default value, which is assigned to new entries if no value is specified in the `INSERT` statement. All that's left for you to do is modify the administration system to allow hidden jokes to be shown. A simple check box in the 'Add Joke' and 'Edit Joke' forms should do the trick.

With new jokes hidden from the public eye, the only security detail that remains is author identification. We want to be able to identify which author in the database submitted a particular joke, but it's inappropriate to rely on the old drop-down list of authors in the "Add New Joke" form, since any author could pose as any other. Obviously, some sort of user name/password authentication scheme is required.

To store a password in the Authors table, simply add another column. You can then require an author to correctly enter his or her email address and password when they submit a joke to the database. You'd want to implement the same login procedure before you allow an author to modify his or her details (name, email address, etc.). You might even like to give each author a "control centre" of sorts, where he or she could view the status of the jokes he or she has submitted to the site.

Summary

While it would be interesting to delve into the details of the content-submission system described above, you should already have all the skills necessary to build it yourself. Want to let users rate the jokes on the site? How about letting joke authors make changes to their jokes, but requiring an administrator to approve the changes before they go live on the site? The power and complexity of the system is limited only by your imagination.

At this point, you should be equipped with all the basic skills and concepts you need to build your very own database-driven Website. In the rest of this book, I'll cover more advanced topics that will help optimize your site's performance. Oh, and of course we'll explore more exciting features of PHP and MySQL.

In"MySQL Administration", we'll take a step away from our joke database and have a close-up look at MySQL server maintenance and administration. We'll learn how to make backups of our database (a critical task for any Web-based company!), administer MySQL users and their passwords, and log into a MySQL server if you've forgotten your password.

Chapter 8: MySQL Administration

Overview

At the core of any well-designed, content-driven site is a relational database. In this book, we've used the MySQL Relational Database Management System (RDBMS) to create our database. MySQL is a popular choice among Web developers not only because it's free for non-commercial use on all platforms, but also because it's fairly simple to activate a MySQL server. As I demonstrated in "Installation", armed with proper instructions, a new user can get a MySQL server up and running in less than 30 minutes, or under 10 if you practice a little!

If all you want to do is have a MySQL server around so you can play with a few examples and experiment a little, then the initial installation process we went through in "Installation" is likely to be all you'll need. If, on the other hand, you want to set up a database backend to a real, live Website-perhaps a site upon which your company depends-then there are a few more things you'll need to learn how to do before you can rely on a MySQL server day-in and day-out.

Backups of data that's important to you or your business should be part of any Internet-based enterprise. Unfortunately, because setting up backups isn't the most interesting part of an administrator's duties, such procedures are usually arranged once out of necessity and deemed "good enough" for all applications. If your answer to "Should we back up our databases?" until now has been "It's okay; they'll be backed up along with everything else," then you really should stick around. I'll show you why a generic file backup solution is inadequate for many MySQL installations, and I'll demonstrate the "right way" to back up and restore a MySQL database.

In"Installation", we set up the MySQL server so that you could connect as `root` with a password you chose. This `root` MySQL user (which, incidentally, has nothing to do with the root user on Linux and similar systems) had read/write access to all databases and tables. In many organizations, it's necessary to create users whose access is limited to particular databases and tables, and to then restrict that access in some way (e.g. read-only access to a particular table). In this chapter, we'll learn how to facilitate such restrictions using two new MySQL commands: `GRANT` and `REVOKE`.

In some situations, such as power outages, MySQL databases can become damaged. Such damage need not always send you scrambling for your backups, however. We'll finish off our review of MySQL database administration by learning how to use the MySQL database check and repair utility to fix simple database corruptions.

Backing up MySQL Databases

Standard Backups Aren't Enough

Like Web servers, most MySQL servers are expected to remain online 24 hours a day, 7 days a week. This makes backups of MySQL database files problematic. Because the MySQL server uses memory caches and buffers to improve the efficiency of updates to the database files stored on disk, these files may be in an inconsistent state at any given time. Since standard backup procedures involve merely copying system and data files, backups of MySQL data files cannot be relied upon, as they can't guarantee that the files that are copied are in a fit state to be used as replacements in the event of a crash.

Furthermore, as many databases receive new information at all hours of the day, standard backups can provide only 'snapshots' of database data. Any information stored in the database that's changed after the last backup will be lost in the event that the MySQL data files are destroyed or become unusable. In many situations, such as when a MySQL server is used to track customer orders on an ecommerce site, this is an unacceptable loss.

Facilities exist in MySQL to keep up-to-date backups that are not adversely affected by server activity at the time at which the backups are generated. Unfortunately, they require you to set up a backup scheme specifically for your MySQL data, completely apart from whatever backup measures you have established for the rest of your data. As with any good backup system, however, you'll appreciate it when the time comes to use it.

In this chapter, the instructions I'll provide will be designed for use on a computer running Linux, or some other UNIX-based operating system. If you're running your MySQL server under Windows, the methods and advice provided here will all apply equally well, but you'll have to come up with some of the specific commands yourself. If you have any trouble, don't hesitate to post your questions in the SitePoint Forums.

Database Backups using *mysqldump*

In addition to *mysqld*, the MySQL server, and *mysql*, the MySQL client, a MySQL installation comes with many useful utility programs. We have seen *mysqladmin*, which is responsible for the control and retrieval of information about an operational MySQL server, for example.

mysqldump is another such program. When run, it connects to a MySQL server (in much the same way as the*mysql* program or the PHP language does) and downloads the complete contents of the database you specify. It then outputs these as a series of SQL `CREATE TABLE` and `INSERT` commands that, if run in an empty MySQL database, would create a MySQL database with exactly the same contents as the original.

If you redirect the output of *mysqldump* to a file, you can store a 'snapshot' of the database as a backup. The following command (typed all on one line) connects to the MySQL server running on `myhost` as user `root` with password `mypass`, and saves a backup of the database called dbname into the file *dbname_backup.sql*:

```
shell%mysqldump -h myhost -u root -pmypass dbname >
dbname_backup.sql
```

To restore this database after a server crash, you would use these commands:

```
shell%mysqladmin -h myhost -u root -pmypass create dbname
shell%mysql -h myhost -u root -pmypass dbname < dbname_backup.sql
```

The first command uses the *mysqladmin* program to create the database; alternatively, you can do this at the MySQL command line. The second connects to the MySQL server using the usual *mysql* program,

and feeds in our backup file as the commands to be executed.

In this way, we can use *mysqldump* to create backups of our databases. *mysqldump* connects through the MySQL server to perform backups, rather than by directly accessing the database files in the MySQL data directory. So the backup it produces is guaranteed to be a valid copy of the database, and not a snapshot of the database files, which may be in a state of flux as long as the MySQL server is online.

But how do we bridge the gap between these snapshots to maintain a backup of a database that is always up to date? The solution is simple: instruct the server to keep an update log.

Incremental Backups using Update Logs

As I mentioned above, many situations in which MySQL databases are used would make the loss of data-any data-unacceptable. In cases like these, we need some way to bridge the gap between the backups we made using *mysqldump* as described above. The solution is to instruct the MySQL server to keep an update log. An update log is a record of all SQL queries that were received by the database, and which modified the contents of the database in some way. This includes `INSERT`,`UPDATE`, and `CREATE TABLE` statements (among others), but doesn't include `SELECT` statements.

The basic idea is that you can restore the contents of the database at the very moment at which a disaster occurred, with the application of a backup (made using *mysqldump*), followed by the application of the contents of the update logs that were generated after that backup was made.

You can also edit update logs to undo mistakes that may have been made. For example, if a co-worker comes to you after having accidentally issued a `DROP TABLE` command, you can edit your update log to remove that command before you restore your database using your last backup and the log application. In this way, you can even keep changes to other tables that were made *after* the accident. And, as a precaution, you should probably also revoke your co-worker's `DROP` privileges (see the next section to find out how).

To tell the MySQL server to keep update logs, simply add an option to the server command line:

```
shell%safe_mysqld --log-update=update
```

The above command starts the MySQL server and tells it to create files named *update.001*,*update.002*, and so on, in the server's data directory (*/usr/local/mysql/var* if you set up the server according to the instructions in "Installation"). A new file will then be created each time the server flushes its log files; in practice, this occurs whenever the server is restarted. If you want to store your update logs somewhere else (usually a good idea-if the disk that contains your data directory dies, you don't want it to take your backups with it!), you can specify the full path to the update files.

However, if you run your MySQL server full time, you probably have your system set up to launch the MySQL server at start-up. The addition of command-line options to the server can be difficult in this case. A simpler way to have update logs created is to add the option to the MySQL configuration file, *my.cnf*, which you should have created in your system's */etc* directory as part of the procedure to auto-start the MySQL server (see "Installation"). To set MySQL to create update logs by default, simply add a `log-update` line below `[mysqld]` in your *my.cnf* file. For Windows users, the *my.cnf* file should be located in the root of your C: drive; alternatively, it may be named *my.ini* and placed in your Windows directory.

```
[mysqld]
log-update=/usr/backups/mysql/update
```

Feel free to specify whatever location to which you'd like the server to write the update logs. Save the file and restart your MySQL server. From now on, the server will behave by default as if you'd specified the `--log-update` option on the command line.

Obviously, update logs can take up a lot of space on an active server. For this reason, and because

MySQL will not automatically delete old log files as it creates new ones, it's up to you to manage your update log files. The following UNIX shell script, for example, tells MySQL to flush its log files, and then deletes all update files that were last modified more than a week ago.

```
#!/bin/sh

/usr/local/mysql/bin/mysqladmin -u root -pmypasswd \
  flush-logs
find /usr/backups/mysql/ -name "update.[0-9]*" \
  -type f -mtime +6 | xargs rm -f
```

This first step (flushing the log files) creates a new update log in case the current one is about to be deleted. This deletion will occur if the server has been online, and has not received any queries that changed database contents, for over a week. If you're an experienced user, setting up a script that uses cron[1] or Windows' Task Scheduler to periodically (say, once a week) perform a database backup and delete old update logs should be fairly easy. If you need a little help with this, speak to your Web host, system administrator, or local guru, or post a message to the SitePoint Forums (we'll be glad to help!).

If you have a backup and a copy of the update logs since the backup was made, then the restoration of your database should be fairly simple. After you create the empty database and apply the backup as described in the previous section, apply the update logs, using the `--one-database` command-line option for *mysql*. This command instructs the server to run only those queries in the update log that pertain to the database you want to restore (db in this example):

```
shell%mysql -u root -pmypasswd --one-database db < update.100
shell%mysql -u root -pmypasswd --one-database db < update.102
...
```

[1]cron is a well-known task scheduling utility available on most Linux and UNIX-based systems. To learn how to set up cron tasks, begin by typing `man crontab` at your server's command prompt.

MySQL Access Control

Early in this book, I mentioned that the database called mysql, which appears on every MySQL server, is used to keep track of users, their passwords, and what they're allowed to do. Until now, however, we've always logged into the server as the `root` user, which gives us access to all databases and tables.

If your MySQL server will only be accessed through PHP, and you're careful about who is given the password to the `root` MySQL account, then the `root` account may be sufficient for your needs. However, in cases where a MySQL server is shared among many users, for example, if a Web host wishes to use a single MySQL server to provide a database to each of its users, it's usually a good idea to set up user accounts with more restricted access.

The MySQL access control system is fully documented in Chapter 6 of the MySQL Reference Manual. In essence, user access is governed by the contents of five tables in the mysql database: user, db, host, tables_priv, and columns_priv. If you plan to edit these tables directly using `INSERT`,`UPDATE`, and `DELETE` statements, I'd suggest you read the section of the MySQL manual on the subject first. For us mere mortals, MySQL provides a simpler method to manage user access. Using `GRANT` and `REVOKE`—nonstandard commands provided by MySQL—you can create users and set their privileges without worrying about the details of how they'll be represented in the tables mentioned above.

Using `GRANT`

The `GRANT` command, used to create new users, assign user passwords, and add user privileges, looks like this:

```
mysql>GRANTprivileges [(columns)] ON what
    ->TOuser [IDENTIFIED BY "password"]
    ->[WITH GRANT OPTION];
```

As you can see, there are a lot of blanks to be filled in with this command. Let's describe each of them in turn, and then look at some examples to give you an idea of how they work together.

`privileges` is a comma-separated list of the privileges you wish to grant. The privileges you can specify can be sorted into three groups:

- *Database/Table/Column privileges*

  `ALTER`
  : Modify existing tables (e.g. add/remove columns) and indexes.

  `CREATE`
  : Create new databases and tables.

  `DELETE`
  : Delete table entries.

  `DROP`
  : Delete tables and/or databases.

  `INDEX`
  : Create and/or delete indexes.

  `INSERT`
  : Add new table entries.

  `SELECT`
  : View/search table entries.

  `UPDATE`
  : Modify existing table entries.

- *Global administrative privileges*

`FILE`
: Read and write files on the MySQL server machine.

`PROCESS`
: View and/or kill server threads that belong to other users.

`RELOAD`
: Reload the access control tables, flush the logs, etc.

`SHUTDOWN`
: Shut down the MySQL server.

- *Special privileges*

`ALL`
: Allowed to do anything (like root).

`USAGE`
: Only allowed to log in—nothing else.

Some of these privileges apply to features of MySQL that we have not yet seen, but many should be familiar to you.

`what` defines the areas of the database sever to which the privileges apply. `*.*` means the privileges apply to all databases and tables. *`dbName`*`.*` means the privileges apply to all tables in the database called dbName. *`dbName.tblName`* means the privileges apply only to the table called tblName in the database called dbName. You can even specify privileges for individual table columns—simply list the columns between the parentheses that follow the privileges to be granted (we'll see an example of this in a moment).

`user` specifies the user to which these privileges should apply. In MySQL, a user is specified both by the user name given at login, and the host name/IP address of the machine from which the user connects. The two values are separated by the `@` sign (i.e. *`username@hostname`*). Both values may contain the `%` wild card character, but you need to put quotes around any value that does (e.g. `kevin@"%"` will allow the user name `kevin` to log in from any host and use the privileges you specify).

`password` specifies the password required by the user to connect to the MySQL server. As indicated by the square brackets above, the `IDENTIFIED BY "`*`password`*`"` portion of the `GRANT` command is optional. Any password specified will replace the existing password for that user. If no password is specified for a new user, a password will not be required to connect.

The optional `WITH GRANT OPTION` portion of the command specifies that the user be allowed to use the `GRANT`/`REVOKE` commands to give to another user any privileges granted to him or her. Be careful with this option—the repercussions are not always obvious! For example, two users who have this option enabled can get together and share their privileges with each other.

Let's consider a few examples. To create a user named `dbmgr` that can connect from `server.host.net` with password `managedb` and have full access to the database named db only (including the ability to grant access to that database to other users), use this `GRANT` command:

```
mysql>GRANT ALL ON db.*
    ->TO dbmgr@server.host.net
    ->IDENTIFIED BY "managedb"
    ->WITH GRANT OPTION;
```

To subsequently change that user's password to `funkychicken`, use:

```
mysql>GRANT USAGE ON *.*
    ->TO dbmgr@server.host.net
    ->IDENTIFIED BY "funkychicken";
```

Notice that we haven't granted any additional privileges (the USAGE privilege doesn't let a user do anything besides log in), but the user's existing privileges remain unchanged.

Now let's create a new user named `jess`, who will connect from various machines in the `host.net` domain. Say she's responsible for keeping the names and email addresses of users in the database up to date, but may need to refer to other database information at times. As a result, she will have read-only (i.e. SELECT) access to the db database, but will be able to UPDATE the name and email columns of the Users table. Here are the commands:

```
mysql>GRANT SELECT ON db.*
    ->TO jess@"%.host.net"
    ->IDENTIFIED BY "jessrules";
mysql>GRANT UPDATE (name,email) ON db.Users
    ->TO jess@"%.host.net";
```

Notice in the first command how we used the % (wild card) character in the host name to indicate the host from which Jess could connect. Notice also that we haven't given her the ability to pass her privileges to other users, as we didn't put `WITH GRANT OPTION` on the end of the command. The second command demonstrates how to grant privileges for specific table columns—it lists the column(s) separated by commas in parentheses after the privilege(s) being granted.

Using REVOKE

TheREVOKE command, as you'd expect, is used to strip previously granted privileges from a user. The syntax for the command is as follows:

```
mysql>REVOKEprivileges [(columns)]
    ->ONwhat FROM user;
```

All the fields in this command work just as they do in GRANT above. To revoke the DROP privileges of a co-worker of Jess's (for instance, if he or she has demonstrated a habit of occasionally deleting tables and databases by mistake), you would use this command:

```
mysql>REVOKE DROP ON *.* FROM idiot@"%.host.net";
```

Revoking a user's login privileges is about the only thing that can't be done using GRANT and REVOKE. `REVOKE ALL ON *.*` would definitely prevent a user from doing anything of consequence besides logging in, but to remove a user completely requires that you delete the corresponding entry in the user table:

```
mysql>DELETE FROM user
    ->WHERE User="idiot" AND Host="%.host.net";
```

Access Control Tips

As a result of the way the access control system in MySQL works, there are a couple of idiosyncrasies that you should be aware of before you launch into user creation.

When you create users that can log into the MySQL server only from the computer on which that server is running (i.e. you require them to use Telnet or SSH to log into the server and run the MySQL client from there, or communicate using server-side scripts like PHP), you may ask yourself what the *user* part of the GRANT command should be. Say the server is running on `www.host.net`. Should you set up the user as

username@www.host.net, or *username*@localhost?

The answer is that you can't rely on either one to handle all connections. In theory, if, when connecting, the user specifies the host name either with the *mysql* client, or with PHP's `mysql_connect` function, that host name will have to match the entry in the access control system. But as you probably don't want to force your users to specify the host name a particular way (in fact, users of the *mysql* client probably won't want to specify the host name at all), it's best to use a work-around.

For users that need to be able to connect from the same machine on which the MySQL server is running, it's best to create two user entries in the MySQL access system: one with the actual host name of the machine (e.g. *username*@www.host.net), the other with `localhost` (e.g. *username*@localhost). Of course, you will have to grant/revoke all privileges to both of these user entries individually, but this is the only work-around that you can really rely upon.

Another common problem faced by MySQL administrators is that user entries with wild cards in their host names (e.g. `jess@%.host.net` above) fail to work. When this happens, it's usually due to the way MySQL prioritizes the entries in the access control system. Specifically, it orders entries so that more specific host names appear first (e.g. `www.host.net` is completely specific, `%.host.net` is less specific, and % is totally unspecific).

In a fresh installation, the MySQL access control system contains two anonymous user entries (which allow connections from the local host that use any user name—the two entries are to support connections from `localhost` and the server's actual host name [2], as described above), and two `root` user entries. The problem described above occurs when the anonymous user entries take precedence over our new entry because their host name is more specific.

Let's look at the abridged contents of the user table on `www.host.net`, our fictitious MySQL server, after we add Jess's entry. The rows here are sorted in the order in which the MySQL server considers them when it validates a connection:

```
+--------------+------+-------------------+
| Host         | User | Password          |
+--------------+------+-------------------+
| localhost    | root | (encrypted value) |
| www.host.net | root | (encrypted value) |
| localhost    |      |                   |
| www.host.net |      |                   |
| %.host.net   | jess | (encrypted value) |
+--------------+------+-------------------+
```

As you can see, since Jess's entry has the least specific host name, it comes last in the list. When Jess attempts to connect from `www.host.net`, the MySQL server matches her connection attempt to one of the anonymous user entries (a blank `User` value matches anyone). Since these anonymous entries don't require a password, and presumably Jess enters her password, MySQL rejects the connection attempt. Even if Jess managed to connect without a password, she would be given the very limited privileges that are assigned to anonymous users, as opposed to the privileges assigned to her entry in the access control system.

The solution to this problem is to either make your first order of business as a MySQL administrator the deletion of those anonymous user entries (`DELETE FROM mysql.user WHERE User=""`), or to give two more entries to all users who need to connect from `localhost` (i.e. entries for `localhost` and the actual host name of the server):

```
+--------------+------+-------------------+
| Host         | User | Password          |
+--------------+------+-------------------+
| localhost    | root | (encrypted value) |
| www.host.net | root | (encrypted value) |
| localhost    | jess | (encrypted value) |
```

```
| www.host.net | jess | (encrypted value) |
| localhost    |      |                   |
| www.host.net |      |                   |
| %.host.net   | jess | (encrypted value) |
+--------------+------+-------------------+
```

Since it's excessive to maintain three user entries (and three sets of privileges) for each user, I recommend that you remove the anonymous users, unless you have a particular need for them:

```
+--------------+------+-------------------+
| Host         | User | Password          |
+--------------+------+-------------------+
| localhost    | root | (encrypted value) |
| www.host.net | root | (encrypted value) |
| %.host.net   | jess | (encrypted value) |
+--------------+------+-------------------+
```

Locked Out?

Like locking your keys in the car, forgetting your password after you've spent an hour installing and tweaking a new MySQL server can be an embarrassment to say the least. Fortunately, if you have root access to the computer on which the MySQL server is running, or if you can log in as the user you set up to run the MySQL server (`mysqlusr` if you followed the instructions in "Installation"), all is not lost. This next procedure will let you regain control of the server.

First, you must shut down the MySQL server. Since you would normally do this using `mysqladmin`, which requires your forgotten password, you'll instead have to kill the server process to shut it down. Under Windows, use the task manager to find and end the MySQL process, or simply stop the MySQL service if you have installed it as such. Under Linux, use the `ps` command, or look in the server's PID file, in the MySQL data directory, to determine the process ID of the MySQL server, and then terminate it with this command:

```
shell%kill pid
```

`pid` is the process ID of the MySQL server. This should be enough to stop the server. Do *not* use `kill -9` unless absolutely necessary, as this may damage your table files. If you're forced to do so, however, the next section provides instructions on how to check and repair those files.

Now that the server's down, you can restart it by running *safe-mysqld* (*mysqld-opt.exe*,*mysqld-nt.exe*, or whichever server executable you decided on under Windows) with the `--skip-grant-tables` command line option. This instructs the MySQL server to allow unrestricted access to anyone. Obviously, you'll want to run the server in this mode as infrequently as possible, to avoid the inherent security risks.

Once you're connected, change your root password to something you'll remember:

```
mysql>USE mysql;
mysql>UPDATE user SET Password=PASSWORD("newpassword")
    ->WHERE User="root";
```

Finally, disconnect, and instruct the MySQL server to reload the grant tables to begin requiring passwords:

```
shell%mysqladmin flush-privileges
```

That does it—and nobody ever has to know what you did. As for locking your keys in your car, you're on

your own there.

[2] On Windows installations of MySQL, the second entry's hostname is set to %, not the server's hostname. It therefore does not contribute to the problem described here. It does, however, permit connections with any user name from any computer, so it's a good idea to delete it anyway.

Checking and Repairing MySQL Data Files

In power outages, situations where you need to `kill -9` the MySQL server process, or when Jess's friend`idiot@%.host.net` kicks the plug out of the wall, there is a risk that the MySQL data files may be damaged. This situation can arise if the server is in the middle of making changes to the files at the time of the disturbance, as the files may be left in a corrupt or inconsistent state. Since this type of damage can be subtle, it can go undetected for days, weeks, or even months. As a result, by the time you do finally discover the problem, all your backups may contain the same corruption.

Chapter 4 of the MySQL Reference Manual describes the *myisamchk* utility that comes with MySQL, and how to use it to check and repair your MySQL data files. While that chapter is recommended reading for anyone who wants to set up a heavy-duty preventative maintenance schedule for their MySQL server, we'll cover all the essentials here.

Before we go any further, though, it's important to realize that the *myisamchk* program expects to have sole access to the MySQL data files that it checks and modifies. If the MySQL server works with the files at the same time, and makes a modification to a file that *myisamchk* is in the middle of checking, *myisamchk* might incorrectly detect an error and try to fix it-which in turn could trip up the MySQL server! Thus, to avoid making things worse instead of better, it's usually a good idea to shut down the MySQL server while you're working on the data files. Alternatively, shut down the server just long enough to make a copy of the files, and then do the work on the copies. When you're done, shut down the server again briefly to replace the files with the new ones, and perhaps apply any update logs that were made in the interim.

The MySQL data directory isn't too difficult to understand. It contains a sub-directory for each database, and each of these sub-directories contains the data files for the tables in the corresponding database. Each table is represented by three files, which have the same name as the table, but three different extensions. The *tblName.frm* file is the table definition, which keeps track of which columns are contained in the table, and their type. The *tblName.MYD* file contains all the table data. The *tblName.MYI* file contains any indexes for the table. For example, it might contain the look-up table that helps the table's primary key column speed up queries that are based on this table.

To check a table for errors, just run *myisamchk* (in the MySQL *bin* directory) and provide either the location of these files and the name of the table, or the name of the table index file:

```
shell%myisamchk /usr/local/mysql/var/dbName/tblName
shell%myisamchk /usr/local/mysql/var/dbName/tblName.MYI
```

Either of the above will perform a check of the specified table. To check all tables in the database, use a wild card:

```
shell%myisamchk /usr/local/mysql/var/dbName/*.MYI
```

And to check all databases in all tables, use two:

```
shell%myisamchk /usr/local/mysql/var/*/*.MYI
```

Without any options, *myisamchk* performs a normal check of the table files. If you suspect problems with a table and a normal check fails to turn up anything, you can perform a much more thorough (but also much slower!) check using the `--extend-check` option:

```
shell%myisamchk --extend-check /path/to/tblName
```

Checking for errors is non-destructive, which means that you don't have to worry that you might make an existing problem worse if you perform a check on your data files. Repair operations, on the other hand, while usually safe, will make changes to your data files that cannot be undone. For this reason, I strongly recommend that you make a copy of any damaged table files before you attempt to repair them. As usual, make sure your MySQL server is shut down before you make copies of live data files.

There are three types of repair that you can use to fix a problem with a damaged table. These should be tried in order with fresh copies of the data files each time (i.e. don't try the second recovery method on a set of files that result from a failed attempt of the first recovery method). If at any point you get an error message that indicates that a temporary file can't be created, delete the file to which the message refers and try again-the offending file is a remnant of a previous repair attempt.

The three repair methods can be executed as follows:

```
shell%myisamchk --recover --quick /path/to/tblName
shell%myisamchk --recover /path/to/tblName
shell%myisamchk --safe-recover /path/to/tblName
```

The first is the quickest, and fixes the most common problems; the last is the slowest, and fixes a few problems that the other methods do not.

If these methods fail to resurrect a damaged table, there are a couple more tricks you can try before you give up:

- If you suspect that the table index file (**.MYI*) is damaged beyond repair, or even missing entirely, it can be regenerated from scratch and used with your existing data (**.MYD*) and table form (**.frm*) files. To begin, make a copy of your table data (*tblName.MYD*) file. Restart your MySQL server and connect to it, then delete the contents of the table with the following command:

  ```
  mysql>DELETE FROM tblName;
  ```

  This command doesn't just delete the contents of your table; it also creates a brand new index file for that table. Log out and shut down the server again, then copy your saved data file (*tblName.MYD*) over the new (empty) data file. Finally, perform a standard repair (the second method above), and use *myisamchk* to regenerate the index data based on the contents of the data and table form files.

- If your table form file (*tblName.frm*) is missing or damaged beyond repair, but you know the table well enough to reproduce the CREATE TABLE statement that defines it, you can generate a new *.frm* file and use it with your existing data file and index file. If the index file is no good, use the above method to generate a new one afterwards. First, make a copy of your data and index files, then delete the originals, and remove any record of the table from the data directory.

  Start up the MySQL server and create a new table using the exact same CREATE TABLE statement. Log out and shut down the server, then copy your two saved files over top of the new, empty files. The new*.frm* file should work with them, but perform a standard table repair-the second method above-for good measure.

Summary

Admittedly this chapter hasn't been the usual nonstop, action-packed code-fest that you may have become accustomed to by now. But our concentration on these topics—the back up and restoration of MySQL data, the administration of the MySQL access control system, and table checking and repair—has armed us with the tools we'll need in order to set up a MySQL database server that will stand the test of time, not to mention the constant traffic that your site will endure during that period.

In"Advanced SQL", we'll get back to the fun stuff and learn some advanced SQL techniques that make a relational database server do things that you may never have thought possible.

Chapter 9: Advanced SQL

As we worked through our example of the Internet Joke Database Website, we had opportunities to explore most aspects of Structured Query Language (SQL). From the basic form of a `CREATE TABLE` query, to the two syntaxes of `INSERT` queries, you probably know many of these commands by heart now.

In this chapter, in an effort to tie up loose ends, we'll look at a few more SQL tricks that we haven't seen before, either because they were too advanced, or simply because "it didn't come up". As is typical, most of these will expand on our knowledge of what is already the most complex and potentially confusing SQL command available to us: the `SELECT` query.

Sorting `SELECT` Query Results

Long lists of information are always easier to use when they're provided in some kind of order. To find a single author in a list from our Authors table, for example, could become an exercise in frustration if we had more than a few dozen registered authors in our database. While at first it might appear that they are sorted in order of database insertion, with the oldest records first and the newest records last, you'll quickly notice that deleting records from the database leaves invisible gaps in this order, which get filled in by newer entries as they're inserted.

What this amounts to is no reliable built-in result sorting capabilities from `SELECT` queries. Fortunately, there is another optional part of the `SELECT` query that lets us specify a column by which to sort our table of results. Let's say we wanted to print out a listing of the entries in our Authors table for future reference. If you'll recall, this table has three columns: ID, Name, and EMail. Since ID isn't really interesting in and of itself (it just provides a means to associate entries in this table with entries in the Jokes table), we will usually just list the remaining two columns when we work with this table. Here's a short list of a table of authors:

```
mysql>SELECT Name, EMail FROM Authors;
+-----------------+----------------------+
| Name            | EMail                |
+-----------------+----------------------+
| Joan Smith      | jsmith@somewhere.net |
| William Shatner | rocketman@earth.net  |
| Kevin Yank      | kevin@sitepoint.com  |
| Amy Mathieson   | amym@hotmail.com     |
+-----------------+----------------------+
```

As you can see, the entries are sorted in no particular order. This result is fine for a short list like this, but it would be easier to find a particular author's email address (that of Amy Mathieson, for example) in a very long list of authors, say a few hundred or so, if the authors' names appeared in alphabetical order. Here's how:

```
mysql>SELECT Name, EMail FROM Authors ORDER BY Name;
+-----------------+----------------------+
| Name            | EMail                |
+-----------------+----------------------+
| Amy Mathieson   | amym@hotmail.com     |
| Joan Smith      | jsmith@somewhere.net |
| Kevin Yank      | kevin@sitepoint.com  |
| William Shatner | rocketman@earth.net  |
+-----------------+----------------------+
```

The entries now appear sorted alphabetically by their names. Just as we can add a `WHERE` clause to a `SELECT` statement to narrow down the list of results, we can also add an `ORDER BY` clause to specify a

column by which a set of results should be sorted. By adding the keyword DESC after the name of the sort column, you can sort the entries in descending order:

```
mysql>SELECT Name, EMail FROM Authors ORDER BY Name DESC;
+-----------------+----------------------+
| Name            | EMail                |
+-----------------+----------------------+
| William Shatner | rocketman@earth.net  |
| Kevin Yank      | kevin@sitepoint.com  |
| Joan Smith      | jsmith@somewhere.net |
| Amy Mathieson   | amym@hotmail.com     |
+-----------------+----------------------+
```

You can actually use a comma-separated list of several column names in the ORDER BY clause, to have MySQL sort the entries by the first column, then sort any sets of tied entries by the second, and so on. Any of the columns listed in the ORDER BY clause may use the DESC keyword to reverse the sort order.

Setting LIMITS

Often you might work with a large database table, but only really be interested in a few entries within it. Let's say you wanted to track the popularity of different jokes on your site. You could add a column named TimesViewed to your Jokes table. Start it with a value of zero for new jokes, and add one to the value of the requested joke every time the joke page is viewed, to keep count of the number of times each joke in your database has been read.

The query for adding one to the TimesViewed column of a joke with a given ID is as follows:

```
$sql = "UPDATE Jokes SET TimesViewed=TimesViewed+1
  WHERE ID='$id'";
if (!mysql_query($sql)) {
  echo("<p>Error adding to times viewed for this joke!</p>\n");
}
```

A common use of this "joke view counter" would be to present a "Top 10 Jokes" list on the front page of the site, for example. Using `ORDER BY TimesViewed DESC` to list the jokes from highest TimesViewed to lowest, we would just have to pick the 10 first values from the top of the list. But if we have thousands of jokes in our database, the retrieval of a list of thousands would be quite wasteful in terms of the processing time and server system resources required, such as memory and CPU load, to use only ten of those retrieved.

But, if we use a `LIMIT` clause, we can specify a certain number of results to be returned. In our example, we need only the first ten:

```
$sql = "SELECT * FROM Jokes ORDER BY TimesViewed DESC LIMIT 10";
```

Although much less interesting, we could get rid of the word `DESC` and retrieve the 10 least popular jokes in the database.

Often, you want to let users view a long list of entries, say, the results of a search, but wish to display only a few at a time. Think of the last time you went looking through pages of search engine results to find a particular Website. You can use a `LIMIT` clause to do this sort of thing—simply specify both the result to begin the list with, and the maximum number of results to display. The query below, for example, will list the 21st to 25th most popular jokes in the database:

```
$sql = "SELECT * FROM Jokes ORDER BY TimesViewed DESC
  LIMIT 20, 5";
```

Remember, the first entry in the list of results is entry number 0. Thus, the 21st entry in the list is entry number 20.

Notice how, in the UPDATE query given above, and repeated here for convenience, we use the existing value of TimesViewed and add one to it to set the new value.

```
$sql = "UPDATE Jokes SET TimesViewed=TimesViewed+1
  WHERE ID='$id'";
```

If you hadn't known that you were allowed use this short cut, you might have performed a separate SELECT to get the current value, added one to it, and then performed an UPDATE using that newly calculated value. Besides the fact that this would have required two queries instead of one, and thus would take about twice as long, there is a danger to using this method. What if, while that new value was being calculated, someone else viewed the same joke? The PHP script would be run a second time for that new request. When it performed the SELECT to get the current value of TimesViewed, it would retrieve the same value as the first script did, because the value had not yet been updated. Both scripts would then add one to the same value, and write the new value into the table. See what happens? Two users view the joke, but the TimesViewed counter increments by just one!

In some situations, this kind of fetch-calculate-update procedure cannot be avoided, and the possibility of interference between simultaneous requests of the nature described above must be dealt with. Other situations where this procedure may be necessary include cases where you need to update several tables in response to a single action (e.g. updating inventory and shipping tables in response to a sale on an ecommerce Website).

By*locking* the table or tables with which you're working in a multiple-query operation, you can obtain exclusive access for the duration of that operation to prevent potentially damaging interference from concurrent operations. The syntax for locking a table is fairly simple:

```
LOCK TABLES tblName READ/WRITE
```

As shown, when you lock a table, you must specify whether you want a *read lock* or a *write lock*. The former prevents other processes from making changes to the table, but allows others to read the table. The latter stops all other access to the table.

When you're done with a table you have locked, you must release the lock to give other processes access to the table again:

```
UNLOCK TABLES
```

ALOCK TABLES query implicitly releases whatever locks you may already have. Therefore, to safely perform a multi-table operation, you must lock all the tables you'll use with a single query. Here's what the PHP code might look like for the ecommerce application we mentioned above:

```
mysql_query("LOCK TABLES inventory WRITE, shipping WRITE");

// Perform the operation...

mysql_query("UNLOCK TABLES");
```

For simple databases that require the occasional multi-table operation, table locking, as described here, will do the trick. More demanding applications, however, can benefit from the increased performance and crash-proof nature of *transactions*.

Transactions in MySQL

Many high-end database servers (e.g. Oracle, MS SQL Server, etc.) support a feature called *transactions*, which lets you perform complex, multi-query operations in a single, uninterrupted step. Consider what would happen if your server were struck by a power failure halfway through a database

update in response to a client order. For example, the server might have crashed after it updated your shipping table, but before it updated your inventory table, in response to a customer's order.

Transactions allow a group of table updates such as this to be defined so that they all occur, or none of them will. You can also manually cancel a transaction halfway through if the logic of your application requires it.

There are currently two versions of MySQL available: MySQL and MySQL-Max. The MySQL-Max version includes built-in support for *InnoDB tables*, which support transactions. The standard version of MySQL does not include this support by default, but if you compile it yourself you have the option of enabling it.

A full discussion of transactions is outside the scope of this book; please refer to the MySQL Reference Manual for a full description of MySQL-Max, InnoDB tables, and transaction support.

Column and Table Name Aliases

In some situations, it may be more convenient to be able to refer to MySQL columns and tables using different names. Let's take the example of a database used by an airline's online booking system; this example actually came up in the SitePoint Forums. The database structure can be found in *airline.sql* in the code archive if you want to follow along.

To represent the flights offered by the airline, the database contains two tables: Flights and Cities. Each entry in the Flights table represents an actual flight between two cities—the origin and destination of the flight. Obviously, Origin and Destination are columns in the Flights table, with other columns for things like the date and time of the flight, the type of aircraft, the flight numbers, and the various fares.

The Cities table contains a list of all the cities to which the airline flies. Thus, both the Origin and Destination columns in the Flights table will just contain IDs referring to entries in the Cities table. Now, consider these queries. To retrieve a list of flights with their origins:

```
mysql>SELECT Flights.Number, Cities.Name
    ->FROM Flights, Cities
    ->WHERE Flights.Origin = Cities.ID;
+--------+-----------+
| Number | Name      |
+--------+-----------+
| CP110  | Montreal  |
| CP226  | Sydney    |
| QF2026 | Melbourne |
  ...      ...
```

To obtain a list of flights with their destinations:

```
mysql>SELECT Flights.Number, Cities.Name
    ->FROM Flights, Cities
    ->WHERE Flights.Destination = Cities.ID;
+--------+----------+
| Number | Name     |
+--------+----------+
| CP110  | Sydney   |
| CP226  | Montreal |
| QF2026 | Sydney   |
  ...      ...
```

Now, what if we wanted to list both the origin and destination of each flight with a single query? That's pretty reasonable, right? Here's a query you might try:

```
mysql>SELECT Flights.Number, Cities.Name, Cities.Name
    ->FROM Flights, Cities
    ->WHERE Flights.Origin = Cities.ID
    ->AND Flights.Destination = Cities.ID;
Empty set (0.01 sec)
```

Why doesn't this work? Have another look at the query, and this time focus on what it actually says, rather than what you expect it to do. It tells MySQL to join the Flights and Cities tables and list the flight number, city name, and city name (yes, twice!) of all entries obtained, by matching up the Origin with the city ID and the Destination with the city ID. In other words, the Origin, Destination, and city ID must all be equal! This results in a list of all flights where the origin and the destination are the same! Unless your airline offers scenic flights, there aren't likely to be any entries that match this description (thus the "Empty set" result above).

What we need is a way to be able to return two different entries from the Cities table, one for the origin and one for the destination, for each result. If we had two copies of the table, one called Origins and one called Destinations, this would be much easier to do, but why maintain two tables that contain the same list of cities? The solution is to give the Cities table two different temporary names (*aliases*) for the purposes of this query.

If we follow the name of a table with AS*Alias* in the FROM portion of the SELECT query, we can give it a temporary name with which we can refer to it elsewhere in the query. Here's that first query again (to display flight numbers and origins only), but this time we've given the Cities table an alias: Origins.

```
mysql>SELECT Flights.Number, Origins.Name
    ->FROM Flights, Cities AS Origins
    ->WHERE Flights.Origin = Origins.ID;
```

This doesn't actually change the way the query works—in fact, it doesn't change the results at all—but for long table names, it can save some typing. Consider, for example, if we had given aliases of F and O to Flights and Cities, respectively. The query would be much shorter as a result.

Let's now return to our problem query. If we refer to the Cities table twice, using two different aliases, we can use a three-table join (where two of the tables are actually one and the same) to get the effect we want:

```
mysql>SELECT Flights.Number, Origins.Name,
    ->  Destinations.Name
    ->FROM Flights, Cities AS Origins,
    ->  Cities AS Destinations
    ->WHERE Flights.Origin = Origins.ID
    ->AND Flights.Destination = Destinations.ID;
+--------+-----------+----------+
| Number | Name      | Name     |
+--------+-----------+----------+
| CP110  | Montreal  | Sydney   |
| CP226  | Sydney    | Montreal |
| QF2026 | Melbourne | Sydney   |
  ...      ...         ...
```

You can also define aliases for column names. We could use this, for example, to differentiate the two Name columns in our result table above:

```
mysql>SELECT F.Number, O.Name AS Origin,
    ->  D.Name AS Destination
    ->FROM Flights AS F, Cities AS O, Cities AS D
    ->WHERE F.Origin = O.ID AND F.Destination = D.ID;
+--------+-----------+-------------+
| Number | Origin    | Destination |
+--------+-----------+-------------+
| CP110  | Montreal  | Sydney      |
| CP226  | Sydney    | Montreal    |
| QF2026 | Melbourne | Sydney      |
  ...      ...         ...
```

GROUPingSELECT Results

In"Getting Started with MySQL", we saw the following query, which tells us how many jokes are stored in our Jokes table:

```
mysql>SELECT COUNT(*) FROM Jokes;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

The MySQL function COUNT used in this query belongs to a special class of functions called *summary functions* or *group-by functions*, depending on where you look. A complete list of these functions is provided in Chapter 6 of the MySQL Manual and in "MySQL Functions". Unlike other functions, which affect each entry in the result of the SELECT query individually, summary functions group together all the results and return a single result. In the above example, for instance, COUNT returns the total number of result rows.

Let's say you wanted to display a list of authors with the number of jokes they have to their names. Your first instinct, if you've paid attention, might be to retrieve a list of all the authors' names and ID's, then use COUNT to count the number of results when you SELECT the jokes with each author's ID. The PHP code, without error handling, for simplicity, would look something like this:

```
// Get a list of all the authors
$authors = mysql_query( 'SELECT Name, ID FROM Authors' );

// Process each author
while ($author = mysql_fetch_array($authors)) {
  $name = $author['Name'];
  $id = $author['ID'];

  // Get count of jokes attributed to this author
  $result = mysql_query(
    "SELECT COUNT(*) AS NumJokes FROM Jokes WHERE AID='$id'" );
  $row = mysql_fetch_array($result);
  $numjokes = $row['NumJokes'];

  // Display the author & number of jokes
  echo("<p>$name ($numjokes jokes)</p>");
}
```

Note the use of AS in the second query above to give a friendlier name (NumJokes) to the result of COUNT(*).

This technique will work, but will require *n*+1 separate queries (where *n* is the number of authors in the database). Having the number of queries rely on a number of entries in the database is always something we want to avoid, as a large number of authors would make this script unreasonably slow and resource-intensive! Fortunately, another advanced feature of SELECT comes to the rescue!

If you add a GROUP BY clause to a SELECT query, you can tell MySQL to group the results of the query into sets that have the same value in the column(s) you specify. Summary functions like COUNT then operate on those groups-not on the entire result set as a whole. The next single query, for example, lists the number of jokes attributed to each author in the database:

```
mysql>SELECT Authors.Name, COUNT(*) AS NumJokes
    ->FROM Jokes, Authors
```

```
    ->WHERE AID = Authors.ID
    ->GROUP BY AID;
+------------+----------+
| Name       | NumJokes |
+------------+----------+
| Kevin Yank |        3 |
| Joan Smith |        1 |
+------------+----------+
```

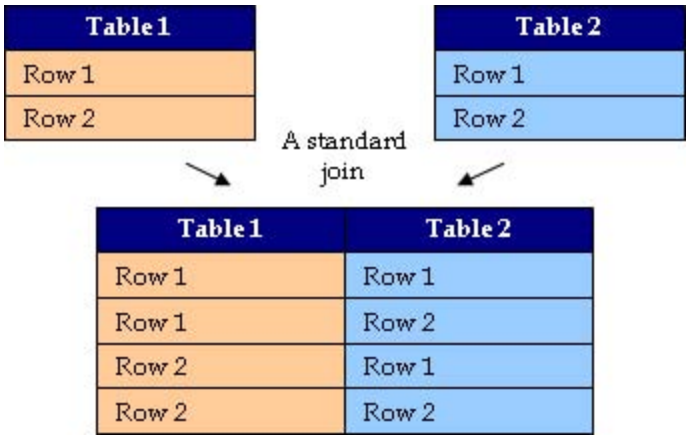
If we group the results by author ID (AID), we receive a breakdown of results for each author. Note that we could have specified `GROUP BY Authors.ID` and achieved the same result (since, as stipulated in the `WHERE` clause, these columns must be equal). `GROUP BY Authors.Name` would also work in most cases, but, as you can't guarantee that two different authors won't have the same name, in which case their results would be lumped together, it's best to stick to the ID columns, which are guaranteed to be unique for each author.

LEFT JOINS

We can see from the results above that Kevin Yank has three jokes to his name, and Joan Smith has one. What these results do not show is that there is a third author, Amy Mathieson, who doesn't have *any* jokes to her name. Since there are no entries in the Jokes table with AIDs that match her author ID, there will be no results that satisfy the `WHERE` clause in the query above, and she will therefore be excluded from the table of results.

About the only practical way to overcome this challenge with the tools we've seen so far would be to add another column to the Authors table and simply store the number of jokes attributed to each author in that column. Keeping that column up to date, however, would be a real pain, because we'd have to remember to update it every time a joke was added to, removed from, or changed (if, for example, the value of AID was changed) in the Jokes table. To keep things synchronized, we'd have to use `LOCK TABLES` whenever we made such changes, as well. Quite a mess, to say the least!

MySQL provides another method for joining tables, which fetches information from multiple tables at once. Called a *left join*, it's designed for just this type of situation. To understand how left joins differ from standard joins, we must first recall how standard joins work.



Standard joins take all possible combinations of rows

As shown in "Standard joins take all possible combinations of rows", MySQL performs a standard join of two tables by listing all possible combinations of the rows of those tables. In a simple case, a standard join of two tables with two rows apiece will contain four rows: row 1 of table 1 with row 1 of table 2, row 1 of table 1 with row 2 of table 2, row 2 of table 1 with row 1 of table 2, and row 2 of table 1 with row 2 of table 2. With all of these result rows calculated, MySQL then looks to the `WHERE` clause for guidance on which rows should actually be kept (e.g. those where the AID column from table 1 matches the ID column from table 2).

The reason the above does not suit our purposes is that we'd like to also include rows in table 1 (i.e. Authors) that don't match any rows in table 2 (i.e. Jokes). A left join does exactly what we need—it forces a row to appear in the results for each row in the first (left-hand) table, even if no matching entries are found in the second (right-hand) table. Such *forced rows* are given `NULL` values for all of the columns in the right-hand table.

To perform a left join between two tables in MySQL, separate the two table names in the `FROM` clause with `LEFT JOIN` instead of a comma. Then follow the second table's name with `ON`*`condition`*, where *`condition`* specifies the criteria for matching rows in the two tables (i.e. what you would normally put in the`WHERE` clause). Here's our revised query for listing authors and the number of jokes to their credit:

```
mysql>SELECT Authors.Name, COUNT(*) AS NumJokes
    ->FROM Authors LEFT JOIN Jokes ON AID = Authors.ID
```

```
    ->GROUP BY AID;
+---------------+----------+
| Name          | NumJokes |
+---------------+----------+
| Amy Mathieson |        1 |
| Kevin Yank    |        3 |
| Joan Smith    |        1 |
+---------------+----------+
```

Wait just a minute! Suddenly Amy Mathieson has one joke? That can't be right! In fact, it is—but only because the query is wrong. `COUNT(*)` counts the number of rows returned for each author. If we look at the ungrouped results of the `LEFT JOIN`, we can see what's happened:

```
mysql>SELECT Authors.Name, Jokes.ID AS JokeID
    ->FROM Authors LEFT JOIN Jokes ON AID = Authors.ID;
+---------------+--------+
| Name          | JokeID |
+---------------+--------+
| Kevin Yank    |      1 |
| Kevin Yank    |      2 |
| Kevin Yank    |      4 |
| Joan Smith    |      3 |
| Amy Mathieson |   NULL |
+---------------+--------+
```

See? Amy Mathieson *does* have a row—the row is forced because it doesn't have any matching rows in the right-hand table of the `LEFT JOIN` (Jokes). The fact that the Joke ID value is `NULL` doesn't affect `COUNT(*)`—it still counts it as a row. If instead of `*`, you specify an actual column name (say `Jokes.ID`) for the `COUNT` function to look at, it will ignore `NULL` values in that column, and give us the count we want:

```
mysql>SELECT Authors.Name, COUNT(Jokes.ID) AS NumJokes
    ->FROM Authors LEFT JOIN Jokes ON AID = Authors.ID
    ->GROUP BY AID;
+---------------+----------+
| Name          | NumJokes |
+---------------+----------+
| Amy Mathieson |        0 |
| Kevin Yank    |        3 |
| Joan Smith    |        1 |
+---------------+----------+
```

Limiting Results with `HAVING`

What if we wanted a list of *only* those authors that had no jokes to their name? Once again, let's look at the query that most users would try first:

```
mysql>SELECT Authors.Name, COUNT(Jokes.ID) AS NumJokes
    ->FROM Authors LEFT JOIN Jokes ON AID = Authors.ID
    ->WHERE NumJokes = 0
    ->GROUP BY AID;
ERROR 1054: Unknown column 'NumJokes' in 'where clause'
```

By now you're probably not surprised that it didn't work as expected. The reason why `WHERE NumJokes = 0` didn't do the job has to do with the way MySQL processes result sets. First, MySQL produces the raw, combined list of authors and jokes from the Authors and Jokes tables. Next, it processes the `WHERE` clause and the `ON` portion of the FROM clause so that only the relevant rows in the list are returned (in this case, rows that match up authors with their jokes). Finally, MySQL processes the `GROUP BY` clause by grouping the results according to their AID, `COUNT`ing the number of entries in each group that have non-`NULL` `Jokes.ID` values, and producing the NumJokes column as a result.

Notice that the NumJokes column isn't actually created until the `GROUP BY` clause is processed, and *that* doesn't happen until after the `WHERE` clause does its thing! If you wanted to exclude jokes that contained the word "chicken" from the count, you could use the `WHERE` clause without a problem, because that exclusion doesn't rely on a value that the `GROUP BY` clause is responsible for producing. Conditions that affect the results after grouping takes place, however, must appear in a special `HAVING` clause. Here's the corrected query:

```
mysql>SELECT Authors.Name, COUNT(Jokes.ID) AS NumJokes
    ->FROM Authors LEFT JOIN Jokes ON AID = Authors.ID
    ->GROUP BY AID
    ->HAVING NumJokes = 0;
+---------------+----------+
| Name          | NumJokes |
+---------------+----------+
| Amy Mathieson |        0 |
+---------------+----------+
```

Some conditions work both in the `HAVING` and the `WHERE` clause. For example, if we wanted to exclude a particular author by name, we could use `Authors.Name != "Author Name"` in either the `WHERE` or the`HAVING` clause to do it, because whether you filter out the author before or after grouping the results, the same results are returned. In such cases, it is always best to use the `WHERE` clause, because MySQL is better at internally optimizing such queries so they happen faster.

Summary

In this chapter, we rounded out your knowledge of Structured Query Language (SQL), as supported by MySQL. We focused predominantly on features of `SELECT` that allow you to view information stored in a database with an unprecedented level of flexibility and power. With judicious use of the advanced features of`SELECT`, you can have MySQL do what it does best and lighten the load on PHP in the process.

There are still a few isolated query types, mainly to do with indexes, that we haven't seen, and MySQL offers a whole library of built-in functions to do things like calculate dates and format text strings (see "MySQL Functions"). To become truly proficient with MySQL, you should also have a firm grasp on the various column types offered by MySQL. The `TIMESTAMP` type, for example, can be a real time saver (no pun intended). All of these are fully documented in the MySQL Manual, and briefly covered in "MySQL Column Types".

In"Advanced PHP", I look at some useful features of PHP we haven't had the opportunity to cover. From tightening security to sending email, from handling file uploads to lightening the load on your server, I guarantee it's a chapter not to be missed!

Chapter 10: Advanced PHP

Overview

PHP's strength lies in its huge library of built-in functions, which allows even a novice user to perform very complicated tasks without having to install new libraries or worry about low-level details, as is often the case with other popular server-side languages like Perl. Because of the focus of this book, we've constrained ourselves to exploring only those functions that were directly related to MySQL databases (in fact, we didn't even see all of *those*). In this final instalment, we'll broaden our horizons a little and explore some of the other useful features PHP has to offer someone building a database driven Website.

We'll begin by learning about PHP's `include` function, which allows us to use a single piece of PHP code in multiple pages, and makes the use of common code fragments much more practical. We'll also see how to add an extra level of security to our site with this feature.

PHP, while generally quick and efficient, nevertheless adds to the load time and the workload of the machine on which the server is run. On high-traffic sites (sitepoint.com, for example!), this load can grow to unacceptable levels. But this challenge doesn't mean we have to abandon the database-driven nature of our site. We'll see how to use PHP behind the scenes to create semi-dynamic pages that don't stress the server as much.

A common question asked on sitepoint.com and in other sites' forums is how to use an `<input type="file">` tag to accept file uploads from site visitors. We'll learn how to do this with PHP, and see how to make this fit into a database-driven site.

Finally, an extremely powerful feature of PHP is the ability to send email messages with dynamically generated content. Whether you want to use PHP to let visitors send email versions of your site's content to their friends, or just provide a way for users to retrieve their forgotten passwords, PHP's email function will serve nicely!

Server-Side Includes with PHP

If you've been working on the Internet for a while, you've probably come across the term Server-Side Includes (SSIs); if not, you can read Matt Mickiewicz's mini-tutorial on the subject.

In essence, SSIs allow you to insert the content of one file stored on your Web server into the middle of another. The most common use for this technology is to encapsulate common design elements of a Website in small HTML files that can then be incorporated into Web pages on the fly. Any changes to these small files immediately affect all files that include them. And, just like a PHP script, the Web browser doesn't need to know about any of it, since the Web server does all the work before it sends the requested page to the browser.

PHP has a function that provides similar capabilities. But in addition to being able to incorporate regular HTML and other static elements into your included files, you can also include common script elements. Let's look at an example:

```
<!-- include-me.php -->
<?php
  echo( '<p>"Make me one with everything!"</p>\n' );
?>
```

The above file, *include-me.php*, contains some simple PHP code. You'll also need the following file:

```
<!-- testinclude.php -->
<html>
<head>
<title> Test of PHP Includes </title>
</head>
<body>
<p>What did the Buddhist monk say to the hot dog vendor?</p>
<?php
  include('include-me.php');
?>
</body>
</html>
```

Notice the call to the `include` function. We specify the name of the file we want to include (*include-me.php*), and PHP will attempt to grab the named file and stick it into the file to replace the call to `include`. Upload both of the above files to your Web server (or copy them to your Web server's document folder if you're running the server on your computer) and load *testinclude.php* in your browser. You'll see a Web page that contains the message from our include file, as expected.

If this example doesn't work, you may need to configure the `include_path` option in your *php.ini* file. Open the file in your favourite text editor and look for a line that begins with `include_path`, about halfway through the file. This setting works in the same way as the system `PATH` environment variable with which you may be familiar. It contains a list of directories where PHP should look for files that you ask it to include. Set it so it contains "." (the current directory).

Depending on whether your server is running under Windows or Linux, you may need to surround your setting with quotes:

Under Linux (or other UNIX-based operating systems):

```
include_path=.:/another/directory
```

Under Windows:

```
include_path=".;c:\another\directory"
```

Increasing Security with Includes

PHP scripts will sometimes contain sensitive information like user names, passwords, and other things you don't want the world to be able to access. By now, you're probably used to the `mysql_connect` function, which requires you to put your MySQL user name and password in a PHP script that needs access to a database. While you can simply set up MySQL so that the user name and password used by PHP cannot be used by potential hackers (by setting the Host field in the user table as described in "MySQL Administration"), you would probably still rest easier knowing that your user name and password are protected by an extra level of security.

"But wait a minute," you might say. "Since the PHP is processed by the server, nobody can see my password anyway, right?" Right, but consider what would happen if PHP stopped working on your server. If, because of an accidental software misconfiguration made by a well-meaning associate, or some other factor, PHP stopped working on your server, the PHP pages would be served up as plain text files, with all your PHP code (including your password) there for the world to see!

To guard against this kind of security breach, you should put any security-sensitive code into an include file, and place that file into a directory that's not part of your Web server's directory structure. If you add that directory to your PHP `include_path` setting (in *php.ini*), you can refer to the files directly with the PHP`include` function, but have them tucked away safely somewhere where your Web server can't display them as Web pages.

For example, if your Web server expects all Web pages to exist in */home/httpd/* and its sub-directories, you could create a directory called */home/phpinc/* to house all of your include files. Add that directory to your `include_path`, and you're done! The next example shows how you can put your database connection code into an include file:

```
<!-- dbConnect.inc (in /home/phpinc/) -->
<?php

  $dbcnx = mysql_connect('localhost', 'root', 'rootpassword');

?>
```

And a file that uses this include:

```
<!-- dbSample.php (in /home/httpd/) -->
<?php

  // Connect to MySQL
  include('dbConnect.inc');

  mysql_select_db('myDatabase',$dbcnx);

  ...
```

As you can see, if PHP stops working on your server, all that will be exposed is a call to the `include` function. The user name and password are safely stored in *dbConnect.inc*, which cannot be accessed directly from the Web.

As usual, it's still important to consider other means that may be available to access those files. For example, if you share your Web server with other people/companies, be certain that the files are not accessible to those other users!

Semi-Dynamic Pages

As the owner of a successful-or soon-to-be so-Website, you probably see site traffic as something you'd like to encourage. Unfortunately, high site traffic is just the kind of thing that a Web server administrator dreads-especially when that site is primarily composed of dynamically generated, database-driven pages. Such pages take a great deal more horsepower from the computer that runs the Web server software than plain, old HTML files do, because every page request is like a miniature program that runs on that computer.

While some pages of a database-driven site must always display up-to-the-second data culled from the database, others don't necessarily. Consider the front page of a Website like sitepoint.com. Typically, it presents a sort of "digest" of what's new and fresh on the site. But how often does that information actually change? Once a day? Once a *week*? And how important is it that visitors to your site see those changes the instant they occur? Would your site really suffer if changes took effect after a bit of a delay?

By converting high-traffic dynamic pages into *semi-dynamic* equivalents, which are static pages that get dynamically regenerated at regular intervals to freshen their content, you can go a long way towards reducing the toll that the database-driven components of your site take on your Web server's performance.

Say you have *index.php*, your front page, which provides a summary of new content on your site. Through examination of server logs, you'll probably find that this is one of the most requested pages on your site. If you ask yourself some of the questions above, you'll realize that this page doesn't have to be dynamically generated for every request. As long as it's updated every time new content is added to your site, it'll be as dynamic as it needs to be. With a PHP script, you can generate a static snapshot of the dynamic page's output and put this snapshot online, in place of the dynamic version, as *index.html*.

This little trick will require some reading, writing, and juggling of files. PHP is perfectly capable of accomplishing this task, but we have not yet seen the functions we'll need:

`fopen`
: Opens a file for reading and/or writing. This file can be stored on the server's hard disk, or PHP can load it from a URL just like a Web browser would.

`fclose`
: Tells PHP you're finished reading/writing a particular file and releases it for other programs or scripts to use.

`fread`
: Reads data from a file into a PHP variable. Allows you to specify how much information (i.e. how many characters or bytes) to read.

`fwrite`
: Writes data from a PHP variable into a file.

`copy`
: Performs a run-of-the-mill file copy operation.

`unlink`
: Deletes a file from the hard disk.

Do you see where we're headed? If not, don't worry-you will in a moment.

Create a file called *generateindex.php*. It will be the responsibility of this file to load *index.php*, the dynamic version of your front page, as a Web browser would, then write the static version of the file as an updated version of *index.html*. If anything goes wrong in this process, you want to avoid the potential destruction of the good copy of *index.html*, so we'll make this script write the new static version into a temporary file (*tempindex.html*) and then copy it over *index.html* if all is well.

Here's the code for *generateindex.php*, with ample comments so you can see what's going on:

```
<!-- generateindex.php -->
<?php

  // Sets the files we'll be using
  $srcurl         = 'http://localhost/index.php';
  $tempfilename   = 'tempindex.html';
  $targetfilename = 'index.html';

?>
<html>
<head>
<title> Generating <?=$targetfilename?> </title>
</head>
<body>
<p>Generating <?=$targetfilename?>...</p>
<?php

  // Begin by deleting the temporary file, in case
  // it was left lying around. This might spit out an
  // error message if it were to fail, so we use
  // @ to suppress it.
  @unlink($tempfilename);

  // Load the dynamic page by requesting it with a
  // URL. The PHP will be processed by the Web server
  // before we receive it (since we're basically
  // masquerading as a Web browser), so what we'll get
  // is a static HTML page. The 'r' indicates that we
  // only intend to read from this "file".
  $dynpage = fopen($srcurl, 'r');

  // Check for errors
  if (!$dynpage) {
    die("<p>Unable to load $srcurl. Static page
          update aborted!</p>");
  }

  // Read the contents of the URL into a PHP variable.
  // Specify that we're willing to read up to 1MB of
  // data (just in case something goes wrong).
  $htmldata = fread($dynpage, 1024*1024);

  // Close the connection to the source "file", now
  // that we're done with it.
  fclose($dynpage);

  // Open the temporary file (creating it in the
  // process) in preparation to write to it (note
  // the 'w').
  $tempfile = fopen($tempfilename, 'w');

  // Check for errors
  if (!$tempfile) {
    die("<p>Unable to open temporary file
         ($tempfilename) for writing. Static page
```

```
        update aborted!</p>");
  }

  // Write the data for the static page into the
  // temporary file
  fwrite($tempfile, $htmldata);

  // Close the temporary file, now that we're done
  // writing to it.
  fclose($tempfile);

  // If we got this far, then the temporary file
  // was successfully written, and we can now copy
  // it on top of the static page.
  $ok = copy($tempfilename, $targetfilename);

  // Finally, delete the temporary file.
  unlink($tempfilename);

?>
<p>Static page successfully updated!</p>
</body>
</html>
```

The above code only looks daunting because of the large comments I've included. Remove them, and you'll see it's actually a fairly simple script.

Now, whenever *generateindex.php* is executed (say, when a browser requests it), a fresh copy of *index.html* will be generated from *index.php*. If we move *index.php* and *generateindex.php* into a restricted-access directory, you can make sure that only site administrators have the ability to update the front page of your site in this way. Expand this script to generate all semi-dynamic pages on your site, and add an "update semi-dynamic pages" link to your content management system!

If you'd rather have your front page updated automatically, you'll need to set up your server to run *generateindex.php* at regular intervals-say, every hour. Under recent versions of Windows, you can use the Task Scheduler (called System Agent in older versions of Windows equipped with MS Plus Pack), to run*php.exe*, a stand-alone version of PHP included with the Windows PHP distribution, automatically every hour. Just create a batch file called *generateindex.bat* that contains this line of text:

```
C:\PHP\php.exe C:\WWW\generateindex.php
```

Adjust the paths and file names as necessary, and then set up Task Scheduler to run *generateindex.bat* every hour. In some versions of Windows, you'll need to set up 24 tasks to be run daily at the appropriate times. Done!

Under Linux, or other UNIX based platforms, you can do a similar thing with *cron*-a program installed on just about every UNIX system out there that lets you define tasks to be run at regular intervals. Ask your friendly neighbourhood Linux know-it-all, check your favourite Linux Website, or post a message on the SitePoint Forums if you need any help getting started with *cron*.

The task you'll set up *cron* to run will be very similar to the Windows task discussed above. The stand-alone version of PHP you'll need, however, doesn't come with the PHP Apache loadable module we compiled way back in "Installation". You'll need to compile it separately from the same package we used to compile the Apache module. Instructions for this are provided with the package and on the PHP Website, but feel free to post in the SitePoint Forums if you need help!

For experienced *cron* users in a hurry, here's what the line in your *crontab* file should look like:

```
0 0-23 * * * php /path/to/generateindex.php > /dev/null
```

Handling File Uploads

All the examples of database-driven Websites in this book so far have dealt with sites based around textual data. Jokes, articles, authors... all of these things can be fully represented with strings of text. But what if you ran, say, an online digital photo gallery where people could upload pictures taken with digital cameras? For this idea to work, we need to be able to let visitors to our site upload their photos and we need to be able to keep track of them.

We'll start with the basics: let's write an HTML form that allows users to upload files. HTML makes this quite easy with its `<input type="file">` tag. By default, however, only the name of the file selected by the user is sent. To have the file itself submitted with the form data, we need to add `enctype="multipart/form-data"` to the `<form>` tag:

```
<form action="fileupload.php" method="post"
  enctype="multipart/form-data">
<p>Select file to upload:
  <input type="file" name="uploadedfile" /></p>
<p><input type="submit" name="submit" value="Submit" /></p>
</form>
```

As we can see, a PHP script (*fileupload.php*) will handle the data submitted with the form above. Information about uploaded files appears in a array called `$_FILES` that is automatically created by PHP[1]. As you'd expect, an entry in this array called `$_FILES['uploadedfile']` (from the `name` attribute of the `<input>` tag) will contain information about the file uploaded in this example. However, instead of storing the contents of the uploaded file, `$_FILES['uploadedfile']` contains yet another array. We therefore use a second set of square brackets to select the information we want:

- `$_FILES['uploadedfile']['tmp_name']`

  - the name of the file stored on the Web server's hard disk, in the directory set by the `TEMP` environment variable (e.g. *C:\Windows\TEMP* on most Windows 9x systems), unless it has been specified explicitly, using the `upload_tmp_dir` setting in your *php.ini* file. This file is only kept for as long as the PHP script responsible for handling the form submission is in operation, so if you want to use it for anything later on (e.g. storing it for display on the site) you need to make a copy of it somewhere else. To do this, use the `copy` function described in the previous section.

- `$_FILES['uploadedfile']['name']`

  - the name of the file on the client machine, before it was submitted. If you make a permanent copy of the temporary file, you might want to give it its original name instead of the automatically-generated temporary file name described above.

- `$_FILES['uploadedfile']['size']`

  - the size (in bytes) of the file.

- `$_FILES['uploadedfile']['type']`

  - the MIME type (e.g. `text/plain`,`image/gif`, etc.) of the file.

Remember,`'uploadedfile'` is just the `name` attribute of the `<input>` tag that submitted the file, so the actual names of these variables will depend on that attribute.

You can use these variables to decide whether to accept or reject an uploaded file. For example, in our photo gallery we would only really be interested in JPEG and possibly GIF files. These files have MIME types of `image/pjpeg` and `image/gif` respectively, but to cater to differences between browsers[2], you should use regular expressions to validate the uploaded file's type:

```
if (eregi('^image/p?jpeg(;.*)?$', $_FILES['uploadedfile']['type'])
    or eregi('^image/gif(;.*)?$', $_FILES['uploadedfile']['type'])
```

```
) {
  // Handle the file...
} else {
  echo("<p>Please submit a JPEG or GIF image file.</p>\n");
}
```

See"Content Formatting and Submission" for help with regular expression syntax.

While you can use a similar technique to disallow files that are too large (by checking the `$_FILES['uploadedfile']['size']` variable), this is not usually a good idea. Before this value can be checked, the file is already uploaded and saved in the *TEMP* directory. If you try to reject files because you have limited disk space and/or bandwidth, the fact that large files can still be uploaded, even though they get deleted almost immediately, may be a problem for you.

Instead, you can tell PHP in advance the maximum file size you wish to accept. There are two ways to do this. The first is to adjust the `upload_max_filesize` setting in your *php.ini* file. The default value is 2MB, so if you want to accept uploads larger than that you'll immediately need to change that value[3].

The second method is to include a hidden `input` field in your form with the `nameMAX_FILE_SIZE`, and the maximum file size you want to accept with this form as its `value`. For security reasons, this value cannot exceed the `upload_max_filesize` setting in your *php.ini*, but it does provide a way for you to accept different maximum sizes on different pages. The following form, for example, will allow uploads of up to 1 kilobyte (1024 bytes):

```
<form action="fileupload.php" method="post"
  enctype="multipart/form-data">
<input type="hidden" name="MAX_FILE_SIZE" value="1024" />
<p>Select file to upload:
  <input type="file" name="uploadedfile" /></p>
<p><input type="submit" name="submit" value="Submit" /></p>
</form>
```

Note that the hidden `MAX_FILE_SIZE` field must come before any `<input type="file">` tags in the form, so that PHP is apprised of this restriction before it receives any submitted files. Note also that this restriction can be easily circumvented by a malicious user who simply writes his or her own form without the`MAX_FILE_SIZE` field. For fail-safe security against large file uploads, use the `upload_max_filesize` setting in *php.ini*.

Assigning Unique File Names

As we said above, to keep an uploaded file we need to copy it to another directory. And while we have access to the name of each uploaded file with its `$_FILE['uploadedfile']['name']` variable, we have no guarantee that two files with the same name will not be uploaded. In such a case, storage of the file with its original name may result in newer uploads overwriting older ones.

For this reason, you'll usually want to adopt a scheme that allows you to assign a unique file name to all uploaded files. Using the system time (which we can access using the PHP `time` function), we can easily produce a name based on the number of seconds since January 1st 1970. But what if two files happen to be uploaded within one second of each other? To help guard against this possibility, we'll also use the client's IP address (automatically stored in `$_SERVER['REMOTE_ADDR']` by PHP) in the file name. Since we're unlikely to receive two files from the same IP address within one second of each other, this is an acceptable solution for our purposes.

```
// Pick a file extension
if ( eregi('^image/p?jpeg(;.*)?$',
     $_FILES['uploadedfile']['type']) )
  $extension = '.jpg';
else $extension = '.gif';
```

```
// The complete path/filename
$filename = "C:/Uploads/" . time() .
  $_SERVER['REMOTE_ADDR'] . $extension;

// Copy the file (if it is deemed safe)
if (is_uploaded_file($_FILES['uploadedfile']['tmp_name']) and
    copy($_FILES['uploadedfile']['tmp_name'], $filename)) {
  echo("<p>File stored successfully as $filename.</p>");
} else {
  echo("<p>Could not save file as $filename!</p>");
}
```

Important to note is my use of the `is_uploaded_file` function to check if the file is 'safe'. All this function does is return true if the file name it is passed as a parameter (`$_FILES['uploadedfile']['tmp_name']` in this case) was in fact uploaded as part of a form submission. If a malicious user loaded this script and manually specified a file name such as */etc/passwd* and we had not used `is_uploaded_file` to check that *$uploadedfile* really referred to an uploaded file, our script might be used to copy sensitive files on our server into a directory where they would become publicly accessible over the Web! Thus, before you ever trust a PHP variable that you expect to contain the file name of an uploaded file, be sure to use `is_uploaded_file` to check it.

A second trick I've used here is to combine `is_uploaded_file` and `copy` together as the condition of an `if` statement. If the result of `is_uploaded_file` is false, PHP knows immediately that the entire condition will be false when it identifies the `and` operator separating the two function calls. To save time, it won't even bother running `copy`, so the file won't get copied when `is_uploaded_file` returns false. On the other hand, if `is_uploaded_file` returns true, PHP goes ahead and copies the file. The result of `copy` then determines whether the success or error message is displayed. Similarly, if we had used the `or` operator instead of `and`, a successful result from the first part of the condition would cause PHP to skip the second part. This characteristic of `if` statements is known as *short-circuit evaluation*, and works in other conditional structures such as `while` and `for` loops, too.

Finally, note in the above script that I have used UNIX-style forward slashes (`/`) in the path, despite it being a Windows path. If I'd used backslashes I'd have had to replace them with double-backslashes (`\\`) so that PHP didn't think we were escaping special characters. However, PHP is smart enough to convert forward slashes in a file path to backslashes when it's running on a Windows system. Because under UNIX we can also use single slashes (`/`) as usual, the adoption of forward slashes in general for file paths in PHP will make your scripts more portable.

Recording Uploaded Files in the Database

So, we've created a system whereby visitors can upload JPEG and GIF images and have them saved on our server… but wasn't this book supposed to be about database-driven Websites? If we used the system as it stands now, someone would have to collect the submitted images out of the folder where they're saved, and then add them to the Website by hand! If you think back to "Content Formatting and Submission", when we developed a system that site visitors could use to submit jokes and have them stored in the database ready for quick approval by an administrator, you know there must be a better way!

MySQL has several column types that allow you to store binary data. In database parlance, these column types let us store BLOB's (Binary Large OBjects). However, the storage of potentially large files in a relational database is not usually a good idea. While there is convenience in having all the data located in one place, large files lead to large databases and large databases lead to reduced performance and much larger backup files.

The best alternative is usually to store the *file names* in the database. As long as you remember to delete files when you delete their corresponding entries in the database, everything should work just the way you need it to. Since we've seen all the SQL code involved in this time and again, I'll leave the details up to you. As usual, the SitePoint Forum community is there to offer a helping hand if you need it!

If your heart is set on storing the files themselves in the database, however, you'll definitely want to check out"Storing Binary Data in MySQL"!

[1]Prior to PHP 4.1, this array was called `$HTTP_POST_FILES`. This name continues to work in current versions of PHP for backwards compatibility.

[2]The exact MIME type depends on the browser in use. Internet Explorer uses the standards-compliant `image/pjpeg` for JPEG images, while Netscape 6 uses `image/jpeg`. Stranger yet, Opera 6 uses `image/jpeg; name="`*`filename`*`.jpg"`!

[3]A second restriction, affecting the total size of form submissions, is enforced by the `post_max_size` setting in *php.ini*. Its default value is 8MB, so if you want to accept *really* big uploads, you'll need to modify that setting too.

Email in PHP

Email is a powerful force on the Internet. Whether you want to provide a weekly "what's new" newsletter to your users, or a way for them to retrieve a lost or forgotten password, email is the way to go. PHP makes working with email very easy, letting you send messages using a single call to the `mail` function.

Before you can send email using the `mail` function, you first have to set up PHP's email related options. Here are the relevant lines of an out-of-the-box *php.ini* file under Windows:

```
[mail function]
SMTP            =
sendmail_from   = me@localhost.com
;sendmail_path  =
```

Depending on whether you use the Windows or Linux/UNIX version, PHP will send mail through an SMTP server or the local sendmail system, respectively. The process we'd use to set up either of these systems is beyond the scope of this book, and there's plenty of information out there to help you with either. If you're running on Windows, however, chances are that your ISP has already provided an SMTP server for your use. It's the same server you set your email program to use when you send messages. Set the `SMTP` setting to the host name/IP address of that server.

Set `sendmail_from` to the default 'from' email address you want to use for messages that are sent by PHP. If you administer this server, then you should probably put your email address here.

Finally, `sendmail_path` under Linux/UNIX should be uncommented (i.e. remove the semicolon from the start of the line) and set to the path and file name of the *sendmail* program on your system[4]. Under Linux, this will usually be */usr/sbin/sendmail*.

With these settings set and your Web server restarted, PHP should be decked out with full email capabilities. Now, to send an email in PHP couldn't be easier:

```
mail("to-address@somewhere.com", "Message Subject",
     "This is the body of the message.");
```

Mailing to multiple recipients can be accomplished if you simply separate each address with commas:

```
mail("to1@mail.net, to2@mail.net, ...", "Message Subject",
     "Message body");
```

It's also very easy to supply additional headers, in order to specify From or Reply-To addresses, for example. Just add them as a fourth parameter, separated by carriage return/new line pairs[5]:

```
mail("to@mail.net", "Message Subject", "Message body",
     "From: webmaster@host.com\r\nReply-to:admin@host.com");
```

Combined with a database, a mailing list becomes very easy to manage! Simply pull the list of addresses out of the database and use the `mail` function to fire off the messages. Personalization of the messages is also very easy. Consider this example:

```
// Retrieve $email and $password from the database based
// on the $username provided in a form.

mail($email, "Your Password",
"Hi there!

You just filled out a form on our Website
indicating that you had lost your password.
As requested, we are sending it to you by
email.

username: $username
```

```
password: $password

Please record this information in a safe
place so you have it on hand for your next
visit to pingpongballs.com!

-The Webmaster.

");
```

[4] If you are unwilling or unable to set up the sendmail system on your server, you can configure PHP to use an SMTP server provided by you or your ISP instead. Simply follow the instructions given above for Windows.

[5] Resist the temptation to use simple new lines (\n) to separate headers—even on Linux/UNIX servers. The standard for MIME email clearly indicates that carriage return/new line pairs must be used.

Summary

In this chapter, we've looked at some of PHP's convenient and powerful built-in functions. With the `include` function, you can increase the security of your site and reuse code that might appear on many pages of your site (e.g. connecting to MySQL and selecting your database). With the file manipulation functions, you can reduce the strain that your semi-dynamic pages place on your server. And with the incredibly simple `mail` function, you can send email from any PHP script with a single line of code. In addition, we looked at how to accept and handle file uploads in PHP.

Chapter 11: Storing Binary Data in MySQL

Overview

In the previous chapter, we developed a system whereby users could upload image files and have them stored on your Web server for display on your site. In developing this system, we had to tackle several issues:

- Ensure that non-conflicting file names are generated.
- Synchronize stored files with the database by adding an entry for each file as it is uploaded.
- Delete old files when their database entries are deleted.

While I've demonstrated that these constraints can be met, after 10 chapters of pushing the advantages of relational databases, you may have been surprised that I didn't opt for a solution where the files were stored in the database.

In cases where you're dealing with relatively small files, for example, head shots for use in a staff directory, the storage of data in MySQL is, in fact, quite practical. In this chapter, I'll demonstrate how to use PHP to store binary files uploaded over the Web in a MySQL database, and how to retrieve those files for download or display.

Binary Column Types

As with most database-driven Web applications, the first thing to consider is the layout of the database. For each of the files that's stored in our database, we will store the file name, the MIME type (e.g. `image/pjpeg` for JPEG image files), a short description of the file, and the binary data itself. Here's the `CREATE TABLE` statement that must be entered in MySQL to create the table:

```
mysql>CREATE TABLE filestore (
    ->  ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    ->  FileName VARCHAR(255) NOT NULL,
    ->  MimeType VARCHAR(50) NOT NULL,
    ->  Description VARCHAR(255) NOT NULL,
    ->  FileData MEDIUMBLOB
    ->);
```

Most of this syntax should be familiar to you; however, the `MEDIUMBLOB` column type is new. If you consult the MySQL Column Type Reference in "MySQL Column Types", you'll find that `MEDIUMBLOB` is the same as`MEDIUMTEXT`, except that it performs case-sensitive searches and sorts. In fact, from MySQL's point of view, there is no difference between binary data and blocks of text-both are just long strings of bytes to be stored in the database. The reason we'll use `MEDIUMBLOB` instead of `MEDIUMTEXT` is simply to anticipate the situation where we might need to compare the contents of one binary file with another. In such cases, we'd want the comparison to be 'case sensitive', as binary files may use byte patterns that are equivalent to alphabetical letters, and we'd want to distinguish between the byte pattern that represents 'A' from that which represents 'a'.

`MEDIUMBLOB` is one of several 'BLOB' column types designed to store variable-length binary data (BLOB stands for Binary Large OBject). These column types differ from one another only in two aspects: the maximum size of the data a particular value in the column can contain, and the number of bytes used to store the length of each data value. The different binary column types are listed with these details in "Binary Column Types in MySQL".

Binary Column Types in MySQL

Column Type	Maximum Size	Space req'd per entry
`TINYBLOB`	255B	Data size + 1 byte
`BLOB`	65KB	Data size + 2 bytes
`MEDIUMBLOB`	16.7MB	Data size + 3 bytes
`LONGBLOB`	4.3GB	Data size + 4 bytes

As you can see, the table we created above will be able to store files up to 16.7MB in size. If you think you'll need larger files, you can bump the FileData column up to a `LONGBLOB`. Each file will occupy 1 more byte in the database, because MySQL will require that extra byte in order to record larger file sizes, but you'll be able to store files up to 4.3GB in size-assuming that your operating system allows files of that size!

Storing Files

The next step is to create a PHP script that lets users upload files and store them in the database. You can hold off copying the code in the next two sections-I'll present it all as a complete script at the end of the chapter. Here's the code for the form-there should be no surprises here:

```
<form action="<?=$_SERVER['PHP_SELF']?>?action=ulfile"
      method="post" enctype="multipart/form-data">
<p>Upload File:<br />
<input type="file" name="uploadfile" /></p>
<p>File Description:<br />
<input type="text" name="desc" maxlength="255" /></p>
<p><input type="submit" name="go" value="Upload" /></p>
</form>
```

As you should already know from our work in "Advanced PHP", this form will create a temporary file on the server and store the file name of that temp file in `$_FILES['uploadfile']['tmp_name']`. It also creates`$_FILES['uploadfile']['name']` (the original name of the file), `$_FILES['uploadfile']['size']` (the file size in bytes), and `$_FILES['uploadfile']['type']` (the MIME type of the file).

Inserting the file into the database is a relatively straightforward process: open the temporary file, read the data it contains into a PHP variable, and then use that variable in a standard MySQL `INSERT` query. Again, we make use of `is_uploaded_file` to make sure the file name we use does, in fact, correspond to an uploaded file before we do any of this. Here's the code:

```
// Bail out if the file isn't really an upload.
if (!is_uploaded_file($_FILES['uploadfile']['tmp_name']))
  die("$uploadfile is not an uploaded file!");
$uploadfile = $_FILES['uploadfile']['tmp_name'];
$uploadname = $_FILES['uploadfile']['name'];
$uploadtype = $_FILES['uploadfile']['type'];
$uploaddesc = $_POST['desc'];

// Open file for binary reading ('rb')
$tempfile = fopen($uploadfile,'rb');

// Read the entire file into memory using PHP's
// filesize function to get the file size.
$filedata = fread($tempfile,filesize($uploadfile));

// Prepare for database insert by adding backslashes
// before special characters.
$filedata = addslashes($filedata);

// Create the SQL query.
$sql = "INSERT INTO filestore SET
        FileName = '$uploadname',
        MimeType = '$uploadtype',
        Description = '$uploaddesc',
        FileData = '$filedata'";

// Perform the insert.
$ok = @mysql_query($sql);
if (!$ok) die("Database error storing file: " .
              mysql_error());
```

Viewing Stored Files

Armed with the code that accepts file uploads and stores them in a database, you're halfway home. But you still need to be able to pull that data out of the database to use it. For our purposes, this will mean sending the file to a requesting browser.

Once again, this turns out to be a relatively straightforward process. We simply retrieve the data for the requested file from the database and send it on to the Web browser. The only tricky part is to send the browser information *about* the file, including:

- the file size (so that the browser can display accurate download progress information to the user)
- the file type (so that the browser knows what to do with the data it receives, e.g. display it as a Web page, a text file, an image, or offer to save the file)
- the file name (if we don't specify it, the browser will assume all files downloaded from our script have the same file name as the script)

All this information is sent to the browser using *HTTP headers*—special lines of information that precede the transmission of the file data itself. Sending HTTP headers via PHP is quite easy using the `header` function, but as headers must be sent before plain content, any calls to this function must come before anything is output by your script.

The*file size* is specified with a `content-length` header:

```
header('content-length: ' . strlen($filedata));
```

`strlen` is a built-in PHP function that returns the length of the given string. Since binary data is just a string of bytes as far as PHP is concerned, you can use this function to count the length in bytes of the file data.

The*file type* is specified with a `content-type` header:

```
header("content-type: $mimetype");
```

Finally, the *file name* is specified with a `content-disposition` header:

```
header("content-disposition: inline; filename=$filename");
```

Use the script below to fetch a file with a given ID from the database and send it to the browser:

```
$sql = "SELECT FileName, MimeType, FileData
        FROM filestore WHERE ID = '$id'";
$result = @mysql_query($sql);
if (!$result) die('Database error: ' . mysql_error());

$file = mysql_fetch_array($result);
if (!$file)
  die('File with given ID not found in database!');

$filename = $file['FileName'];
$mimetype = $file['MimeType'];
$filedata = $file['FileData'];

header("content-disposition: inline; filename=$filename");
header("content-type: $mimetype");
header('content-length: ' . strlen($filedata));

echo($filedata);
```

One final trick we can add to this code is to allow a file to be downloaded, instead of viewed, if the user so

desires. Web standards suggest that the way to do this is to send a `content-disposition` of `attachment` instead of `inline`. Here's the modified code, which checks if the variable `$action` equals `'dnld'`, which would indicate that this special file type should be sent:

```
$sql = "SELECT FileName, MimeType, FileData
        FROM filestore WHERE ID = '$id'";
$result = @mysql_query($sql);
if (!$result) die("Database error: " . mysql_error());

$file = mysql_fetch_array($result);
if (!$file)
  die('File with given ID not found in database!');

$filename = $file['FileName'];
$mimetype = $file['MimeType'];
$filedata = $file['FileData'];
$disposition = 'inline';

if ($action == 'dnld')
  $disposition = 'attachment';

header("content-disposition: $disposition; filename=$filename");
header("content-type: $mimetype");
header('content-length: ' . strlen($filedata));

echo($filedata);
```

Unfortunately, many browsers do not respect the `content-disposition` header. Netscape 4, Internet Explorer 5, and all Opera browsers, for example, will decide what to do with a file based on the `content-type` header.

To ensure the correct behaviour in as many browsers as possible, we can use the built-in `$_SERVER['HTTP_USER_AGENT']` variable to identify the browser in use. Opera 7 and Internet Explorer 5 browsers can be coerced into displaying the download dialogue by sending a made-up `content-type` of`application/x-download`:

```
if ($action == 'dnld') {
  $disposition = 'attachment';
  if (strpos($_SERVER['HTTP_USER_AGENT'],'MSIE 5') or
      strpos($_SERVER['HTTP_USER_AGENT'],'Opera 7'))
    $mimetype = 'application/x-download';
}
```

The`strpos` function used here is a built-in PHP function that takes two strings and searches for the second string within the first. If it doesn't find it, it returns FALSE, but if it does, it returns the position of the second string in the first as an integer.

Older versions of Opera and Netscape 4 are best left to their own devices; when you mess with the `content-type` header, these browsers display binary files as plain text in the browser window and display an error message, respectively.

The Complete Script

Below you'll find the complete example script. It combines all the elements given above with some simple code that will list the files in the database and allow them to be deleted. As always, this file is available in the code archive.

As you look at the code, you may notice two spots where I've used this command:

```
header('location: '. $_SERVER['PHP_SELF']);
```

As described above, the `header` function sends an HTTP header to the browser. We've seen the `content-type`,`content-length`, and `content-disposition` headers already. The above line sends a `location` header, which redirects the browser to the specified URL. As it specifies `$_SERVER['PHP_SELF']` as the URL, this line redirects the browser to the same page.

'What's the point of that?' you might wonder. Well, I use this line after the script adds or deletes a file in the database. In such cases, the action-add or delete-is specified by a query string in the page's URL. If the script simply took the requested action and immediately displayed the updated list of files, the user could inadvertently repeat that action by refreshing the page! To avoid this kind of error, we instead redirect the browser after it completes each action to a URL that's identical in every way, except that it doesn't contain a query string (`$_SERVER['PHP_SELF']`). Thus, the user can refresh the file list that's produced with no ill effects.

```
<?php

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

$action = '';
if (isset($_GET['action'])) $action = $_GET['action'];

if (($action == 'view' or $action == 'dnld')
    and isset($_GET['id'])) {
  $id = $_GET['id'];

  // User is retrieving a file
  $sql = "SELECT FileName, MimeType, FileData
          FROM filestore WHERE ID = '$id'";
  $result = @mysql_query($sql);
  if (!$result) die('Database error: ' . mysql_error());

  $file = mysql_fetch_array($result);
  if (!$file)
    die('File with given ID not found in database!');

  $filename = $file['FileName'];
  $mimetype = $file['MimeType'];
  $filedata = $file['FileData'];
  $disposition = 'inline';

  if ($action == 'dnld') {
    $disposition = 'attachment';
    if (strpos($_SERVER['HTTP_USER_AGENT'],'MSIE 5') or
        strpos($_SERVER['HTTP_USER_AGENT'],'Opera 7'))
      $mimetype = 'application/x-download';
  }

  header("content-disposition: $disposition; filename=$filename");
```

```
  header("content-type: $mimetype");
  header('content-length: ' . strlen($filedata));

  echo($filedata);
  exit();

} elseif ($action == 'del' and isset($_GET['id'])) {
  $id = $_GET['id'];

  // User is deleting a file
  $sql = "DELETE FROM filestore WHERE ID = '$id'";
  $ok = @mysql_query($sql);
  if (!$ok) die('Database error: ' . mysql_error());

  header('location: ' . $_SERVER['PHP_SELF']);
  exit();

} elseif ($action == 'ulfile') {

  // Bail out if the file isn't really an upload.
  if (!is_uploaded_file($_FILES['uploadfile']['tmp_name']))
    die('There was no file uploaded!');
  $uploadfile = $_FILES['uploadfile']['tmp_name'];
  $uploadname = $_FILES['uploadfile']['name'];
  $uploadtype = $_FILES['uploadfile']['type'];
  $uploaddesc = $_POST['desc'];

  // Open file for binary reading ('rb')
  $tempfile = fopen($uploadfile,'rb');

  // Read the entire file into memory using PHP's
  // filesize function to get the file size.
  $filedata = fread($tempfile,filesize($uploadfile));

  // Prepare for database insert by adding backslashes
  // before special characters.
  $filedata = addslashes($filedata);

  // Create the SQL query.
  $sql = "INSERT INTO filestore SET
          FileName = '$uploadname',
          MimeType = '$uploadtype',
          Description = '$uploaddesc',
          FileData = '$filedata'";

  // Perform the insert.
  $ok = @mysql_query($sql);
  if (!$ok) die('Database error storing file: ' .
                mysql_error());

  header('location: ' . $_SERVER['PHP_SELF']);
  exit();

}

// Default page view: lists stored files
```

```
$sql = 'SELECT ID, FileName, MimeType, Description
        FROM filestore';
$filelist = @mysql_query($sql);
if (!$filelist) die('Database error: ' . mysql_error());
?>
<html>
<head>
<title> PHP/MySQL File Repository </title>
</head>
<body>

<h1>PHP/MySQL File Repository</h1>

<form action="<?=$_SERVER['PHP_SELF']?>?action=ulfile"
      method="post" enctype="multipart/form-data">
<p>Upload File:<br />
<input type="file" name="uploadfile" /></p>
<p>File Description:<br />
<input type="text" name="desc" maxlength="255" /></p>
<p><input type="submit" name="go" value="Upload" /></p>
</form>

<p>The following files are stored in the database:</p>
<table width="85%" border="0" cellpadding="0"
       cellspacing="0">
<tr>
 <th align="left">Filename</th>
 <th align="left">Type</th>
 <th align="left">Description</th>
</tr>
<?php

if (mysql_num_rows($filelist) > 0) {
  while ($f = mysql_fetch_array($filelist)) {
    ?>

<tr valign="top">
 <td nowrap>
  <a href="<?=$_SERVER['PHP_SELF']?>?action=view&id=<?=$f['ID']?>"
    ><?=$f['FileName']?></a>
 </td>
 <td nowrap><?=$f['MimeType']?></td>
 <td><?=$f['Description']?></td>
 <td nowrap>
  [<a href="<?=$_SERVER['PHP_SELF']?>?action=dnld&id=<?=$f['ID']?>
    ">Download</a> |
  <a href="<?=$_SERVER['PHP_SELF']?>?action=del&id=<?=$f['ID']?>"
     onClick="return confirm('Delete this file?');"
  >Delete</a>]
 </td>
</tr>

    <?php
  }
} else {
  ?>
```

```
<tr><td colspan="3" align="center">No Files!</td></tr>
  <?php
}
?>
</table>
</body>
</html>
```

This example demonstrates all the techniques you need in order to juggle binary files with PHP and MySQL, and I invite you to think of some creative uses of this code. Consider, for example, a file archive where users must provide a user name and password before they are allowed to view or download the files. If a user enters an incorrect user name/password combination, your script can display an error page instead of sending the file data. Another possibility would be a script that sends different files depending on the details provided by the form.

Advanced Considerations

In systems like that developed above, large files present some unique challenges to the developer. I'll explain these here briefly, but fully-developed solutions to these problems are beyond the scope of this book.

MySQL Packet Size

By default, MySQL does not accept commands (packets) that are longer than 1MB. This default puts a reasonably severe limit on the maximum file size you can store, unless you're prepared to write your file data in 1MB chunks, using an `INSERT` followed by several `UPDATE`s. You can increase the maximum packet size by setting the `max_allowed_packet` option in your *my.cnf* (or *my.ini*) file. Refer to the MySQL manual for more information about this issue.

PHP Script Timeout

PHP is configured by default to kill PHP scripts that run for more than 30 seconds. For large downloads over slow connections, this limit will be reached fairly quickly! Use PHP's `set_time_limit` function to set an appropriate limit for the download to occur, or simply set the time limit to zero, which allows the script to run to completion, however long it takes. Don't do this unless you're positive your script will always terminate, and not run into an infinite loop!

Summary

In this chapter, we explored the methodologies for storing binary data (e.g. image files, encryption keys, programs for download) in MySQL databases, and retrieving it again for dynamic display on the Web. We also developed a fully-coded, but admittedly simple, online file storage system to test out these techniques.

Chapter 12: Cookies and Sessions in PHP

Cookies and sessions are two of those mysterious technologies that are almost always made out to be more intimidating and complex than they really are. In this chapter, I'll debunk those myths by explaining in simple language what they are, how they work, and what they can do for you. I'll also provide practical examples to demonstrate each, with hints as to how these can be expanded to add exciting new features to your own Website!

Cookies

Most computer programs these days preserve some form of *state* when you close them. Whether it be the position of the application window, or the names of the last five files you worked with, the settings are usually stored in a small file somewhere on your system, so they can be read back the next time the program is run. When Web developers took Web design to the next level, and moved from static pages to complete, interactive, online applications, there was a need for similar functionality in Web browsers-so cookies were born.

A*cookie* is a name/value pair associated with a given Website, and stored on the computer that runs the client (browser). Once it's set by a Website, all future page requests to that same site will also include the cookie, until it *expires*. Other Websites cannot access the cookies set by your site, and vice versa, so, contrary to popular belief, they're a relatively safe place to store personal information. Cookies in and of themselves cannot violate a user's privacy.

Illustrated in "Cookie Life Cycle" is the life cycle of a PHP-generated cookie.

- First, a Web browser requests a URL that corresponds to a PHP script. Within that script is a call to the `setcookie` function built into PHP.
- The page produced by the PHP script is sent back to the browser, along with an HTTP `set-cookie` header that contains the name (e.g. `mycookie`) and value pair of the cookie to be set.
- When it receives this HTTP header, the browser creates and stores the specified value as a cookie named `mycookie`.
- Subsequent page requests to that Website contain an HTTP `cookie` header that sends the name/value pair (`mycookie=value`) to the script requested.
- Upon receipt of a page request with a cookie header, PHP automatically creates an entry in the `$_COOKIE` array[1] with the name of the cookie (`$_COOKIE['mycookie']`) and its value.

Cookie Life Cycle

In other words, the PHP `setcookie` function lets you set a variable that will be automatically set by subsequent page requests from the same browser! Before we examine an actual example, let's take a close look at the `setcookie` function.

```
setcookie(name[, value[, expirytime[,path[,domain[,secure]]]]]);
```

Like the `header` function we saw in "Storing Binary Data in MySQL", the `setcookie` function adds HTTP headers to the page, and thus *must be called before any of the actual page content is sent*. Any attempt to call `setcookie` after some HTML, for example, has already been sent to the browser, will produce a PHP error message.

The only required parameter for this function is *`name`*, which specifies the name of the cookie. Calling `setcookie` with only the *`name`* parameter will actually delete the cookie that's stored on the browser, if it exists. The *`value`* parameter allows you to create a new cookie, or modify the value stored in an existing one.

By default, cookies will remain stored on the browser, and thus will continue to be sent with page requests, until the browser is closed by the user. If you want the cookie to persist beyond the current browser session, you must set the *`expirytime`* parameter to show the number of seconds from January 1st, 1970

to the time at which you want the cookie to be deleted automatically. The current time in this format can be obtained using the PHP `time` function. Thus, a cookie could be set to expire in one hour, for example, by setting *`expirytime`* to `time() + 3600`. To delete a cookie that has a preset expiry time, change this expiry time to represent a point in the past (e.g. one year ago: `time() - 3600 * 24 * 365`). Here's an example:

```
// Set a cookie to expire in 1 year
setcookie('mycookie','somevalue',time()+3600*24*365);

// Delete it
setcookie('mycookie','',time()-3600*24*365);
```

The *`path`* parameter lets you restrict access to the cookie to a given path on your server. For instance, if you set a path of `'/~kyank/'` for a cookie, then only requests for pages in the *~kyank* directory (and its sub-directories) will include the cookie as part of the request. Note the trailing '/', which prevents other scripts in other directories beginning with `/~kyank` from accessing the cookie. This is helpful if you're sharing a server with other users, and each user has a home directory. It allows you to set cookies without exposing your visitors' data to the scripts of other users on your server.

The *`domain`* parameter serves a similar purpose; it restricts the cookie's access to a given domain. By default, a cookie will be returned only to the host that originally sent it. Large companies, however, commonly have several host names for their Web presence (e.g. `www.company.com` and `support.company.com`). To create a cookie that is accessible by pages on both servers, you would set the *`domain`* parameter to `'company.com'`. Note the leading '.', which prevents another site at `somecompany.com` from accessing your cookies on the basis that their domain ends with `company.com`.

The *`secure`* parameter, when set to `1`, indicates that the cookie should be sent only with page requests that happen over a secure (SSL) connection (i.e. with a URL that starts with `https://`).

While all parameters except *`name`* are optional, you must specify values for earlier parameters if you want to specify values for later ones. For instance, you can't call `setcokie` with a *`domain`* value if you don't also specify some value for the *`expirytime`* parameter. To omit parameters that require a value, you can set string parameters (*`value`*,*`path`*,*`domain`*) to `''` (the empty string) and numerical parameters (*`expirytime`* and *`secure`*) to `0`.

Let's now look at an example of cookies in use. Say you wanted to display a special welcome message to people on their first visit to your site. You could use a cookie to indicate that someone had been to your site before and then only display the message when the cookie was not set. Here's the code:

```
if (!isset($_COOKIE['visited'])) { // First visit
  echo("Welcome to my Website! Click here for a tour!");
}
setcookie('visited','1',time()+3600*24*365); // 1 year
```

At first glance, this code would seem to do exactly what we want it to. Unfortunately, it runs into a common pitfall-the welcome message is printed out before `setcookie` is called. Instead of setting the cookie, PHP displays an error message to inform you that HTTP headers cannot be added after page content has already been sent.

To fix the problem, we call `setcookie` before we print out the message:

```
setcookie('visited','1',time()+3600*24*365); // 1 year
if (!isset($_COOKIE['visited'])) { // First visit
  echo('Welcome to my Website! Click here for a tour!');
}
```

This may seem wrong at first glance. If you've never worked with cookies before, you might think that the welcome message will never be printed out, because the cookie is always set before the `if` statement. However, if you think back to the cookie life cycle, you'll realize that the cookie isn't actually set until the browser receives the Web page that's generated by this script. So the cookie that is created by the

`setcookie` line above won't actually exist until the next time a page is requested.

Instead of simply tracking whether or not the user has visited before, you could instead track the number of times he or she has visited. Here's the complete example containing all the HTML tags, which is also included in the code archive (*cookiecounter.php*). Notice how I've structured the document to ensure that `setcookie` happens before any page content is output.

```
<?php
if (!isset($_COOKIE['visits'])) $_COOKIE['visits'] = 0;
$visits = $_COOKIE['visits'] + 1;
setcookie('visits',$visits,time()+3600*24*365);
?>
<html>
<head>
<title> Title </title>
</head>
<body>
<?php
if ($visits > 1) {
  echo("This is visit number $visits.");
} else { // First visit
  echo('Welcome to my Website! Click here for a tour!');
}
?>
</body>
</html>
```

Before you go overboard using cookies, be aware that browsers place a limit on the number and size of cookies allowed per Website. Most browsers will start deleting old cookies to make room for new ones after you have set 20 cookies from your site. Browsers also enforce a maximum combined size for all cookies from all Websites, so an especially cookie-heavy site might cause your own site's cookies to be deleted. For these reasons, you should never store information in a cookie if your application relies on that information being available later on. Instead, cookies are best used for convenient features like automatically logging in a user. If the cookie that contains a visitor's automatic login details is deleted before that person's next visit, he or she can simply re-enter the user name and password by hand.

[1] In versions of PHP prior to 4.1, this array was named `$HTTP_COOKIE_VARS` instead of `$_COOKIE`. This old name also remains in current versions of PHP for backwards compatibility.

PHP Sessions

Because of the limitations I've just described, cookies are not appropriate for storing large amounts of information. Also, because of the negative impression that many people have of cookies, it's not uncommon for users to disable cookies in their browsers. So if you run an ecommerce Website that uses cookies to store the items in a user's shopping cart as the user makes his or her way through your site, this can be a big problem.

Sessions were developed in PHP as the solution to all these issues. Instead of storing all your (possibly large) data as cookies in the Web browser, sessions let you store the data on your Web server. The only thing that's stored on the browser is a single cookie that contains the user's *session ID*—a variable that PHP watches for on subsequent page requests, and uses to load the stored data that's associated with that session.

Unless configured otherwise, a PHP session works by automatically setting in the user's browser a cookie that contains the session ID—a long string of letters and numbers that serves to identify that user uniquely for the duration of his or her visit to your site. The browser then sends that cookie along with every request for a page from your site, so that PHP can determine to which of potentially numerous sessions-in-progress the request belongs. Using a set of temporary files that are stored on the Web server, PHP keeps track of the variables that have been registered in each session, and their values.

One of the big selling points of PHP sessions is that they also work when cookies are *disabled*! If PHP detects that cookies are disabled in the user's browser, it will automatically add the session ID as a query string variable on all the relative links on your page, thus passing the session ID onto the next page. Be aware that all of the pages on your site need to be PHP files for this to work, because PHP won't be able to add the session ID to links on non-PHP pages. Also, for this feature to work, `session.use_trans_sid` must be enabled in your *php.ini* file (see below), and PHP must be compiled with the `--enable-trans-sid` option if you're doing it yourself under Linux, or other Unix variants.

Before you can go ahead and use the spiffy session-management features in PHP, you need to ensure that the relevant section of your *php.ini* file has been set up properly. If you're using a server that belongs to your Web host, it's probably safe to assume this has been done for you. Otherwise, open your *php.ini* file in a text editor and look for the section marked `[Session]`. Beneath it, you'll find twenty-some options that begin with the word `session`. Most of them are just fine if left as-is, but here are a few crucial ones you'll want to check:

```
session.save_handler      = files
session.save_path         = C:\WINDOWS\TEMP
session.use_cookies       = 1
session.use_trans_sid     = 1
```

`session.save_path` tells PHP where to create the temporary files used to track sessions. It must be set to a directory that exists on the system, or you'll get ugly error messages when you try to create a session on one of your pages. Under Unix, */tmp* is a popular choice. In Windows, you could use *C:\WINDOWS\TEMP*, or some other directory if you prefer (I use *D:\PHP\SESSIONS*). With these adjustments made, restart your Web server software to allow the changes to take effect.

You're now ready to start working with PHP sessions. But before we jump into an example, let's quickly look at the most common session management functions in PHP. To tell PHP to look for a session ID, or to start a new session if none is found, you simply call `session_start`. If an existing session ID is found when this function is called, PHP restores the variables that belong to that session. Since this function attempts to create a cookie, it must come before any page content is sent to the browser, just as we saw for `setcookie` above.

```
session_start();
```

To create a session variable, which will be available on all pages in the site when accessed by the current user, simply set a value in the special `$_SESSION` array[2]. For example, the following will store the variable called `pwd` in the current session:

```
$_SESSION['pwd'] = 'mypassword';
```

To remove a variable from the current session, you just use PHP's `unset` function:

```
unset($_SESSION['pwd']);
```

Finally, should you want to end the current session, deleting all registered variables in the process, you can clear all the stored values and use `session_destroy`:

```
$_SESSION = array();
session_destroy();
```

For more detailed information on these and the other session-management functions in PHP, see the relevant section of the PHP Manual. Now that we have these basic functions under our belt, let's put them to work in a simple example.

[2] In PHP versions prior to 4.1, this array was called `$HTTP_SESSION_VARS`. This name also remains available in current versions of PHP for backwards compatibility.

A Simple Shopping Cart

This example will consist of two PHP scripts:

- a product catalogue, where you can add items to your shopping cart
- a check-out page, which displays the contents of the user's shopping cart for confirmation

From the check-out page, the order could then be submitted to a processing system that would handle the details of accepting payment and shipping arrangements. That system is beyond the scope of this book, and will not be discussed here.

As in "Storing Binary Data in MySQL", we'll look at various important fragments of code first and then I'll present the full script, so don't bother typing any of the following snippets into your editor.

Let's start with the list of items we'll have for sale in our online store:

```
<table border="1">
  <tr>
    <th>Item Description</th>
    <th>Price</th>
  </tr>
  <tr>
    <td>Canadian-Australian Dictionary</td>
    <td>$24.95</td>
  </tr>
  <tr>
    <td>Used Parachute (never opened)</td>
    <td>$1,000.00</td>
  </tr>
  <tr>
    <td>Songs of the Goldfish (2CD Set)</td>
    <td>$19.99</td>
  </tr>
  <tr>
    <td>Ending PHP4 (O'Wroxey Press)</td>
    <td>$34.95</td>
  </tr>
</table>
<p>All prices are in imaginary dollars.</p>
```

Now, instead of hard-coding the items, we'll place them in a PHP array and generate the page dynamically. Arrays are created with the built-in PHP function `array`, by listing the elements to be included in the array as the parameters of the function. Items and their prices would normally be stored in a database, but I'm using this method so we can focus on sessions. You should already know all you need in order to put together a database-driven product catalogue, so I'll leave that to you as an exercise.

Here's the code for our dynamically generated product catalogue:

```
<?php
$items = array( 'Canadian-Australian Dictionary',
                'Used Parachute (never opened)',
                'Songs of the Goldfish (2CD Set)',
                'Ending PHP4 (O\'Wroxey Press)');
$prices = array( 24.95, 1000, 19.99, 34.95 );
?>
<table border="1">
  <tr>
```

```
    <th>Item Description</th>
    <th>Price</th>
  </tr>
<?php
  for($i = 0; $i < count($items); $i++) {
    echo('<tr>');
    echo('<td>'.$items[$i].'</td>');
    echo('<td>$'.number_format($prices[$i],2).'</td>');
    echo('</tr>');
  }
?>
</table>
<p>All prices are in imaginary dollars.</p>
```

This code produces the HTML we saw above. The table row for each item is created using a `for` loop that counts through the `$items` array (the function `count` returns the number of items in the array). `for` loops were introduced in "Getting Started with PHP". We also use PHP's built-in `number_format` function to display the prices with two digits after the decimal point (see the PHP Manual for more information about this function).

Now, we're going to store the list of items the user placed in the shopping cart in yet another array. Because we'll need this variable to persist throughout a user's visit to your site, we'll store it using PHP sessions. Here's the code responsible for all this:

```
session_start();
if (!isset($_SESSION['cart']))
  $_SESSION['cart'] = array();
```

`session_start` either starts a new session (and sets the session ID cookie), or restores the variables registered in the existing session, if one exists. The code then checks if `$_SESSION['cart']` exits, and, if not, initializes it to an empty array to represent the empty cart.

We can now print out the number of items in the user's shopping cart:

```
<p>Your shopping cart contains
  <?=count($_SESSION['cart'])?> items.</p>
```

Now, the user's shopping cart is going to stay very empty if we don't provide a way to add items to it, so let's modify the `for` loop that displays our table of items, and enable it to provide a 'Buy' link on the end of each row:

```
<?php
  for($i = 0; $i < count($items); $i++) {
    echo('<tr>');
    echo('<td>'.$items[$i].'</td>');
    echo('<td>$'.number_format($prices[$i],2).'</td>');
    echo('<td><a href="' . $_SERVER['PHP_SELF'] .
         '?buy=' . $i . '">Buy</a></td>');
    echo('</tr>');
  }
?>
```

Now each product in our catalogue has a link back to the catalogue with `buy=`*n* in the query string, where *n* is the index of the item that's to be added to the shopping cart in the `$items` array. We can then watch for the `$_GET['buy']` variable to appear at the top of the page, and, when it does, we'll add the item to the`$_SESSION['cart']` array before redirecting the browser back to the same page, but with no query string, thereby ensuring that refreshing the page doesn't repeatedly add the item to the cart.

```
if (isset($_GET['buy'])) {
  // Add item to the end of the $_SESSION['cart'] array
  $_SESSION['cart'][] = $_GET['buy'];
  header('location: ' . $_SERVER['PHP_SELF']);
  exit();
}
```

Now, this works just fine if the user has cookies enabled, but when cookies are disabled, PHP's automatic link adjustment doesn't affect the `header` function, so the session ID gets lost at this point. Fortunately, if PHP identifies that cookies are disabled, it creates a constant named `SID`, a string of the form `PHPSESSID=`*`somevalue`*, which will pass on the session ID. We can make our code compatible with disabled cookie support as follows:

```
if (isset($_GET['buy'])) {
  // Add item to the end of the $_SESSION['cart'] array
  $_SESSION['cart'][] = $_GET['buy'];
  header('location: '. $_SERVER['PHP_SELF'] . '?' . SID);
  exit();
}
```

All that's left is to add a 'view your cart' link to the appropriate page, and we'll have completed the catalogue script. Here is the complete code for this page (*catalog.php*-it's also available in the code archive).

```
<?php
session_start();
if (!isset($_SESSION['cart']))
  $_SESSION['cart'] = array();
if (isset($_GET['buy'])) {
  // Add item to the end of the $_SESSION['cart'] array
  $_SESSION['cart'][] = $_GET['buy'];
  header('location: ' . $_SERVER['PHP_SELF'] . '?' . SID);
  exit();
}
?>
<html>
<head>
<title>Product catalogue</title>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<p>Your shopping cart contains
  <?=count($_SESSION['cart'])?> items.</p>
<p><a href="cart.php">View your cart</a></p>
<?php
$items = array( 'Canadian-Australian Dictionary',
                'Used Parachute (never opened)',
                'Songs of the Goldfish (2CD Set)',
                'Ending PHP4 (O\'Wroxey Press)');
$prices = array( 24.95, 1000, 19.99, 34.95 );
?>
<table border="1">
  <tr>
    <th>Item Description</th>
    <th>Price</th>
  </tr>
<?php
  for($i = 0; $i < count($items); $i++) {
    echo('<tr>');
```

```
    echo('<td>'.$items[$i].'</td>');
    echo('<td>$'.number_format($prices[$i],2).'</td>');
    echo('<td><a href="' . $_SERVER['PHP_SELF'] .
         '?buy=' . $i . '">Buy</a></td>');
    echo('</tr>');
  }
?>
</table>
<p>All prices are in imaginary dollars.</p>
</body>
</html>
```

Now the code for *cart.php* is very similar to the product catalogue. All it does is take the `$_SESSION['cart']` variable (after loading the session, of course), and use the numbers stored there to print out the corresponding items from the `$items` array. We also provide a link that loads the page with query string `empty=1`, and causes the script to `unset` the `$_SESSION['cart']` variable, which results in a new, empty shopping cart. Here's the code (also available in the code archive):

```
<?php
session_start();
if (!isset($_SESSION['cart']))
  $_SESSION['cart'] = array();
if (isset($_GET['empty'])) {
  // Empty the $_SESSION['cart'] array
  unset($_SESSION['cart']);
  header('location: ' . $_SERVER['PHP_SELF'] . '?' . SID);
  exit();
}
?>
<html>
<head>
<title>Shopping cart</title>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<h1>Your Shopping Cart</h1>
<?php
$items = array( 'Canadian-Australian Dictionary',
                'Used Parachute (never opened)',
                'Songs of the Goldfish (2CD Set)',
                'Ending PHP4 (O\'Wroxey Press)');
$prices = array( 24.95, 1000, 19.99, 34.95 );
?>
<table border="1">
  <tr>
    <th>Item Description</th>
    <th>Price</th>
  </tr>
<?php
  $total = 0;
  for($i = 0; $i < count($_SESSION['cart']); $i++) {
    echo('<tr>');
    echo('<td>'.$items[$_SESSION['cart'][$i]].'</td>');
    echo('<td align="right">$');
    echo(number_format($prices[$_SESSION['cart'][$i]],2));
    echo('</td>');
    echo('</tr>');
    $total = $total + $prices[$_SESSION['cart'][$i]];
```

```
  }
?>
<tr>
<th align="right">Total:</th><br />
<th align="right">$<?=number_format($total,2)?></th>
</tr>
</table>
<p><a href="catalog.php">Continue Shopping</a> or
<a href="<?=$_SERVER['PHP_SELF']?>?empty=1">Empty your cart</a>
</p>
</body>
</html>
```

Summary

In this chapter, you learned about the two main methods of creating persistent variables-those variables that continue to exist from page to page in PHP. The first stores the variable in the visitor's browser in the form of a *cookie*. By default, cookies terminate at the end of the browser session, but by specifying an expiry time, they can be preserved indefinitely. Unfortunately, cookies are fairly unreliable because you have no way of knowing when the browser might delete your cookies, and because some users configure their browsers to disable cookies. Obviously, cookies are not something that should be relied upon.

Sessions, on the other hand, free you from all the limitations of cookies. They don't rely on cookies to work, and they let you store an unlimited number of potentially large variables. Sessions are an essential building block in modern ecommerce applications, as we demonstrated in our simple shopping cart example.

Appendix A: MySQL Syntax

Overview

This appendix describes the syntax of the majority of SQL statements implemented in MySQL, as of version 3.23.54a (current as of this writing).

The following conventions are used in this reference:

- Queries are listed in alphabetical order for easy reference.
- Optional components are surrounded by square brackets ([]), while mutually exclusive alternatives appear in braces, separated by vertical bars (|).
- Lists of elements from which one element must be chosen are surrounded by braces ({}).
- An ellipsis (. . .) means that the preceding element may be repeated.

The query syntax documented in this appendix has been simplified in several places by the omission of the alternative syntax, and of keywords that didn't actually perform any function, but were originally included for compatibility with more advanced database systems. Query features having to do with some advanced features such as transactions were also omitted. For a complete, up-to-date reference to supported MySQL syntax, see the MySQL Reference Manual.

ALTER TABLE

```
ALTER [IGNORE] TABLE tbl_name action [, action ...]
```

In this code, each *action* refers to an action as defined below.

ALTER TABLE queries may be used to change the definition of a table, without losing any of the information in the table (except in obvious cases, such as the deletion of a column). Here are the main actions that are possible:

- `ADD [COLUMN] create_definition [FIRST | AFTER column_name]`

  This action adds a new column to the table. The syntax for *create_definition* is as described for "CREATE TABLE". By default, the column will be added to the end of the table, but by specifying FIRST or AFTER column_name, you can place the column wherever you like. The optional word COLUMN does not actually do anything-leave it off unless you particularly like to see it there.

- `ADD INDEX [index_name] (index_col_name, ...)`

  This action creates a new index to speed up searches based on the column(s) specified. It's usually a good idea to assign a name to your indices by specifying the index_name, otherwise, a default name based on the first column in the index will be used. When creating an index based on CHAR and/or VARCHAR columns, you can specify a number of characters to index as part of index_col_name (e.g.myColumn(5) will index the first 5 characters of myColumn). When indexing BLOB and TEXT columns, this number must be specified. For detailed information on indexes, see the MySQL Reference Manual, or Mike Sullivan's excellent article Optimizing your MySQL Application on SitePoint.

- `ADD PRIMARY KEY (index_col_name, ...)`

  This action creates an index for the specified row(s) with the name 'PRIMARY', identifying it as the primary key for the table. All values (or combinations of values) must be unique, as described for the ADD UNIQUE action below. This will cause an error if a primary key already exists for this table. index_col_name is defined as it is for the ADD INDEX action above.

- `ADD UNIQUE [index_name] (index_col_name, ...)`

  This action creates an index on the specified columns, but with a twist: all values in the designated column, or all combinations of values, if more than one column is included in the index, must be unique. The parameters index_name and index_col_name are defined as they are for the ADD INDEX action above.

- `ALTER [COLUMN] col_name {SET DEFAULT value | DROP DEFAULT}`

  This action assigns a new default value to a column (SET DEFAULT), or removes the existing default value (DROP DEFAULT). Again, the word COLUMN is completely optional, and has no effect one way or the other.

- `CHANGE [COLUMN] col_name create_definition`

  This action replaces an existing column (col_name) with a new column, as defined by *create_definition* (the syntax of which is as specified for "CREATE TABLE"). The data in the existing column is converted, if necessary, and placed in the new column. Note that *create_definition* includes a new column name, so this action may be used to rename a column, if you wish. If you want to leave the name of the column unchanged, however, don't forget to include it twice (once for col_name and once for *create_definition*), or use the MODIFY action below.

- `MODIFY [COLUMN] create_definition`

Nearly identical to the `CHANGE` action above, this action lets you specify a new declaration for a column in the table, but assumes you will not be changing its name. Thus, you simply have to re-declare the column with the same name in the *`create_definition`* parameter (as defined for "CREATE TABLE"). As before, `COLUMN` is completely optional and does nothing. Although convenient, this action is not standard SQL syntax, and was added for compatibility with an identical extension in Oracle database servers.

- `DROP [COLUMN] col_name`

  Fairly self-explanatory, this action completely removes a column from the table. The data in that column is irretrievable after this query completes, so be sure of the column name. `COLUMN`, as usual, can be left off-it doesn't do anything but make the query sound better when read aloud.

- `DROP PRIMARY KEY`

- `DROP INDEX index_name`

  The above two actions are quite self-explanatory: they remove from the table the primary key, and a specific index, respectively.

- `RENAME [TO | AS] new_tbl_name`

  This action renames the table. The words `TO` and `AS` are completely optional, and don't do anything. Use ‘em if you like ‘em.

- `ORDER BY col_name`

  Lets you sort the entries in a table by a particular column. However, this is not a persistent state; as soon as new entries are added to the table, or existing entries modified, the ordering can no longer be guaranteed. The only practical use of this action would be to increase performance of a table that you regularly sorted in a certain way in your application's `SELECT` queries. Under some circumstances, arranging the rows in (almost) the right order to begin with will make sorting quicker.

- *`table_options`*

  Using the same syntax as in the `CREATE TABLE` query, this action allows you to set and change advanced table options. These options are fully documented in the MySQL Reference Manual.

ANALYZE TABLE

```
ANALYZE TABLE tbl_name[, tbl_name, ...]
```

This function updates information that's used by the SELECT query in the optimization of queries that take advantage of table indices. It pays in performance to periodically run this query on tables whose contents change a lot over time. The table(s) in question are locked 'read-only' while the analysis runs.

CREATE DATABASE

```
CREATE DATABASE [IF NOT EXISTS] db_name
```

This action simply creates a new database with the given name (`db_name`). This query will fail if the database already exists (unless `IF NOT EXISTS` is specified), or if you don't have the required privileges.

CREATE INDEX

```
CREATE [UNIQUE | FULLTEXT] INDEX index_name ON tbl_name
(col_name[(length)], ...)
```

This query creates a new index on an existing table. It works identically to `ALTER TABLE ADD INDEX`, described in "ALTER TABLE".

CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
[db_name.]tbl_name [(create_definition, ...)]
[table_options] [[IGNORE | REPLACE] select_statement]
```

Where*create_definition* is:

```
{   col_name type [NOT NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [PRIMARY KEY]

  | PRIMARY KEY (index_col_name, ...)

  | INDEX [index_name] (index_col_name, ...)

  | UNIQUE [INDEX] [index_name] (index_col_name, ...) }
```

In this code, *type* is a MySQL column type (see "MySQL Column Types"), and *index_col_name* is as described for ALTER TABLE ADD INDEX in "ALTER TABLE".

CREATE TABLE is used to create a new table called tbl_name in the current database (or in a specific database if db_name.tbl_name is specified instead). If TEMPORARY is specified, the table disappears upon termination of the connection by which it was created. Creating a temporary table with the same name as an existing table will hide the existing table from the current client session until the temporary table is deleted or the session ends; however, other clients will continue to see the original table.

Assuming TEMPORARY is not specified, this query will fail if a table with the given name already exists, unless IF NOT EXISTS is specified. A CREATE TABLE query will also fail if you don't have the required privileges.

Most of the time, the name of the table will be followed by a series of column declarations (*create_definition* above). Each column definition includes the name and data type for the column, and any of the following options:

- NOT NULL

  This specifies that the column may not be left empty (NULL). Note that NULL is a special 'no value' value, which is quite different from, say, an empty string (''). A column of type VARCHAR, for instance, which is set NOT NULL may be set to '' but will not be NULL. Likewise, a NOT NULL column of type INT may contain zero (0), which is a value, but it may not contain NULL, as this is not a value.

- DEFAULT default_value

  DEFAULT lets you specify a value to be given to a column when no value is assigned in an INSERT statement. When this option is not specified, NULL columns (columns that don't have the NOT NULL option set) will be assigned a value of NULL when they are not given a value in an INSERT statement. NOT NULL columns will instead be assigned a 'default default value' (an empty string (''), zero (0), '0000-00-00', or a current timestamp, depending on the data type of the column.

- AUTO_INCREMENT

  As described in "Getting Started with MySQL", an AUTO_INCREMENT column will automatically insert a number that is one greater than the current highest number in that column, when NULL is inserted. AUTO_INCREMENT columns must also be NOT NULL, and either a PRIMARY KEY or UNIQUE.

- PRIMARY KEY

  This option specifies that the column in question should be the primary key for the table; that is, the values in the column must uniquely identify each of the rows in the table. This forces the values in this

column to be unique, and speeds up searches for items based on this column by creating an index of the values it contains.

- `UNIQUE`

  Very similar to `PRIMARY KEY`, this option requires all values in the column to be unique, and indexes the values for high speed searches.

The *`table_options`* portion of the `CREATE TABLE` query is used to specify advanced properties of the table, and is described in detail in the MySQL Reference Manual.

The *`select_statement`* portion of the `CREATE TABLE` query allows you to create a table from the results of a `SELECT` query (see "SELECT"). When you create this table, you don't have to declare the columns that correspond to those results separately. This type of query is useful if you want to obtain the result of a SELECT query, store it in a temporary table, and then perform a second `SELECT` query on it. To some extent, this may be used to make up for MySQL's lack of support for *sub-selects*[1], which allow you to perform the same type of operation in a single query.

[1] Support for sub-selects is planned for inclusion in MySQL 4.1

DELETE

```
DELETE [LOW_PRIORITY] FROM tbl_name [WHERE where_clause]
[LIMIT rows]
```

This query deletes all rows from the specified table, unless the optional (but desirable!) `WHERE` or `LIMIT` clauses are specified. The `WHERE` clause works the same way as its twin in the `SELECT` query (see "SELECT"). The `LIMIT` clause simply lets you specify the maximum number of rows to be deleted.

The`LOW_PRIORITY` option causes the query to wait until there are no clients reading from the table to perform the operation.

DESCRIBE

`{DESCRIBE | DESC} tbl_name [col_name | wild]`

Supplies information about the columns, a specific column (`col_name`), or any columns that match a pattern containing wild cards '%' and '_' (*`wild`*), that make up the specified table. The information returned includes the column name, its type, whether it accepts `NULL` as a value, whether the column has an index, the default value for the column, and any extra features it has (e.g. `AUTO_INCREMENT`).

DROP DATABASE

```
DROP DATABASE [IF EXISTS] db_name
```

This is a dangerous command. It will immediately delete a database along with all of its tables. This query will fail if the database does not exist (unless `IF EXISTS` is specified), or if you don't have the required privileges.

DROP INDEX

```
DROP INDEX index_name ON tbl_name
```

`DROP INDEX` has the exact same effect as `ALTER TABLE DROP INDEX`, described in "ALTER TABLE".

DROP TABLE

```
DROP TABLE [IF EXISTS] tbl_name [, tbl_name, ...]
```

This query completely deletes one or more tables. *This is a dangerous query*, since the data cannot be retrieved once this action is executed, so be very careful with it!

This query will fail with an error if the table doesn't exist (unless `IF EXISTS` is specified) or if you don't have the required privileges.

EXPLAIN

The explain query has two very different forms. The first,

`EXPLAIN tbl_name`
is equivalent to `DESCRIBE tbl_name` or `SHOW COLUMNS FROM tbl_name`.

The second format,

`EXPLAIN`*`select_satement`*

where *`select_statement`* can be any valid `SELECT` query, will produce an explanation of how MySQL would determine the results of the `SELECT` statement. This query is useful for finding out where indexes will help speed up your `SELECT` queries, and also to determine if MySQL is performing multi-table queries in optimal order. See the `STRAIGHT_JOIN` option of the SELECT query in "SELECT" for information on how to override the MySQL optimizer, and control this order manually. See the MySQL Reference Manual for complete information on how to interpret the results of an `EXPLAIN` query.

GRANT

```
GRANTpriv_type [(column_list)], ...
  ON {tbl_name | * | *.* | db_name.*}
  TO user_name [IDENTIFIED BY 'password'], ...
  [WITH GRANT OPTION]
```

GRANT adds new access privileges to a user account, and creates a new account if the specified user_name does not yet exist, or changes the password if IDENTIFIED BY 'password' is used on an account that already has a password.

See"MySQL Access Control" for a complete description of this query.

INSERT

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] tbl_name

{   [(col_name, ...)] VALUES (expression, ...), ...

  | [(col_name, ...)] SELECT ...

  | SET col_name=expression, col_name=expression, ... }
```

TheINSERT query is used to add new entries to a table. It supports three general options:

- LOW_PRIORITY

  The query will wait until there are no clients reading from the table before it proceeds.

- DELAYED

  The query completes immediately from the client's point of view, and the INSERT operation is performed in the background. This option is useful when you wish to insert a large number of rows without waiting for the operation to complete. Be aware that the client will not know the last inserted ID on an AUTO_INCREMENT column when a DELAYED insert is performed (e.g. mysql_insert_id in PHP will not work correctly).

- IGNORE

  Normally, when an inserted operation causes a clash in a PRIMARY KEY or UNIQUE column, the insert fails and produces an error message. This option allows the insert to fail silently-the new row is not inserted, but no error message is displayed.

The word INTO is entirely optional, and has no effect on the operation of the query.

As you can see above, INSERT queries may take three forms. The first form lets you insert one or more rows by specifying the values for the table columns in parentheses. If the optional list of column names is omitted, then the list(s) of column values must include a value for every column in the table, in the order in which they appear in the table.

In the second form of INSERT, the rows to be inserted result from a SELECT query. Again, if the list of column names is omitted, the result set of the SELECT must contain values for each and every column in the table, in the correct order. A SELECT query that makes up part of an insert statement may not contain anORDER BY clause, and you cannot use the table into which you are inserting in the FROM clause.

The third and final form of INSERT can be used only to insert a single row, but it very intuitively allows you to assign values to the columns in that row by giving them in col_name=value format.

Columns to which you assign no value (e.g. if you leave them out of the column list) are assigned their default. By default, inserting a NULL value into a NOT NULL field will also cause that field to be set to its default value; however, if MySQL is configured with the DONT_USE_DEFAULT_FIELDS option enabled, such an INSERT operation will cause an error. For this reason, it's best to avoid them.

LOAD DATA INFILE

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE
  'file_name.txt' [REPLACE | IGNORE] INTO TABLE tbl_name
  [FIELDS
    [TERMINATED BY 'string']
    [[OPTIONALLY] ENCLOSED BY 'char']
    [ESCAPED BY 'char'] ]
  [LINES TERMINATED BY 'string']
  [IGNORE number LINES]
  [(col_name, ...)]
```

The `LOAD DATA INFILE` query is used to import data from a text file either on the MySQL server, or on the `LOCAL` (client) system (for example, a text file created with a `SELECT INTO OUTFILE` query). The syntax of this command is given above; however, the reader is referred to the MySQL Reference Manual for a complete explanation of this query and the issues that surround it.

LOCK/UNLOCK TABLES

```
LOCK TABLES
  tbl_name [AS alias] {READ | [LOW_PRIORITY] WRITE},
  tbl_name ...
```

LOCK TABLES locks the specified table(s) so that the current connection has exclusive access to them, while other connections will have to wait until the lock is released with UNLOCK TABLES, with another LOCK TABLES query, or with the closure of the current connection.

AREAD lock prevents the specified table(s) from being written by this, or any other connection. This allows you to make certain that the contents of a table (or set of tables) are not changed for a certain period of time.

AWRITE lock prevents all other connections from reading or writing the specified table(s). It's useful when a series of INSERT or UPDATE queries must be performed together to maintain the integrity of the data model in the database. New support for *transactions* in MySQL provides more robust support for these types of “grouped queries” (see the sidebar in "LOCK ing TABLES" for details).

By default, a WRITE lock that is waiting for access to a table will take priority over any READ locks that may also be waiting. To specify that a WRITE lock should yield to all other READ lock requests, you can use the LOW_PRIORITY option. Be aware, however, that if there are always READ lock requests pending, a LOW_PRIORITY WRITE lock will never be allowed to proceed.

When locking tables, you must list the same aliases that you’re going to use in the queries you will be performing. If, for example, you are going to refer to the same table with two different aliases in one of your queries, you will need to obtain a lock for each of those aliases beforehand.

For more information on locking tables, see "LOCK ing TABLES".

OPTIMIZE TABLE

```
OPTIMIZE TABLE tbl_name[, tbl_name, ...]
```

Much like a hard disk partition becomes fragmented if old files are deleted or resized, MySQL tables become fragmented over time as you delete rows and modify variable-length columns (such as `VARCHAR` or`BLOB`). This query performs the database equivalent of a 'defrag' on the table, reorganizing the data it contains to eliminate wasted space.

It's important to note that a table is *locked* while an optimize operation is going on, so if your application relies on a large table being constantly available, that application will grind to a halt while the optimization takes place. In such cases, it's better to copy the table, optimize the copy, and then replace the old table with the newly optimized version using a `RENAME` query. Changes made to the original table in the interim will be lost, so this technique is not appropriate for all applications.

RENAME TABLE

```
RENAME TABLE tbl_name TO new_table_name[, tbl_name2 TO ..., ...]
```

This query quickly and conveniently renames one or more tables. This differs from `ALTER TABLE tbl_name RENAME` in that all the tables being renamed in the query are locked for the duration of the query, so that no other connected clients may access them. As the MySQL Reference Manual explains, this assurance of atomicity lets you replace a table with an empty equivalent, for example, if you wanted to start a new table once a certain number of entries was reached, safely:

```
CREATE TABLE new_table (...)
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

You can also move a table from one database to another by specifying the table name as `db_name.tbl_name`, as long as both tables are stored on the same physical disk, which is usually the case.

You must have `ALTER` and `DROP` privileges on the original table, as well as `CREATE` and `INSERT` privileges on the new table to perform this query. A `RENAME TABLE` query that fails to complete halfway through will be automatically reversed, so that the original state is restored.

REPLACE

```
REPLACE [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] tbl_name

{   [(col_name, ...)] VALUES (expression, ...), ...

  | [(col_name, ...)] SELECT ...

  | SET col_name=expression, col_name=expression, ... }
```

REPLACE is identical to INSERT, except that if an inserted row clashes with an existing row in a PRIMARY KEY or UNIQUE column, the old entry is replaced with the new.

REVOKE

```
REVOKEpriv_type [(column_list)], ...
  ON {tbl_name | * | *.* | db_name.*}
  FROM user, ...
```

This function removes access privileges from a user account. If all privileges are removed from an account, the user will still be able to log in, though he or she simply won't be able to access any information.

See"MySQL Access Control" for a complete description of this query.

SELECT

```
SELECT [select_options]
  select_expression, ...
  [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
  [FROM table_references
    [WHERE where_definition]
    [GROUP BY {col_name | col_pos } [ASC | DESC], ...]
    [HAVING where_definition]
    [ORDER BY {col_name | col_pos } [ASC | DESC], ...]
    [LIMIT [offset,] rows]]
```

SELECT is the most complex query in SQL, and is used to perform all data retrieval operations. This query supports the following *select_options*, which may be specified in any sensible combination by simply listing them separated by spaces:

- STRAIGHT_JOIN

  Forces MySQL to join multiple tables specified in the *table_references* argument in the order specified there. If you think MySQL's query optimizer is doing it the 'slow way', this argument lets you override it. For more information on joins, see "Joins" below.

- SQL_SMALL_RESULT

  This option shouldn't be needed in MySQL 3.23 or later; however, it remains available for compatibility. This option informs MySQL that you are expecting a relatively small result set from a query that uses the DISTINCT option or the GROUP BY clause, so it uses the faster, but more memory-intensive, method of generating a temporary table in memory to hold the result set as it is created.

- SQL_BIG_RESULT

  Along the same lines as SQL_SMALL_RESULT, this option informs MySQL that you are expecting a large number of results from a query that makes use of DISTINCT or GROUP BY. When it creates the result set, MySQL will create on disk, as needed, a temporary table in which the results are sorted. This is a quicker solution than generating an index on the temporary table, which would take longer to update for each result row in a large result set.

- SQL_BUFFER_RESULT

  This option forces MySQL to store the result set in a temporary table. This frees up the tables that were used in the query for use by other processes, while the result set is transmitted to the client.

- HIGH_PRIORITY

  This option does exactly what it says—it assigns a high priority to the SELEC query. Normally, if a query is waiting to update a table, all read-only queries (such as SELEC) mut yieto it. A SELEC HIGH_PRIORITY, however, will go first.

- DISTINCT | DISCTINCTROW | ALL

  Any one of these three options may be used to specify the treatment of duplicate rows in the result set.AL (the default) specifies that all duplcate rows appear in the result set, while DISTINCT and DISTINCTROW (they have the same effect) specify that duplicate rows should be eliminated from the result set.

select_expression defines a column of the result set to be returned by the query. Typically, this is a table colmn name, and may be specified as col_name,tbl_name.col_name, or db_name.tbl_name.col_name, depending on how specific you need to be for MySQL to know to which

particular column you are referring. *select_expressions* need not refer to a database column—a simple mathematical formula such as `1 + 1` or a complex expression calculated with MySQL functions may also be used. Here's an example of the latter, which will give the date one month from now in the form "January 1st, 2002":

```
SELECT DATE_FORMAT(
  DATE_ADD(CURDATE(),INTERVAL 1 MONTH),'%M %D, %Y')
```

select_expressions may also contain an alias, or assigned name for the result column, if the expression is followed with `[AS] alias` (the AS is entirely optional). This expression must be used when referring to that column elsewhere in the query (e.g. in `WHERE` and `ORDER BY` clauses), as follows:

```
SELECT JokeDate AS JD FROM Jokes ORDER BY JD ASC
```

MySQL lets you use an `INTO` clause to output the results of a query into a file instead of returning them to the client. The most typical use of this is to export the contents of a table into a text file containing comma-separated values (CSV). Here's an example:

```
SELECT * INTO OUTFILE '/home/user/myTable.txt'
FIELDS TERMINATED BY ',' OPTIONALY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM myTable
```

The file to which the results are dumped must not exist beforehand, or this query will fail. This restriction prevents an SQL query from being used to overwrite critical operating system files. The created file will also be world-readable on systems that support file security, so consider this before you export sensitive data to a text file that anyone on the system can read.

`DUMPFILE` may be used instead of `OUTFILE` to write to the file only a single row, without row or column delimiters. It can be used, for example, to dump a BLOB stored in the table to a file (`SELECT blobCol INTO DUMPFILE ...`). For complete information on the `INTO` clause, see the MySQL Reference Manual. For information on reading data back from a text file, see "LOAD DATA INFILE".

The `FROM` clause contains a list of tables from which the rows composing the result set should be formed. At its most basic, *table_references* is a comma-separated list of one or more tables, which may be assigned aliases with or without using `AS` as described above for *select_expression*. If you specify more than one table name, you are performing a *join*. These are discussed in "Joins" below.

The *where_definition* in the `WHERE` clause sets the condition for a row to be included in the table of results sent in response to the `SELECT` query. This may be a simple condition (e.g. `id=5`) or a complex expression that makes use of MySQL functions and combines multiple conditions using Boolean operators (`AND`,`OR`,`NOT`).

The `GROUP BY` clause lets you specify one or more columns (by name, alias, or column position, where 1 is the first column in the result set) for which rows with equal values should be collapsed into single rows in the result set. This clause should usually be used in combination with the MySQL grouping functions such as `COUNT`,`MAX`, and `AVG`, described in "MySQL Functions", to produce result rows that give summary information about the groups produced. By default, the grouped results are sorted in ascending order of the grouped column(s); however, the `ASC` or `DESC` argument may be added following each column reference to explicitly sort the results in ascending or descending order for that column, respectively. Results are sorted by the first column listed, and then tying sets of rows are sorted by the second, and so on.

Note that the WHERE clause is processed before `GROUP BY` grouping occurs, so conditions in the `WHERE` clause may not refer to columns produced by the grouping operation. To impose conditions on the post-grouping result set, you should use the `HAVING` clause instead. This clause's syntax is identical to that of the `WHERE` clause, except the conditions specified here are processed just prior to returning the set of results, and are not optimized in any way. For this reason, you should use the `WHERE` clause whenever possible. For more information on `GROUP BY` and the `HAVING` clause, see "Advanced SQL".

The `ORDER BY` clause lets you sort results according the values in one or more rows before they are returned. As for the `GROUP BY` clause, each column may be identified by a column name, alias, or position (where 1 is the first column in the result set), and each column may have an `ASC` or `DESC` argument to specify that sorting occurs in ascending or descending order, respectively (ascending is the default). Rows are initially sorted by the first column listed, and then tying sets of rows are sorted by the second, and so on.

The `LIMIT` clause instructs the query to return only a portion of the results it would normally generate. In the simple case, `LIMIT`*n* returns only the first *n* rows of the complete result set. You can also specify an offset by using the form `LIMIT`*x*`,`*n*. In this case, up to *n* rows will be returned, beginning from the *x*th row of the complete result set. The first row corresponds to `x = 0`, the second to `x = 1` and so on.

Joins

As described above, the `FROM` clause of a `SELECT` query lets you specify the tables that are combined to create the result set. When multiple tables are combined in this way, it is called a *join*. MySQL supports several types of joins, as defined by the following supported syntaxes for the *`table_references`* component of the `FROM` clause above:

```
table_ref
table_references,table_ref
table_references [CROSS] JOIN table_ref
table_references INNER JOIN table_refjoin_condition
table_references STRAIGHT_JOIN table_ref
table_references LEFT [OUTER] JOIN table_refjoin_condition
{ oj table_ref LEFT OUTER JOIN table_ref ON cond_expr }
table_references NATURAL [LEFT [OUTER]] JOIN table_ref
table_references RIGHT [OUTER] JOIN table_refjoin_condition
table_references NATURAL [RIGHT [OUTER]] JOIN table_ref
```

Where*`table_ref`* is defined as:

```
table_name [[AS] alias] [USE INDEX (key_list)]
  [IGNORE INDEX (key_list)]
```

and*`join_condition`* is defined as one of the following:

```
ONcond_expr
USING (column_list)
```

Don't be disheartened by the sheer variety of join types; I'll explain how each of them works below.

The most basic type of join, an *inner join*, produces rows made up of all possible pairings of the rows from the first table with the second. You can perform an inner join in MySQL either by separating the table names with a comma (`,`) or with the words `JOIN`,`CROSS JOIN`, or `INNER JOIN` (these are all equivalent).

Normally, the `WHERE` clause of the `SELECT` query is used to specify a condition to narrow down which of the combined rows are actually returned (e.g. to match up a primary key in the first table with a column in the second); however, when the `INNER JOIN` syntax is used, the `ON` form of the *`join_condition`* can play this role as well. As a final alternative, the `USING (`*`column_list`*`)` form of *`join_condition`* lets you specify columns that must match between the two tables. For example,

```
SELECT * FROM t1 INNER JOIN t2 USING (aid)
```

is equivalent to

```
SELECT * FROM t1 INNER JOIN t2 ON t1.aid = t2.aid
```

and also

```
SELECT * FROM t1, t2 WHERE t1.aid = t2.aid
```

`STRAIGHT_JOIN` works the same as an inner join, except that the tables are processed in the order listed (left first, then right). Normally MySQL selects the order that will produce the shortest processing time, but if you think you know better, you can use a `STRAIGHT_JOIN`.

The second type of join is an *outer join*, which is accomplished in MySQL with `LEFT/RIGHT [OUTER] JOIN` (`OUTER` is completely optional, and has no effect). In a `LEFT` outer join, any row in the left-hand table that has no matching rows in the right-hand table (as defined by the *`join_condition`*), will be listed as a single row in the result set. `NULL` values will appear in all the columns that come from the right-hand table.

The`{ oj ... }` syntax is equivalent to a standard left outer join; it is included for compatibility with other ODBC databases.

`RIGHT` outer joins work in the same way as `LEFT` outer joins, except in this case, it is the table on the right whose entries are always included, even if they do not have matching entries in the left-hand table. Since `RIGHT` outer joins are nonstandard, it is usually best to stick to `LEFT` outer joins for cross-database compatibility.

For some practical examples of outer joins and when they are useful, see "Advanced SQL".

Natural joins are kind of 'automatic', in that they will automatically match up rows based on column names that are found to match between the two tables. Thus, if a table called Jokes has an AID column that refers to entries in an Authors table whose primary key is another AID column, you can perform a join of these two tables on that column very simply (assuming there are no other columns with identical names in the two tables):

```
SELECT * FROM Jokes NATURAL JOIN Authors
```

SET

SET*option* = value, ...

TheSET query allows you to set a number of options both on your client and on the server.

The most common uses of the SET OPTION query are changing your password,

SET PASSWORD = PASSWORD('new_password')
and changing another user's password (if you have appropriate access privileges).

SET PASSWORD FOR *user* = PASSWORD('new_password')
For a complete list of the options that may be SET, refer to the MySQL Reference Manual.

SHOW

TheSHOW query may be used in a number of forms to get information about the MySQL server, the databases, and the tables it contains. Many of these forms have an optional LIKE*wild* component, where*wild* is a string that may contain wild card characters ('%' for multiple characters, '_' for just one) to filter the list of results. Each of the forms of the SHOW query are described below:

- SHOW DATABASES [LIKE *wild*]

  This query lists the databases that are available on the MySQL server.

- SHOW [OPEN] TABLES [FROM db_name] [LIKE *wild*]

  This query lists the tables (or, optionally, the currently OPEN tables) in the default or specified database.

- SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE *wild*]

  WhenFULL is not used, this query provides the same information as a DESCRIBE query (see "DESCRIBE"). The FULL option adds a listing of the privileges you have on each column to this information.SHOW FIELDS is equivalent to SHOW COLUMNS.

- SHOW INDEX FROM tbl_name [FROM db_name]

  This query provides detailed information about the indexes that are defined on the specified table. See theMySQL Reference manual for a guide to the results produced by this query. SHOW KEYS is equivalent to SHOW INDEX.

- SHOW TABLE STATUS [FROM db_name] [LIKE *wild*]

  This query displays detailed information about the tables in the default or specified database.

- SHOW STATUS [LIKE *wild*]

  This query displays detailed statistics for the server. See the MySQL Reference Manual for details on the meaning of each of the figures.

- SHOW VARIABLES [LIKE *wild*]

  This query lists the MySQL configuration variables and their settings. See the MySQL Reference Manual for a complete description of these options.

- SHOW [FULL] PROCESSLIST

  This query displays all threads running on the MySQL server, and the queries being executed by each. If you don't have the 'process' privilege, you will only see threads executing your own queries. The FULL option causes the complete queries to be displayed, rather than only the first 100 characters of each (the default).

- SHOW GRANTS FOR *user*

  This query lists the GRANT queries that would be required to recreate the privileges of the specified user.

- SHOW CREATE TABLE table_name

  This query displays the CREATE TABLE query that would be required to reproduce the specified table.

UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
  SET col_name=expr, ...
  [WHERE where_definition]
  [ORDER BY ...]
  [LIMIT #]
```

TheUPDATE query updates existing table entries by assigning new values to the specified columns. Columns that are not listed are left alone, with the exception of the TIMESTAMP column (see "TIMESTAMP[(M)] MySQL column types TIMESTAMP Description: A timestamp (date/time), in YYYYMMDD..."). The WHERE clause lets you specify a condition (*where_definiton*) that rows must satisfy if they are to be updated, while the LIMIT clause lets you specify a maximum number of rows to be updated.
Important IfWHERE and LIMIT are not specified, then every row in the table will be updated!

TheORDER BY clause lets you specify the order in which entries are updated. This is most useful in combination with the LIMIT clause—together they let you create queries like “update the 10 most recent rows”.

AnUPDATE operation will fail with an error if the new value assigned to a row clashes with an existing value in a PRIMARY KEY or UNIQUE column, unless the IGNORE option is specified, in which case the query will simply have no effect on that particular row.

TheLOW_PRIORITY option instructs MySQL to wait until there are no other clients reading the table before it performs the update.

USE

```
USE db_name
```

This simple query sets the default database for MySQL queries in the current session. Tables in other databases may still be accessed as `db_name.tbl_name`.

Appendix B: MySQL Functions

MySQL provides a sizeable library of functions to format and combine data within SQL queries in order to produce the desired results in the desired format. This appendix provides a reference to the most useful of these functions, as implemented in MySQL as of version 3.23.54a, current as of this writing.

For a complete, up-to-date reference to supported SQL functions, see the MySQL Reference Manual.

Control Flow Functions

`IFNULL(expr1,expr2)`	This function returns `expr1` unless it is `NULL`, in which case it returns `expr2`.
`NULLIF(expr1,expr2)`	This function returns `expr1` unless it equals `expr2`, in which case it returns `NULL`.
`IF(expr1,expr2,expr3)`	If `expr1` is `TRUE` (that is, not `NULL` or `0`), returns `expr2`; otherwise, returns `expr3`.
`CASE value WHEN [compare-value1] THEN result1 [WHEN ...] [ELSE else-result] END`	This function returns `result1` when `value=compare-value1` (note that several compare-value/result pairs can be defined); otherwise, returns `else-result`, or `NULL` if none is defined.
`CASE WHEN [condition1] THEN result1 [WHEN ...] [ELSE else-result] END`	This function returns `result1` when `condition` is `TRUE` (note that several condition/result pairs can be defined); otherwise, returns `else-result`, or `NULL` if none is defined.

Mathematical Functions

`ABS(`*`expr`*`)`

This function returns the absolute (positive) value of `expr`.

`SIGN(`*`expr`*`)`

This function returns -1, 0, or 1 depending on whether `expr` is negative, zero, or positive, respectively.

`MOD(`*`expr1`*`,`*`expr2`*`)`,`expr1 % expr2`

This function returns the remainder of dividing `expr1` by `expr2`.

`FLOOR(`*`expr`*`)`

This function rounds down `expr` (i.e. returns the largest integer value that is less than or equal to `expr`).

`CEILING(`*`expr`*`)`

This function rounds up `expr` (i.e. returns the smallest integer value that is greater than or equal to `expr`).

`ROUND(`*`expr`*`)`

This function returns `expr` rounded to the nearest integer. Note that the behaviour when the value is exactly an integer plus 0.5 is system dependant. Thus, you should not rely on any particular outcome when migrating to a new system.

`ROUND(`*`expr`*`,`*`num`*`)`

This function rounds `expr` to a number with `num` decimal places, leaving trailing zeroes in place. Use `num=2`, for example, to format a number as dollars and cents. Note that the same uncertainty about the rounding of 0.5 applies as discussed for `ROUND()` above.

`EXP(`*`expr`*`)`

This function returns e^{expr}, the base of natural logarithms raised to the power of `expr`.

`LOG(`*`expr`*`)`

This function returns ln(`expr`), or $\log_e$(`expr`), the natural logarithm of `expr`.

Remember, a logarithm with an arbitrary base B can be calculated as LOG(`expr`)/LOG(B).

`LOG10(`*`expr`*`)`

This function returns the base-10 logarithm of `expr`.

`POW(`*`expr1`*`,`*`expr2`*`)`,`POWER(`*`expr1`*`,`*`expr2`*`)`

This function returns `expr1` raised to the power of `expr2`.

`SQRT(`*`expr`*`)`

This function returns the square root of `expr`.

`PI()`

This function returns the value of p (pi).

COS(*expr*)	This function returns the cosine of expr in radians (e.g. COS(PI()) = -1).
SIN(*expr*)	This function returns the sine of expr in radians (e.g. SIN(PI()) = 0).
TAN(*expr*)	This function returns the tangent of expr in radians (e.g. TAN(PI()) = 0).
ACOS(*expr*)	This function returns the arc cosine ($\cos^{-1}$ or inverse cosine) of expr (e.g. ACOS(-1) = 3.141593).
ASIN(*expr*)	This function returns the arc sine ($\sin^{-1}$ or inverse sine) ofexpr (e.g. ASIN(0) = 3.141593).
ATAN(*expr*)	This function returns the arc tangent ($\tan^{-1}$ or inverse tangent) of expr (e.g. ATAN(0) = 3.141593).
ATAN(*y,x*),ATAN2(*y,x*)	This function returns the angle (in radians) made at the origin between the positive x axis and the point (x,y) (e.g.ATAN(1,0) = 1.570796).
COT(*expr*)	This function returns the cotangent of expr (e.g. COT(PI()/2) = 0).
RAND(),RAND(*expr*)	This function returns a random, floating point number between 0.0 and 1.0. If expr is specified, a random number will be generated based on that value, which will always be the same.
LEAST(*expr1,expr2,...*)	This function returns the smallest of the values specified.
GREATEST(*expr1,expr2,...*)	This function returns the largest of the values specified.
DEGREES(*expr*)	This function returns the value of expr (in radians) in degrees.
RADIANS(*expr*)	This function returns the value of expr (in degrees) in radians.
TRUNCATE(*expr,num*)	This function returns the value of floating point number expr truncated to num decimal places (i.e. rounded down).

String Functions

`ASCII(str)`

This function returns the ASCII code value of the le
character in `str`, 0 if `str` is an empty string, or `NUL`
is`NULL`.

`ORD(str)`

This function returns the ASCII code of the left-mos
character in `str`, taking into account the possibility
might be a multi-byte character.

`CONV(expr,from_base,to_base)`

This function converts a number (`expr`) in base `fr`
to a number in base `to_base`. Returns `NULL` if any
arguments are `NULL`.

`BIN(expr)`

This function converts decimal `expr` to binary, equi
`CONV(expr,10,2)`.

`OCT(expr)`

This function converts decimal `expr` to octal, equiv
`CONV(expr,10,8)`.

`HEX(expr)`

This function converts decimal `expr` to hexadecima
equivalent to `CONV(expr,10,16)`.

`CHAR(expr,...)`

This function creates a string composed of characte
ASCII code values of which are given by the expres
passed as arguments.

`CONCAT(str1,str2,...)`

This function returns a string made up of the strings
as arguments joined end-to-end. If any of the argun
`NULL`,`NULL` is returned instead.

`CONCAT_WS(separator,str1,str2,...)`

`CONCAT` "With Separator" (`WS`). Same as `CONCAT`,
the first argument is placed between each of the ac
arguments when they are combined.

`LENGTH(str)`,`OCTET_LENGTH(str)`,
`CHAR_LENGTH(str)`,`CHARACTER_LENGTH(str)`

All of these return the length in characters of `str`.
`CHAR_LENGTH` and `CHARACTER_LENGTH`, howeve
multi-byte characters into consideration when perfo
count.

`BIT_LENGTH(str)`

This function returns the length (in bits) of `str` (i.e.
`BIT_LENGTH(str) = 8 * LENGTH(str)`).

`LOCATE(substr,str)`,`POSITION(substr IN str)`

This function returns the position of the first occurre
`substr` in `str` (1 if it occurs at the beginning, 2 if it
after one character, and so on). Returns 0 if `subst`
not occur in `str`.

`LOCATE(substr,str,pos)`	Same as `LOCATE(substr,str)`, but begins sear from character number `pos`.
`INSTR(str,substr)`	This function is the same as `LOCATE(substr,str` with argument order swapped.
`LPAD(str,len,padstr)`	This function shortens or lengthens `str` so that it is `len`. Lengthening is accomplished by inserting `pad` the left of the characters of `str` (e.g. `LPAD('!','5','.') = '....!'`).
`RPAD(str,len,padstr)`	This function shortens or lengthens `str` so that it is `len`. Lengthening is accomplished by inserting `pad` the right of the characters of `str` (e.g. `RPAD('!','5','.') = '!....'`).
`LEFT(str,len)`	This function returns the left-most `len` characters o `str` is shorter than `len` characters, `str` is returned extra padding.
`RIGHT(str,len)`	This function returns the right-most `len` characters `str` is shorter than `len` characters, `str` is returned extra padding.
`SUBSTRING(str,pos,len)`, `SUBSTRING(str FROM pos FOR len)`, `MID(str,pos,len)`	This function returns a string up to `len` characters l taken from `str` beginning at position `pos` (where 1 first character). The second form of `SUBSTRING` is standard.
`SUBSTRING(str,pos)`,`SUBSTRING(str FROM pos)`	This function returns the string beginning from posit `instr` (where 1 is the first character) and going to th `str`.
`SUBSTRING_INDEX(str,delim,count)`	MySQL counts `count` occurrences of `delim` in `st` takes the substring from that point. If `count` is posit MySQL counts to the right from the start of the strin then takes the substring up to but not including that If`count` is negative, MySQL counts to the left from of the string, and then takes the substring that starts after that delimiter, and runs to the end of `str`.
`LTRIM(str)`	This function returns `str` with any leading white spa trimmed off.
`RTRIM(str)`	This function returns `str` with any trailing white spa trimmed off.

`TRIM([[BOTH | LEADING | TRAILING] str) [remstr] FROM]`

This function returns `str` with either white space (b or occurrences of the string `remstr` removed from of the string (`LEADING`), end of the string (`TRAILI` both (`BOTH`, the default).

`SOUNDEX(str)`

This function produces a string that represents how sounds when read aloud. Words that sound similar have the same 'soundex string'.

E.g.:

```
SOUNDEX("tire") = "T600"
SOUNDEX("tyre") = "T600"
SOUNDEX("terror") = "T600"
SOUNDEX("tyrannosaur") = "T6526"
```

`SPACE(num)`

This function returns a string of `num` space characte

`REPLACE(str,from_str,to_str)`

This function returns `str` after replacing all occurre `from_str` to `to_str`.

`REPEAT(str,count)`

This function returns a string made up of `str` repea `count` times, an empty string if count <= 0, or `NULL` argument is `NULL`.

`REVERSE(str)`

This function returns `str` spelled backwards.

`INSERT(str,pos,len,newstr)`

This function takes `str`, and removes the substring beginning at `pos` (where 1 is the first character in th with length `len`, then inserts `newstr` at that positio `= 0`, simply inserts `newstr` at position `pos`.

`ELT(N,str1,str2,str3,...)`

This function returns the `N`th string argument (`str1` `str2` if `N=2` and so on), or `NULL` if there is no argu the given `N`.

`FIELD(str,str1,str2,str3,...)`

This function returns the position of `str` in the subs list of arguments (1 if `str = str1`, 2 if `str = st` so on).

`FIND_IN_SET(str,strlist)`

When `strlist` is a list of strings of the form `'string1,string2,string3,...'` this functio the index of `str` in that list, or 0 if `str` is not in the l function is ideally suited (and optimized) for determ `str` is selected in a column of type `SET` (see "MyS Column Types").

`MAKE_SET(`*`bits`*`,`*`str1`*`,`*`str2`*`,...)`

This function returns a list of strings of the form `'string1,string2,string3,...'` using the s parameters (`str1`,`str2`, etc.) that correspond to th that are set in the number bits. For example, if `bits` (binary 1010) then bits 2 and 4 are set, so the outpu `MAKE_SET` will be `'str2,str4'`.

`EXPORT_SET(`*`bits`*`,`*`on_str`*`,`
`off_str``[,`*`separator`*`[,`*`number_of_bits`*`]])`

This function returns a string representation of whic are, and are not set in `bits`. Set bits are represent `on_str` string, while unset bits are represented by `off_str` string. By default, these bit representation comma-separated, but the optional `separator` str you define your own. By default, up to 64 bits of bits read; however, `number_of_bits` lets you specify smaller number be read.

E.g.:

```
EXPORT_SET(10,'Y','N',',',6) = 'N,Y,N
```

`LCASE(`*`str`*`),LOWER(`*`str`*`)`

This function returns `str` with all letters in lowercas

`UCASE(`*`str`*`),UPPER(`*`str`*`)`

This function returns `str` with all letters in upperca

`LOAD_FILE(`*`filename`*`)`

This function returns the contents of the file specifie `filename` (an absolute path to a file readable by M Your MySQL user should also have file privileges.

Date and Time Functions

`DAYOFWEEK(`*`date`*`)`	This function returns the weekday of `date` in the form of an integer, according to the ODBC standard (1 = Sunday, 2 = Monday, 3 = Tuesday, ... 7 = Saturday).
`WEEKDAY(`*`date`*`)`	This function returns the weekday of `date` in the form of an integer (0 = Monday, 1 = Tuesday, 2 = Wednesday,... 6 = Sunday).
`DAYOFMONTH(`*`date`*`)`	This function returns the day of the month for `date` (from 1 to 31).
`DAYOFYEAR(`*`date`*`)`	This function returns the day of the year for `date` (from 1 to 366-remember leap years!).
`MONTH(`*`date`*`)`	This function returns the month for `date` (from 1, January, to 12, December).
`DAYNAME(`*`date`*`)`	Returns the name of the day of the week for `date` (e.g. 'Tuesday').
`MONTHNAME(`*`date`*`)`	This function returns the name of the month for `date` (e.g. 'April').
`QUARTER(`*`date`*`)`	This function returns the quarter of the year for `date` (e.g.:`QUARTER('2001-04-12') = 2`).
`WEEK(`*`date`*`,`*`first`*`),WEEK(`*`date`*`)`	This function returns the week of the year for `date` (from 1 to 53), assuming by default that the first day of the week is Sunday (if first is not specified or 0), or Monday (if first is 1).
`YEAR(`*`date`*`)`	Returns the year for `date` (from 1000 to 9999).
`YEARWEEK(`*`date`*`),YEARWEEK(`*`date`*`,`*`first`*`)`	This function returns the year and week for `date` in the form`YYYYWW`. Note that the first or last day or two of the year may often belong to a week of the year before or after, respectively. E.g.: `YEARWEEK("2001-12-31") = 200201`
`HOUR(`*`time`*`)`	This function returns the hour for `time` (from 0 to 23).
`MINUTE(`*`time`*`)`	This function returns the minute for `time` (from 0 to 59).

`SECOND(time)`

This function returns the second for `time` (from 0 to 59).

`PERIOD_ADD(period,num_months)`

This function adds `num_months` months to period (specified as `YYMM` or `YYYYMM`) and returns the value in the form `YYYYMM`.

`PERIOD_DIFF(period1,period2)`

This function returns the number of months between `period1` and `period2` (each of which should be specified as `YYMM` or `YYYYMM`).

`DATE_ADD(date, INTERVAL exprtype)`
`,DATE_SUB(date, INTERVAL exprtype),`
`ADDDATE(date, INTERVAL exprtype),`
`SUBDATE(date, INTERVAL exprtype)`

This function returns the result of either adding or subtracting the specified interval of time to or from `date` (a`DATE` or `DATETIME` value). `DATE_ADD` and `ADDDATE` are identical, as are `DATE_SUB` and `SUBDATE`.`expr` specifies the interval to be added or subtracted and may be negative if you wish to specify a negative interval, and`type` specifies the format of `expr`, as shown in "Interval types for date addition/subtraction functions".

If`date` and `expr` involve only date values, the result will be a `DATE` value; otherwise, this function will return a `DATETIME` value.

Interval types for date addition/subtraction functions

type	Format for expr
SECOND	number of seconds
MINUTE	number of minutes
HOUR	number of hours
DAY	number of days
MONTH	number of months
YEAR	number of years
MINUTE_SECOND	'*minutes*:*seconds*'
HOUR_MINUTE	'*hours*:*minutes*'
DAY_HOUR	'*dayshours*'
YEAR_MONTH	'*years-months*'
HOUR_SECOND	'*hours*:*minutes*:*seconds*'
DAY_MINUTE	'*dayshours*:*minutes*'
DAY_SECOND	'*dayshours*:*minutes*:*seconds*'

Here are a few examples to help you see how this family of functions works:

The following both return the date six months from now:

```
ADDDATE(CURDATE(), INTERVAL 6 MONTH)
DATE_ADD(CURDATE(), INTERVAL '0-6' YEAR_MONTH)
```

The following all return this time tomorrow:

```
ADDDATE(NOW(), INTERVAL 1 DAY)
SUBDATE(NOW(), INTERVAL -1 DAY)
DATE_ADD(NOW(), INTERVAL '24:0:0' HOUR_SECOND)
```

```
DATE_ADD(NOW(), INTERVAL '1 0:0' DAY_MINUTE)
```

`TO_DAYS(date)`

This function converts date to a number of days since year 0. Allows you to calculate differences in dates (i.e. `TO_DAYS(date1) - TO_DAYS(date2) = days_in_between`).

`FROM_DAYS(days)`

Given the number of `days` since year 0 (as produced by `TO_DAYS`), this function returns a date.

`DATE_FORMAT(date,format)`

This function takes the date or time value date and returns it formatted according to the formatting string `format`, which may contain any of the symbols shown in "DATE_FORMAT symbols (2004-01-01 01:00:00)" as place-holders.

DATE_FORMAT symbols (2004-01-01 01:00:00)

Symbol	Displays	Example
%M	Month name	January
%W	Weekday name	Thursday
%D	Day of the month with English suffix	1st
%Y	Year, numeric, 4 digits	2004
%y	Year, numeric, 2 digits	03
%a	Abbreviated weekday name	Thu
%d	Day of the month	01
%e	Day of the month	1
%m	Month of the year, numeric	01
%c	Month of the year, numeric	1
%b	Abbreviated month name	Jan
%j	Day of the year	001
%H	Hour of the day (24 hour format, 00-23)	01
%k	Hour of the day (24 hour format, 0-23)	1
%h	Hour of the day (12 hour format, 01-12)	01
%I	Hour of the day (12 hour format, 01-12)	01
%l	Hour of the day (12 hour format, 1-12)	1
%i	Minutes	00
%r	Time, 12 hour (hh:mm:ss AM/PM)	01:00:00 AM
%T	Time, 24 hour (hh:mm:ss)	01:00:00
%S	Seconds	00
%s	Seconds	00
%p	AM or PM	AM
%w	Day of the week, numeric (0=Sunday)	4
%U	Week (00-53), Sunday 1st day of the week	00
%u	Week (00-53), Monday 1st day of the week	01
%X	Year of the week where Sunday is the 1st day of the week, 4 digits (use with %V)	2003

%V	Week (01-53), Sunday 1st day of week (%X)	53
%x	Like%X, Monday 1 st day of week (use with%v)	2004
%v	Week (01-53), Monday 1st day of week (%x)	01
%%	An actual percent sign	%

`TIME_FORMAT(`*`time`*`,`*`format`*`)`

This function is the same as `DATE_FORMAT`, except the`format` string may only contain symbols referring to hours, minutes, and seconds.

`CURDATE(),CURRENT_DATE`

This function returns the current system date in the SQL date format `'YYYY-MM-DD'` (if used as a date) or as `YYYYMMDD` (if used as a number).

`CURTIME(),CURRENT_TIME`

This function returns the current system time in the SQL time format `'HH:MM:SS'` (if used as a time) or as`HHMMSS` (if used as a number).

`NOW(),SYSDATE(),CURRENT_TIMESTAMP`

This function returns the current system date and time in SQL date/time format `'YYYY-MM-DD HH:MM:SS'` (if used as a date/time) or as YYYYMMDDHHMMSS (if used as a number).

`UNIX_TIMESTAMP(),UNIX_TIMESTAMP(`*`date`*`)`

This function returns either the current system date and time, or the specified date/time as the number of seconds since 1970-01-01 00:00:00 GMT.

`FROM_UNIXTIME(`*`unix_timestamp`*`)`

The opposite of `UNIX_TIMESTAMP`, this function converts a number of seconds from 1970-01-01 00:00:00 GMT to "YYYY-MM-DD HH:MM:SS" (if used as a date/time) or `YYYYMMDDHHMMSS` (if used as a number), local time.

`FROM_UNIXTIME(`*`unix_timestamp`*`,`*`format`*`)`

This function formats a UNIX timestamp according to the`format` string, which may contain any of the symbols listed in "DATE_FORMAT symbols (2004-01-01 01:00:00)".

`SEC_TO_TIME(`*`seconds`*`)`

This function converts some number of `seconds` to the format `'HH:MM:SS'` (if used as a time) or `HHMMSS` (if used as a number).

`TIME_TO_SEC(`*`time`*`)`

This function converts a `time` in the format `'HH:MM:SS'` to a number of seconds.

Miscellaneous Functions

`DATABASE()`

This function returns the currently selected database name or database is currently selected.

`USER()`,`SYSTEM_USER()`,`SESSION_USER()`

This function returns the current MySQL user name, including (e.g.`'kevin@localhost'`). The `SUBSTRING_INDEX` functic obtain the user name alone:

```
SUBSTRING_INDEX(USER(),"@",1) = 'kevin'
```

`PASSWORD(`*`str`*`)`

A one-way password encryption function, which converts any s plaintext password) into an encrypted format precisely 16 char particular plaintext string will always yield the same encrypted thus, values encoded in this way can be used to verify the corr without actually storing the password in the database.

This function does not use the same encryption mechanism as `ENCRYPT` for that type of encryption.

`ENCRYPT(`*`str`*`[,`*`salt`*`])`

This function uses standard UNIX encryption (via the `crypt()` encrypt`str`. The `salt` argument is optional, and lets you cont used for the generation of the password. If you want the encryp password file entry, the salt should be the two first characters you are trying to match. Depending on the implementation of c system, the encrypted value may only depend on the first 8 ch value.

On systems where `crypt()` is not available, this function retu

`ENCODE(`*`str`*`,`*`pass_str`*`)`

This function encrypts `str` using a two-way password-based e with password `pass_str`. To subsequently decrypt the value,

`DECODE(`*`crypt_str`*`,`*`pass_str`*`)`

This function decrypts the encrypted `crypt_str` using two-wa encryption, with password `pass_str`. If the same password is provided to `ENCODE` the value originally, the original string will

`MD5(`*`string`*`)`

This function calculates an MD5 hash based on string. The res digit hexadecimal number. A particular string will always produ hash; however, `MD5(NOW())` may be used, for instance, to ob string when one is needed (as a default password, for instance

`LAST_INSERT_ID()`

This function returns the last number that was automatically ge `AUTO_INSERT` column in the current connection.

`FORMAT(`*`expr`*`,`*`num`*`)`

This function formats a number `expr` with commas as "thousa `num` decimal places (rounded to the nearest value, and padde

`VERSION()`

This function returns the MySQL server version (e.g. `'3.23.5`

`CONNECTION_ID()`

This function returns the thread ID for the current connection.

`GET_LOCK(`*`str`*`,`*`timeout`*`)`

If two or more clients must synchronize tasks beyond what tab named locks may be used instead. `GET_LOCK` attempts to obt name (`str`). If the named lock is already in use by another clie up to `timeout` seconds before giving up waiting for the lock to

Once a client has obtained a lock, it can be released either usi by using `GET_LOCK` again to obtain a new lock.

`GET_LOCK` returns 1 if the lock was successfully retrieved, 0 if `timeout` elapsed, or `NULL` if some error occurred.

`GET_LOCK` is not a MySQL command in and of itself-it must ap another query.

E.g.:

```
SELECT GET_LOCK("mylock",10)
```

`RELEASE_LOCK(`*`str`*`)`

This function releases the named lock that was obtained by GE the lock was released, 0 if the lock wasn't locked by this thread doesn't exist.

`BENCHMARK(`*`count`*`,`*`expr`*`)`

This function repeatedly evaluates `exprcount` times, for the p testing. The MySQL command line client allows the operation

`INET_NTOA(`*`expr`*`)`

This function returns the IP address represented by the integer `INET_ATON` to create such integers.

`INET_ATON(`*`expr`*`)`

This function converts an IP address `expr` to a single integer r

E.g.:

```
INET_ATON('64.39.28.1') = 64 * 2553 + 39 * 25
                        = 1063751116
```

Functions for Use with GROUP BY Clauses

Also known as *summary functions*, the following are intended for use with GROUP BY clauses, where they will produce values based on the set of records making up each row of the final result set.

If used without a GROUP BY clause, these functions will cause the result set to be displayed as a single row, with a value calculated based on all of the rows of the complete result set. Without a GROUP BY clause, mixing these functions with columns that do not contain summary functions will cause an error, because you cannot collapse those columns into a single row and get a sensible value.

COUNT(*expr*)	This function returns a count of the number of times in the ungrouped result set that expr had a non-NULL value. If COUNT(*) is used, it will simply provide a count of the number of rows in the group, irrespective of NULL values.
COUNT(DISTINCT*expr*[,*expr...*])	This function returns a count of the number of different non-NULL values (or sets of values, if multiple expressions are provided).
AVG(*expr*)	This function calculates the arithmetic mean (average) of the values appearing in the rows of the group.
MIN(*expr*),MAX(*expr*)	This function returns the smallest or largest value of expr in the rows of the group.
SUM(*expr*)	This function returns the sum of the values for expr in the rows of the group.
STD(*expr*),STDDEV(*expr*)	This function returns the standard deviation of the values for expr in the rows of the group (either of the two function names may be used).
BIT_OR(*expr*),BIT_AND(*expr*)	This function calculates the bit-wise OR and the bit-wise AND of the values for expr in the rows of the group, respectively.

Appendix C: MySQL Column Types

Overview

When you create a table in MySQL, you must specify the data type for each column. This appendix documents all of the column types that MySQL provides as of version 3.23.54a, current as of this writing.

In this reference, many column types can accept *optional parameters* to further customize how data for the column is stored or displayed. First, there are the `M` and `D` parameters, which are indicated (in square brackets when optional) immediately following the column type name.

The parameter `M` is used to specify the display size (i.e. maximum number of characters) to be used by values in the column. In most cases, this will limit the range of values that may be specified in the column. `M` may be any integer between 1 and 255. Note that for numerical types (e.g. `INT`), this parameter does not actually restrict the range of values that may be stored. Instead, it causes spaces (or zeroes in the case of a `ZEROFILL` column—see below for details) to be added to the values so that they reach the desired display width when they're displayed. Note also that the storage of values longer than the specified display width can cause problems when the values are used in complex joins, and thus should be avoided whenever possible.

The parameter `D` lets you specify how many decimal places will be stored for a floating-point value. This parameter may be set to a maximum of `30`, but `M` should always allow for these places (i.e. `D` should always be less than or equal to `M-2` to allow room for a zero and a decimal point).

The second type of parameter is an optional *column attribute*. The attributes supported by the different column types are listed for each, and are enabled by simply typing them after the column type, separated by spaces. Here are the available column attributes, and their meanings:

- `ZEROFILL`

  Values for the column always occupy their maximum display length, as the actual value is padded with zeroes. The option automatically sets the `UNSIGNED` option as well.

- `UNSIGNED`

  The column may accept only positive numerical values (or zero). This restriction frees up more storage space for positive numbers, effectively doubling the range of positive values that may be stored in the column, and should always be set if you know that you won't need to store negative values.

- `BINARY`

  By default, comparisons of character values in MySQL (including sorting) are case-insensitive. However, comparisons for `BINARY` columns are case-sensitive.

For a complete, up-to-date reference to supported SQL column types, see the MySQL Reference Manual.

Numerical Types

`TINYINT[(M)]`

Description: A tiny integer value.

Attributes allowed: `UNSIGNED`,`ZEROFILL`

Range: -128 to 127 (0 to 255 if `UNSIGNED`)

Storage space: 1 byte (8 bits)

`SMALLINT[(M)]`

Description: A small integer value.

Attributes allowed: `UNSIGNED`,`ZEROFILL`

Range: -32768 to 32767 (0 to 65535 if `UNSIGNED`)

Storage space: 2 bytes (16 bits)

`MEDIUMINT[(M)]`

Description: A medium integer value.

Attributes allowed: `UNSIGNED`,`ZEROFILL`

Range: -8588608 to 8388607 (0 to 16777215 if `UNSIGNED`)

Storage space: 3 bytes (24 bits)

`INT[(M)]`

Description: A regular integer value.

Attributes allowed: `UNSIGNED`,`ZEROFILL`

Range: -2147483648 to 2147483647 (0 to 4294967295 if `UNSIGNED`)

Storage space: 4 bytes (32 bits)

Alternative syntax: `INTEGER[(M)]`

`BIGINT[(M)]`

Description: A large integer value.

Attributes allowed: `UNSIGNED`,`ZEROFILL`

Range: -9223372036854775808 to 9223372036854775807 (0 to 18446744073709551615 if `UNSIGNED`)

Storage space: 8 bytes (64 bits)

Notes: MySQL performs all integer arithmetic functions in signed `BIGINT` format; thus, `BIGINT UNSIGNED` values over 9223372036854775807 (63 bits) will only work properly with bit functions (e.g. bit-wise AND, OR, and NOT). Attempting integer arithmetic with larger values may produce inaccurate results due to rounding errors.

`FLOAT[(M,D)]`,`FLOAT(precision)`

Description: A floating point number.

Attributes allowed: `ZEROFILL`

Range: 0 and ±1.175494351E-38 to ±3.402823466E+38

Storage space: 4 bytes (32 bits)

Notes: `precision` (in bits), if specified, must be less than or equal to 24, or else a `DOUBLE` column will be created instead (see below).

`DOUBLE[(M,D)]`,`DOUBLE(precision)`

Description: A high-precision floating point number.

Attributes allowed: `ZEROFILL`

Range: 0 and ±2.2250738585072014-308 to ±1.7976931348623157E+308

Storage space: 8 bytes (64 bits)

Notes: `precision` (in bits), if specified, must be greater than or equal to 25, or else a `FLOAT` column will be created instead (see above). `precision` may not be greater than 53.

Alternative syntax: `DOUBLE PRECISION[(M,D)]` or `REAL([M,D])`

`DECIMAL[(M[,D])]`

Description: A floating point number stored as a character string.

Attributes allowed: `ZEROFILL`

Range: As for `DOUBLE`, but constrained by `M` and `D` (see Notes).

Storage space: `M+2` bytes (`8M+16` bits) (see Notes prior to MySQL 3.23)

Notes: If `D` is not specified, it defaults to 0 and numbers in this column will have no decimal point or fractional part. If `M` is not specified, it defaults to 10. In versions of MySQL prior to 3.23, `M` had to include space for the negative sign and the decimal point, so the storage space required was `M` bytes (`8M` bits). The newer format in MySQL 3.23 or later is ANSI SQL compliant.

Alternative syntax: `NUMERIC([M[,D]])`

Character Types

`CHAR(M)`

Description: A fixed-length character string.

Attributes allowed: `BINARY`

Maximum Length: `M` characters

Storage space: `M` bytes (`8M` bits)

Notes: `CHAR` values are stored as strings of length `M`, even though the assigned value may be shorter. When the string does not occupy the full length of the field, spaces are added to the end of the string to bring it to exactly M characters. Trailing spaces are then stripped off when the value is retrieved.

`CHAR` columns are quicker to search than variable-length character column types such as `VARCHAR`, since their fixed-length nature makes the underlying database file format more regular.

`M` may take any integer value from 0 to 255, with a `CHAR(0)` column able to store only two values: `NULL` and `''` (the empty string), and occupying only a single bit.

Alternative syntax: `CHARACTER(M)`

`VARCHAR(M)`

Description: A variable-length character string.

Attributes allowed: `BINARY`

Maximum Length: `M` characters

Storage space: Length of stored value, plus 1 byte to store length.

Notes: As `VARCHAR` values occupy only the space they require, there is usually no point to specifying a maximum field length `M` of anything less than 255 (the maximum). Values anywhere from 1 to 255 are acceptable, however, and will cause strings longer than the specified limit to be chopped to the maximum length when inserted. Trailing spaces are stripped from values before they are stored.

Alternative syntax: `CHARACTER VARYING(M)`

`TINYBLOB`,`TINYTEXT`

Description: A short, variable-length character string.

Maximum Length: 255 characters

Storage space: Length of stored value, plus 1 byte to store length.

Notes: These types are basically equivalent to `VARCHAR(255) BINARY` and `VARCHAR(255)`, respectively. However, these column types do not trim trailing spaces from inserted values. The only difference between `TINYBLOB` and `TINYTEXT` is that the former performs case-sensitive comparisons and sorts while the latter does not.

`BLOB,TEXT`

Description: A variable-length character string.

Maximum Length: 65535 characters (65KB)

Storage space: Length of stored value, plus 2 bytes to store length.

Notes: The only difference between `BLOB` and `TEXT` is that the former performs case-sensitive comparisons and sorts, while the latter does not.

`MEDIUMBLOB,MEDIUMTEXT`

Description: A medium, variable-length character string.

Maximum Length: 16777215 characters (16.8MB)

Storage space: Length of stored value, plus 3 bytes to store length.

Notes: The only difference between `MEDIUMBLOB` and `MEDIUMTEXT` is that the former performs case-sensitive comparisons and sorts, while the latter does not.

`LONGBLOB,LONGTEXT`

Description: A long, variable-length character string.

Maximum Length: 4294967295 characters (4.3GB)

Storage space: Length of stored value, plus 4 bytes to store length.

Notes: The only difference between `LONGBLOB` and `LONGTEXT` is that the former performs case-sensitive comparisons and sorts, while the latter does not.

`ENUM(`*`value1,value2,...`*`)`

Description: A set of values from which a single value must be chosen for each row.

Maximum Length: One value chosen from up to 65535 possibilities.

Storage space:

- 1 to 255 values: 1 byte (8 bits)
- 256 to 65535 values: 2 bytes (16 bits)

Notes: Values in this type of field are stored as integers that represent the element selected. 1 represents the first element, 2 the second, and so on. The special value 0 represents the empty string `''`, which is stored if a value that does not appear in column declaration is assigned.

`NOT NULL` columns of this type default to the first value in the column declaration if no particular default is assigned.

`SET(value1,value2,...)`

Description: A set of values, each of which may be set or not set.

Maximum Length: Up to 64 values in a given `SET` column.

Storage space:

- 1 to 8 values: 1 byte (8 bits)
- 9 to 16 values: 2 bytes (16 bits)
- 17 to 24 values: 3 bytes (24 bits)
- 25 to 32 values: 4 bytes (32 bits)
- 33 to 64 values: 8 bytes (64 bits)

Notes: Values in this type of field are stored as integers representing the pattern of bits for set and unset values. For example, if a set contains 8 values, and in a particular row the odd values are set, then the binary representation 01010101 becomes the decimal value 85. Values may therefore be assigned either as integers, or as a string of set values, separated by commas (e.g. `'value1,value3,value5,value7' = 85`). Searches should be performed either with the `LIKE` operator, or the `FIND_IN_SET` function.

Date/Time Types

`DATE`

Description: A date.

Range: `'1000-01-01'` to `'9999-12-31'`, and `'0000-00-00'`

Storage space: 3 bytes (24 bits)

`TIME`

Description: A time.

Range: `'-838:59:59'` to `'838:59:59'`

Storage space: 3 bytes (24 bits)

`DATETIME`

Description: A date and time.

Range: `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`

Storage space: 8 bytes (64 bits)

`YEAR`

Description: A year.

Range: 1901 to 2155, and 0000

Storage space: 1 byte (8 bits)

Notes: You can specify a year value with a four-digit number (`1901` to `2155`, or `0000`), a 4-digit string (`'1901'` to `'2155'`, or `'0000'`), a two-digit number (`70` to `99` for 1970 to 1999, `1` to `69` for 2001 to 2069, or `0` for 0000), or a two-digit string (`'70'` to `'99'` for 1970 to 1999, `'00'` to `'69'` for 2000 to 2069). Note that you cannot specify the year 2000 with a two-digit number, and you can't specify the year 0000 with a two-digit string. Invalid year values are always converted to 0000.

`TIMESTAMP[(`*M*`)]`

Description: A timestamp (date/time), in `YYYYMMDDHHMMSS` format.

Range: `19700101000000` to sometime in 2037 on current systems.

Storage space: 4 bytes (32 bits)

Notes: An `INSERT` or `UPDATE` operation on a row that contains one or more `TIMESTAMP` columns will automatically update the first `TIMESTAMP` column in the row with the current date/time. This lets you use such a column as the 'last modified date/time' for the row. Assigning a value of `NULL` to the column will have the same effect, thereby providing a means of 'touching' the date/time. You can also assign actual values as you would for any other column.

Allowable values for `M` are 14, 12, 10, 8, 6, 4, and 2, and correspond to the display formats `YYYYMMDDHHMMSS`, `YYMMDDHHMMSS`, `YYMMDDHHMM`, `YYYYMMDD`, `YYMMDD`, `YYMM`, and `YY` respectively. Odd values from 1 to 13 will automatically be bumped up to the next even number, while values of 0 or greater than 14 are changed to 14.

Appendix D: PHP Functions for Working with MySQL

Overview

PHP provides a vast library of built-in functions that let you perform all sorts of tasks without having to look to third party software vendors for a solution. The online reference to these functions provided by the PHP Official Website is second to none. Obtaining detailed information about a function is as simple as opening your browser and typing:

`http://www.php.net/`*`functionname`*

As a result of the convenience of this facility, we have judged that a complete PHP function reference is beyond the scope of this book. All the same, this appendix contains a reference to those PHP functions specifically designed to interact with MySQL databases, so that if you use this book as your primary reference while building a database-driven Website, you won't have to look elsewhere for the information you need.

This list of functions and their definitions are current as of PHP 4.3.0.

mysql_affected_rows

```
mysql_affected_rows([link_id])
```

This function returns the number of affected rows in the previous MySQL `INSERT`,`UPDATE`,`DELETE`, or `REPLACE` operation performed with the specified `link_id`. If the link is not specified, then the last-opened link is assumed. It returns `-1` if the previous operation failed.

mysql_close

```
mysql_close([link_id])
```

This function closes the current or specified (`link_id`) MySQL connection. If the link is a persistent link opened by `mysql_pconnect` (see below), this function call is ignored. As non-persistent connections are closed automatically by PHP at the end of a script, this function is usually not needed.

This function returns `true` on success, `false` on failure.

mysql_connect

mysql_connect([*hostname*[:*port*|:*/socket/path*][,*username*[,*password*]]])

This function opens a connection to a MySQL server and returns a connection ID (which evaluates to true) that may be used in other MySQL-related functions. The following default values are assumed if they are not specified:

`hostname:port`
: `'localhost:3306'`

`username`
: server process name

`password`
: `''`

If the connection attempt is unsuccessful, an error message will be displayed by default and the function will return `false`. To bypass display of the error message (e.g. to display your own by checking the return value), put '@' at the start of the function name (i.e. `@mysql_connect(...)`).

mysql_create_db

mysql_create_db(*db_name*[,*link_id*])

This function creates a new MySQL database with the specified name, using the default or specified (link_id) MySQL connection. It returns true on success, or false on error. The function name mysql_createdb may also be used, but is deprecated.

> **Note** mysql_create_db is deprecated. Use mysql_query to issue a CREATE DATABASE command to MySQL instead.

mysql_data_seek

`mysql_data_seek(`*`result_id`*`,`*`row_number`*`)`

This function moves the internal result pointer of the result set identified by `result_id` to row number `row_number`, so that the next call to a `mysql_fetch_*` function will retrieve the specified row. It returns `true` on success, and `false` on failure. The first row in a result set is number 0.

mysql_db_name

mysql_db_name(*result_id*,*row_number*)

result_id should refer to a result set produced by a call to mysql_list_dbs (see below), and will retrieve the name of the database listed on the row specified by row_number. The first row in a result set is row 0. The function name mysql_dbname may also be used, but is deprecated.

mysql_db_query

mysql_db_query(*db_name*,*sql_query*[,*link_id*])

This function selects the MySQL database identified by db_name as if with mysql_select_db, and then executes the specified MySQL query (sql_query). If the MySQL connection identifier (link_id) is not specified, PHP will use the currently active connection. If no such connection exists, PHP will attempt to open a connection by implicitly calling mysql_connect with default parameters.

If the query fails, an error message to that effect will be displayed unless '@' is added to the beginning of the function name, and the function will return false instead of a result identifier (which evaluates to true). If the error occurred due to an error in the SQL query, the error number and message can be obtained using mysq_errno and mysql_error respectively.

The function name mysql may also be used, but is deprecated.

Note As of PHP 4.0.6, this function is deprecated. Use mysql_select_db and then mysql_query, or use only mysql_query and fully specify your table names in your query as *dbname*.*tblname* in your query.

mysql_drop_db

mysql_drop_db(*db_name*[,*link_id*])

This function drops (deletes) the specified database and all the tables it contains, using the default or specified (link_id) MySQL connection. It returns true on success or false on failure.

The function name mysql_dropdb may also be used, but is deprecated.

mysql_errno

```
mysql_errno([link_id])
```

This function returns the numerical value of the error message from the last MySQL operation on the default or specified (`link_id`) MySQL connection.

mysql_error

```
mysql_error([link_id])
```

This function returns the text of the error message from the last MySQL operation on the default or specified (`link_id`) MySQL connection.

mysql_escape_string

`mysql_escape_string(`*`string`*`)`

This function returns an escaped version of a `string` (with backslashes before special characters such as quotes) for use in a MySQL query. This function is a little more thorough than `addslashes` or PHP's Magic Quotes feature, but those methods are generally sufficient (and in the case of Magic Quotes, automatic), so this function is rarely used.

mysql_fetch_array

mysql_fetch_array(*result_id*[,*array_type*])

This function fetches the next row of a MySQL result set, and then advances the internal row pointer of the result set to the next row. It returns the row as an associative array, a numeric array, or both, depending on the value of array_type.

Whenarray_type is not specified, or set to MYSQL_BOTH, each field in the row will be given a numerical index ($row[0]) as well as a string index ($row['col_name']) in the returned array. MYSQL_NUM causes only numerical indices to be assigned, while MYSQL_ASSOC assigns only string indices.

This function returns false if there are no rows left in the specified result set.

mysql_fetch_assoc

`mysql_fetch_assoc(`*`result_id`*`)`

This function fetches a result row as an associative array. It's identical to `mysql_fetch_array` called with the `MYSQL_ASSOC` parameter.

mysql_fetch_field

mysql_fetch_field(*result_id*[,*field_position*])

This function returns an object that contains information about a particular column in the supplied result set (result_id). If the field_position (the first column is position 0) is not specified, then repeated calls to mysql_fetch_field will retrieve each of the columns one at a time, from left to right. Assuming the result of this function is stored in $field, then the properties of the retrieved field are accessible as shown in "Object fields for mysql_fetch_field".

Object fields for mysql_fetch_field

Object property	Information contained
$field->name	Column name
$field->table	Name of table the column belongs to
$field->max_length	Maximum length of the column
$field->not_null	1 if the column is set NOT NULL
$field->primary_key	1 if the column is set PRIMARY KEY
$field->unique_key	1 if the column is set UNIQUE
$field->multiple_key	1 if the column is a non-unique key
$field->numeric	1 if the column is numeric
$field->blob	1 if the column is a BLOB
$field->type	The data type of the column
$field->unsigned	1 if the column is UNSIGNED
$field->zerofill	1 if the column is set ZEROFILL

mysql_fetch_lengths

```
mysql_fetch_lengths(result_id)
```

This function returns an array containing the lengths of each of the fields in the last-fetched row of the specified result set.

mysql_fetch_object

`mysql_fetch_object(`*`result_id`*`)`

This function returns the next result row from `result_id` in the form of an object, and advances the internal row pointer of the result set to the next row. Column values for the row become accessible as named properties of the object (e.g. `$row->user` for the value of the user field in the `$row` object).

mysql_fetch_row

`mysql_fetch_row(`*`result_id`*`)`

This function fetches a result row as numerical array. Identical to `mysql_fetch_array` called with the `MYSQL_NUM` parameter.

mysql_field_flags

mysql_field_flags(*result_id*,*field_position*)

This function returns a string containing the flags associated with the specified field (field_position) in the specified result set (result_id). The flags are separated by spaces in the returned string. Possible flags are: not_null,primary_key,unique_key,multiple_key,blob,unsigned,zerofill, binary,enum,auto_increment, and timestamp.

The function name mysql_fieldflags may also be used, but is deprecated.

mysql_field_len

mysql_field_len(*result_id*,*field_position*)

This function returns the length of the specified field (field_position) in a result set (result_id).

The function name mysql_fieldlen may also be used, but is deprecated.

mysql_field_name

mysql_field_name(*result_id*,*field_position*)

This function returns the name of the specified field (field_position) in a result set (result_id).

The function name mysql_fieldname may also be used, but is deprecated.

mysql_field_seek

mysql_field_seek(*result_id*,*field_position*)

This function sets the default field position for the next call to `mysql_fetch_field`.

mysql_field_table

mysql_field_table(*result_id*,*field_position*)

This function returns the name of the table containing the specified field (field_position) of the specified result set (*result_id*).

The function name mysql_fieldtable may also be used, but is deprecated.

mysql_field_type

mysql_field_type(*result_id*,*field_position*)

This function returns the type of the specified field (field_position) in the specified result set (result_id).

The function name mysql_fieldtype may also be used, but is deprecated.

mysql_free_result

mysql_free_result(*result_id*)

This function destroys the specified result set (result_id), freeing all memory associated with it. As all memory is automatically freed at the end of a PHP script, this function is only really useful when working with multiple very large result sets in a single script.

The function name mysql_freeresult may also be used, but is deprecated.

mysql_get_client_info

```
mysql_get_client_info()
```

This function returns a string indicating the version of the MySQL client library that PHP is using (e.g. `'3.23.54a'`).

mysql_get_host_info

```
mysql_get_host_info([link_id])
```

This function returns a string describing the type of connection and server host name for the specified (`link_id`) or last opened MySQL connection (e.g. `'Localhost via UNIX socket'`).

mysql_get_proto_info

```
mysql_get_proto_info([link_id])
```

This function returns an integer indicating the MySQL protocol version in use for the specified (`link_id`) or last opened MySQL connection (e.g. `10`).

mysql_get_server_info

```
mysql_get_server_info([link_id])
```

This function returns a string indicating the version of MySQL server in use on the specified (`link_id`) or last opened MySQL connection (e.g. `'3.23.54-alpha'`).

mysql_insert_id

```
mysql_insert_id ([link_id])
```

This function returns the value that was automatically assigned to an `AUTO_INCREMENT` column in the previous`INSERT` query for the default or specified (`link_id`) MySQL connection. If no `AUTO_INCREMENT` value was assigned in the previous query, `0` is returned instead.

mysql_list_dbs

mysql_list_dbs([*link_id*])

This function returns a result set containing a list of the databases available from the current or specified (link_id) MySQL connection. Use mysql_db_name to retrieve the individual database names from this result set.

The function name mysql_listdbs may also be used, but is deprecated.

mysql_list_fields

mysql_list_fields(*db_name*,*table_name*[,*link_id*])

This function returns a result set with information about all the fields in the specified table (table_name) in the specified database (db_name) using the default or specified (link_id) MySQL connection. The result set produced may be used with mysql_field_flags,mysql_field_len, mysql_field_name, and mysql_field_type.

The function name mysql_listfields may also be used, but is deprecated.

mysql_list_tables

mysql_list_tables(*db_name*[,*link_id*])

This function returns a result set containing a list of the tables in the specified database (db_name) from the current or specified (link_id) MySQL connection. Use mysql_tablename to retrieve the individual table names from this result set.

The function name mysql_listtables may also be used, but is deprecated.

mysql_num_fields

`mysql_num_fields(`*`result_id`*`)`

This function returns the number of fields in a MySQL result set (`result_id`).

The function name `mysql_numfields` may also be used, but is deprecated.

mysql_num_rows

`mysql_num_rows(`*`result_id`*`)`

This function returns the number of rows in a MySQL result set (`result_id`). This method is not compatible with result sets created by `mysql_unbuffered_query`.

mysql_pconnect

mysql_pconnect([*hostname*[:*port*|:*/socket/path*][,*username*[,*password*]]])

This function opens a persistent connection to a MySQL Server. Works the same as `mysql_connect`, except that the connection is not closed by `mysql_close` or at the end of the script. If a persistent connection is already found to exist with the specified parameters, then this is used, avoiding the creation of a new one.

mysql_query

mysql_query(*sql_query*[,*link_id*])

This function executes the specified MySQL query (sql_query) on the currently selected database.

If the MySQL connection identifier (link_id) is not specified, PHP will use the currently active connection. If no such connection exists, PHP will attempt to open a connection by implicitly calling mysql_connect with default parameters.

If the query fails, an error message to that effect will be displayed unless '@' is added to the beginning of the function name, and the function will return false instead of a result identifier (which evaluates to true). If the error occurred due to an error in the SQL query, the error number and message can be obtained using mysq_errno and mysql_error respectively.

mysql_result

mysql_result(*result_id*,*row*[,*field*])

This function returns the value of a particular field of the specified row (`row`) of the specified result set (`result_id`). The *`field`* argument may be the name of the field (either `fieldname` or `dbname.fieldname`), or its numerical position, where the first field in a row is at position 0. If field is not specified, then `0` is assumed.

mysql_select_db

```
mysql_select_db(db_name[,link_id])
```

This function selects the default database (`db_name`) for the current or specified (`link_id`) MySQL connection.

The function name `mysql_selectdb` may also be used, but is deprecated.

mysql_tablename

mysql_tablename(*result_id*,*row_number*)

result_id should refer to a result set produced by a call to mysql_list_tables, and will retrieve the name of the table listed on the row specified by row_number. The first row in a result set is row 0.

mysql_unbuffered_query

mysql_unbuffered_query(*query*[,*link_id*[,*result_mode*]])

This function sends an SQL query to MySQL, without fetching or buffering the result rows automatically, as mysql_query and mysql_db_query do. This method has two advantages: PHP does not need to allocate a large memory buffer to store the entire result set, and you can begin to process the results as soon as PHP receives the first row, instead of having to wait for the full result set to be received.

The down side is that functions that require information about the full result set (such as mysql_num_rows) are not available for result sets produced by mysql_unbuffered_query, and you must use mysql_fetch_* functions to retrieve all of the rows in the result set before you can send another query using that MySQL connection.

Index

Symbols

Index

A

Index

B-C

Index

D

Index

E

Index

F

Index

G-H

Index

I

Index

J-L

Index

M

CURRENT_TIMESTAMP,Date and Time Functions
CURTIME,Date and Time Functions
DATE_ADD,Date and Time Functions
DATE_FORMAT,Date and Time Functions
DATE_SUB,Date and Time Functions
DAYNAME,Date and Time Functions
DAYOFMONTH,Date and Time Functions
DAYOFWEEK,Date and Time Functions
DAYOFYEAR,Date and Time Functions
DECODE,Miscellaneous Functions
DEGREES,Mathematical Functions
ELT,String Functions
ENCODE,Miscellaneous Functions
ENCRYPT,Miscellaneous Functions
EXP,Mathematical Functions
EXPORT_SET,String Functions
FIELD,String Functions
FIND_IN_SET,String Functions
FLOOR,Mathematical Functions
FORMAT,Miscellaneous Functions
FROM_DAYS,Date and Time Functions
FROM_UNIXTIME,Date and Time Functions
GET_LOCK,Miscellaneous Functions
GREATEST,Mathematical Functions
HEX,String Functions
HOUR,Date and Time Functions
IF,Control Flow Functions
IFNULL,Control Flow Functions
INET_ATON,Miscellaneous Functions
INET_NTOA,Miscellaneous Functions
INSERT,String Functions
INSTR,String Functions
LCASE,String Functions
LEAST,Mathematical Functions
LEFT,Viewing Stored Data,String Functions
LENGTH,String Functions
LOAD_FILE,String Functions
LOCATE,String Functions
LOG,Mathematical Functions
LOG10,Mathematical Functions
LPAD,String Functions
LTRIM,String Functions
MAKE_SET,String Functions
MD5,Miscellaneous Functions
MID,String Functions
MINUTE,Date and Time Functions
MOD,Mathematical Functions
MONTH,Date and Time Functions
MONTHNAME,Date and Time Functions
NOW,Date and Time Functions
NULLIF,Control Flow Functions
OCT,String Functions
OCTET_LENGTH,String Functions
ORD,String Functions
PASSWORD,Miscellaneous Functions
PERIOD_ADD,Date and Time Functions

Index

N-O

Index

P

Index

Q-R

Index

S

Index

T

Index

U

Index

V-Z

List of Figures

List of Tables

Chapter 11: Storing Binary Data in MySQL

Appendix B: MySQL Functions

Appendix D: PHP Functions for Working with MySQL

List of Sidebars

Chapter 1: Installation

Chapter 3: Getting Started with PHP

Chapter 9: Advanced SQL