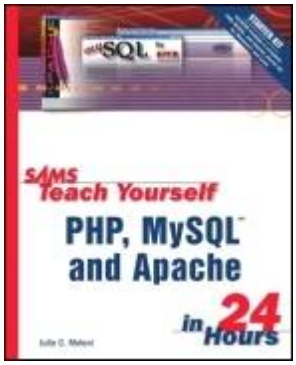


Sams Teach Yourself PHP, MySQL and Apache in 24 Hours

By Julie C Meloni

Start Reading

Publisher: Sams Publishing
Pub Date: December 11, 2002
ISBN: 0-6723-2489-X
Pages: 528

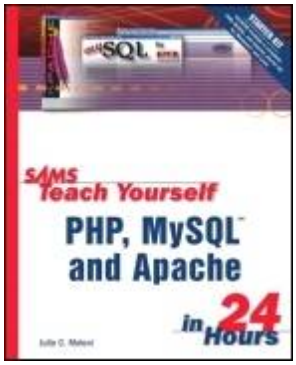
Sams Teach Yourself PHP, MySQL, and Apache in 24 Hours combines coverage of these three popular open-source Web development tools into one easy-to-understand book -- and it comes with one easy-to-use Starter Kit CD-ROM for Windows or Linux.

The book teaches the reader to install, configure and set up the PHP scripting language, the MySQL database system, and the Apache Web server.

By the end of this book the reader will understand how these technologies work, and -- more importantly -- how they can work together to create a dynamic Web site.

After creating a simple Web site using these tools, the reader will be able to manage a simple mailing list, and to create an online address book, shopping cart, and storefront.

The book also teaches the reader how to fine tune Apache and MySQL, and covers simple Web server security.



Sams Teach Yourself PHP, MySQL and Apache in 24 Hours

By Julie C Meloni

Start Reading

Publisher: Sams Publishing
Pub Date: December 11, 2002
ISBN: 0-6723-2489-X
Pages: 528

Index
[Team LiB]
PREVIOUS NEXT

Copyright

Copyright © 2003 by Sams Publishing

Library of Congress Catalog Card Number: 2002115016

Printed in the United States of America

First Printing: December 2002

05 04 03 02 4 3 2 1

Trademarks

Warning and Disclaimer

Credits

ACQUISITIONS EDITOR

Shelley Johnston

DEVELOPMENT EDITOR

Chris Newman

Lead Author

Julie C. Meloni is the technical director for i2i Interactive (www.i2ii.com), a multimedia company located in Los Altos, California. She's been developing Web-based applications since the Web first saw the light of day and remembers the excitement surrounding the first GUI Web browser. She has authored several books and articles on Web-based programming languages and database topics, and you can find translations of her work in several languages, including Chinese, Italian, Portuguese, Polish, and even Serbian!

Contributing Authors

Daniel López Ridruejo is a senior developer with Covalent Technologies, Inc., which provides Apache software, support, and services for the enterprise. He is the author of several popular Apache and Linux guides and of Comanche, a GUI configuration tool for Apache. Daniel is a regular speaker at open source conferences such as LinuxWorld, ApacheCon, and the O'Reilly Open Source Convention. He holds a Master of Science degree in telecommunications from the Escuela Superior de Ingenieros de Sevilla and Danmarks Tekniske Universitet. Daniel is a member of the Apache Software Foundation.

Matt Zandstra (matt@corrosive.co.uk) is a technical consultant. With his business partner, Max Guglielmino, he runs Corrosive Web Design (http://www.corrosive.co.uk), a company specializing in information design, usablity, and the creation of dynamic environments. Before this book took over his life once again, Matt was writing an XML/Java-based scripting language and interpreter for extracting content from Web pages. He is currently keen on design patterns, unit tests, extreme programming, and space operas. Matt is fatter than he was, but is still an urban cyclist. He says he is working on a novel, but he has been saying that for a long time. He lives by the sea in Brighton, Great Britain, with his partner, Louise McDougall, and their daughter, Holly.

Acknowledgments

The Apache Foundation, the PHP Group, and MySQL AB deserve much more recognition than they ever get for creating these super products that drive a great portion of the Web.

Daniel Lopez (author of Sams Teach Yourself Apache 2 in 24 Hours) and Matt Zandstra (author of Sams Teach Yourself PHP in 24 Hours) wrote super books, which form a significant portion of this book. Obviously, this book would not exist without them!

Great thanks especially to all the editors and layout folks at Sams who were involved with this book, for their hard work in seeing this through!

Thanks as always to everyone at i2i Interactive for their never-ending support and encouragement.

We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone number or email address. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: opensource@samspublishing.com

Mail:

Mark Taber
Associate Publisher
Sams Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

Reader Services

For more information about this book or others from Sams Publishing, visit our Web site at www.samspublishing.com . Type the ISBN (excluding hyphens) or the title of the book in the Search box to find the book you're looking for.

Introduction

Welcome to Sams Teach Yourself PHP, MySQL, and Apache in 24 Hours! This book combines the hours found in Sams Teach Yourself PHP in 24 Hours, Sams Teach Yourself MySQL in 24 Hours, and Sams Teach Yourself Apache in 24 Hours, to provide you with a solid and painless introduction to the world of developing Web-based applications using these three technologies.

Through a series of 24 easy hours, you'll learn the basics of programming in PHP, the methods for using and administering the MySQL relational database system, and the concepts necessary for configuring and managing Apache. The overall goal of the book is to provide you with the foundation you need to understand how seamlessly these technologies integrate with one another, and to give you practical knowledge of how to integrate them.

Who Should Read This Book?

This book is geared toward individuals who possess a general understanding of the concepts of working in a Web-based development environment, be it Linux/Unix or Windows. Installation and configuration lessons assume that you have familiarity with your operating system and the basic methods of building (on Linux/Unix systems) or installing (on Windows systems) software.

The lessons that delve into programming with PHP assume no previous knowledge of the language, but if you have experience with other programming languages such as C or Perl, you will find the going much easier. Similarly, if you have worked with other databases before, such as Oracle or Microsoft SQL Server, you will have a good foundation for working through the MySQL-related lessons.

The only real requirement is that you understand static Web content creation with HTML. If you are just starting out in the world of Web development, you will still be able to use this book, though you should consider working through an HTML tutorial. If you are comfortable creating basic documents and can build a basic HTML table, you will be fine.

How This Book Is Organized

This book is divided into five parts, corresponding to particular topics. The lessons within each part are designed to be read one right after another, with each lesson essentially building on the information found in those before it:

- Part I, "Getting Up and Running," will walk you through the installation and configuration of PHP, MySLQ, and Apache. You'll need to complete the lessons in Part I before moving on to the remaining lessons, unless you already have access to a working installation of these technologies. Even if you don't need to install and configure PHP, MySQL, and Apache in your environment, you should still skim these lessons so that you understand the basics.

- Part II, "Basic Language Elements," is predominantly devoted to teaching you the basics of the PHP language, and will get you in the habit of writing code, uploading it to your server, and testing the results. One of the lessons offers a basic SQL primer, and this part wraps up with an hour devoted to the integration of PHP and MySQL.

- Part III, "Getting Involved with the Code," consists of lessons that cover intermediatelevel application development topics, including working with forms and files, access restriction, and other small projects designed to introduce a specific concept.

- Part IV, "Simple Projects," contains lessons devoted to performing a particular task. These lessons consist of projects that integrate all the knowledge you have gained so far, and walk you through the process of building and testing the elements you will create.

- Part V, "Administration and Fine-Tuning," is devoted to administering and tuning MySQL and Apache, and also includes information on virtual hosting and setting up a secure Web server.

If you find that you are already familiar with a topic, you can skip ahead to the next lesson. However, in some instances, lessons will refer to specific concepts learned in previous hours, so be aware that you may have to skim through a skipped lesson so that your development environment remains consistent with the book.

At the end of each hour, there are a few quiz questions that will test how well you've learned the material. Additional activities provide another way to apply the information learned in the lesson and guide you toward using this newfound knowledge in the next hour.

Conventions Used in This Book

This book uses different typefaces to differentiate between code and plain English and also to help you identify important concepts. Throughout the lessons, code, commands, and text you type or see onscreen appear in a computer typeface. New terms appear in italics at the point in the text where they are defined. Additionally, icons accompany special blocks of information:

 A Note presents an interesting piece of information related to the current topic.

 A Tip offers advice or teaches an easier method for performing a task.

 A Caution warns you about potential pitfalls and explains how to avoid them.

 A new term icon will appear next to text introducing terms to the reader for the first time.

Part I: Getting Up and Running

Hour

Hour 1. Installing and Configuring MySQL

Welcome to the first hour of Sams Teach Yourself PHP, MySQL, and Apache in 24 Hours. This is the first of three "installation" hours, in which you will learn how to set up your development environment. We'll tackle the MySQL installation first, because the PHP installation is much simpler when MySQL is already installed.

In this hour, you will learn

- How to install MySQL
- Basic security guidelines for running MySQL
- How to work with the MySQL user privilege system

How to Get MySQL

The method you'll use to get MySQL depends on which distribution you want. Methods range from downloading a large file (or several large files) to buying an off-the-shelf product.

- MySQL AB distributes the open source version of MySQL on their Web site: http://www.mysql.com/. There is no shrink-wrapped product; what you get is what you download from the site, which includes binary distributions for Windows and Linux/Unix, as well as RPMs and source distributions.

- NuSphere Corporation sells a variety of products including the NuSphere Technology Platform, which includes a version of MySQL with NuSphere-specific enhancements, such as the Gemini table type. NuSphere's products are available for purchase from their Web site: http://www.nusphere.com/.

- AbriaSoft distributes MySQL as part of their Merlin Server (a Web development platform), which is available for download and purchase at their Web site: http://www.abriasoft.com/.

- Linux distribution CDs usually contain some version or another of the open source MySQL distribution, although it's usually a bit out-of-date.

The installation instructions in this hour are based on the official MySQL-Pro 4.0 distributions from MySQL AB. The process of installing the 3.23 version of MySQL is virtually identical, but if you choose to install that version, read the instructions that ship with the distribution just to be on the safe side. Any functional differences between versions 3.23 and 4.0 will be noted in later hours.

Installing MySQL on Linux/Unix

The process of installing MySQL on Linux/Unix is straightforward, whether you use RPMs or install the binaries. If you choose to install from RPMs, there are several that make up a full distribution. For a minimal installation you need

- MySQL-VERSION.i386.rpm— The MySQL server
- MySQL-client-VERSION.i386.rpm— The standard MySQL client programs

To perform the minimal installation, type the following at the prompt:

```
#> rpm -i MySQL-VERSION.i386.rpm MySQL-client-VERSION.i386.rpm
```


Replace VERSION in the filename with the actual version you downloaded. For example, the current MySQL-Pro 4.0 server RPM is called MySQL-4.0.4-0.i386.rpm.

Another painless installation method is to install MySQL from a binary distribution. This method requires gunzip and tar to uncompress and unpack the distribution and also requires the ability to create groups and users on the system. The first series of commands in the binary distribution installation process has you adding a group and a user and unpacking the distribution, as follows:

```
#> groupadd mysql
#> useradd -g mysql mysql
#> cd /usr/local
#> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
```

Next, the instructions tell you to create a link with a shorter name:

```
#> ln -s mysql-VERSION-OS mysql
#> cd mysql
```

Once unpacked, the README and INSTALL files will walk you through the remainder of the installation process for the version of MySQL you've chosen. In general, the following series of commands will be used:

```
#> scripts/mysql_install_db
#> chown -R root /usr/local/mysql
#> chown -R mysql /usr/local/mysql/data
#> chgrp -R mysql /usr/local/mysql
#> chown -R root /usr/local/mysql/bin
```

You're now ready to start the MySQL server.

Installing MySQL on Windows

The MySQL installation process on Windows is also quite simple—the developers from MySQL AB have packaged up everything you need in one zip file with a setup program! Once you download the zip file, extract its contents into a temporary directory and run the setup.exe application. After the setup.exe application installs the MySQL server and client programs, you're ready to start the MySQL server.

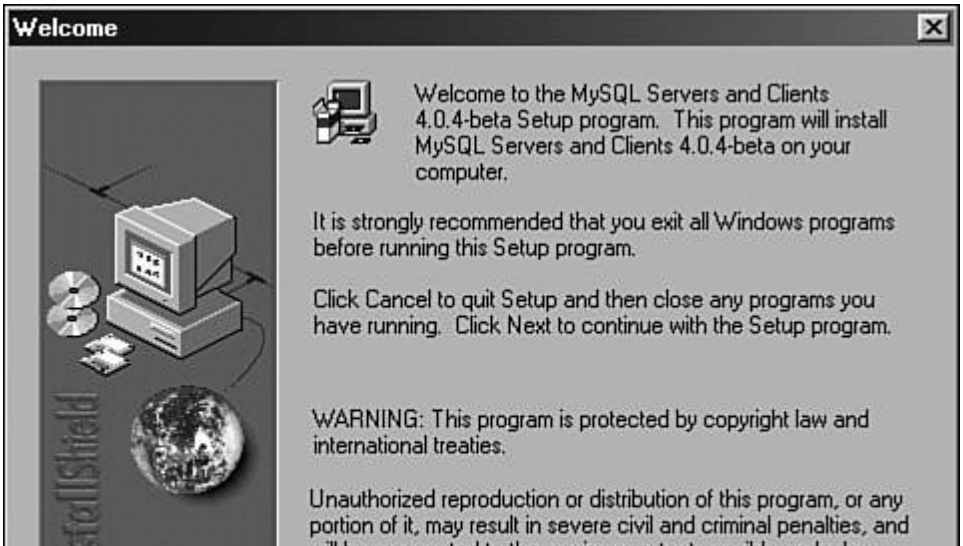
The following steps detail the installation of MySQL 4.0 from MySQL AB on Windows, and show you what you can expect if you install MySQL in a Windows 95/98/NT/2000/XP environment for testing and development. Many users install MySQL on personal Windows machines, to get a feel for working with the database before deploying MySQL in a production environment.

1. Visit the MySQL-Pro 4.0 download page at http://www.mysql.com/downloads/mysql-pro-4.0.html and find the Windows section. You want to download the file under the "Installation files (zip)" heading rather than the one under the "Cygwin downloads (tar.bz2)" heading.

If you have the tools and skills to compile your own Windows binary files, select the Cygwin source download and follow the instructions contained in the source distribution.

2. Clicking the Download link will take you to a page of mirror sites. Select the mirror site closest to you, and download the file. It is a large file, so you may be waiting awhile, depending on your connection speed.
3. Once the zip file is on your hard drive, extract its contents to a temporary directory.
4. From the temporary directory, find the setup.exe file and double-click it to start the installation. You will see the first screen of the installation wizard, as shown in Figure 1.1. Click Next to continue.

Figure 1.1. The first step of the MySQL installation wizard.



Troubleshooting Your Installation

If you have any problems during the installation of MySQL, the first place you should look is the "Problems and Common Errors" chapter of the MySQL manual, which is located at http://www.mysql.com/doc/P/r/Problems.html.

The following are some common problems:

- On Linux/Unix, Incorrect permissions do not allow you to start the MySQL daemon. If this is the case, be sure you have changed owners and groups to match those indicated in the installation instructions.
- If you see the message "Access denied" when connecting to MySQL, be sure you are using the correct username and password.
- If you see the message "Can't connect to server", make sure the MySQL daemon is running.
- When defining tables, if you specify a length for a field whose type does not require a length, the table will not be created. For example, you should not specify a length when defining a field as TEXT (as opposed to CHAR or VARCHAR).

If you still have trouble after reading the manual, sending e-mail to the MySQL mailing list (see http://www.mysql.com/documentation/lists.html for more information) will likely produce results. You can also purchase support contracts from MySQL AB for a very low fee. If you have purchased a version of MySQL other than the one distributed by MySQL AB, you should turn to the documentation and support options for that product. The companies that sell other versions of MySQL usually have additional support contracts that you can purchase.

Basic Security Guidelines

Regardless of whether you are running MySQL on Windows or Linux/Unix, and no matter whether you administer your own server or use a system provided by your Internet service provider, every developer needs to understand basic security guidelines. If you are accessing MySQL through your Internet service provider, there are several aspects of server security that you, as a non-root user, should not be able to modify or circumvent. Unfortunately, many Internet service providers pay no mind to security guidelines, leaving their clients exposed—and for the most part, unaware of the risk.

Starting MySQL

Securing MySQL begins with the server startup procedure. If you are not the administrator of the server, you won't be able to change this, but you can certainly check it out and report vulnerabilities to your Internet service provider.

If your MySQL installation is on Linux/Unix, your primary concern should be the owner of the MySQL daemon—it should not be root. Running the daemon as a non-root user such as mysql or database will limit the ability of malicious individuals to gain access to the server and overwrite files.

You can verify the owner of the process using the ps (process status) command on your Linux/Unix system. The following output shows MySQL running as a non-root user (see the first entry on the second line):

```
#> ps auxw | grep mysqld
mysql 153 0.0 0.6 12068 2624 ? S Nov16 0:00
/usr/local/bin/mysql/bin/mysqld
--defaults-extra-file=/usr/local/bin/mysql/data/my.cnf
--basedir=/usr/local/bin/mysql --datadir=/usr/local/bin/mysql/data
--user=mysql --pid-file=/usr/local/bin/mysql/data/mike.pid --skip-locking
```

The following output shows MySQL running as the root user (see the first entry on the second line):

```
#> ps auxw | grep mysqld
root 21107 0.0 1.1 11176 1444 ? S Nov 27 0:00
/usr/local/mysql/bin/mysqld
--basedir=/usr/local/mysql --datadir=/usr/local/mysql/data --skip-locking
```

If you see that MySQL is running as root on your system, immediately contact your Internet service provider and complain. If you are the server administrator, you should start the MySQL process as a non-root user or specify the username in the startup command line:

```
mysqld --user=non_root_user_name
```

For example, if you want to run MySQL as user mysql, use

```
mysqld --user=mysql
```

Securing Your MySQL Connection

You can connect to the MySQL monitor or other MySQL applications in several different ways, each of which has its own security risks. If your MySQL installation is on your own workstation, you have less to worry about than

Introducing the MySQL Privilege System

The MySQL privilege system is always "on." The first time you try to connect and for each subsequent action, MySQL checks the following three things:

- Where you are accessing from (your host)
- Who you say you are (your username and password)
- What you're allowed to do (your command privileges)

All this information is stored in the database called mysql, which is automatically created when MySQL is installed. There are several tables in the mysql database:

- columns_priv— Defines user privileges for specific fields within a table.
- db— Defines the permissions for all databases on the server.
- func— Defines user-created functions.
- host— Defines the acceptable hosts that can connect to a specific database.
- tables_priv— Defines user privileges for specific tables within a database.
- user— Defines the command privileges for a specific user.

These tables will become more important to you later in this hour as you add a few sample users to MySQL. For now, just remember that these tables exist and must have relevant data in them in order for users to complete actions.

The Two-Step Authentication Process

As you've learned, MySQL checks three things during the authentication process. The actions associated with these three things are performed in two steps:

Working with User Privileges

In most cases when you are accessing MySQL through an Internet service provider, you will have only one user and one database available to you. By default, that one user will have access to all tables in that database and will be allowed to perform all commands.

In this case, the responsibility is yours as the developer to create a secure application through your programming.

If you are the administrator of your own server or have the ability to add as many databases and users as you want, as well as modify the access privileges of your users, these next few sections will take you through the processes of doing so.

Adding Users

Administering your server through a third-party application may afford you a simple method for adding users, using a wizard-like process or a graphical interface. However, adding users through the MySQL monitor is not difficult, especially if you understand the security checkpoints used by MySQL, which you just learned.

The simplest method for adding new users is the GRANT command. By connecting to MySQL as the root user, you can issue one command to set up a new user. The other method is to issue INSERT statements into all the relevant tables in the mysql database, which requires you to know all the fields in the tables used to store permissions. This method works just as well but is more complicated than the simple GRANT command.

The simple syntax of the GRANT command is

```
GRANT privileges
ON databasename.tablename
TO username@host
IDENTIFIED BY "password";
```

The privileges you can grant are

- ALL— Gives the user all of the following privileges
- ALTER— User can alter (modify) tables, columns, and indexes
- CREATE— User can create databases and tables
- DELETE— User can delete records from tables

Summary

Installing MySQL on Windows is a very simple process due to a wizard-based installation method. MySQL AB provides a GUI-based administration tool for Windows users, called WinMySQLadmin. Linux/Unix users do not have a wizard-based installation process, but it's not difficult to follow a simple set of commands to unpack the MySQL client and server. Linux/Unix users can also use RPMs for installation.

Security is always a priority, and there are several steps you can take to ensure a safe and secure installation of MySQL. Even if you are not the administrator of the server, you should be able to recognize breaches and raise a ruckus with the server administrator!

The MySQL server should not run as the root user. Additionally, named users within MySQL should always have a password, and their access privileges should be well defined.

MySQL uses the privilege tables in a two-step process for each request that is made. MySQL needs to know who you are and where you are connecting from, and each of these pieces of information must match an entry in its privilege tables. Also, the user whose identity you are using must have specific permission to perform the type of request you are making.

You can add user privileges using the GRANT command, which uses a simple syntax to add entries to the user table in the mysql database. The REVOKE command, which is equally simple, is used to remove those privileges.

Q&A

Q1:

How do I completely remove a user? The REVOKE command just eliminates the privileges.

A1:

To completely remove a user from the privilege table, you have to issue a specific DELETE command from the user table in the mysql database.

Q2:

What if I tell my Internet service provider to stop running MySQL as root, and they won't?

A2:

Switch providers. If your Internet service provider doesn't recognize the risks of running something as important as your database as the root user, and doesn't listen to your request, find another provider. There are providers with plans as low as $9.95/month that don't run important processes as root!

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

True or False: Telnet is a perfectly acceptable method to securely connect to MySQL from a remote host.

A1:

False. The key word is "secure," and Telnet does not encrypt data between hosts. Instead, use SSH to connect to your server.

2:

Which three pieces of information does MySQL check each time a request is made?

A2:

Who you are, where you are accessing from, and what actions you're allowed to perform.

3:

What command would you use to grant SELECT, INSERT, and UPDATE privileges to a user named bill on localhost to all tables on the BillDB database? Also, what piece of information is missing from this statement that is recommended for security purposes?

A3:

The command is

```
GRANT SELECT, INSERT, UPDATE
ON BillDB.*
TO bill@localhost;
```

The important missing piece is a password for the user!

Activities

1.

Hour 2. Installing and Configuring Apache

In this second of three "installation" hours, you will install the Apache Web server and familiarize yourself with its main components, including log and configuration files. In this hour, you will learn

- How to install the Apache server on Linux/Unix
- How to install the Apache server on Windows
- How to make configuration changes to Apache
- Where Apache log and configuration files are stored

Choosing the Appropriate Installation Method

You have several options when it comes to getting a basic Apache installation in place. Apache is open source, meaning that you can have access to the full source code of the software, which in turn enables you to build your own custom server. Additionally, pre-built Apache binary distributions are available for most modern Unix platforms. Finally, Apache comes already bundled with a variety of Linux distributions, and you can purchase commercial versions from software vendors such as Covalent Technologies and IBM. The examples in this hour will teach you how to build Apache from source if you are using Linux/Unix, and how to use the installer if you plan to run Apache on a Windows system.

Building from Source

Building from source gives you the greatest flexibility, as it enables you to build a custom server, remove modules you do not need, and extend the server with third-party modules. Building Apache from source code enables you to easily upgrade to the latest versions and quickly apply security patches, whereas updated versions from vendors can take days or weeks to appear.

The process of building Apache from the source code is not especially difficult for simple installations, but can grow in complexity when third-party modules and libraries are involved.

Installing a Binary

Linux/Unix binary installations are available from vendors and can also be downloaded from the Apache Software Foundation Web site. They provide a convenient way to install Apache for users with limited system administration knowledge, or with no special configuration needs. Third-party commercial vendors provide prepackaged Apache installations together with an application server, additional modules, support, and so on.

The Apache Software Foundation provides an installer for Windows systems—a platform where a compiler is not as commonly available as in Linux/Unix systems.

Installing Apache on Linux/Unix

This section explains how to install a fresh build of Apache 2.0 on Linux/Unix. The steps necessary to successfully install Apache from source are

1. Downloading the software
2. Running the configuration script
3. Compiling the code and installing it

The following sections describe these steps in detail.

Downloading the Apache Source Code

The official Apache download site is located at http://www.apache.org/dist/httpd. You can find several Apache versions, packaged with different compression methods. The distribution files are first packed with the tar utility and then compressed either with the gzip tool or the compress utility. Download the .tar.gz version if you have the gunzip utility installed in your system. This utility comes installed by default in open source operating systems such as FreeBSD and Linux. Download the tar.Z file if gunzip is not present in your system. (It isn't included in the default installation of many commercial Unix operating systems.)

The file you want to download will be named something similar to httpd-2.0. version.tar.Z or httpd-2.0. version.tar.gz, where version is the most recent release version of Apache. For example, Apache version 2.0.43 is downloaded as a file named httpd-2.0.43.tar.gz. Keep the downloaded file in a directory reserved for source files, such as /usr/src/ or /usr/local/src/.

Uncompressing the Source Code

If you downloaded the tarball compressed with gzip (it will have a tar.gz suffix), you can uncompress it using the gunzip utility (part of the gzip distribution).

 Tarball is a commonly used nickname for software packed using the tar utility.

You can uncompress and unpack the software by typing the following command:

```
#> gunzip < httpd-2.0*.tar.gz | tar xvf -
```

If you downloaded the tarball compressed with compress (tar.Z suffix), you can issue the following command:

```
#> cat httpd-2.0*.tar.Z | uncompress | tar xvf -
```

Installing Apache on Windows

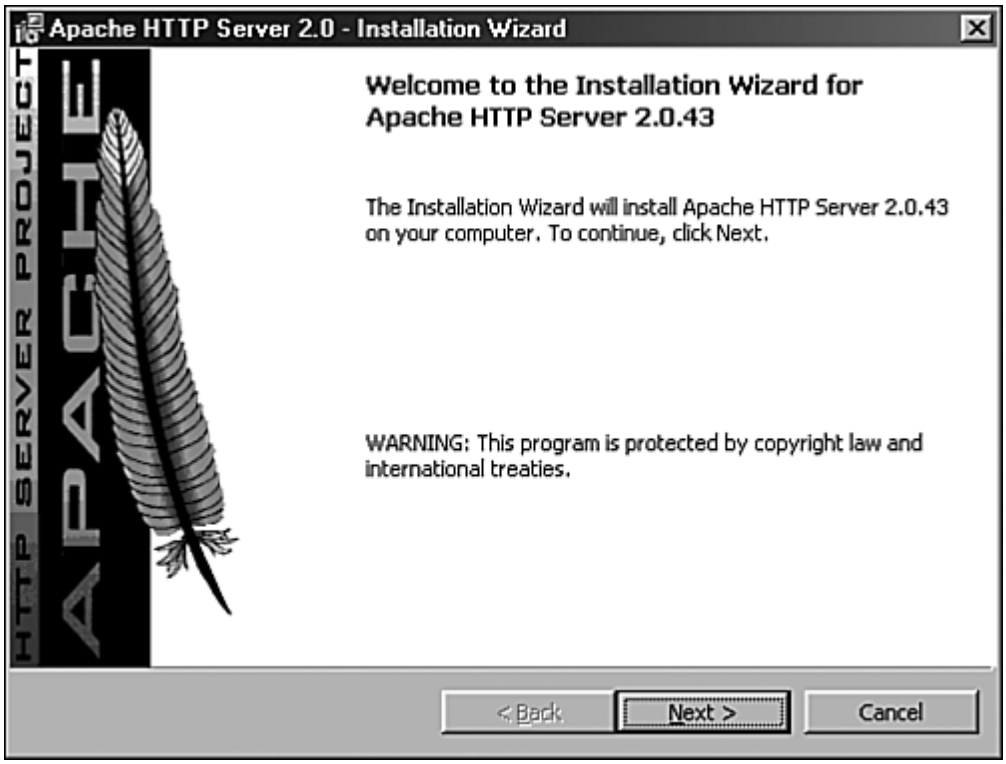
Apache 2.0 runs on most Windows platforms and offers increased performance and stability over the 1.3 versions for Windows. You can build Apache from source, but because not many Windows users have compilers, this section deals with the binary installer.

Before installing Apache, you'll probably want to make sure that you are not currently running a Web server (for instance, a previous version of Apache, Microsoft Internet Information Server, or Microsoft Personal Web Server) in your machine. You might want to uninstall or otherwise disable existing servers. You can run several Web servers, but they will need to run in different address and port combinations.

You can download an installer from http://www.apache.org/dist/httpd/binaries/win32.

After you download the installer, double-click on the file to start the installation process. You will get a welcome screen, as shown in Figure 2.1. Select Next to continue the installation process, and you will be prompted to accept the Apache license. Basically the license says that you can do whatever you want with the software—including making proprietary modifications—except claim that you wrote it, but be sure to read the license so that you fully understand the terms.

Figure 2.1. The Windows installer welcome screen.



After you accept the license, the installer presents you with a brief introduction to Apache. Following that, it asks you to provide basic information about your computer, as shown in Figure 2.2. This includes the full network address for the server (for instance, mycomputer.mydomain.com) and the administrator's email address. The server name will be the name that your clients will use to access your server, and the administrator email address will be added to error messages so that visitors know how to contact you when something goes wrong.

Figure 2.2. The basic information screen.

Apache Configuration File Structure

Apache keeps all of its configuration information in text files. The main file is called httpd.conf. This file contains directives and containers, which enable you to customize your Apache installation. Directives configure specific settings of Apache, such as authorization, performance, and network parameters. Containers specify the context to which those settings refer. For example, authorization configuration can refer to the server as a whole, a directory, or a single file.

Directives

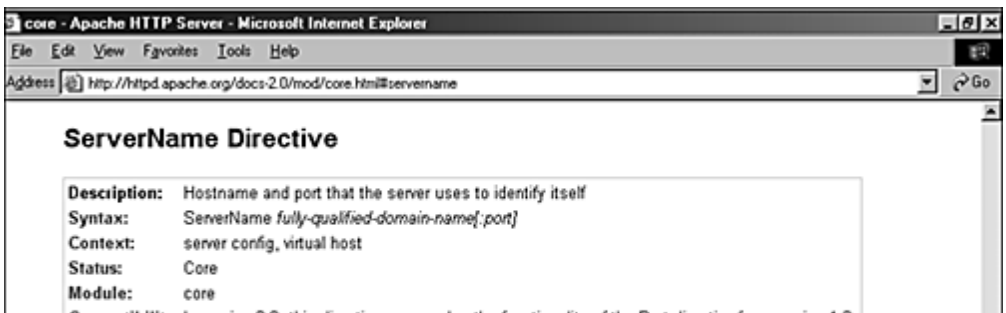
The following rules apply for Apache directive syntax:

- The directive arguments follow the directive name.
- Directive arguments are separated by spaces.
- The number and type of arguments vary from directive to directive; some have no arguments.
- A directive occupies a single line, but you can continue it on a different line by ending the previous line with a backslash character (\).
- The pound sign (#) should precede the directive, and must appear on its own line.

In the Apache server documentation, found online at http://httpd.apache.org/docs-2.0/, you can browse the directives in alphabetical order or by the module to which they belong. You'll soon learn about some of the basic directives, but you should supplement your knowledge using the online documentation.

Figure 2.5 shows an entry from the documentation for the ServerName directive description. You can read this description in the online documentation at http://httpd.apache.org/docs-2.0/mod/core.html#servername.

Figure 2.5. Directive description example.



Apache Log Files

Apache includes two log files by default. The access_log file is used to track client requests. The error_log is used to record important events, such as errors or server restarts. These files don't exist until you start Apache for the first time. The files are named access.log and error.log in Windows platforms.

access_log

When a client requests a file from the server, Apache records several parameters associated with the request, including the IP address of the client, the document requested, the HTTP status code, and the current time. Listing 2.5 shows sample log file entries. Hour 17, "Logging and Monitoring Server Activity," will show you how to modify which parameters are logged.

Listing 2.5 Sample access_log Entries

```
1: 127.0.0.1 - - [01/Sep/2002:09:43:37 -0700] "GET / HTTP/1.1" 200 1494
2: 127.0.0.1 - - [01/Sep/2002:09:43:40 -0700] "GET /manual/ HTTP/1.1" 200 10383
```

error_log

This file includes error messages, startup messages, and any other significant events in the life cycle of the server. This is the first place to look when you have a problem with Apache. Listing 2.6 shows sample error_log entries.

Listing 2.6 Sample error_log Entries

```
1: [Sun Sep 01 09:42:59 2002] [notice] Parent: Created child process -2245
2: [Sun Sep 01 09:42:59 2002] [notice] Child -2242: Child process is running
3: [Sun Sep 01 09:42:59 2002] [notice] Child -2242: Acquired the start mutex.
4: [Sun Sep 01 09:42:59 2002] [notice] Child -2242: Starting 250 worker threads.
```

Additional Files

The httpd.pid file contains the process ID of the running Apache server. You can use this number to send signals to Apache manually, as described in the next section.

The scoreboard file, present on Linux/Unix Apache, is used by the process-based MPMs to communicate with their children.

In general, you do not need to worry about these files.

Apache-Related Commands

The Apache distribution includes several executables. This section covers only the server binary and related scripts. Hour 15, "Restricting Access to Your Applications," and Hour 22, "Apache Performance Tuning and Virtual Hosting," cover additional utilities included with the Apache distribution.

Apache Server Binary

The Apache executable is named httpd in Linux/Unix and apache.exe in Windows. It accepts several command-line options, which are described in Table 2.1. You can get a complete listing of options by typing /usr/local/apache2/bin/httpd -h on Linux/Unix, or by typing apache.exe -h from a command prompt on Windows.

Table 2.1. httpd Options

Option	Meaning
-D	Allows you to pass a parameter that can be used for <IfDefine> section processing
-l	Lists compiled-in modules
-v	Shows version number and server compilation time
-f	Allows you to pass the location of httpd.conf if it is different from the compiletime default

After Apache is running, you can use the kill command on Linux/Unix to send signals to the parent Apache process. Signals provide a mechanism to send commands to a process. To send a signal, execute the following command:

```
#> kill -SIGNAL pid
```

where pid is the process ID and SIGNAL is one of the following:

- HUP— Stop the server
- USR1 or WINCH— Graceful restart; which signal to use depends on the underlying operating system
- SIGHUP — Restart

Starting Apache for the First Time

Before you start Apache, you should verify that the minimal set of information is present in the Apache configuration file, httpd.conf. The following sections describe the basic information needed to configure Apache, and how to start the server.

Check Your Configuration File

You can edit the Apache httpd.conf file with your favorite text editor. In Linux/Unix, this probably means vi or emacs. In Windows, you can use Notepad or WordPad. You must remember to save the configuration file in plain text, which is the only format Apache will understand.

There are only two parameters that you might need to change to enable you to start Apache for the first time: the name of the server and the address and port to which it is listening. The name of the server is the one Apache will use when it needs to refer to itself (for example, when redirecting requests).

Apache can usually figure out its server name from the IP address of the machine, but this is not always the case. If the server does not have a valid DNS entry, you might need to specify one of the IP addresses of the machine. If the server is not connected to a network (you might want to test Apache on a standalone machine), you can use the value 127.0.0.1, which is the loopback address. The default port value is 80. You might need to change this value if there is already a server running in the machine at port 80, or if you do not have administrator permissions—on Linux/Unix systems, only the root user can bind to privileged ports (those with port numbers lower than 1024).

You can change both the listening address and the port values with the Listen directive. The Listen directive takes either a port number or an IP address and a port, separated by a semicolon. If only the port is specified, Apache will listen on that port at all available IP addresses in the machine. If an additional IP address is provided, Apache will listen at only that address and port combination. For example, Listen 80 tells Apache to listen for requests at all IP addresses on port 80. Listen 10.0.0.1:443 tells Apache to listen only at 10.0.0.1 on port 443.

The ServerName directive enables you to define the name the server will report in any self-referencing URLs. The directive accepts a DNS name and an optional port, separated by a colon. Make sure that ServerName has a valid value. Otherwise, the server will not function properly; for example, it will issue incorrect redirects.

On Linux/Unix platforms, you can use the User and Group directives to specify which user and group IDs the server will run as. The nobody user is a good choice for most platforms. However, there are problems in the HP-UX platform with this user ID, so you must create and use a different user ID, such as www.

Starting Apache

To start Apache on Linux/Unix, change to the directory containing the apachectl script and execute the following command:

```
#> ./apachectl start
```

To start Apache on Windows, click on the Start Apache link in the Control Apache section in the Start menu. If you installed Apache as a service, you must start the Apache service instead.

Troubleshooting

The following subsections describe several common problems that you might encounter the first time you start Apache.

Already an Existing Web Server

If there is already a server running in the machine and listening to the same IP address and port combination, Apache will not be able to start successfully. You will get an entry in the error log file indicating that Apache cannot bind to the port:

```
[crit] (48)Address already in use: make_sock: could not bind to address 10.0.0.2:80
[alert] no listening sockets available, shutting down
```

To solve this problem, you need to stop the running server or change the Apache configuration to listen on a different port.

No Permission to Bind to Port

You will get an error if you do not have administrator permissions and you try to bind to a privileged port (between 0 and 1024):

```
[crit] (13)Permission denied: make_sock: could not bind to address 10.0.0.2:80
[alert] no listening sockets available, shutting down
```

To solve this problem, you must either log on as the administrator before starting Apache or change the port number (8080 is a commonly used nonprivileged port).

Access Denied

You might not be able to start Apache if you do not have permission to read the configuration files or to write to the log files. You will get an error similar to the following:

[View full width]

```
(13)Permission denied: httpd: could not open error log file /usr/local/apache2/logs/
➥error_log.
```

This problem can arise if Apache was built and installed by a different user than the one trying to run it.

Wrong Group Settings

You can configure Apache to run under a certain username and group. Apache has default values for the running server username and group. Sometimes the default value is not valid, and you will get an error containing setgid: unable to set group id.

Summary

This hour explained different ways of getting an Apache 2.0 server installed on your Linux/Unix or Windows machine. It covered both binary and source installation and explained the basic build-time options. Additionally, you learned the location of the server configuration files, and the syntax of the commands used to modify your Apache configuration. You learned about the two main log files—access_log and error_log. Finally, you saw how to start and stop the server using the Apache control scripts or the Apache server binary on Linux/Unix and Windows platforms.

Q&A

Q1:

How can I start a clean build?

A1:

If you need to build a new Apache from source and do not want the result of earlier builds to affect the new one, it is always a good idea to run the make clean command. That will take care of cleaning up any existing binaries, intermediate object files, and so on.

Q2:

Why are per-directory configuration files useful?

A2:

Although per-directory configuration files have an impact on server performance, they can be useful for delegated administration. Because per-directory configuration files are read every time a request is made, there is no need to restart the server when a change is made to the configuration.

You can allow users of your Web site to make configuration changes on their own without granting them administrator privileges. In this way, they can password-protect sections of their home pages, for example.

Q3:

What do you mean by a valid ServerName directive?

A3:

The DNS system is used to associate IP addresses with domain names. The value of ServerName is returned when the server generates a URL. If you are using a certain domain name, you must make sure that it is included in your DNS system and will be available to clients visiting your site.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1:

How can you specify the location where you want to install Apache?

A1:

Linux/Unix users can use the --prefix option of the configure script. If an existing installation is present at that location, the configuration files will be preserved but the binaries will be replaced. On Windows, this location is set in the installation wizard.

2:

What is the main difference between <Location> and <Directory> sections?

A2:

Directory sections refer to file system objects; Location sections refer to elements in the address bar of the Web page (also called the URI).

3:

What is the difference between a restart and a graceful restart?

A3:

During a normal restart, the server is stopped and then started, causing some requests to be lost. A graceful restart allows Apache children to continue to serve their current requests until they can be replaced with children running the new configuration.

Activities

1. Practice the various types of server shutdown and restart procedures.
2.

Hour 3. Installing and Configuring PHP

In the last of the three "installation" hours, you will acquire, install, and configure PHP and make some basic changes to your Apache installation. In this hour, you will learn

- How to install PHP with Apache on Linux/Unix
- How to install PHP with Apache server on Windows
- How to test your PHP installation
- How to find help when things go wrong
- The basics of the PHP language

Building PHP on Linux/Unix with Apache

In this section, we will look at one way of installing PHP with Apache on Linux/Unix. The process is more or less the same for any Unix operating system. While you might be able to find pre-built versions of PHP for your system, compiling PHP from the source gives you greater control over the features built into your binary.

To download the PHP distribution files, go to the home of PHP, http://www.php.net/, and follow the link to the Downloads section. Grab the latest version of the source code—for this example, we are using 4.2.3. Your distribution will be named something similar to php-version.tar.gz, where version is the most recent release number. This archive will be a compressed tar file, so you will need to unpack it:

```
#> tar -xvzf php- version.tar.gz
```

Keep the downloaded file in a directory reserved for source files, such as /usr/src/ or /usr/local/src/. After your distribution is unpacked, you should move to the PHP distribution directory:

```
#> cd php-version
```

Within your distribution directory you will find a script called configure. This script accepts additional information that is provided when the configure script is run from the command line. These "command-line arguments" will control the features that PHP will support. For this example, we will include the basic options you need to use to install PHP with Apache and MySQL support. We will discuss some of the available configure options later in the hour.

```
#> ./configure  --prefix=/usr/local/php \
--with-mysql=/usr/local/bin/mysql \
--with-apxs2=/usr/local/apache2/bin/apxs
```

Once the configure script has run, you will be returned to the prompt after receiving several informational notes from the PHP Group:

```
+--------------------------------------------------------------------+
|                        > WARNING >                                 |
|                                                                    |
| Apache 2 Support is EXPERIMENTAL and should NOT be used in         |
| production environment. Before submitting bug reports, try the     |
| latest CVS snapshot from http://snaps.php.net                      |
+--------------------------------------------------------------------+
| License:                                                           |
| This software is subject to the PHP License, available in this     |
| distribution in the file LICENSE.  By continuing this installation |
| process, you are bound by the terms of this license agreement.     |
| If you do not agree with the terms of this license, you must abort |
| the installation process at this point.                            |
+--------------------------------------------------------------------+
|                          > NOTE >                                  |
|            The default for register_globals is now OFF!            |
|                                                                    |
| If your application relies on register_globals being ON, you       |
| should explicitly set it to on in your php.ini file.               |
| Note that you are strongly encouraged to read                      |
| http://www.php.net/manual/en/security.registerglobals.php          |
| about the implications of having register_globals set to on, and   |
| avoid using it if possible.                                        |
+--------------------------------------------------------------------+
#>
```

Installing PHP Files on Windows

Unlike building and installing PHP on Linux/Unix, installing PHP on Windows requires nothing more than downloading the distribution and moving a few files around. To download the PHP distribution files, go to the home of PHP, http://www.php.net/, and follow the link to the Downloads section. Grab the latest version of the zip package from the Windows Binaries section—for this example we are using 4.2.3. Your distribution will be named something similar to php-version.zip, where version is the most recent release number.

Once the file is downloaded to your system, double-click on it to launch your unzipper. The distribution is packed up with pathnames already in place, so if you extract the files to the root of your drive, it will create a directory called php-version-Win32, and place all the files and subdirectories under that new directory.

Now that you have all the basic PHP distribution files, you just need to move a few of them around:

1. In the PHP installation directory, find the php.ini-dist file and rename it php.ini.
2. Move the php.ini file to C:\WINDOWS\ or wherever you usually put your *.ini files.
3. Move the php4ts.dll file to C:\WINDOWS\SYSTEM\ or wherever you usually put your *.dll files.

To get a basic version of PHP working with Apache, you'll need to make a few minor modifications to the Apache configuration file.

Integrating PHP with Apache on Windows

To ensure that PHP and Apache get along with one another, you need to add a few items to the httpd.conf configuration file. First, find a section that looks like this:

```
# Example:
# LoadModule foo_module modules/mod_foo.so
#
LoadModule access_module modules/mod_access.so
LoadModule actions_module modules/mod_actions.so
LoadModule alias_module modules/mod_alias.so
LoadModule asis_module modules/mod_asis.so
LoadModule auth_module modules/mod_auth.so
#LoadModule auth_anon_module modules/mod_auth_anon.so
#LoadModule auth_dbm_module modules/mod_auth_dbm.so
#LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule autoindex_module modules/mod_autoindex.so
#LoadModule cern_meta_module modules/mod_cern_meta.so
LoadModule cgi_module modules/mod_cgi.so
#LoadModule dav_module modules/mod_dav.so
#LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule dir_module modules/mod_dir.so
LoadModule env_module modules/mod_env.so
#LoadModule expires_module modules/mod_expires.so
#LoadModule file_cache_module modules/mod_file_cache.so
#LoadModule headers_module modules/mod_headers.so
```

php.ini Basics

After you have compiled or installed PHP, you can still change its behavior with the php.ini file. On Unix systems, the default location for this file is /usr/local/php/lib, or the lib subdirectory of the PHP installation location you used at configuration time. On a Windows system, this file should be in the Windows directory.

Directives in the php.ini file come in two forms: values and flags. Value directives take the form of a directive name and a value separated by an equals sign. Possible values vary from directive to directive. Flag directives take the form of a directive name and a positive or negative term separated by an equals sign. Positive terms include 1, On, Yes, and True. Negative terms include 0, Off, No, and False. Whitespace is ignored.

You can change your php.ini settings at any time, but after you do, you'll need to restart the server for the changes to take effect. At some point, take time to read through the php.ini file on your own, to see the types of things that can be configured.

Testing Your Installation

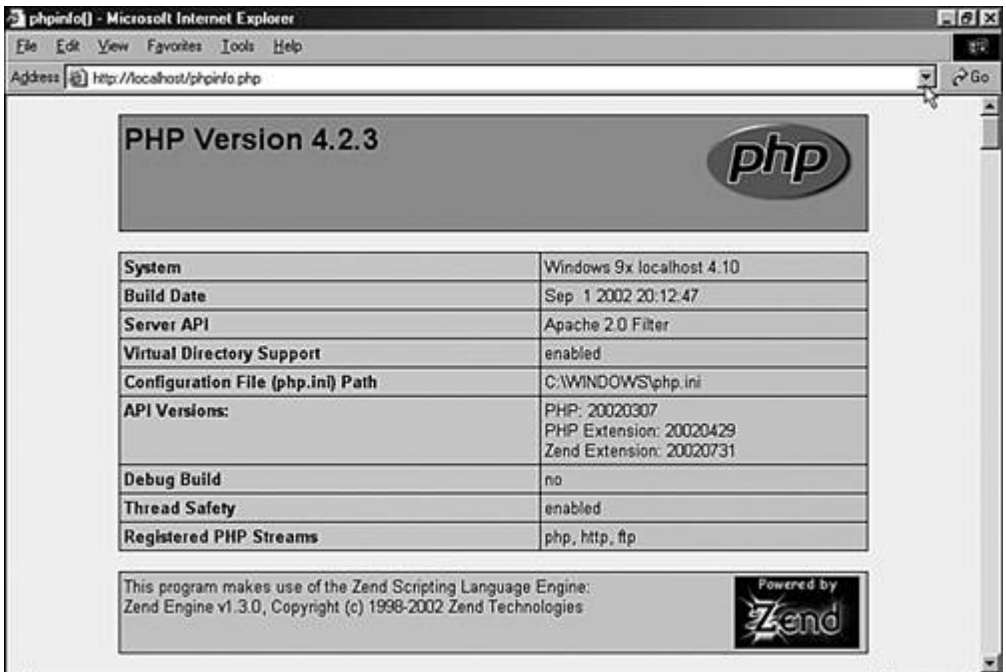
The simplest way to test your PHP installation is to create a small test script utilizing the phpinfo() function. This function will produce a long list of configuration information. Open a text editor and type the following line:

```
<? phpinfo(); ?>
```

Save this file as phpinfo.php and place it in the document root of your Web server—the htdocs subdirectory of your Apache installation. Access this file via your Web browser and you should see something like Figure 3.1 or Figure 3.2 .

Figure 3.1. The results of phpinfo() on a Linux/Unix system.

Figure 3.2. The results of phpinfo() on a Windows system.



Getting Installation Help

Help is always at hand on the Internet, particularly for problems concerning open source software. Wait a moment before you click the send button, however. No matter how intractable your installation, configuration, or programming problem might seem, chances are you are not alone. Someone has probably already answered your question.

When you hit a brick wall, your first recourse should be to the official PHP site at http://www.php.net/ (particularly the annotated manual at http://www.php.net/manual/).

If you still can't find your answer, don't forget that the PHP site is searchable. The advice you are seeking may be lurking in a press release or a Frequently Asked Questions file. You can also search the mailing list archives at http://www.php.net/search.php. These archives represent a huge information resource with contributions from many of the great minds in the PHP community. Spend some time trying out a few keyword combinations.

If you are still convinced that your problem has not been addressed, you may well be doing the PHP community a service by exposing it. You can join the PHP mailing lists at http://www.php.net/support.php. Although these lists often have high volume, you can learn a lot from them. If you are serious about PHP scripting, you should certainly subscribe to at least a digest list. Once you've subscribed to the list that matches your concerns, you might consider posting your problem.

When you post a question, it is a good idea to include as much information as possible (without writing a novel). The following items are often pertinent:

- Your operating system
- The version of PHP you are running or installing
- The configuration options you chose
- Any output from the configure or make commands that preceded an installation failure
- A reasonably complete example of the code that is causing problems

Why all these cautions about posting a question to a mailing list? First, developing research skills will stand you in good stead. A good researcher can generally solve a problem quickly and efficiently. Posting a naive question to a technical list often results in a wait rewarded only by a message or two referring you to the archives where you should have begun your search for answers in the first place.

The Basics of PHP Scripts

Let's jump straight in with a PHP script. To begin, open your favorite text editor. Like HTML documents, PHP files are made up of plain text. You can create them with any text editor, such as Notepad on Windows, Simple Text and BBEdit on Mac OS, or vi and Emacs on Unix operating systems. Most popular HTML editors provide at least some support for PHP.

Keith Edmunds maintains a handy list of PHP-friendly editors at http://www.itworks.demon.co.uk/phpeditors.htm.

Type in the example in Listing 3.1 and save the file, calling it something like first.php.

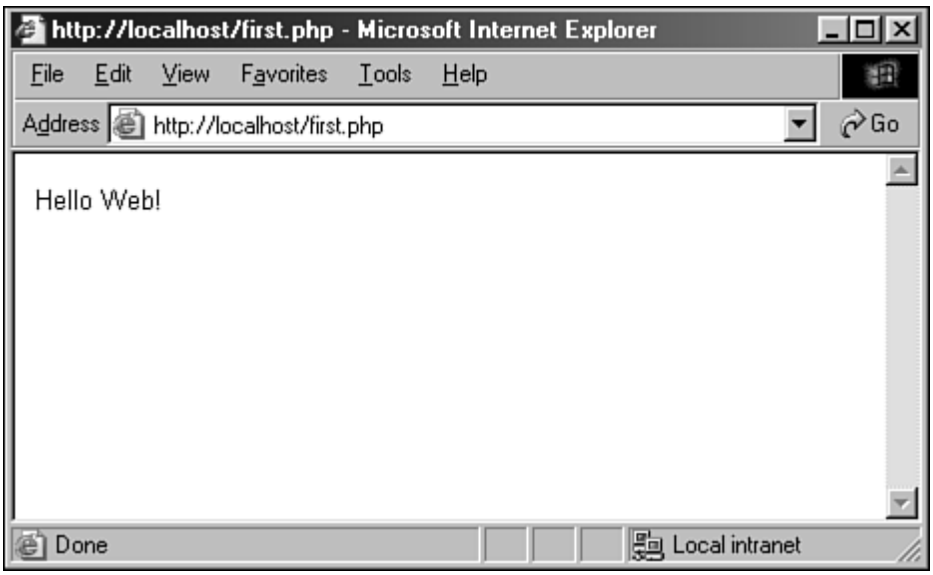
Listing 3.1 A Simple PHP Script

```
1: <?php
2:     print "Hello Web!";
3: ?>
```

If you are not working directly on the machine that will be serving your PHP script, you will probably need to use an FTP client such as WS-FTP for Windows or Fetch for Mac OS to upload your saved document to the server.

Once the document is in place, you should be able to access it via your browser. If all has gone well, you should see the script's output. Figure 3.3 shows the output from the first.php script.

Figure 3.3. Success: the output from Listing 3.1.



Beginning and Ending a Block of PHP Statements

When writing PHP, you need to inform the PHP engine that you want it to execute your commands. If you don't do this, the code you write will be mistaken for HTML and will be output to the browser. You can designate your code

Summary

In this hour, you learned how to install and configure PHP for use with Apache on either Linux/Unix or Windows. You learned that various configure options in the Linux/Unix build script can change the features that are supported. You learned about php.ini and how to change the values of its directives. Using the phpinfo() function, you tested your installation and produced a list of its configuration values. You created a simple PHP script using a text editor. You examined four sets of tags that you can use to begin and end blocks of PHP code. Finally, you learned how to use the print() function to send data to the browser, and you brought HTML and PHP together into the same script. In the next hour, you will use these skills to test some of the fundamental building blocks of the PHP language, including variables, data types, and operators.

Q&A

Q1:

You have covered an installation for Linux/Unix or Windows, and the Apache Web server. Does this mean that the material presented in this book will not apply to my server and operating system?

A1:

No, one of PHP's great strengths is that it runs on multiple platforms. You can find installation instructions for different Web servers and configuration directives for database support in the PHP Manual. While the examples throughout this book are specifically geared toward the combination of PHP, MySQL, and Apache, only slight modifications would be needed to work with the examples using different Web servers or databases.

Q2:

Which are the best start and end tags to use?

A2:

It is largely a matter of preference. For the sake of portability, the standard tags (<?php ?>) are probably the safest bet. Short tags are enabled by default and have the virtue of brevity, but with the increasing popularity of XML, it is safest to avoid them.

Q3:

What editors should I avoid when creating PHP code?

A3:

Do not use word processors that format text for printing (such as Word, for example). Even if you save files created using this type of editor in plain text format, hidden characters are likely to creep into your code.

Q4:

When should I comment my code?

A4:

Once again, this is a matter of preference. Some short scripts will be self-explanatory, even after a long interval. For scripts of any length or complexity, you should comment your code. This often saves you time and frustration in the long run.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1:

From a Linux/Unix operating system, how would you get help on configuration options (the options that you pass to the configure script in your PHP distribution)?

A1:

You can get help on configuration options by calling the configure script in the PHP distribution folder and passing it the --help argument:

```
./configure --help
```

2:

What line should you add to the Apache configuration file to ensure that the .php extension is recognized?

A2:

The line

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

ensures that Apache will treat files ending with the .php extension as PHP scripts.

3:

What is PHP's configuration file called?

A3:

PHP's configuration file is called php.ini.

4:

Can a user read the source code of PHP script you have successfully installed?

A4:

No, the user will only see the output of your script. The exception to this is if you have explicitly created a copy of the script with a .phps extension, which will show the color-coded source.

Part II: Basic Language Elements

Hour

Hour 4. The Building Blocks of PHP

In this hour, you will get your hands dirty with some of the nuts and bolts of the PHP scripting language. Those of you new to programming may feel overwhelmed at times, but don't worry—you can always refer back to this hour later on. Concentrate on understanding the concepts rather than memorizing the features covered.

If you're already an experienced programmer, you should at least skim this hour's lesson, as it covers a few PHP-specific features.

In this hour, you will learn

- About variables—what they are, why you need to use them, and how to use them
- How to define and access variables
- About data types
- About some of the more commonly used operators
- How to use operators to create expressions
- How to define and use constants

Variables

A variable is a special container that you can define to "hold" a value. Variables are fundamental to programming. Without variables, we would be forced to hard-code all the values in our scripts. By adding two numbers together and printing the result, you can achieve something useful:

```
print (2 + 4);
```

This script will only be useful for people who want to know the sum of 2 and 4, however. To get past this, you could write a script for finding the sum of another set of numbers, say 3 and 5. However, this approach to programming is clearly absurd, and this is where variables come into play.

Variables allow us to create templates for operations (adding two numbers, for example), without worrying about what values the variables contain. Values will be given to the variables when the script is run, possibly through user input, or through a database query.

You should use a variable whenever the data that you are subjecting to an operation in your script is liable to change from one script execution to another, or even within the lifetime of the script itself.

A variable consists of a name of your choosing, preceded by a dollar sign ($). Variable names can include letters, numbers, and the underscore character (_). They cannot include spaces. They must begin with a letter or an underscore. The following code defines some legal variables:

```
$a;
$a_longish_variable_name;
$2453;
$sleepyZZZZ;
```

Your variable names should be meaningful as well as consistent in style. For example, if your script deals with name and password values, don't create a variable called $n for the name and $p for the password—those are not meaningful names. If you pick up that script weeks later, you might think that $n is the variable for "number" rather than "name" and that $p stands for "page" rather than "password."

A semicolon (;)—also known as the instruction terminator—is used to end a PHP statement. The semicolons in the previous fragment of code are not part of the variable names.

A variable is a holder for a type of data. It can hold numbers, strings of characters, objects, arrays, or Booleans. The contents of a variable can be changed at any time.

As you can see, you have plenty of choices when naming variables. To declare a variable, you need only include it in your script. When you declare a variable, you usually assign a value to it in the same statement, as shown here:

Data Types

Different types of data take up different amounts of memory and may be treated differently when they are manipulated in a script. Some programming languages therefore demand that the programmer declare in advance which type of data a variable will contain. By contrast, PHP is loosely typed, meaning that it will calculate data types as data is assigned to each variable. This is a mixed blessing. On the one hand, it means that variables can be used flexibly, holding a string at one point and an integer at another. On the other hand, this can lead to problems in larger scripts if you expect a variable to hold one data type when in fact it holds something completely different. For example, suppose you have created code that is designed to work with an array variable. If the variable in question instead contains a number value, errors might occur when the code attempts to perform array-specific operations on the variable.

Table 4.1 shows the six standard data types available in PHP.

Table 4.1. Standard Data Types

Type	Example	Description
Integer	5	A whole number
Double	3.234	A floating-point number
String	"hello"	A collection of characters
Boolean	true	One of the special values true or false
Object		An instance of a class
Array		An ordered set of keys and values

PHP also provides two special data types, listed in Table 4.2.

Table 4.2. Special Data Types

Type	Description
Resource	Reference to a third-party resource (a database, for example)

Operators and Expressions

With what you have learned so far, you can assign data to variables. You can even investigate and change the data type of a variable. A programming language isn't very useful, though, unless you can manipulate the data you can store. Operators are symbols that make it possible to use one or more values to produce a new value. A value that is operated on by an operator is referred to as an operand.


An operator is a symbol or series of symbols that, when used in conjunction with values, performs an action and usually produces a new value.

An operand is a value used in conjunction with an operator. There are usually two operands to one operator.

Let's combine two operands with an operator to produce a new value:

```
4 + 5
```

4 and 5 are operands. They are operated on by the addition operator (+) to produce 9. Operators almost always sit between two operands, though you will see a few exceptions later in this hour.

The combination of operands with an operator to produce a result is called an expression. Although most operators form the basis of expressions, an expression need not contain an operator. In fact, in PHP, an expression is defined as anything that can be used as a value. This includes integer constants such as 654, variables such as $user, and function calls such as gettype(). (4 + 5), for example, is an expression that consists of two further expressions and an operator. When an expression produces a value, it is often said to "resolve to" that value. That is, when all subexpressions are taken into account, the expression can be treated as if it were a code for the value itself.


An expression is any combination of functions, values, and operators that resolve to a value. As a rule of thumb, if you can use it as if it were a value, it is an expression.

Now that we have the principles out of the way, it's time to take a tour of PHP's more common operators.

The Assignment Operator

You have seen the assignment operator each time we have initialized a variable. It consists of the single character =. The assignment operator takes the value of its right-hand operand and assigns it to its left-hand operand:

```
$name = "matt";
```

The variable $name now contains the string "matt". Interestingly, this construct is an expression. It might appear at first glance that the assignment operator simply changes the variable $name without producing a value, but in fact, a

Constants

Variables offer a flexible way of storing data. You can change their values and the type of data they store at any time. If, however, you want to work with a value that you do not want to alter throughout your script's execution, you can define a constant. You must use PHP's built-in define() function to create a constant. After you have done this, the constant cannot be changed. To use the define() function, you must place the name of the constant and the value you want to give it within the call's parentheses. These values must be separated by a comma:

```
define( "CONSTANT_NAME", 42);
```

The value you want to set can be a number, a string, or a Boolean. By convention, the name of the constant should be in capital letters. Constants are accessed with the constant name only; no dollar symbol is required. Listing 4.4 defines and accesses a constant.

Listing 4.4 Defining a Constant

```
 1: <html>
 2: <head>
 3: <title>Listing 4.4 Defining a constant</title>
 4: </head>
 5: <body>
 6: <?php
 7: define( "USER", "Gerald");
 8: print "Welcome ".USER;
 9: ?>
10: </body>
11: </html>
```

Notice that in line 8 we used the concatenation operator to append the value held by our constant to the string "Welcome". This is because the PHP engine has no way of distinguishing between a constant and a string within quotation marks.

Put these lines into a text file called constants.php, and place this file in your Web server document root. When you access this script through your Web browser, it produces the following:

```
Welcome Gerald
```

The define() function can accept a third Boolean argument that determines whether or not the constant name should be case-independent. By default, constants are case-dependent. However, by passing true to the define() function, we can change this behavior, so if we were to set up our USER constant as

```
define( "USER", "Gerald", true );
```

we could access its value without worrying about case:

```
print User;
print usEr;
print USER;
```

These expressions are all equivalent. This feature can make scripts a little friendlier for programmers who work with our code, in that they will not need to consider case when accessing a constant that we have defined. On the other

Summary

In this hour, you covered some of the basic features of the PHP language. You learned about variables and how to assign to them using the assignment operator. You got an introduction to operators and learned how to combine some of the most common of these into expressions. Finally, you learned how to define and access constants.

Now that you have mastered some of the fundamentals of PHP, the next hour will really put you in the driver's seat. You will learn how to make scripts that can make decisions and repeat tasks, with help from variables, expressions, and operators.

Q&A

Q1:

Why is it useful to know the type of data a variable holds?

A1:

Often the data type of a variable constrains what you can do with it. You may want to make sure that a variable contains an integer or a double before using it in a mathematical calculation, for example.

Q2:

Should I obey any conventions when naming variables?

A2:

Your goal should always be to make your code easy to read and understand. A variable such as $ab123245 tells you nothing about its role in your script and invites typos. Keep your variable names short and descriptive.

A variable named $f is unlikely to mean much to you when you return to your code after a month or so. A variable named $filename, on the other hand, should make more sense.

Q3:

Should I learn the operator precedence table?

A3:

There is no reason that you shouldn't, but I would save the effort for more useful tasks. By using parentheses in your expressions, you can make your code easy to read while defining your own order of precedence.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1:

Which of the following variable names is not valid?

```
$a_value_submitted_by_a_user
$666666xyz
$xyz666666
$___counter___
$the first
$file-name
```

A1:

The variable name $666666xyz is not valid because it does not begin with a letter or an underscore character. The variable name $the first is not valid because it contains a space. $file-name is also invalid because it contains a nonalphanumeric character.

2:

What will the following code fragment output?

```
$num = 33;
(boolean) $num;
print $num;
```

A2:

The fragment will print the integer 33. The cast to Boolean produces a converted copy of the value stored in $num. It does not alter the value actually stored there.

3:

What will the following statement output?

```
print gettype("4");
```

A3:

The statement will output the string "string".

4:

What will be the output from the following code fragment?

Hour 5. Flow Control Functions in PHP

The scripts created in the last hour flow only in a single direction. The same statements are executed in the same order every time a script is run. This does not allow for much flexibility.

You will now look at some structures that enable your scripts to adapt to circumstances. In this hour, you will learn

- How to use the if statement to execute code if a test expression evaluates to true
- How to execute alternative blocks of code when the test expression of an if statement evaluates to false
- How to use the switch statement to execute code based on the value returned by a test expression
- How to repeat execution of code using a while statement
- How to use for statements to make neater loops
- How to break out of loops
- How to nest one loop within another
- How to use PHP start and end tags within control structures

Switching Flow

Most scripts evaluate conditions and change their behavior accordingly. The capability to make decisions makes your PHP pages dynamic, able to change their output according to circumstances. Like most programming languages, PHP allows you to do this with an if statement.

The if Statement

An if statement is a way of controlling the execution of a statement that follows it (that is, a single statement or a block of code inside braces). The if statement evaluates an expression between parentheses. If this expression results in a true value, the statement is executed. Otherwise, the statement is skipped entirely. This enables scripts to make decisions based on any number of factors:

```
if ( expression ) {
   // code to execute if the expression evaluates to true
}
```

Listing 5.1 executes a block of code only if a variable contains the string "happy".

Listing 5.1 An if Statement

```
 1: <html>
 2: <head>
 3: <title>Listing 5.1</title>
 4: </head>
 5: <body>
 6: <?php
 7: $mood = "happy";
 8: if ( $mood == "happy" ) {
 9:     print "Hooray, I'm in a good mood";
10: }
11: ?>
12: </body>
13: </html>
```

You use the comparison operator == to compare the variable $mood with the string "happy". If they match, the expression evaluates to true, and the code block below the if statement is executed.

Put these lines into a text file called testif.php, and place this file in your Web server document root. When you access this script through your Web browser, it produces the following:

```
Hooray, I'm in a good mood
```

If you change the value of $mood to "sad" and run the script, the expression in the if statement evaluates to false, and the code block is skipped. The script remains silent.

Using the else Clause with the if Statement

When working with the if statement, you will often want to define an alternative block of code that should be executed if the expression you are testing evaluates to false. You can do this by adding else to the if statement

Loops

So far we've looked at decisions that a script can make about what code to execute. Scripts can also decide how many times to execute a block of code. Loop statements are designed to enable you to achieve repetitive tasks. Almost without exception, a loop continues to operate until a condition is achieved, or you explicitly choose to exit the loop.

The while Statement

The while statement looks similar in structure to a basic if statement:

```
while ( expression ) {
      // do something
}
```

As long as a while statement's expression evaluates to true, the code block is executed over and over again. Each execution of the code block in a loop is called an iteration. Within the block, you usually change something that affects the while statement's expression; otherwise, your loop continues indefinitely. Listing 5.6 creates a while loop that calculates and prints multiples of 2 up to 24.

Listing 5.6 A while Statement

```
 1: <html>
 2: <head>
 3: <title>Listing 5.6</title>
 4: </head>
 5: <body>
 6: <?php
 7: $counter = 1;
 8: while ( $counter <= 12 ) {
 9:     print "$counter times 2 is ".($counter*2)."<br>";
10:     $counter++;
11: }
12: ?>
13: </body>
14: </html>
```

In this example, we initialize a variable $counter in line 7. The while statement in line 8 tests the $counter variable. As long as the integer that $counter contains is less than or equal to 12, the loop continues to run. Within the while statement's code block, the value contained by $counter is multiplied by two, and the result is printed to the browser. Then line 10 increments $counter. This last stage is extremely important. If you were to forget to change $counter, the while expression would never resolve to false, and the loop would never end.

Put these lines into a text file called testwhile.php, and place this file in your Web server document root. When you access this script through your Web browser, it produces the following:

```
1 times 2 is 2
2 times 2 is 4
3 times 2 is 6
4 times 2 is 8
5 times 2 is 10
6 times 2 is 12
7 times 2 is 14
8 times 2 is 16
```

Code Blocks and Browser Output

In Hour 3, "Installing and Configuring PHP,"you learned that you can slip in and out of HTML mode at will, using the PHP start and end tags. In this hour, you have discovered that you can present distinct output to the user according to a decision-making process that we can control with if and switch statements. In this section, we will combine these two techniques.

Imagine a script that outputs a table of values only when a variable is set to the Boolean value true. Listing 5.13 shows a simplified HTML table constructed with the code block of an if statement.

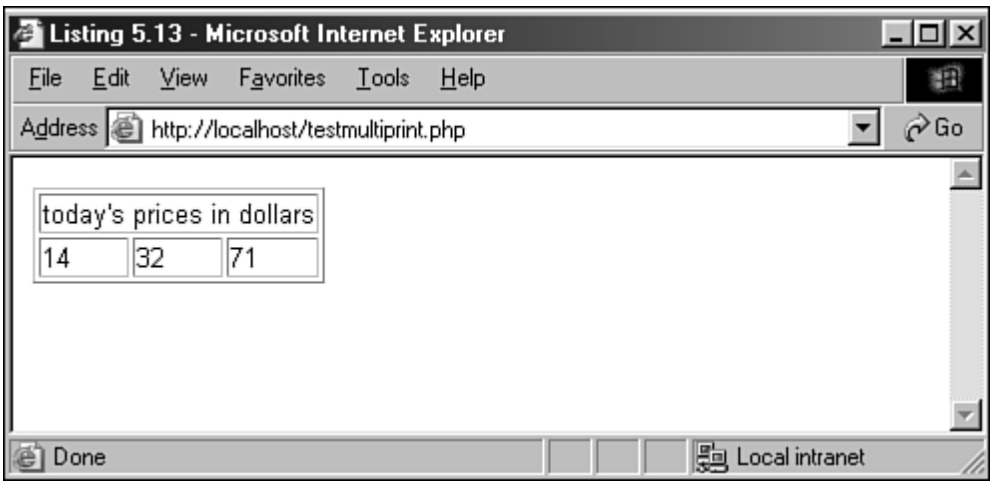
Listing 5.13 A Code Block Containing Multiple print() Statements

```
 1: <html>
 2: <head>
 3: <title>Listing 5.13</title>
 4: </head>
 5: <body>
 6: <?php
 7: $display_prices = true;
 8: if ( $display_prices ) {
 9:     print "<table border=\"1\">";
10:     print "<tr><td colspan=\"3\">";
11:     print "today's prices in dollars";
12:     print "</td></tr>";
13:     print "<tr><td>14</td><td>32</td><td>71</td></tr>";
14:     print "</table>";
15: }
16: ?>
17: </body>
18: </html>
```

If $display_prices is set to true in line 7, the table is printed. For the sake of readability, we split the output into multiple print() statements, and once again escape any quotation marks.

Put these lines into a text file called testmultiprint.php, and place this file in your Web server document root. When you access this script through your Web browser, it should look like Figure 5.2.

Figure 5.2. Output of Listing 5.13.



There's nothing wrong with the way this is coded, but we can save ourselves some typing by simply slipping back into

Summary

In this hour, you learned about control structures and the ways in which they can help to make your scripts flexible and dynamic. Most of these structures will reappear regularly throughout the rest of the book.

You learned how to define an if statement and how to provide for alternative actions with the elseif and else clauses. You learned how to use the switch statement to change flow according to multiple equivalence tests on the result of an expression. You learned about loops—in particular, the while and for statements—and you learned how to use break and continue to prematurely end the execution of a loop or to skip an iteration. You learned how to nest one loop within another and saw a typical use for this structure. Finally, you looked at a technique for using PHP start and end tags in conjunction with conditional code blocks.

You should now have enough information to write scripts of your own. These scripts can make decisions and perform repetitive tasks. In the next hour, we will be looking at a way of adding even more power to your applications. You will learn how functions enable you to organize your code, preventing duplication and improving reusability.

Q&A

Q1:

Must a control structure's test expression result in a Boolean value?

A1:

Ultimately, yes, but in the context of a test expression, zero, an undefined variable, or an empty string will be converted to false. All other values will evaluate to true.

Q2:

Must I always surround a code block in a control statement with brackets?

A2:

If the code you want executed as part of a control structure consists of only a single line, you can omit the brackets.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1:

How would you use an if statement to print the string "Youth message" to the browser if an integer variable, $age, is between 18 and 35? If $age contains any other value, the string "Generic message" should be printed to the browser.

A1:

```
$age = 22;

if ( $age >= 18 && $age <= 35 ) {
    print "Youth message<BR>\n";
} else {
    print "Generic message<BR>\n";
}
```

2:

How would you extend your code in question 1 to print the string "Child message" if the $age variable is between 1 and 17?

A2:

```
$age = 12;

if ( $age >= 18 && $age <= 35 ) {
    print "Youth message<BR>\n";
} elseif ( $age >= 1 && $age <= 17 ) {
    print "Child message<BR>\n";
} else {
    print "Generic message<BR>\n";
}
```

3:

How would you create a while statement that increments through and prints every odd number between 1 and 49?

A3:

```
$num = 1;

while ( $num <= 49 ) {
    print "$num<BR>\n";
    $num += 2;
}
```

Hour 6. Working with Functions

Functions are at the heart of a well-organized script, making code easy to read and reuse. No large project would be manageable without them. Throughout this hour, we will investigate functions and demonstrate some of the ways in which they can save you from repetitive work. In this hour, you will learn

- How to define and call functions
- How to pass values to functions and receive values in return
- How to call a function dynamically using a string stored in a variable
- How to access global variables from within a function
- How to give a function a "memory"
- How to pass data to functions by reference
- How to create anonymous functions
- How to verify that a function exists before calling it

What Is a Function?

You can think of a function as a machine. A machine takes the raw materials you feed it and works with them to achieve a purpose or to produce a product. A function accepts values from you, processes them, and then performs an action (printing to the browser, for example), returns a new value, or both.

If you needed to bake a single cake, you would probably do it yourself. If you needed to bake thousands of cakes, you would probably build or acquire a cake-baking machine. Similarly, when deciding whether to create a function, the most important factor to consider is the extent to which it can save you from repetition.

A function is a self-contained block of code that can be called by your scripts. When called, the function's code is executed. You can pass values to functions, which will then work with them. When finished, a function can pass a value back to the calling code.

Calling Functions

Functions come in two flavors—those built in to the language and those you define yourself. PHP has hundreds of built-in functions. The very first script in this book, which appears in Hour 3, "Installing and Configuring PHP," consists of a single function call:

```
print "Hello Web!";
```

In this example, we call the print() function, passing it the string "Hello Web!". The function then goes about the business of writing the string. A function call consists of the function name (print in this case) followed by parentheses. If you want to pass information to the function, you place it between these parentheses. A piece of information passed to a function in this way is called an argument. Some functions require that more than one argument be passed to them. Arguments in such cases must be separated by commas:

```
some_function( $an_argument, $another_argument);
```

print() is typical in that it returns a value. Most functions give you some information back when they've completed their task—they usually at least tell whether their mission was successful. print() returns a Boolean.

The abs() function, for example, requires a signed numeric value and returns the absolute value of that number. Let's try it out in Listing 6.1.

print() is not a typical function in that it does not require parentheses in order to run successfully:

```
print("Hello Web!");
```

and

```
print "Hello Web!";
```

are equally valid. This is an exception. All other functions require parentheses, whether or not they accept arguments.

Listing 6.1 Calling the Built-in abs() Function

```
 1: <html>
 2: <head>
 3: <title>Listing 6.1</title>
 4: </head>
 5: <body>
 6: <?php
 7: $num = -321;
 8: $newnum = abs( $num );
 9: print $newnum;
10: // prints "321"
11: ?>
12: </body>
```

Defining a Function

You can define a function using the function statement:

```
function some_function( $argument1, $argument2 ) {
     // function code here
}
```

The name of the function follows the function statement and precedes a set of parentheses. If your function requires arguments, you must place comma-separated variable names within the parentheses. These variables will be filled by the values passed to your function. Even if your function doesn't require arguments, you must nevertheless supply the parentheses.

The naming rules for functions are similar to the naming rules for variables, which you learned in Hour 4, "The Building Blocks of PHP." Names cannot include spaces, and they must begin with a letter or an underscore.

Listing 6.2 declares a function.

Listing 6.2 Declaring a Function

```
 1: <html>
 2: <head>
 3: <title>Listing 6.2</title>
 4: </head>
 5: <body>
 6: <?php
 7: function bighello() {
 8:      print "<h1>HELLO!</h1>";
 9: }
10: bighello();
11: ?>
12: </body>
13: </html>
```

The script in Listing 6.2 simply outputs the string "HELLO" wrapped in an HTML <h1> element.

Put these lines into a text file called bighello.php, and place this file in your Web server document root. When you access this script through your Web browser, it should look like Figure 6.1.

Figure 6.1. Output of Listing 6.2.

Returning Values from User-Defined Functions

In the previous example, we output an amended string to the browser within the printBR() function. Sometimes, however, you will want a function to provide you with a value that you can work with yourself. If your function has transformed a string that you have provided, you may wish to get the amended string back so that you can pass it to other functions. A function can return a value using the return statement in conjunction with a value. The return statement stops the execution of the function and sends the value back to the calling code.

Listing 6.4 creates a function that returns the sum of two numbers.

Listing 6.4 A Function That Returns a Value

```
 1: <html>
 2: <head>
 3: <title>Listing 6.4</title>
 4: </head>
 5: <body>
 6: <?php
 7: function addNums( $firstnum, $secondnum ) {
 8:     $result = $firstnum + $secondnum;
 9:     return $result;
10: }
11: print addNums(3,5);
12: // will print "8"
13: ?>
14: </body>
15: </html>
```

Put these lines into a text file called addnums.php, and place this file in your Web server document root. When you access this script through your Web browser, it produces the following:

```
8
```

Notice in line 7 that addNums() should be called with two numeric arguments (line 11 shows those to be 3 and 5 in this case). These are stored in the variables $firstnum and $secondnum. Predictably, addNums() adds the numbers contained in these variables together and stores the result in a variable called $result.

The return statement can return a value or nothing at all. How we arrive at a value passed by return can vary. The value can be hard-coded:

```
return 4;
```

It can be the result of an expression:

```
return ( $a/$b );
```

It can be the value returned by yet another function call:

```
return ( another_function( $an_argument ) );
```

Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function names themselves. Listing 6.5 shows a simple example of this.

Listing 6.5 Calling a Function Dynamically

```
 1: <html>
 2: <head>
 3: <title>Listing 6.5</title>
 4: </head>
 5: <body>
 6: <?php
 7: function sayHello() {
 8:     print "hello<br>";
 9: }
10: $function_holder = "sayHello";
11: $function_holder();
12: ?>
13: </body>
14: </html>
```

A string identical to the name of the sayHello() function is assigned to the $function_holder variable on line 10. Once this is done, we can use this variable in conjunction with parentheses to call the sayHello() function. We do this on line 11.

Put these lines into a text file called sayhello.php, and place this file in your Web server document root. When you access this script through your Web browser, it produces the following:

```
hello
```

Why would we want to do this? In the example, we simply make more work for ourselves by assigning the string "sayHello" to $function_holder. Dynamic function calls are useful when you want to alter program flow according to changing circumstances. We might want our script to behave differently according to a parameter set in a URL's query string, for example. We can extract the value of this parameter and use it to call one of a number of functions.

Variable Scope

A variable declared within a function remains local to that function. In other words, it will not be available outside the function or within other functions. In larger projects, this can save you from accidentally overwriting the contents of a variable when you declare two variables with the same name in separate functions.

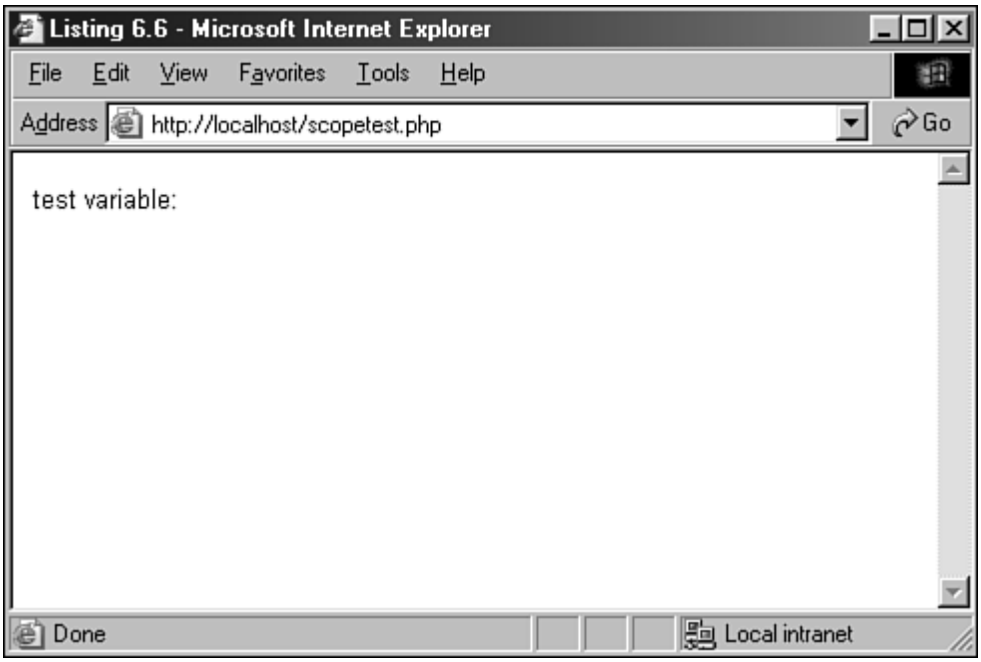
Listing 6.6 creates a variable within a function and then attempts to print it outside the function.

Listing 6.6 Variable Scope: A Variable Declared Within a Function Is Unavailable Outside the Function

```
 1: <html>
 2: <head>
 3: <title>Listing 6.6</title>
 4: </head>
 5: <body>
 6: <?php
 7: function test() {
 8:      $testvariable = "this is a test variable";
 9: }
10: print "test variable: $testvariable<br>";
11: ?>
12: </body>
13: </html>
```

Put these lines into a text file called scopetest.php, and place this file in your Web server document root. When you access this script through your Web browser, it should look like Figure 6.3.

Figure 6.3. Output of Listing 6.6.



The value of the variable $testvariable is not printed. This is because no such variable exists outside the test() function. Note that the attempt in line 10 to access a nonexistent variable does not cause an error.

Similarly, a variable declared outside a function will not automatically be available within it.

Saving State Between Function Calls with the static Statement

Variables within functions have a short but happy life on the whole. They come into being when the function is called and die when execution is finished. Once again, this is as it should be. It is usually best to build a script as a series of self-contained blocks, each with as little knowledge of others as possible. Occasionally, however, you may want to give a function a rudimentary memory.

Let's assume that we want a function to keep track of the number of times it has been called. Why? In our examples, the function is designed to create numbered headings in a script that dynamically builds online documentation.

We could, of course, use the global statement to do this. We have a crack at this in Listing 6.9.

Listing 6.9 Using the global Statement to Remember the Value of a Variable Between Function Calls

```
 1: <html>
 2: <head>
 3: <title>Listing 6.9</title>
 4: </head>
 5: <body>
 6: <?php
 7: $num_of_calls = 0;
 8: function numberedHeading( $txt ) {
 9:     global $num_of_calls;
10:     $num_of_calls++;
11:     print "<h1>$num_of_calls. $txt</h1>";
12: }
13: numberedHeading("Widgets");
14: print("We build a fine range of widgets<p>");
15: numberedHeading("Doodads");
16: print("Finest in the world<p>");
17: ?>
18: </body>
19: </html>
```

Put these lines into a text file called numberedheading.php, and place this file in your Web server document root. When you access this script through your Web browser, it should look like Figure 6.6.

Figure 6.6. Using the global statement to keep track of the number of times a function has been called.

More About Arguments

You've already seen how to pass arguments to functions, but there's more to cover yet. In this section, you'll look at a technique for giving your arguments default values and explore a method of passing variables by reference rather than by value. This means that the function is given an "alias" of the original value rather than a copy of it.

Setting Default Values for Arguments

PHP gives you a nifty feature to help build flexible functions. Until now, we've said that some functions "demand" one or more arguments. By making some arguments optional, you can render your functions a little less autocratic.

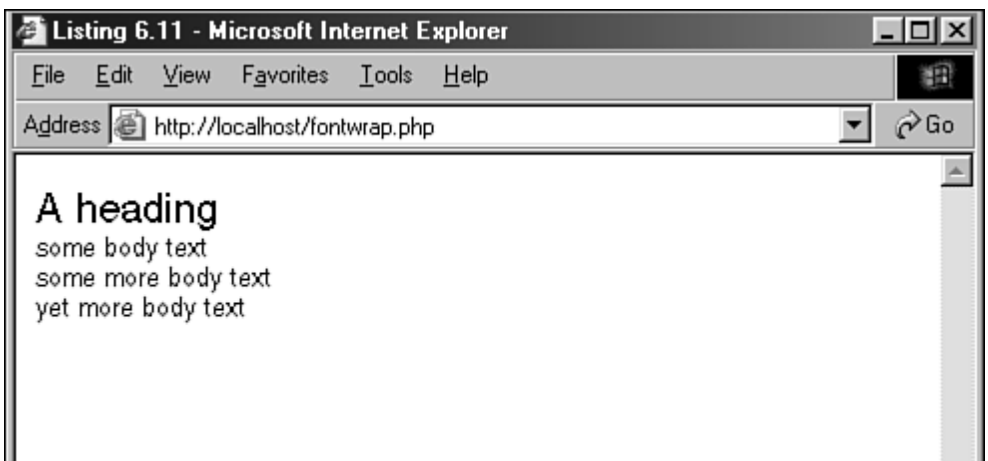
Listing 6.11 creates a useful little function that wraps a string in an HTML font element. We want to give the user of the function the chance to change the font element's size attribute, so we demand a $size argument in addition to the string (line 7).

Listing 6.11 A Function Requiring Two Arguments

```
 1: <html>
 2: <head>
 3: <title>Listing 6.11</title>
 4: </head>
 5: <body>
 6: <?php
 7: function fontWrap( $txt, $size ) {
 8:     print "<font size=\"$size\"
 9:         face=\"Helvetica,Arial,Sans-Serif\">
10:         $txt</font>";
11: }
12: fontWrap("A heading<br>",5);
13: fontWrap("some body text<br>",3);
14: fontWrap("some more body text<BR>",3);
15: fontWrap("yet more body text<BR>",3);
16: ?>
17: </body>
18: </html>
```

Put these lines into a text file called fontwrap.php, and place this file in your Web server document root. When you access this script through your Web browser, it should look like Figure 6.7.

Figure 6.7. A function that formats and outputs strings.



Creating Anonymous Functions

It is possible to create functions on the fly during script execution. Because such functions are not themselves given a name, but are stored in variables or passed to other functions, they are known as anonymous functions. PHP provides the create_function() function for creating anonymous functions. create_function() requires two string arguments. The first argument should contain a comma-delimited list of argument variables, exactly the same as the argument variables you would include in a standard function declaration. The second argument should contain the function body.

Listing 6.15 creates a simple anonymous function to add two numbers together.

Listing 6.15 A Simple Anonymous Function

```
 1: <html>
 2: <head>
 3: <title>Listing 6.15</title>
 4: </head>
 5: <body>
 6: <?php
 7: $my_anon = create_function( '$a, $b', 'return $a+$b;' );
 8: print $my_anon( 3, 9 );
 9: // prints 12
10: ?>
11: </body>
12: </html>
```


As of this writing, the use of anonymous functions will cause a segmentation fault when running the Zend Optimizer.

Put these lines into a text file called anon.php, and place this file in your Web server document root. When you access this script through your Web browser, it produces the following:

```
12
```

Note that we use single quotes when passing arguments to create_function(). That saves us from having to escape the variable names within the arguments. We could have used double quotes, but the function call would have been a little more involved:

```
$my_anon = create_function("\$a, \$b", "return \$a+\$b;");
```

So what is the use of anonymous functions? In practical terms, you will probably only use them when you need to pass callback functions to built-in functions. A callback function is generally written by the user and is designed to be invoked (usually repeatedly) by the function to which it is passed.


The second argument to create_function() is the function body. Don't forget to end the last statement in this string with a semicolon. The interpreter will complain and your anonymous function will not be executed if you omit it.

Testing for the Existence of a Function

You have seen that we do not always know that a function exists before we try to invoke it. If our code were to work with a function name stored in a variable, for example, it would be useful to be able to test whether or not the function exists before we attempt to call it. Furthermore, different builds of the PHP engine may include different functionality. If you are writing a script that may be run on multiple servers, you might want to verify that key features are available. You might write code that will use MySQL if MySQL-related functions are available, but simply log data to a text file otherwise.

You can use function_exists() to check for the availability of a function. function_exists() requires a string representing a function name. It will return true if the function can be located and false otherwise.

Listing 6.16 shows function_exists() in action, and illustrates some of the other topics we have covered in this hour.

Listing 6.16 Testing for a Function's Existence

```
 1: <html>
 2: <head>
 3: <title>Listing 6.16</title>
 4: </head>
 5: <body>
 6: <?php
 7:
 8: function tagWrap( $tag, $txt, $func="" ) {
 9:     if ( ! empty( $txt ) && function_exists( $func ) ) {
10:         $txt = $func( $txt );
11:         return "<$tag>$txt</$tag>\n";
12:     }
13:  }
14:
15: function underline( $txt ) {
16:     return "<u>$txt</u>";
17: }
18:
19: print tagWrap('b', 'make me bold');
20: // <b>make me bold</b>
21:
22: print tagWrap('i', 'underline me too', "underline");
23: // <i><u>underline me too</u></i>
24:
25: print tagWrap('i', 'make me italic and quote me',
26:     create_function('$txt', 'return "&quot;$txt&quot;";'));
27: // <i>&quot;make me italic and quote me&quot;</i>
28:
29: ?>
30: </body>
31: </html>
```

We define two functions, tagWrap() (line 8) and underline() (line 15). The tagWrap() function accepts three strings: a tag, the text to be formatted, and an optional function name. It returns a formatted string. underline() requires a single argument—the text to be formatted—and returns the text wrapped in <u> tags.

When we first call tagWrap() on line 19, we pass it the character b and the string make me bold. Because we haven't passed a value for the function argument, the default value (an empty string) is used. On line 9, we check whether the $func variable contains characters and, if it is not empty, we call function_exists() to check for a function by that

Summary

This hour taught you about functions and how to deploy them. You learned how to define and pass arguments to a function, how to use the global and static statements, how to pass references to functions, and how to create default values for function arguments. Finally, you learned how to create anonymous functions and test for the existence of functions.

Q&A

Q1:

Can you include a function call within a double- or single-quoted string, as you can with a variable?

A1:

No. You must call functions outside quotation marks. However, you can break the string apart and place the function call between the parts of the string, using the concatenation operator to tie them together. For example:

```
$newstring = "I
purchased".numPurchase($somenum)."
items.";
```

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1:

True or false: If a function doesn't require an argument, you can omit the parentheses in the function call.

A1:

The statement is false. You must always include the parentheses in your function calls, whether you are passing arguments to the function or not.

2:

How do you return a value from a function?

A2:

You must use the return keyword.

3:

What would the following code fragment print to the browser?

```
$number = 50;

function tenTimes() {
    $number = $number * 10;
}

tenTimes();
print $number;
```

A3:

It would print 50. The tenTimes() function has no access to the global $number variable. When it is called, it will manipulate its own local $number variable.

4:

What would the following code fragment print to the browser?

```
$number = 50;

function tenTimes() {
```

Hour 7. Learning Basic SQL Commands

This hour takes a break from all that PHP you've been learning and provides a primer on SQL syntax, which you will use to create and manipulate your MySQL database tables. This is a very hands-on hour, and it assumes that you are able to issue queries through the MySQL monitor on Windows or Linux/Unix. Alternatively, if you use a GUI to MySQL, this hour assumes you know the methods for issuing queries through those interfaces.

In this hour, you will learn

- The basic MySQL data types
- How to use the CREATE TABLE command to create a table
- How to use the INSERT command to enter records
- How to use the SELECT command to retrieve records
- How to use basic functions, the WHERE clause, and the GROUP BY clause in SELECT expressions
- How to select from multiple tables, using JOIN
- How to use the UPDATE and REPLACE commands to modify existing records
- How to use the DELETE command to remove records

Learning the MySQL Data Types

Properly defining the fields in a table is important to the overall optimization of your database. You should use only the type and size of field you really need to use. These types of fields (or columns) are also referred to as data types because it's the type of data you will be storing in those fields.

MySQL uses many different data types, which are broken into three categories: numeric, date and time, and string types. Pay close attention because defining the data type is more important than any other part of the table creation process.

Numeric Data Types

MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you. The following list shows the common numeric data types and their descriptions.


The terms signed and unsigned will be used in the list of numeric data types. If you remember your basic algebra, you'll recall that a signed integer is a positive or negative integer, whereas an unsigned integer is a non-negative integer.

- INT— A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from –2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.


INT and INTEGER are synonymous. If it helps you to remember the data type by using INTEGER instead of INT, go for it.

- TINYINT— A very small integer that can be signed or unsigned. If signed, the allowable range is from –128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- SMALLINT— A small integer that can be signed or unsigned. If signed, the allowable range is from –32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- MEDIUMINT— A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from –8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
-

Learning the Table Creation Syntax

The table creation command requires

- Name of the table
- Names of fields
- Definitions for each field

The generic table creation syntax is

```
CREATE TABLE table_name (column_name column_type);
```

The table name is up to you of course, but should be a name that reflects the usage of the table. For example, if you have a table that holds the inventory of a grocery store, you wouldn't name the table s. You would probably name it something like grocery_inventory. Similarly, the field names you select should be as concise as possible and relevant to the function they serve and data they hold. For example, you might call a field holding the name of an item item_name, not n.

This example creates a generic grocery_inventory table with fields for ID, name, description, price, and quantity:

```
mysql> CREATE TABLE grocery_inventory (
    -> id int not null primary key auto_increment,
    -> item_name varchar (50) not null,
    -> item_desc text,
    -> item_price float not null,
    -> curr_qty int not null
    -> );
Query OK, 0 rows affected (0.02 sec)
```

The id field is defined as a primary key. You will learn more about keys in later hours, in the context of creating specific tables as parts of sample applications. By using auto_increment as an attribute of the field, you are telling MySQL to go ahead and add the next available number to the id field for you.

The MySQL server will respond with Query OK each time a query, regardless of type, is successful. Otherwise, an error message will be displayed.

Using the INSERT Command

After you have created some tables, you'll use the SQL command INSERT for adding new records to these tables. The basic syntax of INSERT is

```
INSERT INTO table_name (column list) VALUES (column values);
```

Within the parenthetical list of values, you must enclose strings within quotation marks. The SQL standard is single quotes, but MySQL enables the usage of either single or double quotes. Remember to escape the type of quotation mark used, if it's within the string itself.

 Integers do not require quotation marks around them.

Here is an example of a string where escaping is necessary:

```
O'Connor said "Boo"
```

If you enclose your strings in double quotes, the INSERT statement would look like this:

```
INSERT INTO table_name (column_name) VALUES ("O'Connor said \"Boo\"");
```

If you enclose your strings in single quotes instead, the INSERT statement would look like this:

```
INSERT INTO table_name (column_name) VALUES ('O\'Connor said "Boo"');
```

A Closer Look at INSERT

Besides the table name, there are two main parts of the INSERT statement—the column list and the value list. Only the value list is actually required, but if you omit the column list, you must specifically name each column in your values list in order.

Using the grocery_inventory table as an example, you have five fields: id, item_name, item_desc, item_price, and curr_qty. To insert a complete record, you could use either of these statements:

1. A statement with all columns named:

```
insert into grocery_inventory (id, item_name, item_desc, item_price,
curr_qty) values ('1', 'Apples', 'Beautiful, ripe apples.', '0.25', 1000);
```

2. A statement that uses all columns but does not explicitly name them:

```
insert into grocery_inventory values ('2', 'Bunches of Grapes',
```

Using the SELECT Command

SELECT is the SQL command used to retrieve records. This command syntax can be totally simplistic or very complicated. As you become more comfortable with database programming, you will learn to enhance your SELECT statements, ultimately making your database do as much work as possible and not overworking your programming language of choice.

The most basic SELECT syntax looks like this:

```
SELECT expressions_and_columns FROM table_name
[WHERE some_condition_is_true]
[ORDER BY some_column [ASC | DESC]]
[LIMIT offset, rows]
```

Start with the first line:

```
SELECT expressions_and_columns FROM table_name
```

One handy expression is the * symbol, which stands for "everything." So, to select "everything" (all rows, all columns) from the grocery_inventory table, your SQL statement would be

```
SELECT * FROM grocery_inventory;
```

Depending on how much data you inserted into the grocery_inventory table during the previous hour, your results will vary, but it might look something like this:

```
mysql> select * from grocery_inventory;
+---+----------------------+-----------------------+----------+---------+
| id| item_name            | item_desc             | item_price| curr_qty|
+---+----------------------+-----------------------+----------+---------+
|  1| Apples               | Beautiful, ripe apples.|      0.25|     1000|
|  2| Bunches of Grapes    | Seedless grapes.      |      2.99|      500|
|  3| Bottled Water (6-pack)| 500ml spring water.  |      2.29|      250|
+---+----------------------+-----------------------+----------+---------+
3 rows in set (0.00 sec)
```

As you can see, MySQL creates a lovely table with the names of the columns along the first row as part of the result set. If you only want to select specific columns, replace the * with the names of the columns, separated by commas. The following statement selects just the id, item_name, and curr_qty fields from the grocery_inventory table.

```
mysql> select id, item_name, curr_qty from grocery_inventory;
+----+----------------------+----------+
| id | item_name            | curr_qty |
+----+----------------------+----------+
|  1 | Apples               |     1000 |
|  2 | Bunches of Grapes    |      500 |
|  3 | Bottled Water (6-pack) |      250 |
+----+----------------------+----------+
3 rows in set (0.00 sec)
```

Ordering SELECT Results

Using WHERE in Your Queries

You have learned numerous ways to retrieve particular columns from your tables, but not specific rows. This is when the WHERE clause comes in to play. From the basic SELECT syntax, you see that WHERE is used to specify a particular condition:

```
SELECT expressions_and_columns FROM table_name
[WHERE some_condition_is_true]
```

An example would be to retrieve all the records for items with a quantity of 500:

```
mysql> select * from grocery_inventory where curr_qty = 500;
+----+------------------+-----------------------+------------+----------+
| id | item_name        | item_desc             | item_price | curr_qty |
+----+------------------+-----------------------+------------+----------+
|  2 | Bunches of Grapes | Seedless grapes.      |       2.99 |      500 |
|  5 | Pears            | Anjou, nice and sweet. |        0.5 |      500 |
+----+------------------+-----------------------+------------+----------+
2 rows in set (0.00 sec)
```

As shown previously, if you use an integer as the WHERE clause comes in to play. From part of your WHERE clause, quotation marks are not required. Quotation marks are required around strings, and the same rules apply with regard to escaping characters, as you learned in the section on INSERT.

Using Operators in WHERE Clauses

You've used the equal sign (=) in your WHERE clauses to determine the truth of a condition—is one thing equal to another. You can use many types of operators, with comparison operators and logical operators being the most popular types.

Comparison operators, shown in Table 7.1, should look familiar to you if you think about the first day of algebra class.

Table 7.1. Basic Comparison Operators and Their Meanings

Operator	Meaning
=	Equal to
!=	Not equal to
<=	Less than or equal to
<	Less than

Selecting from Multiple Tables

You're not limited to selecting only one table at a time. That would certainly make application programming a long and tedious task! When you select from more than one table in one SELECT statement, you are said to be joining the tables together.

Suppose you have two tables, fruit and color. You can select all rows from each of the two tables, using two separate SELECT statements:

```
mysql> select * from fruit;
+----+-----------+
| id | fruitname |
+----+-----------+
|  1 | apple     |
|  2 | orange    |
|  3 | grape     |
|  4 | banana    |
+----+-----------+
4 rows in set (0.00 sec)

mysql> select * from color;
+----+-----------+
| id | colorname |
+----+-----------+
|  1 | red       |
|  2 | orange    |
|  3 | purple    |
|  4 | yellow    |
+----+-----------+
4 rows in set (0.00 sec)
```

When you want to select from both tables at once, there are a few differences in the syntax of the SELECT statement. First, you must ensure that all the tables you're using in your query appear in the FROM clause of the SELECT statement. Using the fruit and color example, if you simply want to select all columns and rows from both tables, you might think you would use the following SELECT statement:

```
mysql> select * from fruit, color;
+----+-----------+----+-----------+
| id | fruitname | id | colorname |
+----+-----------+----+-----------+
|  1 | apple     |  1 | red       |
|  2 | orange    |  1 | red       |
|  3 | grape     |  1 | red       |
|  4 | banana    |  1 | red       |
|  1 | apple     |  2 | orange    |
|  2 | orange    |  2 | orange    |
|  3 | grape     |  2 | orange    |
|  4 | banana    |  2 | orange    |
|  1 | apple     |  3 | purple    |
|  2 | orange    |  3 | purple    |
|  3 | grape     |  3 | purple    |
|  4 | banana    |  3 | purple    |
|  1 | apple     |  4 | yellow    |
|  2 | orange    |  4 | yellow    |
|  3 | grape     |  4 | yellow    |
|  4 | banana    |  4 | yellow    |
+----+-----------+----+-----------+
16 rows in set (0.00 sec)
```

Using JOIN

Several types of JOINs can be used in MySQL, all of which refer to the order in which the tables are put together and the results are displayed. The type of JOIN used with the fruit and color tables is called an INNER JOIN, although it wasn't written explicitly as such. To rewrite the SQL statement using the proper INNER JOIN syntax, you would use

```
mysql> select fruitname, colorname from fruit inner join color
    -> on fruit.id = color.id;
+-----------+-----------+
| fruitname | colorname |
+-----------+-----------+
| apple     | red       |
| orange    | orange    |
| grape     | purple    |
| banana    | yellow    |
+-----------+-----------+
4 rows in set (0.00 sec)
```

The ON clause replaced the WHERE clause, in this instance telling MySQL to join together the rows in the tables where the IDs match each other. When joining tables using ON clauses, you can use any conditions that you would use in a WHERE clause, including all the various logical and arithmetic operators.

Another common type of JOIN is the LEFT JOIN. When joining two tables with LEFT JOIN, all rows from the first table will be returned, no matter if there are matches in the second table or not. Suppose you have two tables in an address book, one called master_name, containing basic records, and one called email, containing email records. Any records in the email table would be tied to a particular id of a record in the master_name table. For example

```
mysql> select name_id, firstname, lastname from master_name;
+---------+-----------+----------+
| name_id | firstname | lastname |
+---------+-----------+----------+
|       1 | John      | Smith    |
|       2 | Jane      | Smith    |
|       3 | Jimbo     | Jones    |
|       4 | Andy      | Smith    |
|       7 | Chris     | Jones    |
|      45 | Anna      | Bell     |
|      44 | Jimmy     | Carr     |
|      43 | Albert    | Smith    |
|      42 | John      | Doe      |
+---------+-----------+----------+
9 rows in set (0.00 sec)
mysql> select name_id, email from email;
+---------+------------------+
| name_id | email            |
+---------+------------------+
|      42 | jdoe@yahoo.com   |
|      45 | annabell@aol.com |
+---------+------------------+
2 rows in set (0.00 sec)
```

Using LEFT JOIN on these two tables, you can see that if a value from the email table doesn't exist, NULL will appear in place of an email address:

```
mysql> select firstname, lastname, email fom master_name left join email
    -> on master_name.name_id = email.name_id;
```

Using the UPDATE Command to Modify Records

UPDATE is the SQL command used to modify the contents of one or more columns in an existing record. The most basic UPDATE syntax looks like this:

```
UPDATE table_name
SET column1='new value',
column2='new value2'
[WHERE some_condition_is_true]
```

The guidelines for updating a record are similar to those used when inserting a record—the data you're entering must be appropriate to the data type of the field, and you must enclose your strings in single or double quotes, escaping where necessary.

For example, assume you have a table called fruit containing an ID, a fruit name, and the status of the fruit (ripe or rotten):

```
mysql> SELECT * FROM fruit;
+----+------------+--------+
| id | fruit_name | status |
+----+------------+--------+
|  1 | apple      | ripe   |
|  2 | pear       | rotten |
|  3 | banana     | ripe   |
|  4 | grape      | rotten |
+----+------------+--------+
4 rows in set (0.00 sec)
```

To update the status of the fruit to "ripe", use

```
mysql> update fruit set status = 'ripe';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 4 Changed: 2 Warnings: 0
```

Take a look at the result of the query. It was successful, as you can tell from the Query OK message. Also note that only 2 rows were affected—if you try to set the value of a column to the value it already is, the update won't occur for that column.

The second line of the response shows that 4 rows were matched, and only 2 were changed. If you're wondering "matched what?" the answer is simple—because you did not specify a particular condition for matching, the match would be "all rows".

You must be very careful and use a condition when updating a table, unless you really intend to change all the columns for all records to the same value. For the sake of argument, assume that "grape" is spelled incorrectly in the table, and you want to use UPDATE to correct this mistake. This query would have horrible results:

```
mysql> update fruit set fruit_name = 'grape';
Query OK, 4 rows affected (0.00 sec)
Rows matched: 4 Changed: 4 Warnings: 0
```

When you read the result, you should be filled with dread: 4 of 4 records were changed, meaning your fruit table now

Using the REPLACE Command

Another method for modifying records is to use the REPLACE command, which is remarkably similar to the INSERT command.

```
REPLACE INTO table_name (column list) VALUES (column values);
```

The REPLACE statement works like this: if the record you are inserting into the table contains a primary key value that matches a record already in the table, the record in the table will be deleted and the new record inserted in its place.

The REPLACE command is a MySQL-specific extension to ANSI SQL. This command mimics the action of a DELETE and re-INSERT of a particular record. In other words, you get two commands for the price of one.

Using the grocery_inventory table, the following command will replace the entry for Apples:

```
mysql> replace into grocery_inventory values
    -> (1, 'Granny Smith Apples', 'Sweet!', '0.50', 1000);
Query OK, 2 rows affected (0.00 sec)
```

In the query result, notice that the result states, "2 rows affected". In this case, because id is a primary key that had a matching value in the grocery_inventory table, the original row was deleted and the new row inserted—2 rows affected.

Select the records to verify that the entry is correct, which it is

```
mysql> select * from grocery_inventory;
+----+----------------------+----------------------+-----------+-----------+
| id | item_name            | item_desc            | item_price | curr_qty |
+----+----------------------+----------------------+-----------+-----------+
|  1 | Granny Smith Apples  | Sweet!               |       0.5 |     1000 |
|  2 | Bunches of Grapes    | Seedless grapes.     |      2.99 |      500 |
|  3 | Bottled Water (6-pack)| 500ml spring water. |      2.29 |      250 |
|  4 | Bananas              | Bunches, green.      |      1.99 |      150 |
|  5 | Pears                | Anjou, nice and sweet.|       0.5 |      500 |
|  6 | Avocado              | Large Haas variety.  |      0.99 |      750 |
+----+----------------------+----------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

If you use a REPLACE statement, and the value of the primary key in the new record does not match a value for a primary key already in the table, the record would simply be inserted and only one row would be affected.

Using the DELETE Command

The basic DELETE syntax is

```
DELETE FROM table_name
[WHERE some_condition_is_true]
[LIMIT rows]
```

Notice there is no column specification in the delete command—when you use DELETE, the entire record is removed. You might recall the fiasco earlier in this hour, regarding grapes in the fruit table, when updating a table without specifying a condition caused all records to be updated. You must be similarly careful when using DELETE.

Assuming the structure and data in a table called fruit:

```
mysql> select * from fruit;
+----+------------+--------+
| id | fruit_name | status |
+----+------------+--------+
|  1 | apple      | ripe   |
|  2 | pear       | rotten |
|  3 | banana     | ripe   |
|  4 | grape      | rotten |
+----+------------+--------+
4 rows in set (0.00 sec)
```

This statement will remove all records in the table:

```
mysql> delete from fruit;
Query OK, 0 rows affected (0.00 sec)
```

You can always verify the deletion by attempting to SELECT data from the table:

```
mysql> select * from fruit;
Empty set (0.00 sec)
```

All your fruit is gone.

Conditional DELETE

A conditional DELETE statement, just like a conditional SELECT or UPDATE statement, means you are using WHERE clauses to match specific records. You have the full range of comparison and logical operators available to you, so you can pick and choose which records you want to delete.

A prime example would be to remove all records for rotten fruit from the fruit table:

```
mysql> delete from fruit where status = 'rotten';
Query OK, 2 rows affected (0.00 sec)
```

Two records were deleted, and only ripe fruit remains:

Summary

In this hour, you learned the basics of SQL, from table creation to manipulating records. The table creation command requires three important pieces of information—the table name, the field name, and the field definitions. Field definitions are important because a well-designed table will help speed along your database. MySQL has three different categories of data types: numeric, date and time, and string.

The INSERT command, used to add records to a table, names the table and columns you want to populate, and then defines the values. When placing values in the INSERT statement, strings must be enclosed with single or double quotes. The SELECT SQL command is used to retrieve records from specific tables. The * character enables you to easily select all fields for all records in a table, but you can also specify particular column names. If the result set is too long, the LIMIT clause provides a simple method for extracting slices of results if you indicate a starting position and the number of records to return. To order the results, use the ORDER BY clause to select the columns to sort. Sorts can be performed on integers, dates, and strings, in either ascending or descending order. The default order is ascending. Without specifying an order, results are displayed in the order they appear in the table.

You can pick and choose which records you want to return using WHERE clauses to test for the validity of conditions. Comparison or logical operators are used in WHERE clauses, and sometimes both types are used for compound statements. Selecting records from multiple tables within one statement is as advanced as it gets, as these types of statements—called JOIN—require forethought and planning to produce correct results. Common types of JOIN are INNER JOIN, LEFT JOIN, and RIGHT JOIN, although MySQL supports many different kinds of JOIN.

The UPDATE and REPLACE commands are used to modify existing data in your MySQL tables. UPDATE is good for changing values in specific columns or to change values in multiple records based on specific conditions. REPLACE is a variation of INSERT that deletes, and then reinserts a record with a matching primary key. Be very careful when using UPDATE to change values in a column because failure to add a condition will result in the given column being updated throughout all records in the table.

The DELETE command is a simple one—it simply removes whole records from tables. This also makes it very dangerous, so be sure you give DELETE privileges only to users who can handle the responsibility. You can specify conditions when using DELETE so that records are removed only if a particular expression in a WHERE clause is true. Also, you can delete smaller portions of the records in your table using a LIMIT clause. If you have an exceptionally large table, deleting portions is less resource-intensive than deleting each record in a huge table.

Q&A

Q1:

What characters can I use to name my tables and fields, and what is the character limit?

A1:

The maximum length of database, table, or field names is 64 characters. Any character that you can use in a directory or filename, you can use in database and table names—except (/) and (.). These limitations are in place because MySQL creates directories and files in your file system, which correspond to database and table names. There are no character limitations (besides length) in field names.

Workshop

The Workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1: The integer 56678685 could be which data type(s)?

A1: MEDIUMINT, INT, or BIGINT.

2: How would you define a field that could only contain the following strings: apple, pear, banana, cherry?

A2: ENUM ('apple', 'pear', 'banana', 'cherry') or SET ('apple', 'pear', 'banana', 'cherry')

3: What would be the LIMIT clauses for selecting the first 25 records of a table? Then the next 25?

A3: LIMIT 0, 25 and LIMIT 26, 25

4: How would you formulate a string comparison using LIKE to match first names of "John" or "Joseph"?

A4: LIKE 'Jo%'

5: How would you explicitly refer to a field called id in a table called table1?

A5: Use table1.id instead of id in your query.

6: Write an SQL statement that joins two tables, orders,

Hour 8. Interacting with MySQL Using PHP

Now that you've learned about PHP and MySQL, you're ready to make the two interact. Think of PHP as a conduit to MySQL—the commands you learned in the previous hour are the same commands that you will send to MySQL in this hour, only this time you'll send them with PHP. In this hour, you will learn

- How to connect to MySQL using PHP
- How to insert and select data through PHP scripts

Connecting to MySQL with PHP

To successfully use the PHP functions to talk to MySQL, you must have MySQL running at a location to which your Web server can connect (not necessarily the same machine as your Web server). You also must have created a user (with a password), and you must know the name of the database to which you want to connect. If you followed the instructions in Hour 1, "Installing and Configuring MySQL," and Hour 3, "Installing and Configuring PHP," you should already have taken care of this.

In all sample scripts in this hour, the sample database name is testDB, the sample user is joeuser, and the sample password is somepass. Substitute your own information when you use these scripts.

You can find the section of the PHP Manual that covers all MySQL-related functions at http://www.php.net/manual/en/ref.mysql.php. Use it!

Using mysql_connect()

The mysql_connect() function is the first function you must call when utilizing a PHP script to connect to MySQL—without an open connection to MySQL, you won't get very far! The basic syntax for the connection is

```
mysql_connect("hostname", "username", "password");
```

Using actual sample values, the connection function looks like this:

```
mysql_connect("localhost", "joeuser", "somepass");
```

This function returns a connection index if the connection is successful or returns false if the connection fails. Listing 8.1 is a working example of a connection script. It assigns the value of the connection index to a variable called $conn, then prints the value of $conn as proof of a connection.

Listing 8.1 A Simple Connection Script

```
1: <?php
2: $conn = mysql_connect("localhost", "joeuser", "somepass");
3: echo "$conn";
4: ?>
```

Save this script as mysqlconnect.php and place it in the document area of your Web server. Access the script with your Web browser and you will see something like the following in your Web browser:

```
Resource id #1
```

Connecting to MySQL using the mysql_connect() function is pretty straightforward. The connection closes when the script finishes its execution, but if you would like to explicitly close the connection, simply add the mysql_close() function at the end of the script, as in Listing 8.2.

Working with MySQL Data

Inserting, updating, deleting, and retrieving data all revolve around the use of the mysql_query() function to execute the basic SQL queries. For INSERT, UPDATE, and DELETE, no additional scripting is required after the query has been executed because you're not displaying any results (unless you want to). For SELECT, you have a few options for displaying the data retrieved by your query. Let's start with the basics and insert some data, so you'll have something to retrieve later on.

Inserting Data with PHP

The easiest method for inserting data is to simply hard-code the INSERT statement, as shown in Listing 8.6.

Listing 8.6 A Script to Insert a Record

```
 1: <?php
 2: // open the connection
 3: $conn = mysql_connect("localhost", "joeuser", "somepass");
 4: // pick the database to use
 5: mysql_select_db("testDB",$conn);
 6: // create the SQL statement
 7: $sql = "INSERT INTO testTable values ('', 'some value')";
 8: // execute the SQL statement
 9: $result = mysql_query($sql, $conn) or die(mysql_error());
10: // echo the result identifier
11: echo $result;
12: ?>
```

You might wonder why you need to echo the result identifier if you're just inserting data. Well, you don't have to; it's just there for kicks. You can clean this script up a bit by replacing the query execution line so that it simply executes and prints a relevant statement if successful, as shown in Listing 8.7.

Listing 8.7 The Modified Insert Script

```
 1: <?php
 2: // open the connection
 3: $conn = mysql_connect("localhost", "joeuser", "somepass");
 4: // pick the database to use
 5: mysql_select_db("testDB",$conn);
 6: // create the SQL statement
 7: $sql = "INSERT INTO testTable values ('', 'some value')";
 8: // execute the SQL statement
 9: if (mysql_query($sql, $conn)) {
10:     echo "record added!";
11: } else {
12:     echo "something went wrong";
13: }
14: ?>
```

Running this script will result in the addition of a row to the testTable table. To enter more records than just the one shown in the script, you can either make a long list of hard-coded SQL statements and use mysql_query() multiple times to execute these statements, or you can create a form-based interface to the record addition script.

To create the form for this script, you really only need one field because the id field can automatically increment. The action of the form is the name of the record-addition script; let's call it insert.php. Your HTML form might look

Summary

Using PHP and MySQL to create dynamic, database-driven Web sites is a breeze. Just remember that the PHP functions are essentially a gateway to the database server; anything you'd enter using the MySQL command-line interface, you can use with mysql_query().

To connect to MySQL with PHP, you need to know your MySQL username, password, and database name. Using mysql_connect() and mysql_select_db(), you can connect to and select a database to use throughout the life of the script.

Once connected, you can issue standard SQL commands with the mysql_query() function. If you have issued a SELECT command, you can use mysql_numrows() to count the records returned in the resultset. If you want to display the data found, you can use mysql_fetch_array() to get all the results during a loop and display them onscreen.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin to put your knowledge into practice.

Quiz

1: What is the primary function used to make the connection between PHP and MySQL, and what information is necessary?

A1: The mysql_connect() function creates a connection to MySQL and requires the hostname, username, and password.

2: Which PHP function retrieves a MySQL error message?

A2: The mysql_error() function returns a MySQL error message.

3: Which PHP function is used to count the number of records in a resultset?

A3: The mysql_numrows() function counts the number of records in a resultset.

Activity

Create a PHP script that displays the contents of the grocery_inventory table that was used in the previous hour.

Part III: Getting Involved with the Code

Hour

Hour 9. Working with Forms

Until now, the PHP examples in this book have been missing a crucial dimension. Sure, you know the basics, can set variables and arrays, create and call functions, and connect to MySQL to do great things with a database. But that's all meaningless if users can't reach into a language's environment to offer it information. In this hour, you look at strategies for acquiring and working with user input. On the World Wide Web, HTML forms are the principal means by which substantial amounts of information pass from the user to the server.

In this hour, you will learn

- How to access information from form fields
- How to work with form elements that allow multiple selections
- How to create a single document that contains both an HTML form and the PHP code that handles its submission
- How to save state with hidden fields
- How to redirect the user to a new page
- How to build HTML forms and PHP code that send mail
- How to build HTML forms that upload files and how to write the PHP code to handle them

Predefined Variables

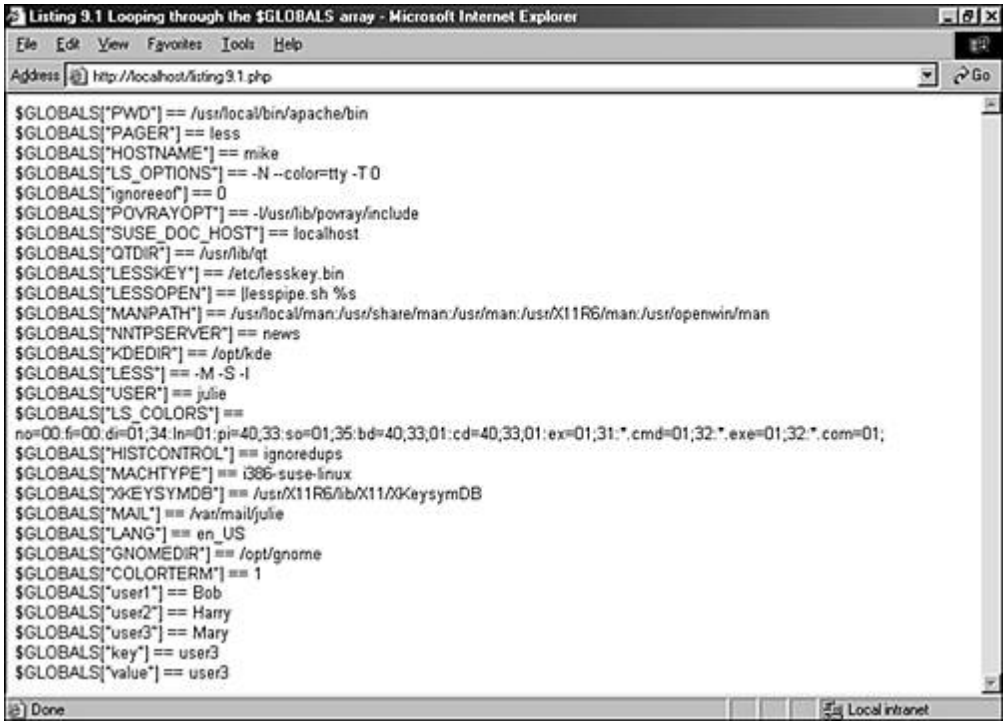
Before you actually build a form and use it to acquire data, you must make a small detour and look again at global variables. You first met global variables in Hour 6, "Working with Functions." To refresh your memory, a global variable is any variable declared at the top level of a script—that is, declared outside of any function. All functions are made available in a built-in associative array named $GLOBALS. This is useful in Listing 9.1 because we can take a peek at all of our script's global variables with a single loop.

Listing 9.1 Looping Through the $GLOBALS Array

```
 1: <html>
 2: <head>
 3: <title>Listing 9.1 Looping through the $GLOBALS array</title>
 4: </head>
 5: <body>
 6: <?php
 7: $user1 = "Bob";
 8: $user2 = "Harry";
 9: $user3 = "Mary";
10: foreach ($GLOBALS as $key=>$value) {
11:    print "\$GLOBALS[\"$key\"] == $value<br>";
12: }
13: ?>
14: </body>
15: </html>
```

Put these lines into a text file named listing9.1.php, and place that file in your Web server document root. When you access this script through your Web browser, it should look something like Figure 9.1 (with your own values, of course).

Figure 9.1. Output of Listing 9.1.



In this listing, we declare three variables (lines 7-9) and then loop through the built-in $GLOBALS associative array (lines 10 and 11), writing both array keys and values to the browser. In the output, we can locate the variables we defined (look toward the bottom of your screen), but we also see an awful lot more in addition to our variables. PHP

Creating a Simple Input Form

For now, let's keep our HTML separate from our PHP code. Listing 9.2 builds a simple HTML form.

Listing 9.2 A Simple HTML Form

```
 1: <html>
 2: <head>
 3: <title>Listing 9.2 A simple HTML form</title>
 4: </head>
 5: <body>
 6: <form action="listing9.3.php" method="POST">
 7: Name: <br>
 8: <input type="text" name="user">
 9: <br>
10: Address: <br>
11: <textarea name="address" rows="5" cols="40"></textarea>
12: <br>
13: <input type="submit" value="hit it!">
14: </form>
15: </body>
16: </html>
```

Put these lines into a text file called listing9.2.php, and place that file in your Web server document root. This listing defines a form that contains a text field with the name "user" on line 8, a text area with the name "address" on line 11, and a submit button on line 13. The FORM element's ACTION argument points to a file called listing9.3.php, which processes the form information. The method of this form is POST, so the variables are stored in the $_POST superglobal.

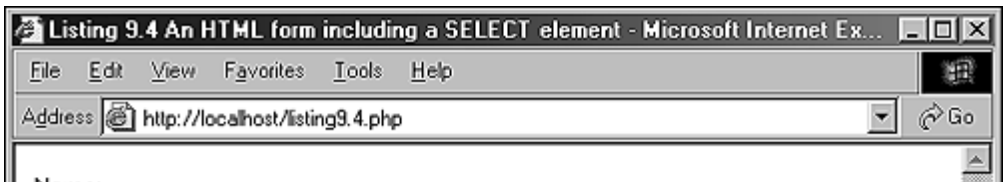
Listing 9.3 creates the code that receives our users' input.

Listing 9.3 Reading Input from the Form in Listing 9.2

```
 1: <html>
 2: <head>
 3: <title>Listing 9.3 Reading input from the form in Listing 9.2</title>
 4: </head>
 5: <body>
 6: <?php
 7: print "Welcome <b>$_POST[user]</b><P>\n\n";
 8: print "Your address is:<P>\n\n<b>$_POST[address]</b>";
 9: ?>
10: </body>
11: </html>
```

Put these lines into a text file called listing9.3.php, and place that file in your Web server document root. Now access the form itself with your Web browser, and you should see something like Figure 9.2.

Figure 9.2. Form created in Listing 9.2.



Accessing Form Input with User-Defined Arrays

The examples so far enable us to gather information from HTML elements that submit a single value per element name. This leaves us with a problem when working with SELECT elements. These elements make it possible for the user to choose multiple items. If we name the SELECT element with a plain name, like so

```
<select name="products" multiple>
```

the script that receives this data has access to only a single value corresponding to this name. We can change this behavior by renaming an element of this kind so that its name ends with an empty set of square brackets. We do this in Listing 9.4.

Listing 9.4 An HTML Form Including a SELECT Element

```
 1: <html>
 2: <head>
 3: <title>Listing 9.4 An HTML form including a SELECT element</title>
 4: </head>
 5: <body>
 6: <form action="listing9.5.php" method="POST">
 7: Name: <br>
 8: <input type="text" name="user">
 9: <br>
10: Address: <br>
11: <textarea name="address" rows="5" cols="40"></textarea>
12: <br>
13: Pick Products: <br>
14: <select name="products[]" multiple>
15: <option>Sonic Screwdriver</option>
16: <option>Tricorder</option>
17: <option>ORAC AI</option>
18: <option>HAL 2000</option>
19: </select>
20: <br><br>
21: <input type="submit" value="hit it!">
22: </form>
23: </body>
24: </html>
```

Put these lines into a text file called listing9.4.php, and place that file in your Web server document root. Next, in the script that processes the form input, we find that input from the "products[]" form element created on line 14 is available in an array called $_POST[products]. Because products[] is a SELECT element, we offer the user multiple choices using the option elements on lines 15 through 18. We demonstrate that the user's choices are made available in an array in Listing 9.5.

Listing 9.5 Reading Input from the Form in Listing 9.4

```
 1: <html>
 2: <head>
 3: <title>Listing 9.5 Reading input from the form in Listing 9.4</title>
 4: </head>
 5: <body>
 6: <?php
 7: print "Welcome <b>$_POST[user]</b><p>\n\n";
 8: print "Your address is:<p>\n\n<b>$_POST[address]</b><p>\n\n";
 9: print "Your product choices are:<p>\n\n";
10: if (!empty($_POST[products])) {
11:     print "<ul>\n\n";
```

Combining HTML and PHP Code on a Single Page

In some circumstances, you might want to include form-parsing code on the same page as a hard-coded HTML form. Such a combination can be useful if you need to present the same form to the user more than once. You would have more flexibility if you were to write the entire page dynamically, of course, but you would miss out on one of the great strengths of PHP. The more standard HTML you can leave in your pages, the easier they are for designers and page builders to amend without reference to you. You should avoid scattering substantial chunks of PHP code throughout your documents, however. Doing so makes them hard to read and maintain. Where possible, you should create functions that can be called from within your HTML code and can be reused in other projects.

For the following examples, imagine that we're creating a site that teaches basic math to preschool children, and have been asked to create a script that takes a number from form input and tells the user whether it's larger or smaller than a predefined integer.

Listing 9.6 creates the HTML. For this example, we need only a single text field, but even so, we'll include a little PHP.

Listing 9.6 An HTML Form That Calls Itself

```
 1: <html>
 2: <head>
 3: <title>Listing 9.6 An HTML form that calls itself</title>
 4: </head>
 5: <body>
 6: <form action="<?php print $_SERVER[PHP_SELF] ?>" method="POST">
 7: Type your guess here: <input type="text" name="guess">
 8: </form>
 9: </body>
10: </html>
```

The action of this script is $_SERVER[PHP_SELF]. This variable is the equivalent of the name of the current script. In other words, the action tells the script to reload itself.

The script in Listing 9.6 doesn't produce any output. In Listing 9.7, we begin to build up the PHP element of the page. First, we must define the number that the user guesses. In a fully working version, we'd probably randomly generate this number, but for now, we keep it simple. We assign 42 to the $num_to_guess variable on line 2. Next, we must determine whether the form has been submitted; otherwise, we'd attempt to assess variables that aren't yet made available. We can test for submission by testing for the existence of the variable $_POST[guess], which is made available if your script has been sent a "guess" parameter. If $_POST[guess] isn't present, we can safely assume that the user arrived at the page without submitting a form. If the value is present, we can test the value it contains. The test for the presence of the $_POST[guess] variable takes place on line 4.

Listing 9.7 A PHP Number-Guessing Script

```
 1: <?php
 2: $num_to_guess = 42;
 3: $message = "";
 4: if (!isset($_POST[guess])) {
 5:    $message = "Welcome to the guessing machine!";
 6: } elseif ($_POST[guess] > $num_to_guess) {
 7:    $message = "$_POST[guess] is too big! Try a smaller number";
 8: } elseif ($_POST[guess] < $num_to_guess) {
 9:    $message = "$_POST[guess] is too small! Try a larger number";
```

Using Hidden Fields to Save State

The script in Listing 9.7 has no way of knowing how many guesses a user has made, but we can use a hidden field to keep track of this. A hidden field behaves exactly the same as a text field, except that the user cannot see it unless he views the HTML source of the document that contains it. Listing 9.8 adds a hidden field to the number-guessing script and some PHP to work with it.

Listing 9.8 Saving State with a Hidden Field

```
 1: <?php
 2: $num_to_guess = 42;
 3: $num_tries = (isset($_POST[num_tries])) ? $num_tries + 1 : 0;
 4: $message = "";
 5: if (!isset($_POST[guess])) {
 6:      $message = "Welcome to the guessing machine!";
 7: } elseif ($_POST[guess] > $num_to_guess) {
 8:      $message = "$_POST[guess] is too big! Try a smaller number";
 9: } elseif ($_POST[guess] < $num_to_guess) {
10:      $message = "$_POST[guess] is too small! Try a larger number";
11: } else { // must be equivalent
12:      $message = "Well done!";
13: }
14: $guess = $_POST[guess];
15: ?>
16: <html>
17: <head>
18: <title>Listing 9.8 Saving state with a hidden field</title>
19: </head>
20: <body>
21: <h1>
22: <?php print $message ?>
23: </h1>
24: Guess number: <?php print $num_tries?>
25: <form action="<?php print $_SERVER[PHP_SELF] ?>" method="POST">
26: Type your guess here:
27: <input type="text" name="guess" value="<?php print $guess?>">
28: <input type="hidden" name="num_tries" value="<?php print $num_tries?>">
29: </form>
30: </body>
31: </html>
```

The hidden field on line 28 is given the name "num_tries". We also use PHP to write its value. While we're at it, we do the same for the "guess" field on line 27 so that the user can always see his last guess. This technique is useful for scripts that parse user input. If we reject a form submission for some reason, we can at least allow our user to edit his previous query.

Within the main PHP code, we use a ternary operator to increment the $num_tries variable. If the $num_tries variable is set, we add one to it and reassign this incremented value; otherwise, we initialize $num_tries to 0. Within the body of the HTML, we can now report to the user how many guesses he's made.

Put these lines into a text file called listing9.8.php, and place that file in your Web server document root. Access the form a few times with your Web browser, and try to guess the number (pretend you don't already know it).

Redirecting the User

Our simple script still has one major drawback. The form is rewritten whether or not the user guesses correctly. The fact that the HTML is hard-coded makes it difficult to avoid writing the entire page. We can, however, redirect the user to a congratulations page, thereby sidestepping the issue altogether.

When a server script communicates with a client, it must first send some headers that provide information about the document to follow. PHP usually handles this for you automatically, but you can choose to send your own header lines with PHP's header() function.

To call the header() function, you must be sure that absolutely no output has been sent to the browser. The first time content is sent to the browser, PHP sends out headers and it's too late for you to send your own. Any output from your document, even a line break or a space outside of your script tags, causes headers to be sent. If you intend to use the header() function in a script, you must make certain that nothing precedes the PHP code that contains the function call. You should also check any libraries that you might be using.

Listing 9.9 shows typical headers sent to the browser by PHP, beginning with line 3, in response to the request in line 1.

Listing 9.9 Typical Server Headers Sent from a PHP Script

```
 1: HEAD /listing9.9.php HTTP/1.0
 2:
 3: HTTP/1.1 200 OK
 4: Date: Sun, 15 Sep 2002 12:32:28 GMT
 5: Server: Apache/2.0.43 (Unix) PHP/4.2.3 mod_ssl/2.8.9 OpenSSL/0.9.6
 6: X-Powered-By: PHP/4.2.3
 7: Connection: close
 8: Content-Type: text/html
```

By sending a "Location" header instead of PHP's default, you can cause the browser to be redirected to a new page:

```
header("Location: http://www.samspublishing.com");
```

Assuming that we've created a suitably upbeat page called "congrats.html", we can amend our number-guessing script to redirect the user if she guesses correctly, as shown in Listing 9.10.

Listing 9.10 Using header() to Send Raw Headers

```
 1: <?php
 2: $num_to_guess = 42;
 3: $num_tries = (isset($_POST[num_tries])) ? $num_tries + 1: 0;
 4: $message = "";
 5: if (!isset($_POST[guess])) {
 6:    $message = "Welcome to the guessing machine!";
 7: } elseif ($_POST[guess] > $num_to_guess) {
 8:    $message = "$_POST[guess] is too big! Try a smaller number";
 9: } elseif ($_POST[guess] < $num_to_guess) {
10:    $message = "$_POST[guess] is too small! Try a larger number";
11: } else { // must be equivalent
12:    header("Location: congrats.html");
13:    exit;
14: }
```

Sending Mail on Form Submission

You've already seen how to take form responses and print the results to the screen. You're only one step away from sending those responses in an email message, as you'll soon see. Before learning about sending mail, however, read through the next section to make sure that your system is properly configured.

System Configuration for the mail() Function

Before you can use the mail() function to send mail, a few directives must be set up in the php.ini file so that the function works properly. Open php.ini with a text editor and look for these lines:

```
[mail function]
; For Win32 only.
SMTP = localhost

; For Win32 only.
sendmail_from = me@localhost.com

; For Unix only. You may supply arguments as well (default: "sendmail -t -i").
;sendmail_path =
```

If you're using Windows as your Web server platform, the first two directives apply to you. For the mail() function to send mail, it must be able to access a valid outgoing mail server. If you plan to use the outgoing mail server of your ISP (in the following example, we use EarthLink), the entry in php.ini should look like this:

```
SMTP = mail.earthlink.net
```

The second configuration directive is sendmail_from, which is the email address used in the From header of the outgoing email. It can be overwritten in the mail script itself, but normally operates as the default value. For example:

```
sendmail_from = youraddress@yourdomain.com
```

A good rule of thumb for Windows users is that whatever outgoing mail server you've set up in your email client on that machine, you should also use as the value of SMTP in php.ini.

If your Web server is running on a Linux/Unix platform, you use the sendmail functionality of that particular machine. In this case, only the last directive applies to you: sendmail_path. The default is sendmail -t -i, but if sendmail is in an odd place or if you need to specify different arguments, feel free to do so, as in the following example:

```
sendmail_path = /opt/sendmail -odd -arguments
```

After making any changes to php.ini on any platform, you must restart the Web server process for the changes to take effect.

Creating the Form

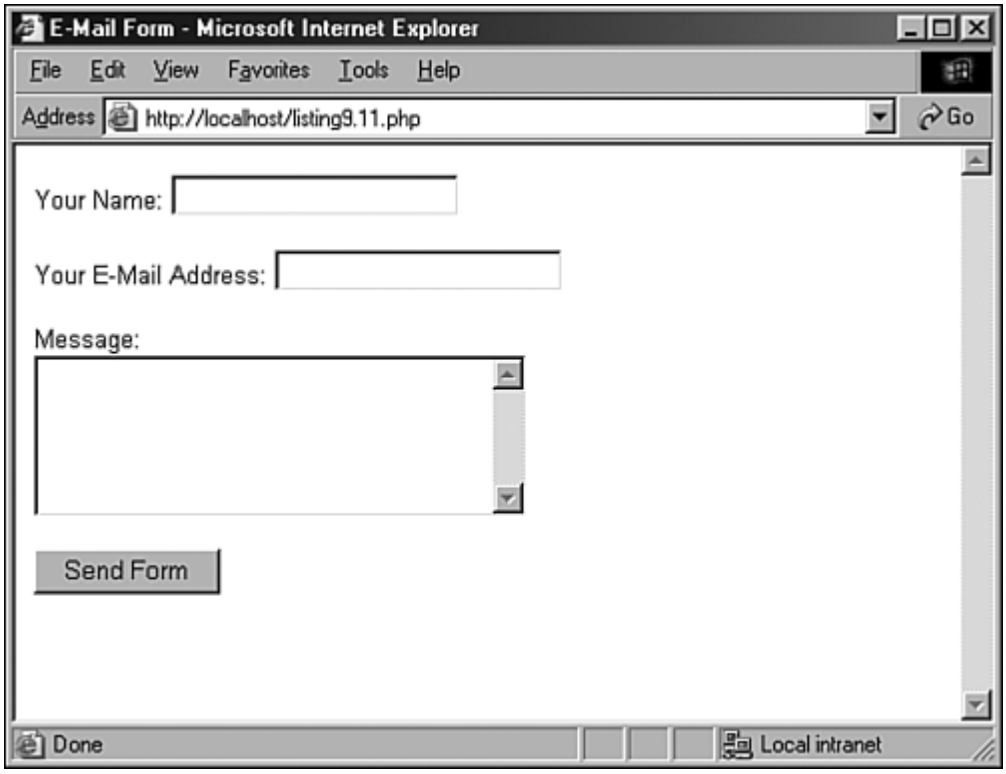
In Listing 9.11, you see the basic HTML for creating a simple feedback form. This form has an action of listing9.12.php, which we create in the next section. The fields are very simple: Line 7 contains a name field, line 8 contains the return email address field, and line 10 contains the text area for the user's message.

Listing 9.11 Creating a Simple Feedback Form

```
 1: <HTML>
 2: <HEAD>
 3: <TITLE>E-Mail Form</TITLE>
 4: </HEAD>
 5: <BODY>
 6: <FORM action="listing9.12.php" method="POST">
 7: Your Name: <INPUT type="text" name="name"><br><br>
 8: Your E-Mail Address: <INPUT type="text" name="email"><br><br>
 9: Message:<br>
10: <textarea name="message" cols=30 rows=5></textarea><br><br>
11: <INPUT type="submit" value="Send Form">
12: </FORM>
13: </BODY>
14: </HTML>
```

Put these lines into a text file called listing9.11.php, and place this file in your Web server document root. Now access the script with your Web browser, and you should see something like Figure 9.6.

Figure 9.6. Form created in Listing 9.11.



In the next section, you create the script that sends this form to a recipient.

Creating the Script to Send the Mail

This script is only slightly different in concept than the script in Listing 9.5, which simply printed form responses to the screen. In this script, in addition to printing the responses to the screen, you send them to an email address as well.

Listing 9.12 Sending the Simple Feedback Form

```
 1: <html>
 2: <head>
 3: <title>Listing 9.12 Sending mail from the form in Listing 9.11</title>
 4: </head>
 5: <body>
 6: <?php
 7: print "Thank you, <b>$_POST[name]</b>, for your message!<br><br>\n\n";
 8: print "Your e-mail address is: <b>$_POST[email]</b><br><br>\n\n";
 9: print "Your message was:<br><br>\n\n";
10: print "$_POST[message] <br><br>";
11: //start building the mail string
12: $msg = "Name: $_POST[name]\n";
13: $msg .= "E-Mail: $_POST[email]\n";
14: $msg .= "Message: $_POST[message]\n";
15: //set up the mail
16: $recipient = "you@yourdomain.com";
17: $subject = "Form Submission Results";
18: $mailheaders = "From: My Web Site <defaultaddress@yourdomain.com> \n";
19: $mailheaders .= "Reply-To: $_POST[email]";
20: //send the mail
21: mail($recipient, $subject, $msg, $mailheaders);
22: ?>
23: </body>
24: </html>
```

The variables you use in lines 7-9 are $_POST[name], $_POST[email], and $_POST[message]—the names of the fields in the form, as part of the $_POST superglobal. That's all well and good for printing the information to the screen, but in this script, you also want to create a string that's sent in email. For this task, you essentially build the email by concatenating strings to form one long message string, using the newline (\n) character to add line breaks where appropriate.

Lines 12 through 14 create the $msg string, which contains the values typed by the user in the form fields. This string is the one sent in the email. Note the use of the concatenation operator (.=) when adding to the variable $msg, in lines 13 and 14.

Lines 16 and 17 are hard-coded variables for the email recipient and the subject of the email message. Replace you@yourdomain.com with your own email address, obviously. If you want to change the subject, feel free!

Lines 18 and 19 set up some mail headers, namely From: and Reply-to: headers. You could put any value in the From: header; this is the information that displays in the From or Sender column of your email application when you receive this mail.

The mail() function takes four parameters: the recipient, the subject, the message, and any additional mail headers. The order of these parameters is shown in line 21, and your script is complete after you close up your PHP block and your HTML elements in lines 22-24.

Working with File Uploads

So far, we've looked at simple form input. However, all popular Web browsers support file uploads, and so, of course, does PHP. In this section, you examine the features that PHP makes available to deal with this kind of input.

Information about the uploaded file becomes available to you in the $_FILES superglobal, which is indexed by the name of the upload field (or fields) in the form. The corresponding value for each of these keys is an associative array. These fields are described in Table 9.2, using fileupload as the name of the form field used for the upload.

Table 9.2. File Upload Global Variables

Element	Contains	Example
$_FILES['fileupload']['name']	Original name of uploaded file	test.gif
$_FILES['fileupload']['tmp_name']	Path to temporary file	/tmp/phprDfZvN
$_FILES['fileupload']['size']	Size (in bytes) of uploaded file	6835
$_FILES['fileupload']['type']	MIME type of uploaded file (where given by client)	image/gif

Keep these elements in the back of your mind for a moment, while we create the upload form in the next section.

Creating the File Upload Form

First, we must create the HTML form to handle the upload. HTML forms that include file upload fields must include an ENCTYPE argument:

```
ENCTYPE="multipart/form-data"
```

PHP also works with an optional hidden field that can be inserted before the file upload field. This field must be called MAX_FILE_SIZE and should have a value representing the maximum size in bytes of the file that you're willing to accept. This size cannot override the maximum size set in the upload_max_filesize field in your php.ini file that defaults to 2MB. The MAX_FILE_SIZE field is obeyed at the browser's discretion, so you should rely on the php.ini setting to cap unreasonable uploads. After the MAX_FILE_SIZE field has been entered, you're ready to add the upload field itself. This is simply an INPUT element with a TYPE argument of "file". You can give it any name you want. Listing 9.13 brings all this together into an HTML upload form.

Listing 9.13 A Simple File Upload Form

```
1: <html>
2: <head>
```

Summary

Things are really getting exciting now! You have the tools to create truly sophisticated and interactive environments. A few things are still missing, of course. Now that you can get information from the user, it would be nice to be able to do something with it. Write it to a file, perhaps. That's the subject of the next hour.

Throughout this hour, you learned how to work with various superglobals and form input. You also learned how to send raw headers to the client to redirect a browser. You learned how to acquire list information from form submissions and how to pass information from script call to script call using hidden fields. Finally, you learned how to send your form results in email, and also how to upload files through your Web browser via a PHP script.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1: Which predefined variable do you use to find the name of the script?

A1: The variable $_SERVER['PHP_SELF']' holds the name of the script.

2: Which built-in associative array contains all values submitted as part of a POST request?

A2: The $_POST superglobal.

3: Which built-in associative array contains all values submitted as part of a file upload?

A3: The $_FILES superglobal.

4: What function do you use to redirect the browser to a new page?

A4: The header() function.

5: What are the four arguments used by the mail() function?

A5: The recipient, the subject, the message string, and additional headers.

Hour 10. Working with Files

Testing, reading, and writing to files are staple activities for any full-featured programming language. PHP is no exception, providing you with functions that make the process straightforward. In this hour, you will learn

- How to include files in your documents
- How to test files and directories
- How to open a file before working with it
- How to read data from files
- How to write or append to a file
- How to lock a file
- How to work with directories

Including Files with include()

The include() statement enables you to incorporate files into your PHP documents. PHP code in these files can be executed as if it were part of the main document. This can be useful for including library code in multiple pages.

Having created a killer function, your only option until now would have been to paste it into every document that needs to use it. Of course, if you discover a bug or want to add a feature, you would have to find every page that uses the function to make the change. The include() statement can save you from this chore. You can add the function to a single document and, at runtime, read it into any page that needs it. The include() statement requires a single argument: a relative path to the file to be included. Listing 10.1 creates a simple PHP script that uses include() to incorporate and output the contents of a file.

Listing 10.1 Using include()

```
 1: <html>
 2: <head>
 3: <title>Listing 10.1 Using include()</title>
 4: </head>
 5: <body>
 6: <?php
 7: include("listing10.2.php");
 8: ?>
 9: </body>
10: </html>
```

The include() statement in Listing 10.1 incorporates the document listing10.2.php, the contents of which you can see in Listing 10.2.

Listing 10.2 The File Included in Listing 10.1

```
1: I have been included!!
```

Put the contents of Listing 10.1 in a file named listing10.1.php, and the contents of Listing 10.2 in a file named listing10.2.php. Place both files in your Web server document root. When you access listing10.1.php through your Web browser, the output on the screen is

```
I have been included!!
```

This might seem strange to you, given that we've included plain text within a block of PHP code. In fact, the contents of an included file are displayed as text by default. If you want to execute PHP code in an included file, you must enclose it in PHP start and end tags. In Listings 10.3 and 10.4, we amend the previous example so that code is executed in the included file.

Listing 10.3 Using the include() Statement to Execute PHP in Another File

```
 1: <html>
 2: <head>
 3: <title>Listing 10.3 Using include to execute PHP in another file</title>
 4: </head>
 5: <body>
 6: <?php
 7: include("listing10.4.php");
 8: ?>
 9: </body>
```

include_once()

One of the problems caused by using multiple libraries within your code is the danger of calling include() twice on the same file. This can occur in larger projects when different library files call include() on a common file. Including the same file twice often results in repeated declarations of functions and classes, thereby causing the PHP engine great unhappiness.

The situation is saved by the include_once() statement. include_once() requires the path to an include file and behaves the same way as include() the first time it's called. However, if include_once() is called again for the same file during script execution, the file is not included again. This makes include_once() an excellent tool for the creation of reusable code libraries!

The include_path Directive

Using include() and include_once() to access libraries can increase the flexibility and reusability of your projects. However, there are still headaches to overcome. Portability in particular can suffer if you hard-code the paths to included files. Imagine that you create a lib directory and reference it throughout your project:

```
include_once("/home/user/bob/htdocs/project4/lib/mylib.inc.php");
```

When you move your project to a new server, you might find that you have to change a hundred or more include paths. You can escape this fate by setting the include_path directive in your php.ini file:

```
include_path .:/home/user/bob/htdocs/project4/lib/
```

The include_path can include as many directories as you want, separated by colons (semicolons in Windows). The first dot (.) before the first colon indicates "current directory." You can then reference your library file by only its name:

```
include_once("mylib.inc.php");
```

When you move your project, you need to change only the include_path directive.

PHP has both a require() statement, which performs a similar function to include(), and a require_once() statement. require() is executed regardless of a script's flow, and therefore shouldn't be used as part of conditional or loop structures.

A file included as a result of a require() statement cannot return a value.

Testing Files

Before you work with a file or directory, it's often a good idea to learn more about it. PHP provides many functions to help you to discover information about files on your system. This section briefly covers some of the most useful functions.

Checking for Existence with file_exists()

You can test for the existence of a file with the file_exists() function. This function requires a string representing an absolute or relative path to a file that might or might not be there. If the file is found, file_exists() returns true; otherwise, it returns false.

```
if (file_exists("test.txt")) {
       print "The file exists!";
}
```

A File or a Directory?

You can confirm that the entity you're testing is a file, as opposed to a directory, with the is_file() function. is_file() requires the file path and returns a Boolean value.

```
if (is_file("test.txt")) {
        print "test.txt is a file!";
}
```

Conversely, you might want to check that the entity you're testing is a directory. You can do this with the is_dir() function. is_dir() requires the path to the directory and returns a Boolean value.

```
if (is_dir("/tmp")) {
        print "/tmp is a directory";
}
```

Checking the Status of a File

When you know that a file exists, and it's what you expect it to be, you can find out some things that you can do with it. Typically, you might want to read, write to, or execute a file. PHP can help you with all of these operations.

is_readable() tells you whether you can read a file. On Unix systems, you might be able to see a file but still be barred from reading its contents. is_readable() accepts the file path as a string and returns a Boolean value.

```
if (is_readable("test.txt")) {
        print "test.txt is readable";
}
```

is_writable() tells you whether you can write to a file. As with is_readable(), the is_writable() function requires the file path and returns a Boolean value.

```
if (is_writable("test.txt")) {
        print "test.txt is writable";
}
```

Creating and Deleting Files

If a file does not yet exist, you can create it with the touch() function. Given a string representing a file path, touch() attempts to create an empty file of that name. If the file already exists, the contents aren't disturbed, but the modification date is updated to the time at which the function executed.

```
touch("myfile.txt");
```

You can remove an existing file with the unlink() function. As did the touch() function, unlink() accepts a file path:

```
unlink("myfile.txt");
```

All functions that create, delete, read, write, and modify files on Unix systems require the correct file or directory permissions to be set.

Opening a File for Writing, Reading, or Appending

Before you can work with a file, you must first open it for reading, writing, or both. PHP provides the fopen() function for doing so. fopen() requires a string that contains the file path followed by a string containing the mode in which the file is to be opened. The most common modes are read (r), write (w), and append (a). fopen() returns a file resource you'll use later to work with the open file. To open a file for reading, you use the following:

```
$fp = fopen("test.txt", 'r');
```

You use the following to open a file for writing:

```
$fp = fopen("test.txt", 'w');
```

To open a file for appending (that is, to add data to the end of a file), you use this:

```
$fp = fopen("test.txt", 'a');
```

fopen() returns false if the file cannot be opened for any reason. Therefore, it's a good idea to test the function's return value before proceeding to work with it. You can do so with an if statement:

```
if ($fp = fopen("test.txt", "w")) {
// do something with $fp
}
```

Or you can use a logical operator to end execution if an essential file can't be opened:

```
($fp = fopen("test.txt", "w")) or die ("Couldn't open file, sorry");
```

If the fopen() function returns true, the rest of the expression won't be parsed, and the die() function (which writes a message to the browser and ends the script) is never reached. Otherwise, the right side of the or operator is parsed and the die() function is called.

Assuming that all is well and you go on to work with your open file, you should remember to close it when you finish. You can do so by calling fclose(), which requires the file resource returned from a successful fopen() call as its argument:

```
fclose($fp);
```

Reading from Files

PHP provides a number of functions for reading data from files. These enable you to read by the byte, the line, and even by the character.

Reading Lines from a File with fgets() and feof()

After you open a file for reading, you often need to access it line by line. To read a line from an open file, you can use fgets(), which requires the file resource returned from fopen() as an argument. You must also pass fgets() an integer as a second argument. The integer argument specifies the number of bytes that the function should read if it doesn't first encounter a line end or the end of the file. The fgets() function reads the file until it reaches a newline character ("\n"), the number of bytes specified in the length argument, or the end of the file.

```
$line = fgets($fp, 1024); // where $fp is the file resource returned by fopen()
```

Although you can read lines with fgets(), you need some way to tell when you reach the end of the file. The feof() function does this by returning true when the end of the file has been reached and false otherwise. feof() requires a file resource as its argument.

```
feof($fp); // where $fp is the file resource returned by fopen()
```

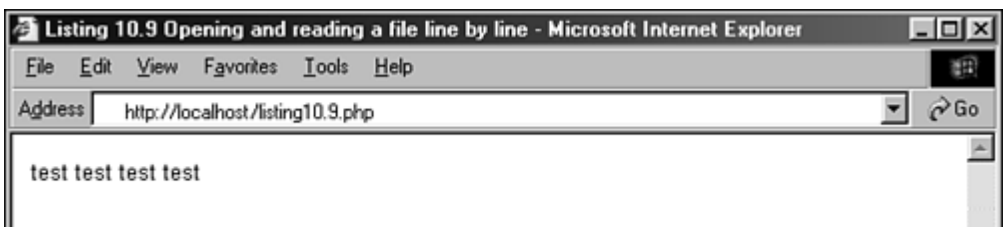
You now have enough information to read a file line by line, as shown in Listing 10.9.

Listing 10.9 Opening and Reading a File Line by Line

```
 1: <html>
 2: <head>
 3: <title>Listing 10.9 Opening and reading a file line by line</title>
 4: </head>
 5: <body>
 6: <?php
 7: $filename = "test.txt";
 8: $fp = fopen($filename, "r") or die("Couldn't open $filename");
 9: while (!feof($fp)) {
10:      $line = fgets($fp, 1024);
11:      print "$line<br>";
12: }
13: ?>
14: </body>
15: </html>
```

If this code were saved to the document root of your Web server and run through your Web browser, the output would look something like Figure 10.3 (the contents of your sample text file might be different).

Figure 10.3. Output of Listing 10.9.



Writing or Appending to a File

The processes for writing to and appending to a file are the same. The difference lies in the fopen() call. When you write to a file, you use the mode argument "w" when you call fopen():

```
$fp = fopen("test.txt", "w");
```

All subsequent writing occurs from the start of the file. If the file doesn't already exist, it is created. If the file already exists, any prior content is destroyed and replaced by the data you write.

When you append to a file, you use mode argument "a" in your fopen() call:

```
$fp = fopen("test.txt", "a");
```

Any subsequent writes to your file are added to the existing content, but if you attempt to append content to a nonexistent file, the file is first created.

Writing to a File with fwrite() or fputs()

fwrite() accepts a file resource and a string, and then writes the string to the file. fputs() works in exactly the same way.

```
fwrite($fp, "hello world");
fputs($fp, "hello world");
```

Writing to files is as straightforward as that. Listing 10.13 uses fwrite() to print to a file. We then append a further string to the same file using fputs().

Listing 10.13 Writing and Appending to a File

```
 1: <html>
 2: <head>
 3: <title>Listing 10.13 Writing and appending to a file</title>
 4: </head>
 5: <body>
 6: <?php
 7: $filename = "test.txt";
 8: print "Writing to $filename<br>";
 9: $fp = fopen($filename, "w") or die("Couldn't open $filename");
10: fwrite($fp, "Hello world\n");
11: fclose($fp);
12: print "Appending to $filename<br>";
13: $fp = fopen($filename, "a") or die("Couldn't open $filename");
14: fputs($fp, "And another thing\n");
15: fclose($fp);
16: ?>
17: </body>
18: </html>
```

The screen output of this script, when run from your Web browser, is

```
Writing to test.txt
```

Working with Directories

Now that you can test, read, and write to files, let's turn our attention to directories. PHP provides many functions for working with directories. Let's look at how to create, remove, and read them.

Creating Directories with mkdir()

mkdir() enables you to create a directory. mkdir() requires a string that represents the path to the directory you want to create, and an octal number integer that represents the mode you want to set for the directory. You specify an octal (base 8) number with a leading 0. The mode argument has an effect only on Unix systems. The mode should consist of three numbers between 0 and 7, representing permissions for the directory owner, group, and everyone, respectively. mkdir() returns true if it successfully creates a directory, or false if it doesn't. If mkdir() fails, it's usually because the containing directory has permissions that preclude processes with the script's user ID from writing. If you're not comfortable setting Unix directory permissions, you should find that one of the following examples fits your needs. Unless you really need your directory to be world writable, you should probably use 0755, which allows the world to read your directory but not to write to it.

```
mkdir("testdir", 0777); // global read/write/execute permissions
mkdir("testdir", 0755); // world and group: read/execute only
                        // owner: read/write/execute
```

Removing a Directory with rmdir()

rmdir() enables you to remove a directory from the file system if the process running your script has the right to do so and if the directory is empty. rmdir() requires only a string representing the path to the directory you want to create.

```
rmdir("testdir");
```

Opening a Directory for Reading with opendir()

Before you can read the contents of a directory, you must first obtain a directory resource. You can do so with the opendir() function. opendir() requires a string that represents the path to the directory you want to open. opendir() returns a directory handle unless the directory isn't present or readable; in that case, it returns false.

```
$dh = opendir("testdir");
```

Reading the Contents of a Directory with readdir()

Just as you use gets() to read a line from a file, you can use readdir() to read a file or directory name from a directory. readdir() requires a directory handle and returns a string containing the item name. If the end of the directory is reached, readdir() returns false. Note that readdir() returns only the names of its items, rather than full paths. Listing 10.14 shows the contents of a directory.

Listing 10.14 Listing the Contents of a Directory with readdir()

```
 1: <html>
 2: <head>
 3: <title>Listing 10.14 Listing the contents
 4: of a directory with readdir()</title>
 5: </head>
 6: <body>
```

Summary

In this hour, you learned how to use include() to incorporate files into your documents and to execute any PHP code contained in include files. You learned how to use some of PHP's file test functions. You explored functions for reading files by the line, by the character, and in arbitrary chunks. You learned how to write to files, either replacing or appending to existing content. Finally, you learned how to create, remove, and read directories.

Now that we can work with files, we can save and access substantial amounts of data. If we need to look up data from large files, however, our scripts begin to slow down quite considerably. What we need is some kind of database.

Q&A

Q1:

Does the include() statement slow down my scripts?

A1:

Because an included file must be opened and parsed by the engine, it adds some overhead. However, the benefits of reusable code libraries often outweigh the relatively low performance overhead.

Q2:

Should I always end script execution if a file cannot be opened for writing or reading?

A2:

You should always allow for this possibility. If your script absolutely depends on the file you want to work with, you might want to use the die() function, writing an informative error message to the browser. In less critical situations, you still need to allow for the failure, perhaps by adding it to a log file.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1:

What functions do you use to add library code to the currently running script?

A1:

You can use the require() or include() statement to incorporate PHP files into the current document. You could also use include_once() or require_once().

2:

What function do you use to find out whether a file is present on your file system?

A2:

You can test for the existence of a file with the file_exists() function.

3:

How do you determine the size of a file?

A3:

The filesize() function returns a file's size in bytes.

4:

What function do you use to open a file for reading or writing?

A4:

The fopen() function opens a file. It accepts the path to a file and a character representing the mode. It returns a file resource.

5:

What function do you use to read a line of data from a file?

A5:

The fgets() function reads data up to the buffer size

Hour 11. Working with Dates and Times

Dates are so much a part of everyday life that it becomes easy to work with them without thinking. However, the quirks of the Gregorian calendar can be difficult to work with in programs. Fortunately, PHP provides powerful tools for date arithmetic that make date manipulation an easy task. Similarly, MySQL comes with its own set of date-related functions. You learn about these in this hour as well, and find that MySQL can take a lot of the programming burden off your hands.

In this hour, you will learn

- How to acquire the current date and time
- How to get information about a date
- How to format date information
- How to test dates for validity
- How to set dates
- How to use MySQL's date and time-related functions
- How to format date and time results in MySQL
- How to find and express intervals between dates and times using MySQL

Using Date and Time Functions in PHP

The several sections that follow introduce you to the date- and time-related functions specifically in PHP. You learn about the MySQL functions later in this hour. Try each listing yourself, as you march along through this hour.

Getting the Date with time()

PHP's time() function gives you all the information that you need about the current date and time. It requires no arguments and returns an integer. For us humans, the returned number is a little hard on the eyes, but it's extremely useful nonetheless.

```
print time();
// sample output: 1127732399
```

The integer returned by time() represents the number of seconds elapsed since midnight GMT on January 1, 1970. This moment is known as the Unix epoch, and the number of seconds that have elapsed since then is referred to as a timestamp. PHP offers excellent tools to convert a timestamp into a form that humans are comfortable with. Even so, isn't a timestamp a needlessly convoluted way of storing a date? In fact, the opposite is true. From just one number, you can extract enormous amounts of information. Even better, a timestamp can make date arithmetic much easier than you might imagine.

Think of a homegrown date system in which you record days of the month as well as months and years. Now imagine a script that must add one day to a given date. If this date happened to be 31 December 1999, rather than adding 1 to the date, you'd have to write code to set the day of the month to 1, the month to January, and the year to 2000. Using a timestamp, you need only add a day's worth of seconds to your current figure and you're done. You can convert this new figure into something more friendly at your leisure.

Converting a Timestamp with getdate()

Now that you have a timestamp to work with, you must convert it before you present it to the user. getdate() optionally accepts a timestamp and returns an associative array containing information about the date. If you omit the timestamp, getdate() works with the current timestamp as returned by time(). Table 11.1 lists the elements contained in the array returned by getdate().

Table 11.1. The Associative Array Returned by getdate()

Key	Description	Example
seconds	Seconds past the minute (0–59)	28
minutes	Minutes past the hour (0–59)	7
hours	Hours of the day (0–23)	12

Using Date and Time Functions in MySQL

MySQL's built-in date-related functions can be used in SELECT statements, with or without specifying a table, to retrieve a result of the function. Or you can use the functions with any type of date field: date, datetime, timestamp, and year. Depending on the type of field in use, the results of the date-related functions are more or less useful.

Working with Days

The DAYOFWEEK() and WEEKDAY() functions do similar things with slightly different results. Both functions are used to find the weekday index of a date, but the difference lies in the starting day and position.

If you use DAYOFWEEK(), the first day of the week is Sunday, at position 1, and the last day of the week is Saturday, at position 7. For example:

```
mysql> select dayofweek('2001-11-13');
+-------------------------+
| DAYOFWEEK('2001-11-13') |
+-------------------------+
|                       3 |
+-------------------------+
1 row in set (0.00 sec)
```

The result shows that November 13, 2001 was weekday index 3, or Tuesday. Using the same date with WEEKDAY() gives you a different result with the same meaning:

```
mysql> select weekday('2001-11-13');
+-----------------------+
| WEEKDAY('2001-11-13') |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

The result shows that November 13, 2001 was weekday index 1. Because WEEKDAY() uses Monday as the first day of the week at position 0 and Sunday as the last day at position 6, 1 is accurate: Tuesday.

The DAYOFMONTH() and DAYOFYEAR() functions are more straightforward, with only one result and a range that starts at 1 and ends at 31 for DAYOFMONTH() and 366 for DAYOFYEAR(). Some examples follow:

```
mysql> select dayofmonth('2001-11-13');
+--------------------------+
| DAYOFMONTH('2001-11-13') |
+--------------------------+
|                       13|
+--------------------------+
1 row in set (0.00 sec)

mysql> select dayofyear('2001-11-13');
+-------------------------+
| DAYOFYEAR('2001-11-13') |
+-------------------------+
|                     317|
+-------------------------+
1 row in set (0.00 sec)
```

Summary

In this hour, you learned how to use various PHP functions to perform date- and time-related actions. The time() function gets a date stamp for the current date and time, and you can use getdate() to extract date information from a timestamp and date() to convert a timestamp into a formatted string. You learned how to create a timestamp using mktime(), and how to test a date for validity with checkdate().

Additionally, you discovered that MySQL's built-in date and time functions can definitely take some of the load off your application by internally formatting dates and times and performing the date and time arithmetic. The formatting options used for the DATE_FORMAT() function provide a simple method to produce a custom display string from any sort of date field. The DATE_ADD() and DATE_SUB() functions and their numerous available interval types help you determine dates and times in the past or future. Additionally, functions such as DAY(), WEEK(), MONTH(), and YEAR() are useful for extracting parts of dates for use in WHERE or ORDER BY clauses.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin putting your knowledge into practice.

Quiz

1:

Using PHP, how do you acquire a Unix timestamp that represents the current date and time? What about using MySQL?

A1:

In PHP, use time(). In MySQL, use UNIX_TIMESTAMP().

2:

Which PHP function accepts a timestamp and returns an associative array that represents the given date?

A2:

The getdate() function returns an associative array whose elements contain aspects of the given date.

3:

Which PHP function do you use to format date information? What about using MySQL?

A3:

In PHP, use date(). In MySQL, use DATE_FORMAT().

4:

Which PHP function could you use to check the validity of a date?

A4:

You can check a date with the checkdate() function.

Activity

Create a birthday countdown script. Given form input of month, day, and year, output a message that tells the user how many days, hours, minutes, and seconds until the big day. Use whatever combination of PHP and MySQL

Hour 12. Creating a Simple Calendar

This hour continues the date and time lesson from the previous hour, this time in the context of creating a small calendar.

In this hour, you will learn

- How to build a simple calendar script
- How to build a class library to generate date pull-downs in HTML forms

Building a Simple Display Calendar

Let's bring together the date and time functions you learned in the previous hour, and use them to build a calendar that can display the dates for any month between 1980 and 2010. The user will be able to select both month and year with pull-down menus, and the dates for the selected month will be organized according to the days of the week. We will be working with two variables—one for month and one for year—which the user will supply.

These pieces of information will be used to build a timestamp based on the first day of the month defined. If the user input is invalid or absent, we will default to the first day of the current month.

Checking User Input

When the user opens our calendar application for the first time, he or she will not be submitting any information. We must therefore make sure that our script can handle the fact that the variables for month and year may not be defined. We could use the isset() function for this. (isset() returns false if the variable passed to it has not been defined.) However, let's use checkdate() instead. Listing 12.1 shows the fragment of code that checks for month and year variables coming from a form, and builds a timestamp based on them.

Listing 12.1 Checking User Input for the Calendar Script

```
 1: <?php
 2: if (!checkdate($_POST[month], 1, $_POST[year])) {
 3:     $nowArray = getdate();
 4:     $month = $nowArray['mon'];
 5:     $year = $nowArray['year'];
 6: } else {
 7:     $month = $_POST[month];
 8:     $year = $_POST[year];
 9: }
10: $start = mktime (12, 0, 0, $month, 1, $year);
11: $firstDayArray = getdate($start);
12: ?>
```

Listing 12.1 is a fragment of a larger script, so it does not produce any output itself. In the if statement on line 2, we use checkdate() to test whether the month and year have been provided by a form. If they have not been defined, checkdate() returns false because the script cannot make a valid date from undefined month and year arguments. This approach has the added bonus of ensuring that the data submitted by the user constitutes a valid date.

If the date is not valid, we use getdate() on line 3 to create an associative array based on the current time. We then set values for $month and $year ourselves, using the array's mon and year elements (lines 4 and 5). If the variables have been set from the form, we put the data into $month and $year variables so as not to touch the values in the original $_POST superglobal.

Now that we are sure that we have valid data in $month and $year, we can use mktime() to create a timestamp for the first day of the month (line 10). We will need information about this timestamp later on, so on line 11 we create a variable called $firstDayArray that will store an associative array returned by getdate() and based on this timestamp.

Building the HTML Form

We need to create an interface by which users can ask to see data for a month and year. For this, we will use

Creating a Calendar Library

Because dates are ubiquitous in Web interfaces, and because working with dates is often comparatively complicated, let's look at creating a class library to automate some of the work that dates can present. Along the way we will revisit some of the techniques we have already covered.


If the concept of classes is completely foreign to you, you can supplement your knowledge by reading through Chapter 14 of the PHP Manual. It is titled "Classes and Objects," and you can find it at http://www.php.net/manual/en/language.oop.php.

The simple date_pulldown library we will look at was born during the creation of a freelance job listing site. The project necessarily involved the presentation of multiple date pull-downs allowing employers to select both the start and end of contract periods, and for candidates to indicate periods of availability. A date pull-down, in this instance, consists of three separate select elements, one for day of the month, one for month, and one for year.

When a user submits a page, the script will check his input. If there is a problem, we will need to represent the page with the user's input still in place. This is very easy to accomplish with text boxes but is more of a chore with pull-down menus. Pages that display information pulled from a database present a similar problem. Data can be entered straight into the value attributes of text type input elements. Dates will need to be split into month, day, and year values, and then the correct option elements selected.

The date_pulldown class aims to make date pull-downs sticky (so they will remember settings from page to page) and easy to set. In order to create our class, we first need to declare it and create a constructor.


A constructor is a function that exists within a class, and which is automatically called when a new instance of the class is created.

We can also declare some class properties. The following snippet shows the beginning of the class:

```
 1: class date_pulldown
 2:     var $name;
 3:     var $timestamp = -1;
 4:     var $months = array("Jan", "Feb", "Mar", "Apr", "May", "Jun",
 5:           "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
 6:     var $yearstart = -1;
 7:     var $yearend = -1;
 8:
 9:  function date_pulldown($name) {
10:       $this->name = $name;
11:   }
```

We first declare the $name property. This will be used to name the HTML select elements. The $timestamp property will hold a Unix timestamp. The $months array property contains the strings we will display in our month pull-down. The $yearstart and $yearend properties are both set to –1 pending initialization. They will eventually hold the first and last years of the range that will be presented in the year pull-down.

Summary

In this hour, you pulled together the PHP date-related functions you learned about in the previous hour to work within a calendar application. You learned how to test the validity of an input date using checkdate(). You worked through an example script, which applies some of the tools you have looked at, and built a class library that automates some of the more tedious aspects of working with dates in forms.

Q&A

Q1:

Are there any functions for converting between different calendars?

A1:

Yes. PHP provides an entire suite of functions that cover alternative calendars. You can read about these in the official PHP Manual at http://www.php.net/manual/en/ref.calendar.php.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1: What PHP function did we use to check the validity of a date?

A1: checkdate()

2: What PHP function did we use to create a timestamp?

A2: mktime()

3: What PHP function did we use to create an associative array of date-related information?

A3: getdate()

Activity

Use your fancy new date pull-down class in the context of your own form. Create a back-end script that takes the selected dates and displays their input.

Hour 13. Working with Strings

The World Wide Web is very much a plain text environment. No matter how rich Web content becomes, HTML lies behind it all. It is no accident, then, that PHP provides many functions with which you can format, investigate, and manipulate strings. As you might expect, MySQL also comes with its own set of string-related functions, which you will also learn about in this hour.

In this hour, you will learn

- How to format strings
- How to determine the length of a string
- How to find a substring within a string
- How to break a string down into component parts
- How to remove whitespace from the beginning or end of a string
- How to replace substrings
- How to change the case of a string
- How to use MySQL to put strings together or extract pieces of strings
- How to use MySQL to create variations of original strings
- How to use MySQL to find alternate representations of strings, in different bases

Formatting Strings with PHP

Until now, we have simply printed any strings that we want to display directly to the browser. PHP provides two functions that allow you first to apply formatting, whether to round doubles to a given number of decimal places, define alignment within a field, or display data according to different number systems. In this section, you will look at a few of the formatting options provided by printf() and sprintf().

Working with printf()

If you have any experience with C, you will be familiar with the printf() function. The PHP version is similar but not identical. printf() requires a string argument, known as a format control string. It also accepts additional arguments of different types. The format control string contains instructions indicating how to display these additional arguments. The following snippet, for example, uses printf() to output an integer as a decimal:

```
printf("This is my number: %d", 55);
// prints "This is my number: 55"
```

NEW TERM Within the format control string (the first argument), we have included a special code, known as a conversion specification.

A conversion specification begins with a percent (%) symbol and defines how to treat the corresponding argument to printf(). You can include as many conversion specifications as you want within the format control string, as long as you send an equivalent number of arguments to printf().

The following snippet outputs two numbers using printf():

```
printf("First number: %d<br>\nSecond number: %d<br>\n", 55, 66);
// Output:
// First number: 55
// Second number: 66
```

The first conversion specification corresponds to the first of the additional arguments to printf(), which is 55. The second conversion specification corresponds to 66. The d following the percent symbol requires that the data be treated as a decimal integer. This part of a conversion specification is a type specifier.

printf() and Type Specifiers

You have already come across one type specifier, d, which displays data in decimal format. Table 13.1 lists the other available type specifiers.

Table 13.1. Type Specifiers

Specifier	Description
d	Display argument as a decimal number

Investigating Strings in PHP

You do not always know everything about the data that you are working with. Strings can arrive from many sources, including user input, databases, files, and Web pages. Before you begin to work with data from an external source, you often will need to find out more about it. PHP provides many functions that enable you to acquire information about strings.

A Note About Indexing Strings

We will frequently use the word index in relation to strings. You will have come across the word more frequently in the context of arrays. In fact, strings and arrays are not as different as you might imagine. You can think of a string as an array of characters. So you can access individual characters of a string as if they were elements of an array:

```
$test = "scallywag";
print $test[0]; // prints "s"
print $test[2]; // prints "a"
```

It is important to remember, therefore, that when we talk about the position or index of a character within a string, characters, like array elements, are indexed from 0.

Finding the Length of a String with strlen()

You can use strlen() to determine the length of a string. strlen() requires a string and returns an integer representing the number of characters in the variable you have passed it. strlen() might typically be used to check the length of user input. The following snippet tests a membership code to ensure that it is four characters long:

```
if (strlen($membership) == 4) {
    print "Thank you!";
} else {
    print "Your membership number must have 4 digits<P>";
}
```

The user is thanked for his input only if the global variable $membership contains four characters; otherwise, an error message is generated.

Finding a Substring Within a String with strstr()

You can use strstr() to test whether a string exists embedded within another string. strstr() requires two arguments: a source string and the substring you want to find within it. The function returns false if the substring is absent. Otherwise, it returns the portion of the source string beginning with the substring. For the following example, imagine that we want to treat membership codes that contain the string AB differently from those that do not:

```
$membership = "pAB7";
if (strstr($membership, "AB")) {
   print "Thank you. Don't forget that your membership expires soon!";
} else {
   print "Thank you!";
}
```

Because our test variable $membership does contain the string AB, strstr() returns the string AB7. This resolves to

Manipulating Strings with PHP

PHP provides many functions that will transform a string argument, subtly or radically, as you'll soon see.

Cleaning Up a String with trim() and strip_tags()

When you acquire text from the user or a file, you can't always be sure that you haven't also picked up whitespace at the beginning and end of your data. The trim() function shaves any whitespace characters, including newlines, tabs, and spaces, from both the start and end of a string. It accepts the string to be modified, returning the cleaned-up version:

```
$text = "\t\t\tlots of room to breathe          ";
$text = trim( $text );
print $text;
// prints "lots of room to breathe";
```

Of course, removing all the whitespace might be more work than you require. You might want to keep whitespace at the beginning of a string but remove it from the end. You can use PHP's rtrim() function exactly the same as you would use trim(). Only whitespace at the end of the string argument is removed, however:

```
$text = "\t\t\tlots of room to breathe          ";
$text = rtrim( $text );
print $test;
// prints "            lots of room to breathe";
```

PHP provides the ltrim() function to strip whitespace only from the beginning of a string. Once again, this is called with the string you want to transform and returns a new string, shorn of tabs, newlines, and spaces:

```
$text = "\t\t\tlots of room to breathe     ";
$text = ltrim( $text );
print $test;
// prints "lots of room to breathe     ";
```

PHP, by its nature, tends to work with markup text. It is not unusual to have to remove tags from a block to be able to present it without formatting. PHP provides the strip_tags() function for this purpose. The strip_tags() function accepts two arguments. The first is the text to transform. The second argument is optional and should be a list of HTML tags that strip_tags() can leave in place. The tags in the exception list should not be separated by any characters.

```
$string = "I <i>simply</i> will not have it,<br>said Mr Dean<p><b>The end</b>";
print strip_tags($string, "<br><p>");
```

In the preceding code snippet, we create an HTML-formatted string. When we call strip_tags(), we pass it the $string variable and a list of exceptions. The result is that the <p> and
 tags are left in place and all other tags are stripped out.

Replacing a Portion of a String Using substr_replace()

The substr_replace() function works similarly to substr() except that it allows you replace the portion of a string that you extract. The function requires three arguments: the string you are transforming, the text you want to add to it, and

Frequently Used String Functions in MySQL

MySQL's built-in string-related functions can be used several ways. You can use functions in SELECT statements without specifying a table to retrieve a result of the function. Or you can use functions to enhance your SELECT results by concatenating two fields to form a new string.

Even if you never use these functions in your applications, it's good to know they exist, and, if nothing else, you'll get some good practice in this hour using the MySQL monitor's command-line interface.

Length and Concatenation Functions

The group of length and concatenation functions focuses on the length of strings and concatenating strings together. Length-related functions include LENGTH(), OCTET_LENGTH(), CHAR_LENGTH(), and CHARACTER_LENGTH(), which do virtually the same thing—count characters in a string.

```
mysql> select length('This is cool!');
+-------------------------+
| LENGTH('This is cool!') |
+-------------------------+
|                      13 |
+-------------------------+

1 row in set (0.00 sec)
```

The fun begins with the CONCAT() function, which is used to concatenate two or more strings:

```
mysql> select concat('My', 'S', 'QL');
+-------------------------+
| CONCAT('My', 'S', 'QL') |
+-------------------------+
| MySQL                   |
+-------------------------+
1 row in set (0.00 sec)
```

Imagine using this function with a table containing names, split into firstname and lastname fields. Instead of using two strings, use two field names to concatenate the firstname and the lastname fields. By concatenating the fields, you reduce the lines of code necessary to achieve the same result in your application:

```
mysql> select concat(firstname, lastname) from table_name;
+----------------------------+
| CONCAT(firstname, lastname) |
+----------------------------+
| JohnSmith                  |
| JaneSmith                  |
| JimboJones                 |
| AndySmith                  |
| ChrisJones                 |
| AnnaBell                   |
| JimmyCarr                  |
| AlbertSmith                |
| JohnDoe                    |
+----------------------------+
9 rows in set (0.00 sec)
```

Summary

In this hour, you examined some of the functions that enable you to take control of the strings in your PHP scripts. You learned how to format strings with printf() and sprint(). You should be able to use these functions both to create strings that transform data and lay it out. You learned about functions that investigate strings. You should be able to discover the length of a string with strlen(), determine the presence of a substring with strpos(), or extract a substring with substr(). You should be able to tokenize a string with strtok().

You also learned about functions that transform strings. You can now remove whitespace from the beginning or end of a string with trim(), ltrim(), or rtrim(). You can change case with strtoupper(), strtolower(), or ucwords(). You can replace all instances of a string with str_replace().

After learning the PHP methods for string manipulation, you were introduced to MySQL functions that perform actions on strings. If you have strings in MySQL you want to concatenate or for which you want to count characters, you can use functions such as CONCAT(), CONCAT_WS(), and LENGTH(). To pad or remove padding from strings, use RPAD(), LPAD(), TRIM(), LTRIM(), and RRIM() to get just the strings you want. You can also find the location of a string within another, or to return a part of a given string, using the LOCATE(), SUBSTRING(), LEFT(), and RIGHT() functions. Functions such as LCASE(), UCASE(), REPEAT(), and REPLACE() also return variations of the original strings. MySQL also has numerous functions for representing strings, such as ASCII(), BIN(), OCT(), HEX(), and CONV() for converting between bases.

Q&A

Q1:

Are there any other string functions that might be useful to me?

A1:

Yes. PHP has more than 60 string functions! You can read about them all in the PHP Manual online at http://www.php.net/manual/ref.strings.php.

Q2:

Can I use multiple functions in one statement, such as making a concatenated string all uppercase?

A2:

Sure—just be mindful of your opening and closing parentheses. This example shows how to uppercase the concatenated first and last names from the master name table:

```
mysql> SELECT UCASE(CONCAT_WS(' ',
firstname, lastname)) FROM table_name;
+---------------------------------------
----+
| UCASE(CONCAT_WS(' ', firstname,
lastname)) |
+---------------------------------------
----+
| JOHN SMITH
   |
| JANE SMITH
   |
| JIMBO JONES
   |
| ANDY SMITH
   |
| CHRIS JONES
   |
| ANNA BELL
   |
| JIMMY CARR
   |
| ALBERT SMITH
   |
| JOHN DOE
   |
+---------------------------------------
----+
9 rows in set (0.00 sec)
```

If you want to uppercase just the last name, use

```
mysql> SELECT CONCAT_WS(' ', firstname,
UCASE(lastname)) FROM master_name;
```

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

What conversion specifier would you use with printf() to format an integer as a double? Indicate the full syntax required to convert the integer 33.

A1:

You use the conversion specifier f to format an integer as a double:

```
printf("%f", 33);
```

2:

How would you pad the conversion you effected in question 1 with zeros so that the part before the decimal point is four characters long?

A2:

You can pad the output from printf() with the padding specifier—that is, a space or a zero followed by a number representing the number of characters you want to pad by.

```
printf("%04f", 33);
```

3:

How would you specify a precision of two decimal places for the floating-point number you have been formatting in the previous questions?

A3:

The precision specifier consists of a dot (.) followed by a number representing the precision you want to apply. It should be placed before the conversion specifier:

```
printf("%04.2f", 33);
```

Hour 14. Creating a Simple Discussion Forum

Although the ultimate goal of this hour is to create a simple discussion forum, a majority of the hour is devoted to understanding the thought processes behind designing an application using a relational database.

In this hour, you will learn

- Three types of table relationships
- How to normalize your database
- How to implement a good database design process
- How to create the tables, input forms, and display of a simple discussion forum

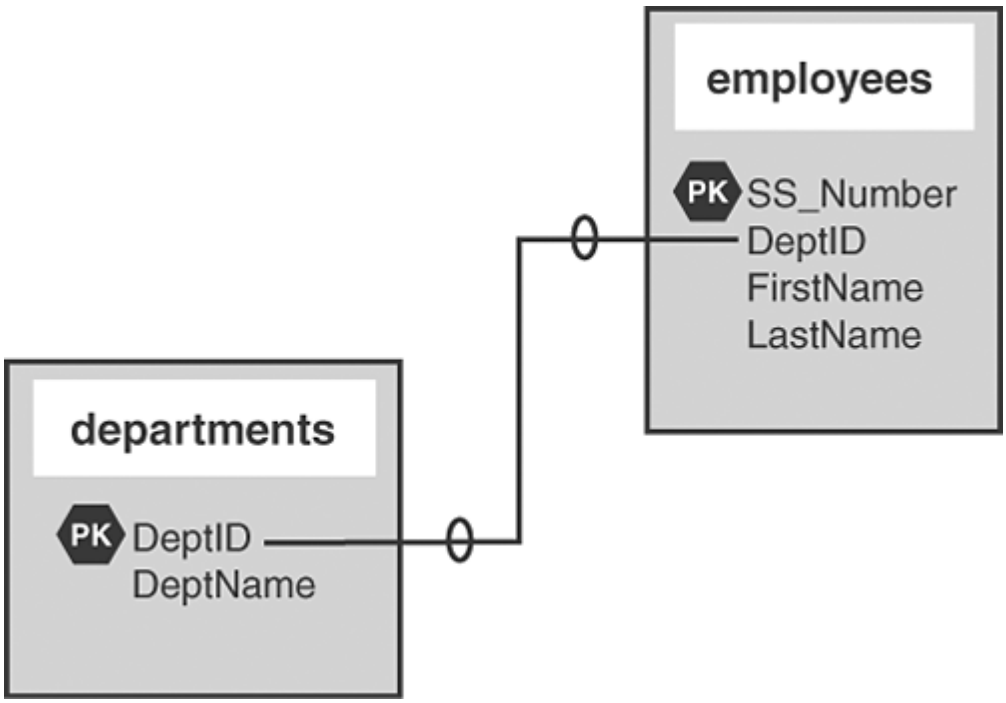
Types of Table Relationships

Table relationships come in several forms:

- One-to-one relationships
- One-to-many relationships
- Many-to-many relationships

NEW TERM For example, suppose you have a table called employees that contains each person's Social Security number, name, and the department in which he or she works. Suppose you also have a separate table called departments, containing the list of all available departments, made up of a Department ID and a name. In the employees table, the Department ID field matches an ID found in the departments table. You can see this type of relationship in Figure 14.1. The "PK" next to the field named stands for primary key, which you'll learn about during this hour.

Figure 14.1. The employees and departments tables are related through the DeptID.



In the following sections, you will take a closer look at each of the relationship types.

One-to-One Relationships

In a one-to-one relationship, a key appears only once in a related table. The employees and departments tables do

Understanding Normalization

Normalization is simply a set of rules that will ultimately make your life easier when you're acting as a database administrator. It's the art of organizing your database in such a way that your tables are related where appropriate and flexible for future growth.

The sets of rules used in normalization are called normal forms. If your database design follows the first set of rules, it's considered in the first normal form. If the first three sets of rules of normalization are followed, your database is said to be in the third normal form.

Throughout this hour, you'll learn about each rule in the first, second, and third normal forms and, we hope, will follow them as you create your own applications. You'll be using a sample set of tables for a students and courses database and taking it to the third normal form.

Problems with the Flat Table

Before launching into the first normal form, you have to start with something that needs to be fixed. In the case of a database, it's the flat table. A flat table is like a spreadsheet—it has many, many columns. There are no relationships between multiple tables; all the data you could possibly want is right there in that flat table. This scenario is inefficient and consumes more physical space on your hard drive than a normalized database.

In your students and courses database, assume you have the following fields in your flat table:

- StudentName— The name of the student.
- CourseID1— The ID of the first course taken by the student.
- CourseDescription1— The description of the first course taken by the student.
- CourseInstructor1— The instructor of the first course taken by the student.
- CourseID2— The ID of the second course taken by the student.
- CourseDescription2— The description of the second course taken by the student.
- CourseInstructor2— The instructor of the second course taken by the student.

Following the Design Process

The greatest problem in application design is a lack of forethought. As it applies to database-driven applications, the design process must include a thorough evaluation of your database—what it should hold, how data relates to each other, and most importantly, whether it is scalable.

The general steps in the design process are

Define the objective.

Design the data structures (tables, fields).

Discern relationships.

Define and implement business rules.

Create the application.

Creating the application is the last step—not the first! Many developers take an idea for an application, build it, and then go back and try to make a set of database tables fit into it. This approach is completely backward, inefficient, and will cost a lot of time and money.

Before you start any application design process, sit down and talk it out. If you can't describe your application—including the objectives, audience, and target market—then you're not ready to build it, let alone model the database.

After you can describe the actions and nuances of your application to other people and have it make sense to them, you can start thinking about the tables you want to create. Start with big flat tables because, after you write them down, your newfound normalization skills will take over. You will be able to find your redundancies and visualize your relationships.

The next step is to do the normalization. Go from flat table to first normal form and so on up to the third normal form if possible. Use paper, pencils, sticky notes, or whatever helps you to visualize the tables and relationships. There's no shame in data modeling on sticky notes until you're ready to create the tables themselves. Plus, using them is a lot cheaper than buying software to do it for you; software ranges from one hundred to several thousands of dollars!

After you have a preliminary data model, look at it from the application's point of view. Or look at it from the point of view of the person using the application you're building. This is the point where you define business rules and see whether your data model will break. An example of a business rule for an online registration application is, "Each user must have one e-mail address, and it must not belong to any other user." If EmailAddress wasn't a unique field in your data model, your model would be broken based on the business rule.

Creating a Discussion Forum

In the following sections, you'll learn the design process behind a simple discussion forum. This includes developing the database tables, user input forms, and display of the results. When broken into pieces like this, such a task seems quite simple—and it is!

Designing the Database Tables

Think of the basic components of a forum: topics and posts. A forum—if properly used by its patrons—should have several topics, and each of those topics will have one or more posts by users. Knowing that, you should realize that the posts are tied to the topics through a key field. This key forms the relationship between the two tables.

Think about the requirements for the topics themselves. You definitely need a field for the title, and subsequently you may want fields to hold the creation time and the identification of the user who created the topic. Similarly, think of the requirements for the posts: You want the text of the post, the time it was created, and the person creating it. Most importantly, you need that key to tie the post to the topic.

The following two table creation statements create these tables, called forum_topics and forum_posts:

```
mysql> create table forum_topics (
    -> topic_id int not null primary key auto_increment,
    -> topic_title varchar (150),
    -> topic_create_time datetime,
    -> topic_owner varchar (150)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> create table forum_posts (
    -> post_id int not null primary key auto_increment,
    -> topic_id int not null,
    -> post_text text,
    -> post_create_time datetime,
    -> post_owner varchar (150)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

In this forum example, we will identify users by their e-mail addresses and not require any sort of login sequence. In the activity at the end of this hour, you'll be given some hints on extending this forum example to fit within an environment of registered users.

You should now have two empty tables, waiting for some input. In the next section, you'll create the input forms for adding a topic and a post.

Creating the Input Forms and Scripts

Before you can add any posts, you must add a topic to the forum. It is common practice in forum creation to add the topic and the first post in that topic at the same time. From a user's point of view, it doesn't make much sense to add a topic and then go back, select the topic, and add a reply. You want the process to be as smooth as possible.

Summary

Following proper database design is the only way your application will be efficient, flexible, and easy to manage and maintain. An important aspect of database design is to use relationships between tables instead of throwing all your data into one long flat file. Types of relationships include one-to-one, one-to-many, and many-to-many.

Using relationships to properly organize your data is called normalization. There are many levels of normalization, but the primary levels are the first, second, and third normal forms. Each level has a rule or two that you must follow. Following all the rules helps ensure that your database is well organized and flexible.

To take an idea from inception through to fruition, you should follow a design process. This process essentially says "think before you act." Discuss rules, requirements, and objectives; then create the final version of your normalized tables. In this hour, you applied this knowledge to the creation of a simple discussion form, using PHP and MySQL to create input forms and display pages for topics and posts.

Q&A

Q1:

Are there only three normal forms?

A1:

No, there are more than three normal forms. Additional forms are the Boyce-Codd normal form, fourth normal form, and fifth normal form/Join-Projection normal form. These forms are not often followed because the benefits of doing so are outweighed by the cost in man-hours and database efficiency.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1: Name three types of data relationships.

A1: One-to-one, one-to-many, many-to-many.

2: Because many-to-many relationships are difficult to represent in an efficient database design, what should you do?

A2: Create a series of one-to-many relationships using intermediary mapping tables.

Activity

Explain each of the three normal forms to a person who works with spreadsheets and flat tables.

Hour 15. Restricting Access to Your Applications

This hour explains how to use Apache to restrict access to parts of a Web site based on the identity of the user or on information about the request. On the application side, you can create your own mechanism for user validation and check the validity of your users through cookies.

In this hour, you will learn

- How to restrict access based on the user, client IP address, domain name, and browser version
- How to use the user management tools provided with Apache
- How to store and retrieve cookie information
- How to use cookies for authentication

Authentication Overview

Authorization and authentication are common requirements for many Web sites. Authentication establishes the identity of parties in a communication.

You can authenticate yourself by something you know (a password, a cookie), something you have (an ID card, a key), something you are (your fingerprint, your retina), or a combination of these elements. In the context of the Web, authentication is usually restricted to the use of passwords and certificates.

Authorization deals with protecting access to resources. You can authorize based on several factors, such as the IP address the user is coming from, the user's browser, the content the user is trying to access, or who the user is (which is previously determined via authentication).

Apache includes several modules that provide authentication and access control and that can be used to protect both dynamic and static content. You can either use one of these modules or implement your own access control at the application level and provide customized login screens, single sign-on, and other advanced functionality.

Client Authentication

Users are authenticated for tracking or authorization purposes. The HTTP specification provides two authentication mechanisms: basic. and digest. In both cases, the process is the following:

1. A client tries to access restricted content in the Web server.
2. Apache checks whether the client is providing a username and password. If not, Apache returns an HTTP 401 status code, indicating user authentication is required.
3. The client reads the response and prompts the user for the required username and password (usually with a pop-up window).
4. The client retries accessing the Web page, this time transmitting the username and password as part of the HTTP request. The client remembers the username and password and transmits them in later requests to the same site, so the user does not need to retype them for every request.
5. Apache checks the validity of the credentials and grants or denies access based on the user identity and other access rules.

In the basic authentication scheme, the username and password are transmitted in clear text, as part of the HTTP request headers. This poses a security risk because an attacker could easily peek at the conversation between server and browser, learn the username and password, and reuse them freely afterward.

The digest authentication provides increased security because it transmits a digest instead of the clear text password.

Apache Authentication Module Functionality

Apache provides the basic framework and directives to perform authentication and access control. The authentication modules provide support for validating passwords against a specific back end. Users can optionally be organized in groups, easing management of access control rules.

Apache provides three built-in directives related to authentication that will be used with any of the authentication modules: AuthName, AuthType, and Require.

AuthName accepts a string argument, the name for the authentication realm. A realm is a logical area of the Web server that you are asking the password for. It will be displayed in the browser pop-up window.

AuthType specifies the type of browser authentication: basic or digest.

Require enables you to specify a list of users or groups that will be allowed access. The syntax is Require user followed by one or more usernames, or Require group followed by one or more group names. For example:

```
Require user joe bob
```

or

```
Require group employee contractor
```

If you want to grant access to anyone who provides a valid username and password, you can do so with

```
Require valid-user
```

With the preceding directives, you can control who has access to specific virtual hosts, directories, files, and so on. Although authentication and authorization are separate concepts, in practice they are tied together in Apache. Access is granted based on specific user identity or group membership. Some third-party modules, such as certain LDAP-based modules, allow for clearer separation between authentication and authorization.

The authentication modules included with Apache provide

- Back-end storage— Provide text or database files containing the username and group information
- User management— Supply tools for creating and managing users and groups in the back-end storage
- Authoritative information— Specify whether the results of the module are authoritative

Using Apache for Access Control

The mod_access module, enabled by default, allows you to restrict access to resources based on parameters of the client request, such as the presence of a specific header or the IP address or hostname of the client.

Implementing Access Rules

You can specify access rules using the Allow and Deny directives. Each of these directives takes a list of arguments such as IP addresses, environment variables, and domain names.

Allow/Deny Access by IP Addresses

You can deny or grant access to a client based on its IP address:

```
Allow from 10.0.0.1 10.0.0.2 10.0.0.3
```

You can also specify IP address ranges, with a partial IP address or a network/mask pair. Additionally, you can specify the first one, two, or three bytes of an IP address. Any IP address containing those will match this rule. For example, the rule

```
Deny from 10.0
```

will match any address starting with 10.0, such as 10.0.1.0 and 10.0.0.1.

You can also utilize the IP address and the netmask; the IP address specifies the network and the mask specifies which bits belong to the network prefix and which ones belong to the nodes. The rule

```
Allow from 10.0.0.0/255.255.255.0
```

will match IP addresses 10.0.0.1, 10.0.0.2, and so on, to 10.0.0.254.

You can also specify the network mask via high-order bits. For example, you could write the previous rule as

```
Allow from 10.0.0.0/24
```

Allow/Deny Access by Domain Name

You can control access based on specific hostnames or partial domain names. For example, Allow from example.com will match www.example.com, foo.example.com, and so on.

Enabling access rules based on domain names forces Apache to do a reverse DNS lookup on the client address, bypassing the settings of the HostNameLookups directive. This has performance implications.

Combining Apache Access Methods

In previous sections, you learned how to restrict access based on user identity or request information. The Satisfy directive enables you to determine whether both types of access restrictions must be satisfied in order to grant access. Satisfy accepts one parameter, which can be either all or any.

Satisfy all means that the client will be granted access if it provides a valid username and password and passes the access restrictions. Satisfy any means the client will be granted access if it provides a valid username and password or passes the access restrictions.

Why is this directive useful? For example, you might want to provide free access to your Web site to users coming from an internal, trusted address, but require users coming from the Internet to provide a valid username and password. Listing 15.4 demonstrates just that.

Listing 15.4 Mixing Authentication and Access Control Rules

```
1: <Location /restricted>
2: Allow from 10.0.0.0/255.255.255.0
3: AuthType Basic
4: AuthName "Intranet"
5: AuthUserFile /usr/local/apache2/conf/htusers
6: AuthAuthoritative on
7: Require valid-user
8: Satisfy any
9: </Location>
```

Access control based on connection or request information is not completely secure. Although it provides an appropriate level of protection for most cases, the rules rely on the integrity of your DNS servers and your network infrastructure. If an attacker gains control of your DNS servers, or your routers or firewalls are incorrectly configured, he can easily change authorized domain name records to point to his machine or pretend he is coming from an authorized IP address.

Limiting Access Based on HTTP Methods

In general, you want your access control directives to apply to all types of client requests, and this is the default behavior. In some cases, however, you want to apply authentication and access rules to only certain HTTP methods such as GET and HEAD.

The <Limit> container takes a list of methods and contains the directives that apply to requests containing those methods. The complete list of methods that can be used is GET, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, and UNLOCK.

The <LimitExcept> section provides complementary functionality, containing directives that will apply to requests not containing the listed methods.

Listing 15.5 shows an example from the default Apache configuration file. The <Limit> and <LimitExcept> sections allow read-only methods but deny requests to any other methods that can modify the content of the file system, such as PUT.

Listing 15.5 Restricting Access Based on Rule

```
 1: <Directory /home/*/public_html>
 2:     AllowOverride FileInfo AuthConfig Limit
 3:     Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
 4:     <Limit GET POST OPTIONS PROPFIND>
 5:         Order Allow,Deny
 6:         Allow from all
 7:     </Limit>
 8:     <LimitExcept GET POST OPTIONS PROPFIND>
 9:         Order Deny,Allow
10:         Deny from all
11:     </LimitExcept>
12: </Directory>
```

In the next section, you'll learn about restricting access on the application side based on information found in cookies.

Introducing Cookies

On the application side, you can use cookies in your PHP scripts to control access to certain areas of your Web site. A cookie is a small amount of data stored by the user's browser in compliance with a request from a server or script. A host can request that up to 20 cookies be stored by a user's browser. Each cookie consists of a name, value, and expiry date, as well as host and path information. An individual cookie is limited to 4KB.

After a cookie is set, only the originating host can read the data, ensuring that the user's privacy is respected. Furthermore, the user can configure her browser to notify her of all cookies set, or even to refuse all cookie requests. For this reason, cookies should be used in moderation and should not be relied on as an essential element of an environment design without first warning the user.

The Anatomy of a Cookie

A PHP script that sets a cookie might send headers that look something like this:

```
HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 13:39:58 GMT
Server: Apache/1.3.26 (Unix) PHP/4.2.3
X-Powered-By: PHP/4.2.3
Set-Cookie: vegetable=artichoke; path=/; domain=yourdomain.com
Connection: close
Content-Type: text/html
```

As you can see, this Set-Cookie header contains a name/value pair, path, and domain. The name and value will be URL encoded. Should it be present, an expires field is an instruction to the browser to "forget" the cookie after the given time and date. The path field defines the position on a Web site below which the cookie should be sent back to the server. The domain field determines the Internet domains to which the cookie should be sent. The domain cannot be different from the domain from which the cookie was sent, but can nonetheless specify a degree of flexibility. In the preceding example, the browser will send the cookie to the server yourdomain.com and the server www.yourdomain.com.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
Host: www.yourdomain.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en,pdf
Accept-Charset: iso-8859-1,*,utf-8
Cookie: vegetable=artichoke
```

A PHP script will then have access to the cookie in the environment variable HTTP_COOKIE or as part of the $_COOKIE superglobal:

```
print "$_SERVER[HTTP_COOKIE]<BR>";     // prints "vegetable=artichoke"
print getenv("HTTP_COOKIE")."<BR>";    // prints "vegetable=artichoke"
print $_COOKIE['vegetable']."<BR>";    // prints "artichoke"
```

Setting a Cookie with PHP

You can set a cookie in a PHP script in two ways. You can use the header() function to set the Set-Cookie header. The header() function requires a string that will then be included in the header section of the server response. Because headers are sent automatically for you, header() must be called before any output at all is sent to the browser:

```
header ("Set-Cookie: vegetable=artichoke; expires=Wed, 19-Sep-02 14:39:58 GMT;
path=/; domain=yourdomain.com");
```

Although not difficult, this method of setting a cookie would require you to build a function to construct the header string. Formatting the date as in this example and URL encoding the name/value pair would not be a particularly arduous task. It would, however, be an exercise in wheel reinvention because PHP provides a function that does just that.

The setcookie() function does what the name suggests—it outputs a Set-Cookie header. For this reason, it should be called before any other content is sent to the browser. The function accepts the cookie name, cookie value, expiry date in Unix epoch format, path, domain, and integer that should be set to 1 if the cookie is only to be sent over a secure connection. All arguments to this function are optional apart from the first (cookie name) parameter.

Listing 15.6 uses setcookie() to set a cookie.

Listing 15.6 Setting and Printing a Cookie Value

```
 1:     <?php
 2:     setcookie("vegetable", "artichoke", time()+3600, "/", "yourdomain.com", 0);
 3:     ?>
 4:     <html>
 5:     <head>
 6:     <title>Listing 15.6 Setting and printing a cookie value</title>
 7:     </head>
 8:     <body>
 9:     <?php
10:     if (isset($_COOKIE[vegetable])) {
11:         print "<p>Hello again, your chosen vegetable is $_COOKIE[vegetable]</p>";
12:     } else {
13:         print "<p>Hello you. This may be your first visit</p>";
14:     }
15:     ?>
16:     </body>
17:     </html>
```

Even though we set the cookie (line 2) when the script is run for the first time, the $_COOKIE[vegetable] variable will not be created at this point. A cookie is read only when the browser sends it to the server. This will not happen until the user revisits a page in your domain. We set the cookie name to "vegetable" on line 2 and the cookie value to "artichoke". We use the time() function to get the current time stamp and add 3600 to it (there are 3600 seconds in an hour). This total represents our expiry date. We define a path of "/", which means that a cookie should be sent for any page within our server environment. We set the domain argument to "yourdomain.com", which means that a cookie will be sent to any server in that group. Finally, we pass 0 to setcookie(), signaling that cookies can be sent in an insecure environment.

Passing setcookie() an empty string ("") for string arguments or 0 for integer fields will cause these arguments to be skipped.

Restricting Access Based on Cookie Values

Now for the fun part—using your cookie skills to restrict access to parts of your Web site! Suppose you created a login form that checked for values against a database. If the user is authorized, you send a cookie that says as much. Then, for all pages you want to restrict only to authorized users, you check for the specific cookie. If the cookie is present, the user can see the page. If the cookie is not present, the user is either sent back to the login form, or a message regarding access restrictions can be printed to the screen.

We'll go through each of these steps in the next few sections.

Creating the Authorized Users Table

When you're integrating user accounts into a Web-based application, it is most common to store the user-specific information in a database table. The information in this table can then be used to authorize the user and grant access to areas of the site that are specifically for these "special" users.

The following table creation command will create a table called auth_users in your MySQL database, with fields for the ID, first name, last name, email address, username, and password:

```
create table auth_users (
       id int not null primary key auto_increment,
       f_name varchar(50),
       l_name varchar(50),
       email varchar(150),
       username varchar(25),
       password varchar (75)
);
```

The following INSERT command puts a record in the auth_users table for a user named John Doe, with an email address of john@doe.com, a username of jdoe, and a password of doepass:

```
mysql> insert into auth_users values ('', 'John', 'Doe', 'john@doe.com',
    -> 'jdoe', password('doepass'));
Query OK, 1 row affected (0.00 sec)
```

This INSERT command should be self-explanatory, with the exception of the use of the password() function. When this function is used in the INSERT command, what is stored in the table is in fact not the actual password, but a hash of the password.

When you view the contents of the auth_users table, you will see the hash in the password field, as follows:

```
mysql> select * from auth_users;
+ - - + - - - - + - - - - + - - - - - - - - + - - - - - + - - - - - - - - - - - - +
| id  | f_name  | l_name  | email          | username  | password           |
+ - - + - - - - + - - - - + - - - - - - - - + - - - - - + - - - - - - - - - - - - +
|  1  | John    | Doe     | john@doe.com   | jdoe      | 2fae5c9d478ec4b1   |
+ - - + - - - - + - - - - + - - - - - - - - + - - - - - + - - - - - - - - - - - - +
1 row in set (0.01 sec)
```

NEW TERM

Summary

This hour explained how to use Apache features to restrict access to your Web site based on the identity of the remote user and information from the HTTP request or network connection. It also covered some authentication modules included with Apache and additional tools that you can use to create and manage your user and group databases.

Additionally, you were introduced to using cookies and learned to use the setcookie() function to set cookies on the user's browser. You then learned to use cookie values to allow access to specific parts of your PHP application.

Q&A

Q1:

I have a Unix system. Can I use /etc/passwd as my user database?

A1:

Although using /etc/passwd might seem convenient, it is advisable that you do not use the existing /etc/passwd file for authenticating users of your Web site. Otherwise, an attacker who gains access to a user of your Web site will also gain access to the system. Keep separate databases and encourage users to choose different passwords for their system accounts and Web access. Periodically run password checkers that scan for weak passwords and accounts in which the username is also the password.

Q2:

Why am I asked for my password twice in some Web sites?

A2:

Your browser keeps track of your password so that you do not have to type it for every request. The stored password is based on the realm (AuthName directive) and the hostname of the Web site. Sometimes you can access a Web site via different names, such as yourdomain.com and www.yourdomain.com. If you are authorized to access a certain restricted area of yourdomain.com but are redirected or follow a link to www.yourdomain.com, you will be asked again to provide the username and password because your browser thinks it is a completely different Web site.

Q3:

Are there any serious security or privacy issues raised by cookies?

A3:

A server can access a cookie set only from its own domain. Although a cookie can be stored on the user's hard drive, there is no other access to the user's file system. It is possible, however, to set a cookie in response to a request for an image. So if many sites include images served from a third-party ad server or counter script, the third party may be able to track a user across multiple domains.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

What are the advantages of database files over plain text files for storing user authentication information?

A1:

Database files are much more scalable because they can be indexed. This means that Apache does not need to read the file sequentially until a match is found for a particular user, but rather can jump to the exact location.

2:

Can you name some disadvantages of HTTP basic authentication?

A2:

One disadvantage is that the information is transmitted in clear text over the network. This means that unless you are using SSL, it is possible for an attacker to read the packets your browser sends to the server and steal your password. Another disadvantage is that HTTP authentication does not provide a means for customizing the login (except the realm name). It is very common for Web sites to implement custom login mechanisms using HTML forms and cookies.

3:

What function is designed to allow you to set a cookie on a visitor's browser?

A3:

The setcookie() function allows you to set a cookie (although you could also output a Set-Cookie header using the header() function).

Activity

Practice using the various types of authentication—both server-based and with PHP—on your development server.

Hour 16. Working with User Sessions

In Hour 15, "Restricting Access to Your Applications," we looked at using cookies to store user-related values, but once again, PHP is one step ahead of us. PHP contains numerous functions for managing user sessions, which can be stored in the $_SESSION superglobal. Sessions use techniques similar to those explored in the preceding hour but build them into the language; thus, saving state is as easy as calling a function.

In this hour, you will learn

- What session variables are and how they work
- How to start or resume a session
- How to store variables in a session
- How to destroy a session
- How to unset session variables

Session Function Overview

Session functions implement a concept that you have already seen; that is, the provision to users of a unique identifier, which can then be used from access to access to acquire information linked to that ID. The difference is that most of the work is already done for you. When a user accesses a session-enabled page, the user is either allocated a new identifier or re-associated with one that was already established in a previous access. Any variables that have been associated with the session will become available to your code, through the $_SESSION superglobal.

When you use sessions, cookies are used by default to store the session identifier, but you can ensure success for all clients by encoding the session ID into all links in your session-enabled pages.

Session state is usually stored in a temporary file, though you can implement database storage using a function called session_set_save_handler(). The use of session_set_save_handler() is beyond the scope of this book, but you can find more information at http://www.php.net/session-set-save-handler.

Starting a Session

To work with a session, you need to explicitly start or resume that session unless you have changed your php.ini configuration file. By default, sessions do not start automatically. If you want to start a session this way, you will have to find the following line in your php.ini file and change the value from 0 to 1 (and restart the Web server):

```
session.auto_start = 0
```

By changing the value of session.auto_start to 1, you ensure that a session is initiated for every PHP document. If you don't change this setting, you need to call the session_start() function in each script.

After a session is started, you instantly have access to the user's session ID via the session_id() function. session_id() allows you to either set or get a session ID. Listing 16.1 starts a session and prints the session ID to the browser.

Listing 16.1 Starting or Resuming a Session

```
 1: <?php
 2: session_start();
 3: ?>
 4: <html>
 5: <head>
 6: <title>Listing 16.1 Starting or resuming a session</title>
 7: </head>
 8: <body>
 9: <?php
10: print "<p>Your session ID is ".session_id()."</p>\n\n";
11: ?>
12: </body>
13: </html>
```

When this script is run for the first time from a browser, a session ID is generated by the session_start() function call on line 2. If the page is later reloaded or revisited, the same session ID is allocated to the user. This action assumes that the user has cookies enabled. For example, when I run this script the first time, the output is

```
Your session ID is fa963e3e49186764b0218e82d050de7b
```

When I reload the page, the output is still

```
Your session ID is fa963e3e49186764b0218e82d050de7b
```

because I have cookies enabled and the session ID still exists.

Because start_session() attempts to set a cookie when initiating a session for the first time, it is imperative that you call this function before you output anything else at all to the browser. If you do not follow this rule, your session will not be set, and you will likely see warnings on your page.

Sessions remain current as long as the Web browser is active. When the user restarts the browser, the cookie is no longer stored. You can change this behavior by altering the session.cookie_lifetime setting in your php.ini file. The default value is 0, but you can set an expiry period in seconds.

Working with Session Variables

Accessing a unique session. identifier in each of your PHP documents is only the start of session functionality. When a session is started, you can store any number of variables in the $_SESSION superglobal and then access them on any session-enabled page.

If you are using a pre-4.1.x version of PHP, the $_SESSION superglobal is not present, and session functionality is much different. If you cannot upgrade to the current version of PHP, read the PHP manual section on sessions, which includes notes for early releases.

Listing 16.2 adds two variables into the. $_SESSION superglobal: product1 and product2 (lines 10 and 11).

Listing 16.2 Storing Variables in a Session

```
 1: <?php
 2: session_start();
 3: ?>
 4: <html>
 5: <head>
 6: <title>Listing 16.2 Storing variables in a session</title>
 7: </head>
 8: <body>
 9: <?php
10: $_SESSION[product1] = "Sonic Screwdriver";
11: $_SESSION[product2] = "HAL 2000";
12: print "The products have been registered.";
13: ?>
14: </body>
15: </html>
```

The magic in Listing 16.2 will not become apparent until the user moves to a new page. Listing 16.3 creates. a separate PHP script that accesses the variables stored in the $_SESSION superglobal in Listing 16.2.

Listing 16.3 Accessing Stored Session Variables

```
 1: <?php
 2: session_start();
 3: ?>
 4: <html>
 5: <head>
 6: <title>Listing 16.3 Accessing stored session variables</title>
 7: </head>
 8: <body>
 9: <?php
10: print "Your chosen products are:\n\n";
11: print "<ul><li>$_SESSION[product1]\n<li>$_SESSION[product2]\n</ul>\n";
12: ?>
13: </body>
14: </html>
```

Figure 16.1 shows the output from Listing 16.3. As you can see, we have access to the $_SESSION[product1] and $_SESSION[product2] variables in an entirely new page.

Passing Session IDs in the Query String

So far you have relied on a cookie to save the session ID between script requests. On its own, this method is not the most reliable way of saving state because you cannot be sure that the browser will accept cookies. You can build in a failsafe, however, by passing the session ID from script to script embedded in a query string. PHP makes a name/value pair available in a constant called SID if a cookie value for a session ID cannot be found. You can add this string to any HTML links in session-enabled pages:

```
<a href="anotherpage.html?<?php print SID; ?>">Another page</a>
```

It will reach the browser as

```
<a href="anotherpage.html?PHPSESSID=08ecedf79fe34561fa82591401a01da1">Another
page</a>
```

The session ID passed in this way will automatically be recognized in the target page when session_start() is called, and you will have access to session variables in the usual way.

Destroying Sessions and Unsetting Variables

You can use session_destroy() to end a session, erasing all session variables. The session_destroy() function requires no arguments. You should have an established session for this function to work as expected. The following code fragment resumes a session and abruptly destroys it:

```
session_start();
session_destroy();
```

When you move on to other pages that work with a session, the session you have destroyed will not be available to them, forcing them to initiate new sessions of their own. Any registered variables will be lost.

The session_destroy() function does not instantly destroy registered variables, however. They remain accessible to the script in which session_destroy() is called (until it is reloaded). The following code fragment resumes or initiates a session and registers a variable called test, which we set to 5. Destroying the session does not destroy the registered variable.

```
session_start();
$_SESSION[test] = 5;
session_destroy();
print $_SESSION[test]; // prints 5
```

To remove all registered variables from a session, you simply unset the variable:

```
session_start();
$_SESSION[test] = 5;
session_destroy();
unset($_SESSION[test]);
print $_SESSION[test]; // prints nothing.
```

Summary

In this hour, you looked at different ways of saving state in a stateless protocol. All methods use some combination of cookies and query strings, sometimes combined with the use of files or databases. These approaches all have their benefits and problems.

You learned that a cookie alone is not intrinsically reliable and cannot store much information. On the other hand, it can persist over a long period of time. Approaches that write information to a file or database involve some cost to speed and might become a problem on a popular site. Nonetheless, a simple ID can unlock large amounts of data stored on disk. To ensure that as many users as possible get the benefit of your session-enabled environment, you can use the SID constant to pass a session ID to the server as part of a query string.

With regards to sessions themselves, you learned how to initiate or resume a session with session_start(). When in a session, you learned how to add variables to the $_SESSION superglobal, check that they exist, unset them if you want, and destroy the entire session.

Q&A

Q1:

Should I be aware of any pitfalls with session functions?

A1:

The session functions are generally reliable. However, remember that cookies cannot be read across multiple domains, so if your project uses more than one domain name on the same server (perhaps as part of an e-commerce environment), you might need to consider disabling cookies for sessions by setting the

```
session.use_cookies
```

directive to 0 in the php.ini file.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1: Which function would you use to start or resume a session?

A1: You can start a session by using the session_start() function.

2: Which function contains the current session's ID?

A2: You can access the session's ID by using the session_id() function.

3: How would you end a session and erase all traces of it for future visits?

A3: The session_destroy() function removes all traces of a session for future requests.

4: What does the SID constant return?

A4: If cookies are not available, the SID constant contains a name/value pair that can be incorporated in a query string. It will pass the session ID from script request to script request.

Activity

Create a script that uses session functions to remember which pages in your environment the user has visited. Provide the user with a list of links on each page to make it easy for her to retrace her steps.

Hour 17. Logging and Monitoring Server Activity

This hour describes how the logging system in Apache works and how you can customize it—which information to store and where to do it. Additionally, you will learn to use PHP and MySQL to log specific items of interest to you, outside the realm of the Apache log files.

In this hour, you will learn how to

- Understand Apache log formats and logging levels
- Rotate and analyze Apache logs
- Interpret common errors that might appear in your logs
- Create scripts that log specific items to database tables
- Create custom reports based on these logging tables

Standard Apache Access Logging

Using Apache's basic logging features, you can keep track of who visits your Web sites by logging accesses to the servers hosting them. You can log every aspect of the requests and responses, including the IP address of the client, user, and resource accessed. You need to take three steps to create a request log:

1. Define what you want to log—your log format.
2. Define where you want to log it—your log files, a database, an external program.
3. Define whether or not to log—conditional logging rules.

Deciding What to Log

You can log nearly every aspect associated with the request. You can define how your log entries look by creating a log format. A log format is a string that contains text mixed with log formatting directives. Log formatting directives start with a % and are followed by a directive name or identifier, usually a letter indicating the piece of information to be logged. When Apache logs a request, it scans the string and substitutes the value for each directive. For example, if the log format is This is the client address %a, the log entry is something like This is the client address 10.0.0.2. That is, the logging directive %a is replaced by the IP address of the client making the request. Table 17.1 provides a comprehensive list of all formatting directives.

Table 17.1. Log Formatting Directives

Formatting Options	Explanation
Data from the Client	
%a	Remote IP address, from the client.
%h	Hostname or IP address of the client making the request. Whether the hostname is logged depends on two factors: The IP address of the client must be able to resolve to a hostname using a reverse DNS lookup, and Apache must be configured to do that lookup using the HostNameLookups directive, explained later in this hour. If these conditions are not met, the IP address of the client will be logged instead.
%l	Remote user, obtained via the identd protocol. This option is not very useful because this protocol is not supported on the majority of the client machines, and the

Standard Apache Error Logging

Apache can be configured to log error messages and debug information. In addition to errors generated by Apache itself, CGI errors also will be logged.

Each error log entry is prefixed by the time the error occurred and the client IP address or hostname, if available. As with HTTP request logging, you can log error information to a file or program. On Unix systems, you can also log to the syslog daemon. Modules for Apache 1.3 allow you to log to the Windows event log and will likely be ported to Apache 2.0 over time.

You can use the ErrorLog directive to define where you want your logs to go. This directive takes one argument, which can be a file, a program, or the syslog daemon.

Logging Errors to a File

A file argument indicates the path to the error log file. If the path is relative, it is assumed to be relative to the server root. By default, the error log file will be located in the logs directory and will be named error_log on Unix and error.log on Windows. The following is an example:

```
ErrorLog logs/my_error_log
```

Logging Errors to a Program

You can specify the path to a program, prefixed by a pipe |. Apache will log errors to the standard input of the program, and the program will further process them. The following is an example:

```
ErrorLog "|/usr/local/bin/someprogram"
```

The syslog Daemon Argument

On a Unix system, if you specify syslog as an argument, you can log error messages to the Unix system log daemon syslogd. By default, log errors are logged to the syslog facility local7. The facility is the part of the system generating the error. You can specify a facility by providing syslog:facility as an argument. Examples of syslog facilities are mail, uucp, local0, local1, and so on. For a complete list, look at the documentation for syslog included with your system (try man syslogd or man syslogd.conf at the command line). The following is an example of logging to syslog:

```
ErrorLog syslog:local6
```

The LogLevel Directive

The error information provided by Apache has several degrees of importance. You can choose to log only important messages and disregard informational or trivial warning messages. The LogLevel directive takes an error-level argument. Only errors of that level of importance or higher will be logged.

Table 17.2 specifies the valid values for the LogLevel directive, as specified by the Apache documentation. By default, the LogLevel value is warn. That should be enough for most Apache installations. If you are trying to troubleshoot a specific configuration, you can alter the level to debug.

Managing Apache Logs

Apache provides several tools for managing your logs. Other Apache-specific third-party tools are available and are mentioned here. Because Apache can log requests in the Common Log Format, most generic log processing tools can be used with Apache as well.

Resolving Hostnames

Earlier in the hour, you learned how to use the HostNameLookups directive to enable or disable hostname resolution at the time the request is made. If HostNameLookups is set to off (the default), the log file will contain only IP addresses. Later, you can use the command-line logresolve utility on Unix or logresolve.exe on Windows to process the log file and convert the IP addresses to hostnames.

logresolve reads log entries from standard input and outputs the result to its standard output. To read to and from a file, you can use redirection, on both Unix and Windows:

```
logresolve < access.log > resolved.log
```

Log-resolving tools are efficient because they can cache results and they do not cause any delay when serving requests to clients.

Fastresolve is an alternative, freely available log-resolving utility that can be found at http://www.pix.net/staff/djm/sw/fastresolve/.

Log Rotation

In Web sites with high traffic, the log files can quickly grow in size. You need to have a mechanism to rotate logs periodically, archiving and compressing older logs at well-defined intervals.

Log files cannot be removed directly while Apache is running because the server is writing directly to them. The solution is to use an intermediate program to log the requests. The program will, in turn, take care of rotating the logs.

Apache provides the rotatelogs program on Unix and rotatelogs.exe on Windows for this purpose. It accepts three arguments: a filename, a rotate interval in seconds, and an optional offset in minutes against UTC (Coordinated Universal Time).

For example,

```
TransferLog "|bin/rotatelogs /var/logs/apachelog 86400"
```

will create a new log file and move the current log to the /var/logs directory daily. (At the end of the command, 86400 is the number of seconds in one day.)

Logging Custom Information to a Database

Creating your own logging tables in MySQL, matched up with snippets of PHP code, can help you to capture access-related information for specific pages of your site. Using this information, you can create customized reports. This method can be much less cumbersome than wading through Apache log files, especially when you are just searching for a subset of access information.

Creating the Database Table

The first step in your custom logging method is to create the database table. The following table creation command will create a table called access_tracker in your MySQL database, with fields for an ID, page title, user agent, and date of access:

```
create table access_tracker (
       id int not null primary key auto_increment,
       page_title varchar(50),
       user_agent text,
       date_accessed date
);
```

Next, you'll create the code snippet that will write to this table.

Creating the PHP Code Snippet

As you may have gathered already, "code snippet" essentially means "a little bit of code." In other words, something that doesn't qualify as a long script, but just serves a simple purpose. In this case, the code snippet in Listing 17.1 will write some basic information to the access_tracker table.

Listing 17.1 Code Snippet for Access Tracking

```
 1: <?
 2: //set up static variables
 3: $page_title = "sample page A";
 4: $user_agent = getenv("HTTP_USER_AGENT");
 5: $date_accessed = date("Y-m-d");
 6:
 7: //connect to server and select database
 8: $conn = mysql_connect("localhost", "joeuser", "somepass")
 9:     or die(mysql_error());
10: $db = mysql_select_db("testDB", $conn) or die(mysql_error());
11:
12: //create and issue query
13: $sql = "insert into access_tracker values
14:    ('', '$page_title', '$user_agent', '$date_accessed')";
15: mysql_query($sql,$conn);
16: ?>
```

What you'll do with this snippet is simple: Place it at the beginning of every page you want to track. For each page, change the value of $page_title in the snippet to represent the actual title of the page.

Now create a sample script called sample1.php, containing the contents of Listing 17.1 and then the content in Listing 17.2.

Summary

This hour's lesson explained how to log specific information about the requests and errors generated by Apache. You can store the logs in files or databases, or pass them to external programs. You learned about the different utilities available for managing, processing, and analyzing logs, both the ones included with Apache and those available from third parties.

Finally, you saw a simple method for using PHP code snippets and a MySQL database to perform simple access tracking of specific pages. This information was then displayed in a simple access report, built with PHP.

Q&A

Q1:

Why wouldn't I want to log images?

A1:

In heavily loaded servers, logging can become a bottleneck. If the purpose of logging is to count the number of visitors and analyze their usage of the Web site, you can achieve this result by logging only the HTML pages, not the images contained in them. This reduces the number of hits stored in the logs and the time spent writing them.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

How would you avoid logging hits from a client accessing your Web site from a particular network?

A1:

In some situations, you may want to ignore requests coming from a particular network, such as your own, so that they do not skew the results. You can do this either by post-processing the logs and removing them or by using the SetEnvIf directive:

```
SetEnvIf Remote_Addr 10\.0\.0\. intranet
CustomLog logs/access_log "%h %l %u %t
\"%r\" %>s %b" !intranet
```

2:

How can you log images to a different file?

A2:

Earlier in the hour, you learned how to avoid logging images. Instead of ignoring images altogether, you can easily log them to a separate file, using the same environment variable mechanism:

```
SetEnvIf Request_URI "(\.gif|\.jpeg)$"
image
CustomLog logs/access_log common
env=!image
CustomLog logs/images_log common
env=image
```

Part IV: Simple Projects

Hour

Hour 18. Managing a Simple Mailing List

This hour provides the first of several hands-on, small projects designed to pull together your PHP and MySQL knowledge. In this hour, you'll learn the methods for creating a managed distribution list, which can be used to send out newsletters or anything else that you want to send, to a list of email addresses in a database.

The mailing mechanism you'll use in this hour is not meant to be a replacement for mailing list software, which is specifically designed for bulk messages. The type of system you'll build in this lesson should be used for only small lists of fewer than a few hundred email addresses.

In this hour, you will learn how to

- Create a subscribe/unsubscribe form and script
- Create a front end for sending your message
- Create the script that sends your message

Developing the Subscription Mechanism

You learned in earlier lessons that planning is the most important aspect of creating any product. In this case, think of the elements you will need for your subscription mechanism:

- A table to hold email addresses
- A way for users to add or remove their email addresses
- A form and script for sending the message

The following sections will describe each item individually.

Creating the subscribers Table

You really need only one field in the subscribers table: to hold the email address of the user. However, you should have an ID field just for consistency among your tables, and also because referencing an ID is a lot simpler than referencing a long email address in where clauses. So, in this case, your MySQL query would look something like

```
mysql> create table subscribers (
    -> id int not null primary key auto_increment,
    -> email varchar (150) unique not null
    -> );
Query OK, 0 rows affected (0.00 sec)
```

Note the use of unique in the field definition for email. This means that although id is the primary key, duplicates should not be allowed in the email field either. The email field is a unique key, and id is the primary key.

This relationship is represented in the table information as MUL (or "multiple") in the Key field:

```
mysql> describe subscribers;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     |      | PRI | NULL    | auto_increment |
| email | varchar(150) | YES  | MUL | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

Now that you have a table, you can create the form and script that place values in there.

Creating the Subscription Form

The subscription form will actually be an all-in-one form and script called manage.php, which will handle both

Developing the Mailing Mechanism

With the subscription mechanism in place, you can create a basic form interface for a script that will take the contents of your form and send it to every address in your subscribers table. This is another one of those all-in-one scripts, called sendmymail.php, and it is shown in Listing 18.2.

Listing 18.2 Send Mail to Your List of Subscribers

```
 1: <?php
 2: if ($_POST[op] != "send") {
 3:    //haven't seen the form, so show it
 4:    print "
 5:    <HTML>
 6:    <HEAD>
 7:    <TITLE>Send a Newsletter</TITLE>
 8:    </HEAD>
 9:    <BODY>
10:    <h1>Send a Newsletter</h1>
11:    <form method=\"post\" action=\"$_SERVER[PHP_SELF]\">
12:    <P><strong>Subject:</strong><br>
13:    <input type=\"text\" name=\"subject\" size=30></p>
14:    <P><strong>Mail Body:</strong><br>
15:    <textarea name=\"message\" cols=50 rows=10 wrap=virtual></textarea>
16:    <input type=\"hidden\" name=\"op\" value=\"send\">
17:    <p><input type=\"submit\" name=\"submit\" value=\"Send It\"></p>
18:    </FORM>
19:    </BODY>
20:    </HTML>";
21:
22: } else if ($_POST[op] == "send") {
23:     //want to send form, so check for required fields
24:     if (($_POST[subject] =="") || ($_POST[message] == "")) {
25:        header("Location: sendmymail.php");
26:        exit;
27:    }
28:
29:    //connect to database
30:    $conn = mysql_connect("localhost", "joeuser", "somepass")
31:         or die(mysql_error());
32:    mysql_select_db("testDB",$conn) or die(mysql_error());
33:
34:    //get emails from subscribers list
35:    $sql = "select email from subscribers";
36:    $result = mysql_query($sql,$conn) or die(mysql_error());
37:
38:    //create a From: mailheader
39:    $headers = "From: Your Mailing List <you@yourdomain.com>\n";
40:
41:    //loop through results and send mail
42:    while ($row = mysql_fetch_array($result)) {
43:        set_time_limit(0);
44:        $email = $row['email'];
45:        mail("$email", stripslashes($_POST[subject]),
46:             stripslashes($_POST[message]), $headers);
47:        print "newsletter sent to: $email<br>";
48:    }
49: }
50: ?>
```

The main logic of the script starts right there at line 2, where we determine whether the user has seen the form yet. If the value of $_POST[op] is not "send", we know the user has not submitted the form; therefore, we must show it to

Summary

In this hands-on hour, you applied your basic PHP and MySQL knowledge to the creation of a personal mailing list. Included were the database table creation, the subscribe and unsubscribe mechanisms, and the form and script for sending the mail.

Q&A

Q1:

How can I ease the burden on my mail server?

A1:

Besides looking into package mailing list software, you can bypass the mail() function and talk directly to your SMTP server via a socket connection. Such an example is shown in the PHP Manual for the fsockopen() function (http://www.php.net/manual/en/function.fsockopen.php), as well as in other developer resource sites.

Q2:

Where do bounced messages go?

A2:

Bounces go to whatever address you specify in your From: or Reply-to: mail headers.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1: What function sends mail?

A1: This is not a trick question. It's the mail() function!

2: What function call causes the script to execute for as long as it needs to run?

A2: set_time_limit(0)

Hour 19. Creating an Online Address Book

In this hour's hands-on lesson, the project is creating a managed, online address book. You will learn the methods for creating the relevant database tables, as well as the forms and scripts for adding, deleting, and viewing database records.

In this hour, you will learn how to

- Create relational tables for an online address book
- Create the forms and scripts for adding and deleting records in the address book
- Create the forms and scripts for viewing records

Planning and Creating the Database Tables

When you think of an address book, the obvious fields come to mind—name, address, telephone number, email address. However, if you look at your own paper-based address book, you may note that you have several entries for one person. Maybe that person has three telephone numbers, or two email addresses, and so forth. In your online address book, a set of related tables will help alleviate the redundancy and repetition of information.

Table 19.1 shows sample table and field names to use for your online address book. In a minute, you'll create the actual SQL statements, but first you should look at this information and try to see the relationships appear. Ask yourself which of the fields should be primary or unique keys.

Table 19.1. Address Book Table and Field Names

Table Name	Field Names
master_name	id, date_added, date_modified, f_name, l_name
address	id, master_id, date_added, date_modified, address, city, state, zipcode, type
telephone	id, master_id, date_added, date_modified, tel_number, type
fax	id, master_id, date_added, date_modified, fax_number, type
email	id, master_id, date_added, date_modified, email, type
personal_notes	id, master_id, date_added, date_modified, note

Notice the use of date-related fields; each table has a date_added and date_modified field in it. The fields will help maintain your data; you may at some point want to issue a query that removes all records that are older than a certain number of months or years, or that removes all records that haven't been updated within a certain period of time.

As you can see in the following SQL statements, the master_name table has two fields besides the ID and date-related fields: f_name and l_name, for first name and last name. The id field is the primary key. No other keys need to be primary or unique, unless you really want to limit your address book to one John Smith, one Mary Jones, and so forth.

Creating a Menu

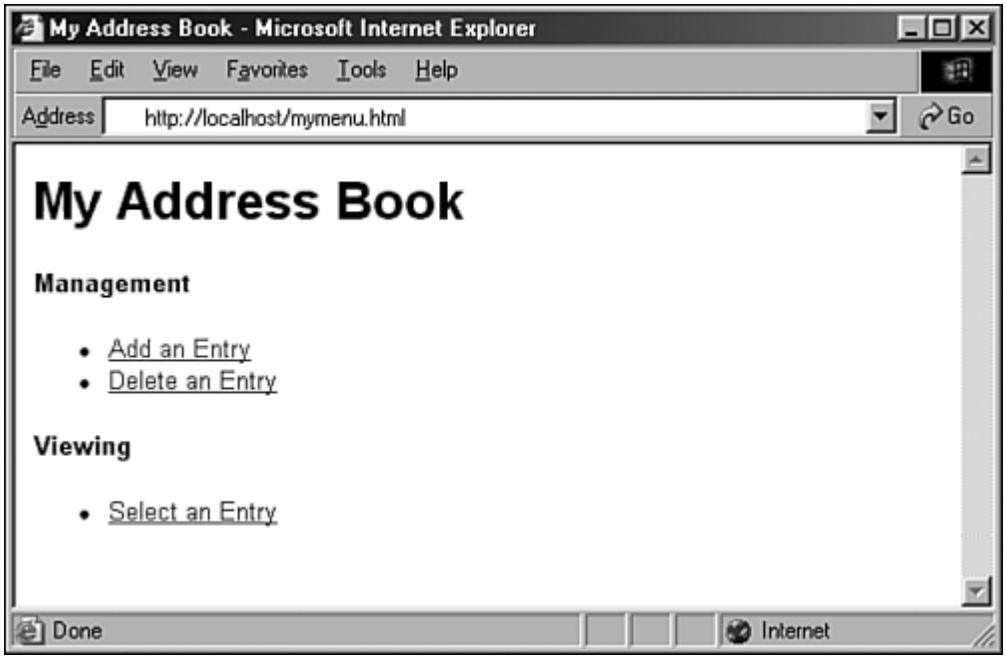
Your online address book will contain several actions, so it makes sense to create a menu for your links. Listing 19.1 creates a menu for all the scripts you will create in this section, called mymenu.php.

Listing 19.1 Address Book Menu

```
 1: <html>
 2: <head>
 3: <title>My Address Book</title>
 4: </head>
 5: <body>
 6: <h1>My Address Book</h1>
 7:
 8: <P><strong>Management</strong>
 9: <ul>
10: <li><a href="addentry.php">Add an Entry</a>
11: <li><a href="delentry.php">Delete an Entry</a>
12: </ul>
13:
14: <P><strong>Viewing</strong>
15: <ul>
16: <li><a href="selentry.php">Select a Record</a>
17: </ul>
18: </body>
19: </html>
```

Figure 19.1 shows the output of Listing 19.1. You'll tackle each of these items in order, starting with "Add an Entry" in the next section.

Figure 19.1. Address book menu.



Creating the Record Addition Mechanism

Just because you'll potentially be adding information to six different tables doesn't mean your form or script will be monstrous. In fact, your scripts won't look much different from any of the ones you created in previous lessons, and with practice, you will be able to make these verbose scripts much more streamlined and efficient.

In Listing 19.2, you can see a basic record addition script, called addentry.php.

Listing 19.2 Basic Record Addition Script Called addentry.php

```
 1: <?php
 2: if ($_POST[op] != "add") {
 3:      //haven't seen the form, so show it
 4:      $display_block = "<h1>Add an Entry</h1>
 5:     <form method=\"post\" action=\"$_SERVER[PHP_SELF]\">
 6:     <P><strong>First/Last Names:</strong><br>
 7:     <input type=\"text\" name=\"f_name\" size=30 maxlength=75>
 8:     <input type=\"text\" name=\"l_name\" size=30 maxlength=75>
 9:
10:     <P><strong>Address:</strong><br>
11:     <input type=\"text\" name=\"address\" size=30>
12:
13:     <P><strong>City/State/Zip:</strong><br>
14:     <input type=\"text\" name=\"city\" size=30 maxlength=50>
15:     <input type=\"text\" name=\"state\" size=5 maxlength=2>
16:     <input type=\"text\" name=\"zipcode\" size=10 maxlength=10>
17:
18:     <P><strong>Address Type:</strong><br>
19:     <input type=\"radio\" name=\"add_type\" value=\"home\" checked> home
20:     <input type=\"radio\" name=\"add_type\" value=\"work\"> work
21:    <input type=\"radio\" name=\"add_type\" value=\"other\"> other
22:
23:    <P><strong>Telephone Number:</strong><br>
24:    <input type=\"text\" name=\"tel_number\" size=30 maxlength=25>
25:    <input type=\"radio\" name=\"tel_type\" value=\"home\" checked> home
26:    <input type=\"radio\" name=\"tel_type\" value=\"work\"> work
27:    <input type=\"radio\" name=\"tel_type\" value=\"other\"> other
28:
29:    <P><strong>Fax Number:</strong><br>
30:    <input type=\"text\" name=\"fax_number\" size=30 maxlength=25>
31:    <input type=\"radio\" name=\"fax_type\" value=\"home\" checked> home
32:    <input type=\"radio\" name=\"fax_type\" value=\"work\"> work
33:    <input type=\"radio\" name=\"fax_type\" value=\"other\"> other
34:
35:    <P><strong>Email Address:</strong><br>
36:    <input type=\"text\" name=\"email\" size=30 maxlength=150>
37:    <input type=\"radio\" name=\"email_type\" value=\"home\" checked> home
38:    <input type=\"radio\" name=\"email_type\" value=\"work\"> work
39:    <input type=\"radio\" name=\"email_type\" value=\"other\"> other
40:
41:    <P><strong>Personal Note:</strong><br>
42:    <textarea name=\"note\" cols=35 rows=5 wrap=virtual></textarea>
43:     <input type=\"hidden\" name=\"op\" value=\"add\">
44:
45:    <p><input type=\"submit\" name=\"submit\" value=\"Add Entry\"></p>
46:    </FORM>";
47:
48: } else if ($_POST[op] == "add") {
49:    //time to add to tables, so check for required fields
50:     if (($_POST[f_name] == "") || ($_POST[l_name] == "")) {
51:        header("Location: addentry.php");
52:        exit;
```

Viewing Records

If you verified your work in the preceding section by issuing queries through the MySQL monitor or other interface, you probably became tired of typing SELECT * FROM... for every table. In this section, you'll create the two-part script that shows you how to select and view records in your database.

Listing 19.3 shows the select-and-view script called selentry.php.

Listing 19.3 Script Called selentry.php for Selecting and Viewing a Record

```
 1: <?php
 2: //connect to database
 3: $conn = mysql_connect("localhost", "joeuser", "somepass")
 4:    or die(mysql_error());
 5: mysql_select_db("testDB",$conn) or die(mysql_error());
 6:
 7: if ($_POST[op] != "view") {
 8:    //haven't seen the form, so show it
 9:    $display_block = "<h1>Select an Entry</h1>";
10:
11:    //get parts of records
12:    $get_list = "select id, concat_ws(', ', l_name, f_name) as display_name
13:        from master_name order by l_name, f_name";
14:    $get_list_res = mysql_query($get_list) or die(mysql_error());
15:
16:    if (mysql_num_rows($get_list_res) < 1) {
17:        //no records
18:        $display_block .= "<p><em>Sorry, no records to select!</em></p>";
19:
20:    } else {
21:        //has records, so get results and print in a form
22:        $display_block .= "
23:        <form method=\"post\" action=\"$_SERVER[PHP_SELF]\">
24:        <P><strong>Select a Record to View:</strong><br>
25:        <select name=\"sel_id\">
26:        <option value=\"\">— Select One —</option>";
27:
28:        while ($recs = mysql_fetch_array($get_list_res)) {
29:            $id = $recs['id'];
30:            $display_name = stripslashes($recs['display_name']);
31:
32:            $display_block .= "<option value=\"$id\">
33:                 $display_name</option>";
34:        }

35:        $display_block .= "
36:        </select>
37:        <input type=\"hidden\" name=\"op\" value=\"view\">
38:        <p><input type=\"submit\" name=\"submit\"
39:            value=\"View Selected Entry\"></p>
40:        </FORM>";
41:     }
42:
43: } else if ($_POST[op] == "view") {
44:
45:     //check for required fields
46:      if ($_POST[sel_id] == "") {
47:         header("Location: selentry.php");
48:         exit;
49:     }
50:
51:     //get master info
```

Creating the Record Deletion Mechanism

The record deletion mechanism is virtually identical to the script used to view a record. In fact, you can just take the first 42 lines of Listing 19.3 and paste them into a new file, called delentry.php, and make the following changes:

- In lines 7, 37, and 43, change "view" to "delete"
- In lines 24 and 39, change "View" to "Delete"

Starting with a new line 45, the remainder of the code for delentry.php is shown in Listing 19.4.

Listing 19.4 Script Called delentry.php for Selecting and Deleting a Record

```
45:    //check for required fields
46:      if ($_POST[sel_id] == "") {
47:        header("Location: delentry.php");
48:        exit;
49:    }
50:
51:    //issue queries
52:    $del_master = "delete from master_name where id = $_POST[sel_id]";
53:    mysql_query($del_master);
54:
55:    $del_address = "delete from address where id = $_POST[sel_id]";
56:    mysql_query($del_address);
57:
58:    $del_tel = "delete from telephone where id = $_POST[sel_id]";
59:    mysql_query($del_tel);
60:
61:    $del_fax = "delete from fax where id = $_POST[sel_id]";
62:    mysql_query($del_fax);
63:
64:    $del_email = "delete from email where id = $_POST[sel_id]";
65:    mysql_query($del_email);
66:
67:    $del_note = "delete from personal_notes where id = $_POST[sel_id]";
68:    mysql_query($del_master);
69:
70:    $display_block = "<h1>Record(s) Deleted</h1>
71:    <P>Would you like to
72:    <a href=\"$_SERVER[PHP_SELF]\">delete another</a>?</p>";
73: }
74: ?>
75: <HTML>
76: <HEAD>
77: <TITLE>My Records</TITLE>
78: </HEAD>
79: <BODY>
80: <? print $display_block; ?>
81: </BODY>
82: </HTML>
```

Picking up with Line 45, the script looks for the required field, $_POST[sel_id]. If that value does not exist, the user is redirected to the selection form. In lines 52–68, queries delete all information related to the selected individual,

Adding Subentries to a Record

At this point, you've learned to add, delete, and view records. What's missing is adding those additional entries to the related tables—entries for home versus work telephone number, for example. All you need to do is make a few changes to existing scripts.

In the selentry.php script in Listing 19.3, change lines 153–154 to read

```
$display_block .= "<P align=center>
<a href=\"addentry.php?master_id=$_POST[sel_id]\">add info</a> ...
<a href=\"$_SERVER[PHP_SELF]\">select another</a></p>";
```

This change simply adds a link to the addentry.php script and also passes it a variable called $master_id.

Now we need to modify the addentry.php script in Listing 19.2 to account for its dual purposes. Here is a summary of the changes to the original script.

Replace the first 10 lines of the original addentry.php script with the following snippet:

```
<?php
if (($_POST[op] != "add") || ($_GET[master_id] != "")) {
//haven't seen the form, so show it
$display_block = "
<h1>Add an Entry</h1>
<form method=\"post\" action=\"$_SERVER[PHP_SELF]\">";

if ($_GET[master_id] != "") {
    //connect to database
    $conn = mysql_connect("localhost", "joeuser", "somepass")
                or die(mysql_error());
    mysql_select_db("testDB",$conn) or die(mysql_error());

    //get first, last names for display/tests validity
    $get_names = "select concat_ws(' ', f_name, l_name) as
                display_name from master_name where id = $_GET[master_id]";
    $get_names_res = mysql_query($get_names) or die(mysql_error());

    if (mysql_num_rows($get_names_res) == 1) {
        $display_name = mysql_result($get_names_res,0,'display_name');
    }
}

if ($display_name != "") {
    $display_block .= "<P>Adding information for
                    <strong>$display_name</strong>:</p>";
} else {
    $display_block .= "
    <P><strong>First/Last Names:</strong><br>
    <input type=\"text\" name=\"f_name\" size=30 maxlength=75>
    <input type=\"text\" name=\"l_name\" size=30 maxlength=75>";
}
$display_block .= "<P><strong>Address:</strong><br>
```

This snippet simply moves around the form elements, printing the first and last name fields only if they contain a new record. If they contain an addition to a record, the individual's name is extracted from the database for aesthetic

Summary

In this hands-on hour, you applied your basic PHP and MySQL knowledge to the creation of a personal address book. You learned how to create the database table and scripts for record addition, deletion, and simple viewing. You also learned the process for adding multiple records attached to a single master entry.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

When passing a variable through the query string, which superglobal does it belong in? (Hint: You passed a variable this way in the last section.)

A1:

The $_GET superglobal.

2:

How many records in the address, email, telephone, and fax tables can you have for each individual in your master_name table?

A2:

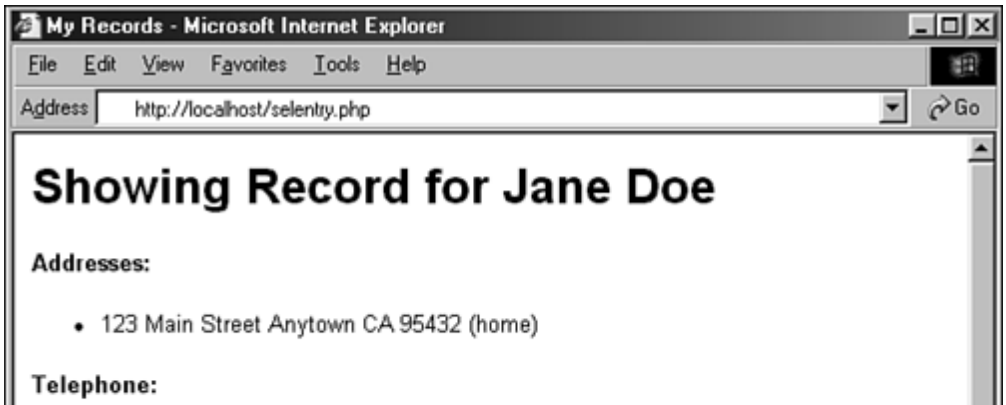
As many as you want—it's relational!

Activities

1. Go through each of the administration scripts and modify the code so that a link to the menu is printed at the bottom of each screen.

2. Use the second version of the addentry.php script to add secondary contact information to records in your database. Figure 19.8 shows how a record will look, after secondary contact information is added to it.

Figure 19.8. An individual's record, with multiple entries in tables.



Hour 20. Creating an Online Storefront

In this hour's hands-on lesson, the project is creating a generic online storefront. You will learn the methods for creating the relevant database tables, as well as the scripts for displaying the information to the user. The examples used in this hour represent one of an infinite number of possibilities to complete these tasks, and are meant to provide a foundation of knowledge rather than a definitive method for completing this task.

In this hour, you will learn how to

- Create relational tables for an online store
- Create the scripts to display store categories
- Create the scripts to display individual items

Planning and Creating the Database Tables

Before you tackle the process of creating database tables for a store, think about how you shop in real life. When you walk into a store, items are ordered in some fashion: The hardware and the baby clothes aren't mixed together, the electronics and the laundry detergent aren't side by side, and so on. Applying that knowledge to database normalization, already you know that you will need a table to hold categories and a table to hold items. These items will each belong to one category.

Next, think about the items themselves. Depending on the type of store you have, your items may or may not have colors, and may or may not have sizes. But all your items will have a name, a description, and a price. Again, thinking in terms of normalization, you know that you will have one general items table and two additional tables that relate to the general items table.

Table 20.1 shows sample table and field names to use for your online storefront. In a minute, you'll create the actual SQL statements, but first you should look at this information and try to see the relationships appear. Ask yourself which of the fields should be primary or unique keys.

Table 20.1. Storefront Table and Field Names

Table Name	Field Names
store_categories	id, cat_title, cat_desc
store_items	id, cat_id, item_title, item_price, item_desc, item_image
store_item_size	item_ id, item_size
store_item_color	item_id, item_color

As you can see in the following SQL statements, the store_categories table has two fields besides the id field: cat_title and cat_desc, for title and description. The id field is the primary key, and cat_title is a unique field because there's no reason you would have two identical categories.

```
mysql> create table store_categories (
    -> id int not null primary key auto_increment,
    -> cat_title varchar (50) unique,
    -> cat_desc text
    -> );
Query OK, 0 rows affected (0.03 sec)
```

The store_items table has five fields besides the id field, none of which are unique keys. The lengths specified in the field definitions are arbitrary; you should use whatever best fits your store. The cat_id field relates the item to a particular category in the store_categories table. This field is not unique because you will want more than one item in each category. The item_title, item_price, and item_desc (for description) fields are self-explanatory. The item_image

Displaying Categories of Items

Believe it or not, the most difficult task in this project is finished. Compared to thinking up categories and items, creating the scripts used to display the information is easy! The first script you will make is one that lists categories and items. Obviously, you wouldn't want to list all categories and all items all at once as soon as the user walks in the door, but you do want to give the user the option of immediately picking a category, seeing its items, and then picking another category. In other words, this script will serve two purposes: It will show the categories and then show the items in that category.

Listing 20.1 shows the code for seestore.php.

Listing 20.1 Script to View Categories

```
<?php
//connect to database
$conn = mysql_connect("localhost", "joeuser", "somepass")
    or die(mysql_error());
mysql_select_db("testDB",$conn) or die(mysql_error());

$display_block = "<h1>My Categories</h1>
<P>Select a category to see its items.</p>";

//show categories first
$get_cats = "select id, cat_title, cat_desc from
    store_categories order by cat_title";
$get_cats_res = mysql_query($get_cats) or die(mysql_error());

if (mysql_num_rows($get_cats_res) < 1) {
   $display_block = "<P><em>Sorry, no categories to browse.</em></p>";
} else {

   while ($cats = mysql_fetch_array($get_cats_res)) {
       $cat_id = $cats[id];
       $cat_title = strtoupper(stripslashes($cats[cat_title]));
       $cat_desc = stripslashes($cats[cat_desc]);

        $display_block .= "<p><strong><a
        href=\"$_SERVER[PHP_SELF]?cat_id=$cat_id\">$cat_title</a></strong>
        <br>$cat_desc</p>";

       if ($_GET[cat_id] == $cat_id) {
           //get items
           $get_items = "select id, item_title, item_price
           from store_items where cat_id = $cat_id
            order by item_title";
           $get_items_res = mysql_query($get_items) or die(mysql_error());

           if (mysql_num_rows($get_items_res) < 1) {
                $display_block = "<P><em>Sorry, no items in
                 this category.</em></p>";
            } else {

                $display_block .= "<ul>";

                while ($items = mysql_fetch_array($get_items_res)) {
                    $item_id = $items[id];
                    $item_title = stripslashes($items[item_title]);
                    $item_price = $items[item_price];

                    $display_block .= "<li><a
                     href=\"showitem.php?item_id=$item_id\">$item_title</a>
```

Displaying Items

The item display page in this hour will simply show all the item information. In the next hour, you'll add a few lines to it to make it function with an "add to cart" button. So for now, just assume this is a paper catalog.

Listing 20.2 shows the code for showitem.php.

Listing 20.2 Script to View Item Information

```
 1: <?php
 2: //connect to database
 3: $conn = mysql_connect("localhost", "joeuser", "somepass")
 4:     or die(mysql_error());
 5: mysql_select_db("testDB",$conn) or die(mysql_error());
 6:
 7: $display_block = "<h1>My Store - Item Detail</h1>";
 8:
 9: //validate item
10: $get_item = "select c.cat_title, si.item_title, si.item_price,
11: si.item_desc, si.item_image from store_items as si left join
12: store_categories as c on c.id = si.cat_id where si.id = $_GET[item_id]";
13:
14: $get_item_res = mysql_query($get_item) or die (mysql_error());
15:
16: if (mysql_num_rows($get_item_res) < 1) {
17:    //invalid item
18:    $display_block .= "<P><em>Invalid item selection.</em></p>";
19: } else {
20:    //valid item, get info
21:    $cat_title = strtoupper(stripslashes(
22:         mysql_result($get_item_res,0,'cat_title')));
23:    $item_title = stripslashes(mysql_result($get_item_res,0,'item_title'));
24:    $item_price = mysql_result($get_item_res,0,'item_price');
25:    $item_desc = stripslashes(mysql_result($get_item_res,0,'item_desc'));
26:    $item_image = mysql_result($get_item_res,0,'item_image');
27:
28:    //make breadcrumb trail
29:    $display_block .= "<P><strong><em>You are viewing:</em><br>
30:    <a href=\"seestore.php?cat_id=$cat_id\">$cat_title</a>
31:     &gt; $item_title</strong></p>
32:
33:    <table cellpadding=3 cellspacing=3>
34:    <tr>
35:    <td valign=middle align=center><img src=\"$item_image\"></td>
36:    <td valign=middle><P><strong>Description:</strong><br>$item_desc</p>
37:    <P><strong>Price:</strong> \$$item_price</p>";
38:
39:    //get colors
40:    $get_colors = "select item_color from store_item_color where
41:     item_id = $item_id order by item_color";
42:    $get_colors_res = mysql_query($get_colors) or die(mysql_error());
43:
44:    if (mysql_num_rows($get_colors_res) > 0) {
45:
46:        $display_block .= "<P><strong>Available Colors:</strong><br>";
47:
48:        while ($colors = mysql_fetch_array($get_colors_res)) {
49:            $item_color = $colors['item_color'];
50:
51:            $display_block .= "$item_color<br>";
52:        }
53:    }
```

Summary

In this hands-on hour, you applied your basic PHP and MySQL knowledge to the creation of a storefront display. You learned how to create the database table and scripts for viewing categories, item lists, and single items.

Workshop

The workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

Which PHP function was used to uppercase the category title strings?

A1:

strtoupper()

2:

Why don't the store_item_size and store_item_color tables contain any primary or unique keys?

A2:

Presumably, you will have items with more than one color and more than one size. Therefore, item_id is not a primary or unique key. Also, items may have the same colors or sizes, so the item_color and item_size fields must not be primary or unique either.

Hour 21. Creating a Shopping Cart Mechanism

In the last of the directly hands-on hours, the project is to integrate a shopping cart and checkout procedure into the storefront you created in the previous hour. You will be shown the methods for creating the relevant database tables as well as the scripts for adding and deleting cart items. Once again, the examples used in this hour represent one of an infinite number of possibilities to complete these tasks and are meant as working examples rather than the definitive guide for building an online store.

In this hour, you will learn

- How to create relational tables for the shopping cart and checkout portion of an online store
- How to create the scripts to add and remove cart items
- Some methods for processing transactions, and how to create your checkout sequence

Planning and Creating the Database Tables

Because the goal of this hour is to provide the user with a way to select and order items, you can imagine what the tables will be—first and foremost you need a cart table! In addition to the cart table, you'll need a table to store orders, along with one to store the items purchased as part of each order.

The following SQL statements were used to create the three new tables, starting with the store_shoppertrack table. This is the table used to hold items as users add them to their shopping cart.


The field lengths used to define these tables were chosen arbitrarily to try and accommodate several possible inputs. Please feel free to modify the lengths to meet your specific needs.

```
mysql> create table store_shoppertrack (
    -> id int not null primary key auto_increment,
    -> session_id varchar (32),
    -> sel_item_id int,
    -> sel_item_qty smallint,
    -> sel_item_size varchar(25),
    -> sel_item_color varchar(25),
    -> date_added datetime
    -> );
Query OK, 0 rows affected (0.01 sec)
```

In this table, the only key is the id field for the record. The session_id cannot be unique; otherwise users could only order one item from your store, which is not a good business practice. The session_id identifies the user. The sel_* fields are the selections by the user: the selected item, the selected quantity of the item, and the selected color and size of the item. Finally, there's a date_added field. Many times, users place items in their cart and never go through the checkout process. This practice leaves straggling items in your tracking table, which you may want to clear out periodically. For example, you might want to delete all cart items more than a week old—this is where the date_added field is helpful.

The next table holds the order information:

```
mysql> create table store_orders (
    -> id int not null primary key auto_increment,
    -> order_date datetime,
    -> order_name varchar (100),
    -> order_address varchar (255),
    -> order_city varchar (50),
    -> order_state char(2),
    -> order_zip varchar(10),
    -> order_tel varchar(25),
    -> order_email varchar(100),
    -> item_total float(6,2),
    -> shipping_total float(6,2),
    -> authorization varchar (50),
    -> status enum('processed', 'pending')
    -> );
Query OK, 0 rows affected (0.00 sec)
```

The only key field in the store_orders table is the id. For the sake of brevity in this lesson, an assumption is made that

Integrating the Cart with Your Storefront

In this section, you'll make modifications to the showitem.php script from Hour 20, "Creating an Online Storefront." The goal is to transform the item information page into an item information page with a form for selecting colors, sizes, and quantities.

In the original script, insert the following before Line 2:

```
session_start();
```

Because the shopping cart elements are attached to the user through a session id, the session must be started. The next changes don't occur until what was line 37 of the original script, so that's where we start Listing 21.1.

Listing 21.1 New Lines in showitem.php

```
37:    <P><strong>Price:</strong> \$$item_price</p>
38:    <form method=post action=\"addtocart.php\">";
39:    //get colors
40:    $get_colors = "select item_color from store_item_color where
41:         item_id = $item_id order by item_color";
42:    $get_colors_res = mysql_query($get_colors) or die(mysql_error());
43:
44:    if (mysql_num_rows($get_colors_res) > 0) {
45:
46:        $display_block .= "<P><strong>Available Colors:</strong>
47:        <select name=\"sel_item_color\">";
48:
49:        while ($colors = mysql_fetch_array($get_colors_res)) {
50:            $item_color = $colors['item_color'];
51:            $display_block .=
52:             "<option value=\"$item_color\">$item_color</option>";
53:        }
54:
55:        $display_block .= "</select>";
56:    }
57:
58:    //get sizes
59:    $get_sizes = "select item_size from store_item_size where
60:         item_id = $item_id order by item_size";
61:    $get_sizes_res = mysql_query($get_sizes) or die(mysql_error());
62:
63:    if (mysql_num_rows($get_sizes_res) > 0) {
64:
65:        $display_block .= "<P><strong>Available Sizes:</strong>
66:        <select name=\"sel_item_size\">";
67:
68:        while ($sizes = mysql_fetch_array($get_sizes_res)) {
69:            $item_size = $sizes['item_size'];
70:               $display_block .= "
71:                 <option value=\"$item_size\">$item_size</option>";
72:        }
73:
74:        $display_block .= "</select>";
75:    }
76:
77:    $display_block .= "
78:    <P><strong>Select Quantity:</strong>
79:    <select name=\"sel_item_qty\">";
80:
81:    for($i=1; $i<11; $i++) {
```

Payment Methods and the Checkout Sequence

Several commerce methods exist when it comes time to pay for the purchases in the shopping cart. The "right" method for you depends on your business—merchant accounts through banking institutions often require you to have a business license, a reseller's permit, and other pieces of paper proving you're a legitimate business. If you're simply a person who has a few items to sell, you might not want to go through all that paperwork. However, you still have options!

Regardless of the payment method you choose, one thing is certain—if you are passing credit-card information over the Web, you must do so over an SSL connection. Obtaining an SSL certificate and installing it on your system is covered in Hour 23, "Setting Up a Secure Web Server." You do not have to use this secure connection during the user's entire shopping experience, just from the point at which sensitive information is captured, such as the checkout form.

Creating the Checkout Form

At this point in the book, you should be well versed in creating a simple form. At the beginning of this hour, the store_orders table was created with fields to be used as a guideline for your form:

- order_name
- order_address
- order_city
- order_state
- order_zip
- order_tel
- order_email

Additionally, your form will need fields for the credit-card number, expiration date, and the name on the credit card. Another nice feature is to repeat the user's shopping cart contents with an item subtotal, so the customer remembers what he's paying for and approximately how much the order will cost. Also at this point of the checkout sequence, you offer any shipping options you might have. Shipping and sales tax would be calculated in the next step of the

Summary

In this hands-on lesson, you applied your basic PHP and MySQL knowledge to the integration of a shopping cart into the storefront from the previous chapter. Included were the database table creation, modifications to the item detail page, and new scripts for adding and removing cart items.

Workshop

The Workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

When removing an item from the cart, why do you suppose the query validates the session id of the user against the record?

A1:

Users should only be able to remove their own items.

2:

What would be a reason not to store the price in a hidden field when adding to the cart?

A2:

If you stored the price in a hidden field, a rogue user could change that value before posting the form, therefore, writing whatever price they wanted into the store_shoppertrack table, as opposed to the actual price.

Part V: Administration and Fine-Tuning

Hour

Hour 22. Apache Performance Tuning and Virtual Hosting

In this administration-related hour, consideration will be given to increasing the performance and scalability of your Apache installation. Additionally, you will learn about name-based and IP-based virtual hosting, and DNS and client issues. It explains different mechanisms that can be used to isolate clients from each other and the associated security tradeoffs.

In this hour, you will learn

- Which operating system and Apache-related settings can limit the server scalability or degrade performance
- About several tools for load testing Apache
- How to fine-tune Apache for optimum performance
- How to configure Apache to detect and prevent abusive behavior from clients
- How to configure name-based virtual hosts, IP-based virtual hosts, and the difference between the two
- About the dependencies virtual hosting has on DNS
- How to set up scaled-up cookie-cutter virtual hosts

Scalability Issues

This section covers scalability problems and how to prevent them. This is more of a "don't do this" list, explaining limiting factors that can degrade performance or prevent the server from scaling. We will also investigate the proactive tuning of Apache for optimal performance.

Operating System Limits

Several operating system factors can prevent Apache from scaling. These factors are related to process creation, memory limits, and maximum simultaneous number of open files or connections.


The Unix ulimit command enables you to set several of the limits covered in this section on a per-process basis. Please refer to your operating system documentation for details on ulimit's syntax.

Processes

Apache provides settings for preventing the number of server processes and threads from exceeding certain limits. These settings affect scalability because they limit the number of simultaneous connections to the Web server, which in turn affects the number of visitors that you can service simultaneously.

The Apache MPM settings are in turn constrained by OS settings limiting the number of processes and threads. How to change those limits varies from operating system to operating system. In Linux 2.0.x and 2.2.x kernels, it requires changing the NR_TASKS defined in /usr/src/linux/include/linux/tasks.h and recompiling the kernel. In the 2.4.x series, the limit can be accessed at runtime from the /proc/sys/kernel/threads-max file. You can read the contents of the file with

```
cat /proc/sys/kernel/threads-max
```

and write to it using

```
echo value > /proc/sys/kernel/threads-max
```

In Linux (unlike most other Unix versions), there is a mapping between threads and processes, and they are similar from the point of view of the OS.

In Solaris, those parameters can be changed in the /etc/system file. Those changes don't require rebuilding the kernel, but might require a reboot to take effect. You can change the total number of processes by changing the max_nprocs entry and the number of processes allowed for a given user with maxuproc.

File Descriptors

Whenever a process opens a file (or a socket), a structure called a file descriptor is assigned until the file is closed.

Load Testing with ApacheBench

You can test the scalability and performance of your site with benchmarking and traffic generation tools. There are many commercial and open-source tools, with varying degrees of sophistication. It is difficult to accurately simulate real-world request traffic because visitors have different navigation patterns, access the Internet using connections with different speeds, stop a download if it is taking too long, press the reload button repeatedly if they get impatient, and so on. That is why some tools record actual network traffic for later replay.

The Apache server comes with a simple, but useful, load-testing tool, called ApacheBench, or ab. You can find it in the /bin directory of the Apache distribution.

This tool enables you to request a certain URL a number of times and display a summary of the result. The following command requests the main page of the www.example.com server 1000 times, with 10 simultaneous clients at any given time:

```
#> /usr/local/apache2/bin/ab -n 1000 -c 10 http://www.example.com/
```

If you invoke ab without any arguments, you will get a complete listing of command-line options and syntax. Additionally, the trailing slash on the target URL is required, unless a specific page is named.

The result will look similar to the following:

```
This is ApacheBench, Version 2.0.40 <$Revision: 1.87 $>
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd,
http://www.zeustech.net/org/
Copyright (c) 1998-2001 The Apache Software Foundation, http://www.apache.org/

Benchmarking www.example.com (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Finished 1000 requests
Server Software:        Apache/2.0.40
Server Hostname:        www.example.com
Server Port:            80

Document Path:          /
Document Length:        8667 bytes

Concurrency Level:      10
Time taken for tests:   64.525026 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      8911000 bytes
HTML transferred:       8667000 bytes
```

Proactive Performance Tuning

Although previous sections explained which settings might prevent Apache from scaling, the following are some techniques for proactively increasing the performance of your server.

Mapping Files to Memory

As explained previously, accesses to disk affect performance significantly. Although most modern operating systems keep a cache of the most frequently accessed files, Apache also enables you to explicitly map a file into memory so that access to disk isn't necessary. The module that performs this mapping is mod_file_cache. You can specify a list of files to memory map by using the MMapFile directive, which applies to the server as a whole. An additional directive in Apache 2.0, CacheFile, takes a list of files, caches the file descriptors at startup, and keeps them around between requests, saving time and resources for frequently requested files.

Distributing the Load

Another way to increase performance is to distribute the load among several servers. This can be done in a variety of ways:

- A hardware load balancer directing network and HTTP traffic across several servers, making it look like a single server from the outside.
- A software load balancer solution using a reverse proxy with mod_rewrite.
- Separate servers providing images, large download files, and other static material. For example, you can place your images in a server called images.example.com and link to them from your main server.

Caching

The fastest way to serve content is not to serve it! This can be achieved by using appropriate HTTP headers that instruct clients and proxies of the validity in time of the requested resources. In this way, some resources that appear in multiple pages, but don't change frequently, such as logos or navigation buttons, are transmitted only once for a certain period of time.

Additionally, you can use mod_cache in Apache 2.0 to cache dynamic content so that it doesn't need to be created for every request. This is potentially a big performance boost because dynamic content usually requires accessing databases, processing templates, and so on, which can take significant resources.

As of this writing, mod_cache is still experimental. You can read more about it at http://httpd.apache.org/docs-2.0/mod/mod_cache.html.

Preventing Abuse

Denial of service (DoS) attacks work by swamping your server with a great number of simultaneous requests, slowing down the server or preventing access altogether to legitimate clients. DoS attacks are difficult to prevent in general, and usually the most effective way to address them is at the network or operating system level. One example is blocking specific addresses from making requests to the server; although you can block those addresses at the Web server level, it is more efficient to block them at the network firewall/router or with the operating system network filters.

Other kinds of abuse include posting extremely big requests or opening a great number of simultaneous connections. You can limit the size of requests and timeouts to minimize the effect of attacks. The default request timeout is 300 seconds, but you can change it with the TimeOut directive. A number of directives enable you to control the size of the request body and headers: LimitRequestBody, LimitRequestFields, LimitRequestFieldSize, LimitRequestLine, and LimitXMLRequestBody.

To prevent abuse, the mod_bwshare module enables you to limit the number of files or bytes that a given client can download from the server. You can learn more about mod_bwshare at http://www.topology.org/src/bwshare/README.html.

Robots

Robots, Web spiders, and Web crawlers are names that define a category of programs that download pages from your Web site, recursively following your site's links. Web search engines use these programs to scan the Internet for Web servers, download their content, and index it. Normal users use them to download an entire Web site or portion of a Web site for later offline browsing. Normally, these programs are well behaved, but sometimes they can be very aggressive and swamp your Web site with too many simultaneous connections or become caught in cyclic loops.

Well-behaved spiders will request a special file, called robots.txt, that contains instructions about how to access your Web site and which parts of the Web site won't be available to them. The syntax for the file can be found at http://www.robotstxt.org/. You can stop the requests at the router or operating system levels.

Implementing Virtual Hosting

Early Web servers were designed to handle the contents of a single site. The standard way of hosting several Web sites in the same machine was to install and configure different, and separate, Web server instances. As the Internet grew, so did the need for hosting multiple Web sites and a more efficient solution was developed: virtual hosting. Virtual hosting allows a single instance of Apache to serve different Web sites, identified by their domain names. IP-based virtual hosting means that each of the domains is assigned a different IP address; name-based virtual hosting means that several domains share a single IP address. As is explained later in the hour, name-based virtual hosting requires HTTP/1.1 support.

Web clients use the domain name server system (DNS) to translate hostnames into IP addresses, and vice versa. Several mappings are possible:

- One to one— Means that each hostname is assigned a single, unique IP address. This is the foundation for IP-based virtual hosting.
- One to many— Means that a single hostname is assigned to several IP addresses. This is useful for having several Apache instances serving the same Web site. If each of the servers is installed in a different machine, it is possible to balance the Web traffic among them, improving scalability.
- Many to one— Means that you can assign the same IP address to several hostnames. The client will specify the Web site it is accessing by using the Host: header in the request. This is the foundation for name-based virtual hosting.

When a many-to-one mapping is in place, a DNS server usually can be configured to respond with a different IP address for each DNS query, which helps to distribute the load. This is known as round robin DNS.

IP-Based Virtual Hosting

The simplest virtual host configuration is when each host is assigned a unique IP address. Each IP address maps the HTTP requests that Apache handles to separate content trees in their own VirtualHost containers, as shown in the following snippet:

```
Listen 192.168.128.10:80
Listen 192.168.129.10:80
<VirtualHost 192.168.128.10:80>
    DocumentRoot /usr/local/www-docs/host1
</VirtualHost>
<VirtualHost 192.168.129.10:80>
    DocumentRoot /usr/local/www-docs/host2
</VirtualHost>
```

Summary

This hour provided you with information on Apache and operating system settings that can affect scalability and performance. In most cases, however, the problems in Web site scalability relate to dynamic content generation and database access. Hardware-related improvements, such as high-quality network cards and drivers, increased memory, and disk arrays can also provide enhanced performance.

With regard to virtual hosting, Apache can be configured to handle virtual hosts in a variety of ways. Whether you need a large number of cookie-cutter virtual hosts, a varied set of different virtual host configurations, or the number of IP addresses you can use is limited, there's a way to configure Apache for your application. Name-based virtual hosting is a common technique for deploying virtual hosts without using up IP addresses. IP-based virtual hosting is still necessary when a virtual host is used for SSL. If you cannot change your DNS configuration, your only recourse is to use separate port numbers for your virtual hosts.

Q&A

Q1:

How can I measure whether my site is fast enough?

A1:

Many developers test their sites locally or over an internal network, but if you run a public Web site, chances are good that many of your users will access it over slow links. Try navigating your Web site from a dialup account and make sure that your pages load fast enough, with the rule of thumb being that pages should load in less than three seconds.

Q2:

How can I migrate an existing name-based virtual host to its own machine while maintaining continuous service?

A2:

If a virtual host is destined to move to a neighboring machine, which by definition cannot have the same IP address, there are some extra measures to take. A common practice is to do the following:

1. Set the time-to-live of the DNS mapping to a very low number. This increases the frequency of client lookups of the hostname.
2. Configure an IP alias on the old host with the new IP address.
3. Configure the virtual host's content to be served by both name- and IP-address-based virtual hosts.
4. After all the requests for the virtual host at the old IP address diminish (due to DNS caches expiring their old lookups), the server can be migrated.

Q3:

Can I mix IP- and name-based virtual hosting?

A3:

Yes. If multiple IP addresses are bound, you can allocate their usage a number of different ways. A family of name-based virtual hosts might be associated with each; just

Workshop

The Workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

Name some Apache settings that might limit scalability or affect Apache performance.

A1:

Some of the Apache settings that might affect scalability include the FollowSymLinks, SymLinksIfOwnerMatch arguments to the Options directive, enabling per-directory configuration files, hostname lookups, having a scoreboard file, and statistics collection with mod_status.

2:

Name some operating system settings that might limit scalability.

A2:

Some operating system settings that might affect scalability include limits for number of processes, open file descriptors, and memory allowed per process.

3:

Name some approaches to improve performance.

A3:

The following are some suggestions for improving performance: load distribution via a hardware load balancer or reverse proxy, data compression, caching, mapping files to memory, and compiling modules statically.

4:

Which VirtualHost container gets a request if the connection uses NameVirtualHost, but no Host header is sent?

A4:

Reading the configuration top-to-bottom, the first VirtualHost container is favored. The same behavior

Hour 23. Setting Up a Secure Web Server

This hour explains how to set up an Apache server capable of secure transactions. In this hour, you will learn

- The installation and configuration of the mod_ssl Apache module
- The SSL/TLS family of protocols and the underlying cryptography concepts
- What certificates are and how to create and manage them

The Need for Security

As the Internet became mainstream and the number of companies, individuals, and government agencies using it grew, so did the number and type of transactions that needed protection. Those included financial transactions, such as banking operations and electronic commerce, as well as exchange of sensitive information, such as medical records and corporate documents. Three requirements are necessary to carry on secure communications on the Internet: confidentiality, integrity, and authentication.

Confidentiality

Confidentiality is the most obvious requirement for secure communications. If you are transmitting or accessing sensitive information such as your credit-card number or your personal medical history, you certainly don't want a stranger to get hold of it.

Integrity

The information contained in the exchanged messages must be protected from external manipulation. That is, if you place an order online to buy 100 shares of stock, you don't want to allow anyone to intercept the message, change it to an order to buy 1000 shares, or replace the original message. Additionally, you want to prevent an attacker from performing replay attacks, which, instead of modifying the original message, simply resend it several times to achieve a cumulative effect.

Authentication

You need to decide whether to trust the organization or individual you are communicating with. To achieve this, you must authenticate the identity of the other party in the communication.

The science of cryptography studies the algorithms and methods used to securely transmit messages, ensuring the goals of confidentiality, integrity, and authenticity. Cryptanalysis is the science of breaking cryptographic systems.

The SSL Protocol

SSL stands for Secure Sockets Layer and TLS stands for Transport Layer Security. They are a family of protocols that were originally designed to provide security for HTTP transactions, but that also can be used for a variety of other Internet protocols such as IMAP and NNTP. HTTP running over SSL is referred to as secure HTTP.

Netscape released SSL version 2 in 1994 and SSL version 3 in 1995. TLS is an IETF standard designed to standardize SSL as an Internet protocol. It is just a modification of SSL version 3 with a small number of added features and minor cleanups. The TLS acronym is the result of arguments between Microsoft and Netscape over the naming of the protocol because each company proposed its own name. However, the name has not stuck and most people refer to these protocols simply as SSL. Unless otherwise specified, the rest of this hour refers to SSL/TLS as SSL.

You specify that you want to connect to a server using SSL by replacing http with https in the protocol component of a URI. The default port for HTTP over SSL is 443.

The following sections explain how SSL addresses the confidentiality, integrity, and authentication requirements outlined in the previous section. In doing so, it explains, in a simplified manner, the underlying mathematical and cryptographic principles SSL is based on.

Addressing the Need for Confidentiality

The SSL protocol protects data from eavesdropping by encrypting it. Encryption is the process of converting a message, the plaintext, into a new encrypted message, the ciphertext. Although the plaintext is readable by everyone, the ciphertext will be completely unintelligible to an eavesdropper. Decryption is the reverse process, which transforms the ciphertext into the original plaintext.

Usually, encryption and decryption processes involve an additional piece of information: a key. If both sender and receiver share the same key, the process is referred to as symmetric cryptography. If sender and receiver have different, complementary keys, the process is called asymmetric or public key cryptography.

Symmetric Cryptography

If the key used to both encrypt and decrypt the message is the same, the process is known as symmetric cryptography. DES, Triple-Des, RC4, and RC2 are algorithms used for symmetric key cryptography. Many of these algorithms can have different key sizes, measured in bits. In general, given an algorithm, the greater the number of bits in the key, the more secure the algorithm is and the slower it will run because of the increased computational needs of performing the algorithm.

Symmetric cryptography is relatively fast compared to public key cryptography, which is explained in the next section. Symmetric cryptography has two main drawbacks, however. One is that keys should be changed periodically to avoid providing an eavesdropper with access to large amounts of material encrypted with the same key. The other is the key distribution problem: How to get the keys to each one of the parties in a safe manner? This was one of the original limiting factors, and before the invention of public key cryptography, the problem was solved by periodically having people traveling around with suitcases full of keys.

Public Key Cryptography

Installing SSL

Now that you've learned all about SSL, you need to install SLL support for Apache. SSL support is provided by mod_ssl, a module that is included with Apache, but is not enabled by default. mod_ssl, in turn, requires the OpenSSL library—an open-source implementation of the SSL/TLS protocols and a variety of other cryptographic algorithms. OpenSSL is based on the SSLeay library developed by Eric A. Young and Tim J. Hudson.

OpenSSL

This section explains how to download and install the OpenSSL toolkit for both Windows and Unix variants.

Windows

The required OpenSSL libraries are included with the Windows installer of Apache 2.0 and no further installation or download is necessary. openssl.exe is included in the bin/ directory of the Apache distribution. It is a utility for generating certificates, keys, certificate signing requests, and so on.

Unix

If you are running a recent Linux or FreeBSD distribution, OpenSSL might already be installed in your system. Use the package management tools bundled with your distribution to determine whether that is the case or, otherwise, to install it.

If you need to install OpenSSL from source, you can download OpenSSL from http://www.openssl.org. After you have downloaded the software, you need to uncompress it and cd into the created directory:

```
#> gunzip < openssl*.tar.gz | tar xvf -
#> cd openssl*
```

OpenSSL contains a config script to help you build the software. You must provide the path to which the software will install. The path used in this hour is /usr/local/ssl/install, and you probably need to have root privileges to install the software there. You can install the software as a regular user, but to do so, you will need to change the path. Then, you must build and install the software:

```
#> ./config --prefix=/usr/local/ssl/install \
--openssldir=/usr/local/ssl/install/openssl
#> make
#> make install
```

If everything went well, you have now successfully installed the OpenSSL toolkit. The openssl command-line tool will be located in /usr/local/ssl/install/bin/.

This tool is used to create and manipulate certificates and keys, and its usage is described in a later section on certificates.

mod_ssl

Managing Certificates

To have a working SSL server implementation, the first step is to create a server certificate. This section explains in detail how to create and manage certificates and keys by using the openssl command-line tool. For example, if you are using SSL for an e-commerce site, encryption prevents customer data from eavesdroppers, and the certificate enables customers to verify that you are who you claim to be.

The examples refer to the Unix version of the command-line program openssl. If you are running under Windows, you need to use openssl.exe instead and change the paths of the examples to use backslashes instead of forward slashes. The examples also assume that OpenSSL was installed in the path described earlier in the OpenSSL installation section.

Creating a Key Pair

You must have a public/private key pair before you can create a certificate request. Assume that the FQDN for the certificate you want to create is www.example.com. (You will need to substitute this name for the FQDN of the machine on which you have installed Apache.) You can create the keys by issuing the following command:

```
#> ./usr/local/ssl/install/bin/openssl genrsa -des3 -rand file1: file2: file3 \
           -out
www.example.com. key 1024
```

genrsa indicates to OpenSSL that you want to generate a key pair.

des3 indicates that the private key should be encrypted and protected by a pass phrase.

The rand switch is used to provide OpenSSL with random data to ensure that the generated keys are unique and unpredictable. Substitute file1, file2, and so on, for the path to several large, relatively random files for this purpose (such as a kernel image, compressed log files, and so on). You can also use /dev/random if it exists on your system. The rand switch is not necessary on Windows because the random data is automatically generated by other means.

The out switch indicates where to store the results.

1024 indicates the number of bits of the generated key.

The result of invoking this command looks like this:

```
625152 semi-random bytes loaded
Generating RSA private key, 1024 bit long modulus
.....++++++
.........................++++++
e is 65537 (0x10001)
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

SSL Configuration

The previous sections introduced the (not-so-basic) concepts behind SSL, and you have learned how to generate keys and certificates. Now, finally, you can configure Apache to support SSL. mod_ssl must either be compiled statically or, if you have compiled as a loadable module, the appropriate LoadModule directive must be present in the file.

If you compiled Apache yourself, a new Apache configuration file, named ssl.conf, should be present in the /conf directory. That file contains a sample Apache SSL configuration, and is referenced from the main httpd.conf file via an Include directive.

If you want to start your configuration from scratch, you can add the following configuration snippet to your Apache configuration file:

```
Listen 80
Listen 443
<VirtualHost _default_:443>
ServerName www.example.com
SSLEngine on
SSLCertificateFile \
/usr/local/ssl/install/openssl/certs/www.example.com.cert
SSLCertificateKeyFile \
/usr/loca/ssl/install/openssl/certs/www.example.com.key
</VirtualHost>
```

With the previous configuration, you set up a new virtual host that will listen to port 443 (the default port for HTTPS), and you enable SSL on that virtual host with the SSLEngine directive.

You need to indicate where to find the server's certificate and the file containing the associated key. You do so by using SSLCertificateFile and SSLCertificateKeyfile directives.

Starting the Server

Now you can stop the server if it is running, and start it again. If your key is protected by a pass phrase, you will be prompted for it. After this, Apache will start, and you should be able to connect securely to it via the https://www.example.com/ URL.

If you compiled and installed Apache yourself, in many of the vendor configuration files, you can see that an <IFDEFINE SSL> block surrounds the SSL directives. That allows for conditional starting of the server in SSL mode. If you start the httpd server binary directly, you can pass it the -DSSL flag at startup. You can also use the apachectl script by issuing the apachectl startssl command. Finally, if you always want to start Apache with SSL support, you can just remove the <ifDefine> section and start Apache in the usual way.

If you are unable to successfully start your server, check the Apache error log for clues about what might have gone wrong. For example, if you cannot bind to the port, make sure that another Apache is not running already. You must have administrator privileges to bind to port 443.

Summary

This hour explained the fundamentals of the SSL protocol and mod_ssl, the Apache module that implements support for SSL. You learned how to install and configure mod_ssl and the OpenSSL libraries, and how to use the openssl command-line tool for certificate and key generation and management. You can access the mod_ssl reference documentation for in-depth syntax explanation and additional configuration information. Bear in mind also that SSL is just part of maintaining a secure server, which includes applying security patches, OS configuration, access control, physical security, and so on.

Q&A

Q1:

Can I have SSL with name-based virtual hosting?

A1:

A question that comes up frequently is how to make name-based virtual hosts work with SSL. The answer is that you can't, at least currently. Name-based virtual hosts depend on the Host header of the HTTP request, but the certificate verification happens when the SSL connection is being established and no HTTP request can be sent. There is a protocol for upgrading an existing HTTP connection to TLS, but it is mostly unsupported by current browsers (see RFC 2817).

Q2:

Can I use SSL with other protocols?

A2:

mod_ssl implements the SSL protocol as a filter. Other protocols using the same Apache server can easily take advantage of the SSL.

Workshop

The Workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

Name three requirements to carry on secure communications on the Internet.

A1:

Confidentiality, integrity, and authentication

2:

How do you start an SSL-enabled instance of Apache?

A2:

Use the apachectl control script and the command apachectl startssl.

Hour 24. Optimizing and Tuning MySQL

Proper care and feeding of your MySQL server will keep it running happily and without incident. The optimization of your system consists of proper hardware maintenance and software tuning. In this hour, you will learn

- Basic hardware and software optimization tips for your MySQL server
- Key start-up parameters for your MySQL server
- How to use the OPTIMIZE TABLE command
- How to use the EXPLAIN command
- How to use the FLUSH command to clean up tables, caches, and log files
- How to use SHOW commands to retrieve information about databases, tables, and indexes
- How to use SHOW commands to find system status information

Building an Optimized Platform

Designing a well-structured, normalized database schema is just half of the optimization puzzle. The other half is building and fine-tuning a server to run this fine database. Think about the four main components of a server: CPU, memory, hard drive, and operating system. Each of these better be up to speed or no amount of design or programming will make your database faster!

- CPU— The faster the CPU, the faster MySQL will be able to process your data. There's no real secret to this, but a 750MHz processor is significantly faster than a 266MHz processor. With processor speeds now more than 1GHz and with reasonable prices all around, it's not difficult to get a good bang for your buck.

- Memory— Put as much RAM in your machine as you can. You can never have enough, and RAM prices will be at rock bottom for the foreseeable future. Having available RAM can help balance out sluggish CPUs.

- Hard Drive— The proper hard drive will be both large enough and fast enough to accommodate your database server and its traffic. An important measurement of hard-drive speed is its seek time, or the amount of time it takes for the drive to spin around and find a specific piece of information. Seek time is measured in milliseconds, and an average disk-seek time is around eight or nine milliseconds. When buying a hard drive, make sure it's big enough to accommodate all the data you'll eventually store in your database and fast enough to find it quickly.

- Operating System— If you use an operating system that's a resource hog, you have two choices: buy enough resources (that is, RAM) so that it doesn't matter, or use an operating system that doesn't suck away all your resources just so that you can have windows and pretty colors. Also, if you are blessed with a machine that has multiple processors, be sure your operating system can handle this condition and handle it well.

If you put the proper pieces together at the system level, you'll have taken several steps toward overall server optimization.

Using the benchmark() Function

A quick test of your server speed is to use the benchmark() MySQL function to see how long it takes to process a given expression. You can make the expression something simple, such as 10+10, or something more extravagant, such as extracting pieces of dates.

No matter the result of the expression, the result of benchmark() will always be 0. The purpose of benchmark() is not to retrieve the result of the expression, but to see how long it takes to repeat the expression for a specific number of times. For example, the following command executes the expression 10+10 one million times:

```
mysql> select benchmark(1000000,10+10);
+--------------------------+
| benchmark(1000000,10+10) |
+--------------------------+
|                        0 |
```

MySQL Startup Options

MySQL AB provides a wealth of information regarding the tuning of server parameters, much of which the average user will never need to use. So, as not to completely overwhelm you with information, this section will contain a few of the more common startup options for a finely tuned MySQL server.

When you start MySQL, a configuration file called my.cnf is loaded. This file contains information ranging from port number to buffer sizes, but can be overruled by command-line startup options. At installation time, my.cnf is placed in the /etc directory, but you can also specify an alternate location for this file during start-up.

In the support-files subdirectory of your MySQL installation directory, you'll find four sample configuration files, each tuned for a specific range of installed memory:

- my-small.cnf— For systems with less than 64MB of RAM, where MySQL is used occasionally.
- my-medium.cnf— For systems with less than 64MB of RAM, where MySQL is the primary activity on the system, or for systems with up to 128MB of RAM, where MySQL shares the box with other processes. This is the most common configuration, where MySQL is installed on the same box as a Web server and receives a moderate amount of traffic.
- my-large.cnf— For a system with 128MB to 512MB of RAM, where MySQL is the primary activity.
- my-huge.cnf— For a system with 1GB to 2GB of RAM, where MySQL is the primary activity.

To use any of these as the base configuration file, simply copy the file of your choice to /etc/my.cnf (or wherever my.cnf is on your system) and change any system-specific information, such as port or file locations.

Key Startup Parameters

There are two primary start-up parameters that will affect your system the most: key_buffer_size and table_cache. If you get only two server parameters correctly tuned, make sure they're these two!

The value of key_buffer_size is the size of the buffer used with indexes. The larger the buffer, the faster the SQL command will finish and a result will be returned. Try to find the fine line between finely tuned and over-optimized; you might have a key_buffer_size of 256MB on a system with 512MB of RAM, but any more than 256MB could cause degraded server performance.

A simple way to check the actual performance of the buffer is to examine four additional variables: key_read_requests, key_reads, key_write_requests, and key_writes. You can find the values of these variables by issuing the SHOW STATUS command:

Optimizing Your Table Structure

An optimized table structure is different from a well-designed table. Table structure optimization has to do with reclaiming unused space after deletions and basically cleaning up the table after structural modifications have been made. The OPTIMIZE TABLE SQL command takes care of this, using the following syntax:

```
OPTIMIZE TABLE table_name[,table_name]
```

For example, if you want to optimize the grocery_inventory table in the testDB database, use

```
mysql> optimize table grocery_inventory;
+--------------------------+----------+----------+----------+
| Table                    | Op       | Msg_type | Msg_text |
+--------------------------+----------+----------+----------+
| testDB.grocery_inventory | optimize | status   | OK       |
+--------------------------+----------+----------+----------+
1 row in set (0.08 sec)
```

The output doesn't explicitly state what was fixed, but the text in the Msg_text column shows that the grocery_inventory table was indeed optimized. If you run the command again, the text will change, showing that it is a useful message:

```
mysql> optimize table grocery_inventory;
+-------------------------+----------+----------+-----------------------------+
| Table                   | Op       | Msg_type | Msg_text                    |
+-------------------------+----------+----------+-----------------------------+
| testDB.grocery_inventory| optimize | status   | Table is already up to date |
+-------------------------+----------+----------+-----------------------------+
1 row in set (0.03 sec)
```

Be aware that the table is locked while it is optimized, so if your table is large, optimize it during scheduled downtime or when little traffic is flowing to your system.

Optimizing Your Queries

Query optimization has a lot to do with the proper use of indexes. The EXPLAIN command will examine a given SELECT statement to see whether it's optimized the best that it can be, using indexes wherever possible. This is especially useful when looking at complex queries involving JOINs. The syntax for EXPLAIN is

```
EXPLAIN SELECT statement
```

The output of the EXPLAIN command is a table of information containing the following columns:

- table— The name of the table.
- type— The join type, of which there are several.
- possible_keys— This column indicates which indexes MySQL could use to find the rows in this table. If the result is NULL, no indexes would help with this query. You should then take a look at your table structure and see whether there are any indexes that you could create that would increase the performance of this query.
- key— The key actually used in this query, or NULL if no index was used.
- key_len— The length of the key used, if any.
- ref— Any columns used with the key to retrieve a result.
- rows— The number of rows MySQL must examine to execute the query.
- extra— Additional information regarding how MySQL will execute the query. There are several options, such as Using index (an index was used) and Where (a WHERE clause was used).

The following EXPLAIN command output shows a nonoptimized query:

```
mysql> explain select * from grocery_inventory;
+------------------+------+--------------+-----+--------+-----+-----+------+
| table            | type | possible_keys | key | key_len| ref | rows| Extra|
+------------------+------+--------------+-----+--------+-----+-----+------+
| grocery_inventory | ALL  | NULL          | NULL| NULL   | NULL| 6   |      |
```

Using the FLUSH Command

Users with reload privileges for a specific database can use the FLUSH command to clean up the internal caches used by MySQL. Often, only the root-level user has the appropriate permissions to issue administrative commands such as FLUSH.

The FLUSH syntax is

```
FLUSH flush_option
```

The common options for the FLUSH command are

- PRIVILEGES
- TABLES
- HOSTS
- LOGS

You've used the FLUSH PRIVILEGES command before, after adding new users. This command simply reloads the grant tables in your MySQL database, enabling the changes to take effect without stopping and restarting MySQL. When you issue a FLUSH PRIVILEGES command, the Query OK response will assure you that the cleaning process occurred without a hitch.

```
mysql> flush privileges;
Query OK, 0 rows affected (0.10 sec)
```

The FLUSH TABLES command will close all tables currently open or in use and essentially give your MySQL server a millisecond of breathing room before starting back to work. When your caches are empty, MySQL can better utilize available memory. Again, you're looking for the Query OK response:

```
mysql> flush tables;
Query OK, 0 rows affected (0.21 sec)
```

The FLUSH HOSTS command works specifically with the host cache tables. If you are unable to connect to your MySQL server, a common reason is that the maximum number of connections has been reached for a particular host, and it's throwing errors. When MySQL sees numerous errors on connection, it will assume something is amiss and simply block any additional connection attempts to that host. The FLUSH HOSTS command will reset this process and again allow connections to be made:

```
mysql> flush hosts;
```

Using the SHOW Command

There are several different uses of the SHOW command, which will produce output displaying a great deal of useful information about your MySQL database, users, and tables. Depending on your access level, some of the SHOW commands will not be available to you or will provide only minimal information. The root-level user has the capability to use all the SHOW commands, with the most comprehensive results.

The common uses of SHOW include the following, which you'll soon learn about in more detail:

```
SHOW GRANTS FOR user
SHOW DATABASES [LIKE something]
SHOW [OPEN] TABLES [FROM database_name] [LIKE something]
SHOW CREATE TABLE table_name
SHOW [FULL] COLUMNS FROM table_name [FROM database_name] [LIKE something]
SHOW INDEX FROM table_name [FROM database_name]
SHOW TABLE STATUS [FROM db_name] [LIKE something]
SHOW STATUS [LIKE something]
SHOW VARIABLES [LIKE something]
```

The SHOW GRANTS command will display the privileges for a given user at a given host. This is any easy way to check on the current status of a user, especially if you have a request to modify a user's privileges. With SHOW GRANTS, you can check first to see that the user doesn't already have the requested privileges. For example, see the privileges available to the joeuser user:

```
mysql> show grants for joe@localhost;
+-----------------------------------------------------------+
| Grants for joeuser@localhost                              |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO 'joeuser'@'localhost' \
IDENTIFIED BY PASSWORD '34f3a6996d856efd'                   |
| GRANT ALL PRIVILEGES ON testDB.* TO 'joeuser'@'localhost' |
+-----------------------------------------------------------+
```

If you're not the root-level user or the joeuser user, you'll get an error. Unless you're the root-level user, you can only see the information relevant to your user. For example, the joeuser user isn't allowed to view information about the root-level user:

```
mysql> show grants for root@localhost;
ERROR 1044: Access denied for user:'joeuser@localhost' to database 'mysql'
```

Be aware of your privilege level throughout the remainder of this hour. If you are not the root-level user, some of these commands will not be available to you or will display only limited information.

Retrieving Information About Databases and Tables

You've used a few of the basic SHOW commands earlier in this book to view the list of databases and tables on your MySQL server. As a refresher, the SHOW DATABASES command does just that—it lists all the databases on the MySQL server:

```
mysql> show databases;
+------------------+
| Database         |
+------------------+
```

Summary

Running an optimized MySQL server starts with the hardware and operating system in use. Your system's CPU should be sufficiently fast, and you should have enough RAM in use to pick up the slack when your CPU struggles. This is especially true if MySQL shares resources with other processes, such as a Web server. Additionally, the hard drive in use is important because a small hard drive will limit the amount of information you can store in your database. The seek time of your hard drive is important—a slow seek time will cause the overall performance of the server to be slower. Your operating system should not overwhelm your machine and should share resources with MySQL rather than using all the resources itself.

Some key startup parameters for MySQL are the values of key_buffer_size and table_cache, among others. Baseline values can be found in sample MySQL configuration files, or you can modify the values of these variables and watch the server performance to see whether you hit on the right result for your environment.

Beyond hardware and software optimization is the optimization of tables, as well as SELECT queries. Table optimization, using the OPTIMIZE command, enables you to reclaim unused space. You can see how well (or not) optimized your queries are by using the EXPLAIN command. The resulting output will show if and when indexes are used, and whether you can use any indexes to speed up the given query.

Paying attention to your MySQL server will ensure that it continues to run smoothly. Basic administration commands, such as FLUSH and SHOW, will help you to recognize and quickly fix potential problems. All these commands are designed to give MySQL a millisecond of rest time and breathing room if it's under a heavy load. Numerous SHOW commands will display structural information about databases, tables, and indexes, as well as how the system is performing.

Q&A

Q1:

Can MySQL take advantage of multiple CPUs in a single server?

A1:

Absolutely, if your operating system supports multiple CPUs, MySQL will take advantage of them. However, the performance and tuning of MySQL using multiple processors varies, depending on the operating system. For more information, please see the MySQL manual section for your specific operating system:

```
http://www.mysql.com/doc/O/p/Operating_System_Specific_Notes.html
```

Q2:

What permission level must I have to use the OPTIMIZE command?

A2:

Any user with INSERT privileges for a table can perform OPTIMIZE commands. If a user has only SELECT permissions, the OPTIMIZE command will not execute.

Workshop

The Workshop is designed to help you anticipate possible questions, review what you've learned, and begin learning how to put your knowledge into practice.

Quiz

1:

Which MySQL function will enable you to run an expression many times over to find the speed of the iterations?

A1:

The benchmark() function.

2:

Which SQL command will clean up the structure of your tables?

A2:

OPTIMIZE

3:

Which FLUSH command resets the MySQL log files?

A3:

FLUSH LOGS

4:

To quickly determine if MySQL has support for InnoDB tables, would you use SHOW STATUS or SHOW VARIABLES?

A4:

SHOW VARIABLES

5:

Write a SQL statement that will enable you to see the SQL statement used to create a table called myTable.

A5:

SHOW CREATE TABLE myTable

[Team LiB]