VISUAL QUICKPRO GUIDE

COVERS PHP 5 & 7

PHP and MySQL for Dynamic Web Sites

Fifth Edition

LARRY ULLMAN

LEARN THE QUICK AND EASY WAY!

VISUAL QUICKPRO GUIDE

PHP and MySQL for Dynamic Web Sites

Fifth Edition

LARRY ULLMAN

Peachpit Press

Visual QuickPro Guide

PHP and MySQL for Dynamic Web Sites, Fifth Edition

Larry Ullman

Peachpit Press
www.peachpit.com

To report errors, please send a note to: errata@peachpit.com
Peachpit Press is a division of Pearson Education.

Editor: Mark Taber
Copy Editor: Elizabeth Welch
Technical Reviewer: Timothy Boronczyk
Production Coordinator: David Van Ness
Compositor: Danielle Foster
Proofreader: Scout Festa
Indexer: Valerie Haynes Perry
Cover Design: RHDG / Riezebos Holzbaur Design Group, Peachpit Press
Interior Design: Peachpit Press
Logo Design: MINE™ www.minesf.com

ISBN-13: 978-0-13-430184-6
ISBN-10: 0-13-430184-6

1 17

Printed and bound in the United States of America

Dedication

Dedicated to the fine faculty at my alma mater, Northeast Missouri State University. In particular, I would like to thank Dr. Monica Barron, Dr. Dennis Leavens, Dr. Ed Tyler, and Dr. Cole Woodcox, whom I also have the pleasure of calling my friend. I would not be who I am as a writer, as a student, as a teacher, or as a person if it were not for the magnanimous, affecting, and brilliant instruction I received from these educators.

Special Thanks to:

My heartfelt thanks to everyone at Peachpit Press, as always.

My gratitude to the fine editor on this project, Mark Taber, for leading the way and putting up with too many delayed emails and chapters!

Thanks to David Van Ness and Elizabeth Welch for their hard work, helpful suggestions, and impressive attention to detail. Thanks to Scout Festa for ensuring the writing is “pixel perfect.” Thanks also to Valerie Perry for indexing and Danielle Foster for laying out the book, and thanks to Timothy Boronczyk for his technical review.

Kudos to the good people working on PHP, MySQL, Apache, phpMyAdmin, MAMP, and XAMPP, among other great projects. And a hearty “cheers” to the denizens of the various newsgroups, mailing lists, support forums, etc., who offer assistance and advice to those in need.

Thanks, as always, to the readers, whose support gives my job relevance. An extra helping of thanks to those who provided the translations in Chapter 17, “Example—Message Board,” and who offered up recommendations as to what they’d like to see in this edition.

Finally, I would not be able to get through a single book if it weren’t for the love and support of my wife, Jessica. And a special shout-out to Zoe and Sam, who give me reasons to, and not to, write books!

Table of Contents

Introduction

Today's web users expect exciting pages that are updated frequently and provide a customized experience. For them, web sites are more like communities, to which they'll return time and again. At the same time, site administrators want pages that are easier to update and maintain, understanding that's the only reasonable way to keep up with visitors' expectations. For these reasons and more, PHP and MySQL have become the de facto standards for creating dynamic, database-driven web sites.

This book represents the culmination of my many years of web development experience coupled with the value of having written several previous books on the technologies discussed herein. The focus of this book is on covering the most important knowledge in the most efficient manner. It will teach you how to begin developing dynamic web sites and give you plenty of example code to get you started. All you need to provide is an eagerness to learn.

Well, that and a computer.

What Are Dynamic Web Sites?

Dynamic web sites are flexible and potent creatures, more accurately described as *applications* than merely sites. Dynamic web sites

- Respond to different parameters (for example, the time of day or the version of the visitor's browser)
- Have a "memory," allowing for user registration and login, e-commerce, and similar processes
- Almost always integrate HTML forms, allowing visitors to perform searches, provide feedback, and so forth
- Often have interfaces where administrators can manage the content
- Are easier to maintain, upgrade, and build upon than statically made sites

Many technologies are available for creating dynamic web sites. The most common are ASP.NET (Active Server Pages, a Microsoft construct), JSP (JavaServer Pages), ColdFusion, Ruby on Rails (a web development framework for the Ruby programming language), and PHP. Dynamic sites don't always rely on a database, but more and more of them do, particularly as excellent database applications like MySQL and MongoDB are available at little to no cost.

What Happened to PHP 6?

When I wrote a previous edition of this book, *PHP 6 and MySQL 5 for Dynamic Web Sites: Visual QuickPro Guide*, the next major release of PHP—PHP 6—was approximately 50 percent complete. Thinking that PHP 6 would therefore be released sometime after the book was published, I relied on a beta version of PHP 6 for a bit of that edition's material. And then... PHP 6 died.

One of the key features planned for PHP 6 was support for Unicode, meaning that PHP 6 would be able to work natively with any language. This would be a great addition to an already popular programming tool. Unfortunately, implementing Unicode support went from being complicated to quite difficult, and the developers behind the language tabled development of PHP 6. Not all was lost, however; some of the other features planned for PHP 6, such as support for *namespaces* (an object-oriented programming concept), were added to PHP 5.3.

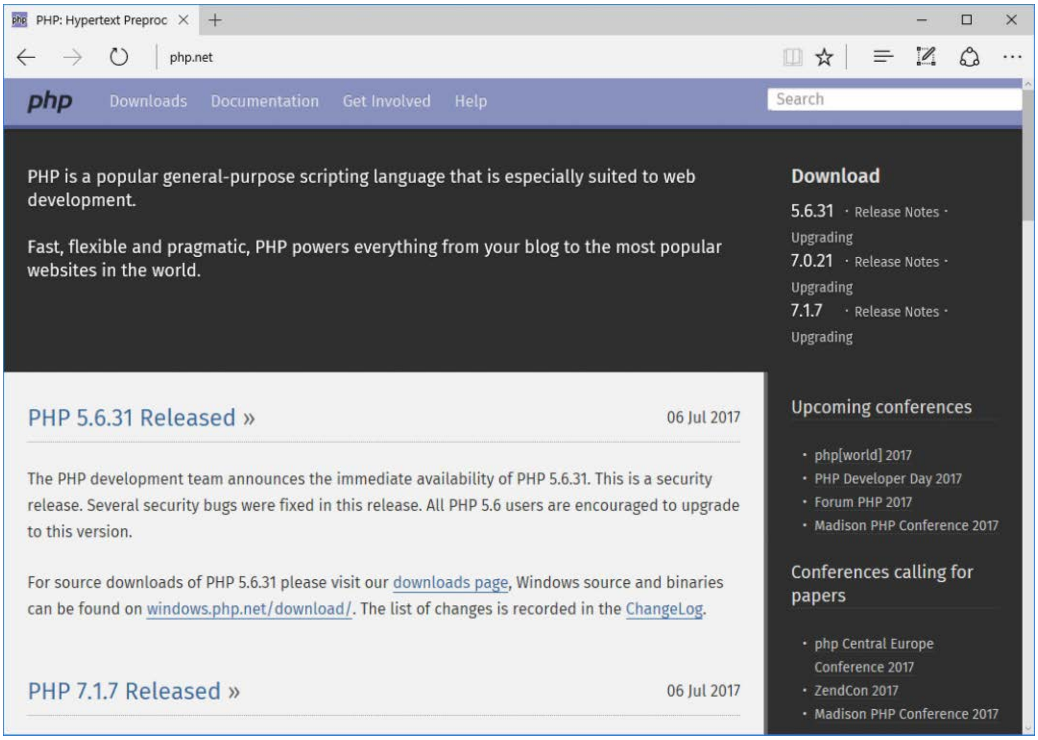
When it was time to release the next major version of PHP, it was decided to name it PHP 7 to avoid confusion with the PHP 6 version that was started but never completed.

What is PHP?

PHP originally stood for "Personal Home Page" when it was created in 1994 by Rasmus Lerdorf to track the visitors to his online résumé. As its usefulness and capabilities grew (and as it started being used in more professional situations), it came to mean "PHP: Hypertext Preprocessor."

According to the official PHP web site, found at **`www.php.net`** **A**, PHP is a "popular general-purpose scripting language that is especially suited to web development." It's a long but descriptive definition, whose meaning I'll explain.

continues on next page



PHP is a popular general-purpose scripting language that is especially suited to web development.

Fast, flexible and pragmatic, PHP powers everything from your blog to the most popular websites in the world.

Download

5.6.31 · Release Notes · Upgrading
7.0.21 · Release Notes · Upgrading
7.1.7 · Release Notes · Upgrading

PHP 5.6.31 Released » 06 Jul 2017

The PHP development team announces the immediate availability of PHP 5.6.31. This is a security release. Several security bugs were fixed in this release. All PHP 5.6 users are encouraged to upgrade to this version.

For source downloads of PHP 5.6.31 please visit our downloads page, Windows source and binaries can be found on windows.php.net/download/. The list of changes is recorded in the ChangeLog.

PHP 7.1.7 Released » 06 Jul 2017

Upcoming conferences

- php[world] 2017
- PHP Developer Day 2017
- Forum PHP 2017
- Madison PHP Conference 2017

Conferences calling for papers

- php Central Europe Conference 2017
- ZendCon 2017
- Madison PHP Conference 2017

A The home page for PHP.

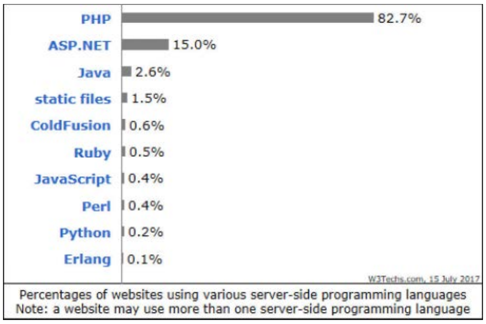
Starting at the end of that statement, to say that PHP *is especially suited to web development* means that although you can use PHP for non-web development purposes, it's best suited for that. The corollary is that although many other technologies can be used for web development, that may not be what they're best suited for. Simply put, if you're hoping to do web development, PHP is an excellent choice.

Also, PHP is a *scripting* language, as opposed to a *compiled* language: PHP was designed to write web scripts, not stand-alone applications (although, with some extra effort, you can create applications in PHP). PHP scripts run only after an event occurs—for example, when a user submits a form or goes to a URL (uniform resource locator, the technical term for a web site address).

I should add to this definition that PHP is a server-side, cross-platform technology, both descriptions being important. *Server-side* refers to the fact that everything PHP does occurs on the server. A web server application, like Apache or Microsoft's IIS (Internet Information Services), is required and all PHP scripts must be accessed through a URL (**`http://`*`something`***). Its cross-platform nature means that PHP runs on most operating systems, including Windows, Unix (and its many variants), and Macintosh. More important, the PHP scripts written on one server will normally work on another with little or no modification.

At the time this book was written, PHP was at version 7.1.7. Although PHP 7 is a major release, the most important changes are in its core, with PHP 7 being significantly more performant than PHP 5.

For the most part, the examples in this book will work fine so long as you're using at least version 5.4. Some functions and



B The Web Technology Surveys site provides this graphic regarding server-side technologies (`www.w3techs.com/technologies/overview/programming_language/all`).

features covered will require more specific or current versions, like PHP 5.6 or greater. In those cases, I will make it clear when the functionality was added to PHP, and provide alternative solutions if you have a slightly older version of the language.

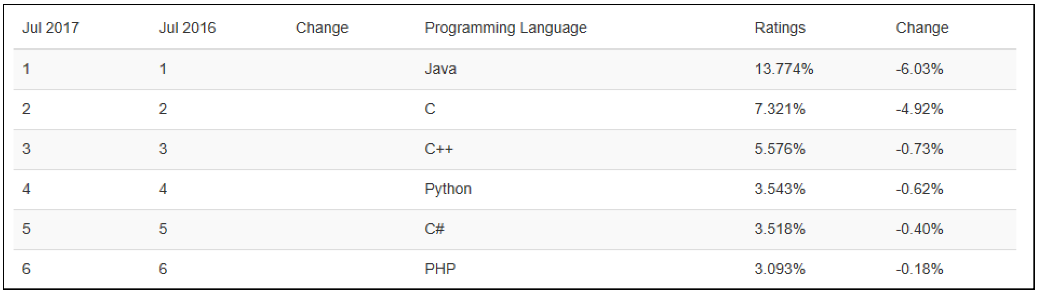
More information about PHP can always be found at PHP.net.

Why use PHP?

Put simply, when it comes to developing dynamic web sites, PHP is better, faster, and easier to learn than the alternatives. What you get with PHP is excellent performance, a tight integration with nearly every database available, stability, portability, and a nearly limitless feature set due to its extendibility. All of this comes at no cost (PHP is open source) and with a very manageable learning curve. PHP is one of the best marriages I've ever seen between the ease with which beginning programmers can start using it and the ability for more advanced programmers to do everything they require.

Finally, the proof is in the pudding: PHP has seen an exponential growth in use since its inception, and is the server-side technology of choice on over 82 percent of all web sites **B**. In terms of all programming languages, PHP is the sixth most popular **C**.

continues on next page



Jul 2017	Jul 2016	Change	Programming Language	Ratings	Change
1	1		Java	13.774%	-6.03%
2	2		C	7.321%	-4.92%
3	3		C++	5.576%	-0.73%
4	4		Python	3.543%	-0.62%
5	5		C#	3.518%	-0.40%
6	6		PHP	3.093%	-0.18%

C The Tiobe Index (`https://www.tiobe.com/tiobe-index/`) uses a combination of factors to rank the popularity of programming languages.

Of course, you might assume that I, as the author of a book on PHP (several, actually), have a biased opinion. Although not nearly to the same extent as I have with PHP, I've also developed sites using JavaServer Pages (JSP), Ruby on Rails (RoR), Sinatra (another Ruby web framework), and ASP. NET. Each has its pluses and minuses, but PHP is the technology I always return to. You might hear that it doesn't perform or scale as well as other technologies, but Yahoo, Wikipedia, and Facebook all use PHP, and you can't find many sites more visited or demanding than those.

You might have heard that PHP is less secure. But *security isn't in the language*; it's in how that language is used. Rest assured that a complete and up-to-date discussion of all the relevant security concerns is provided by this book.

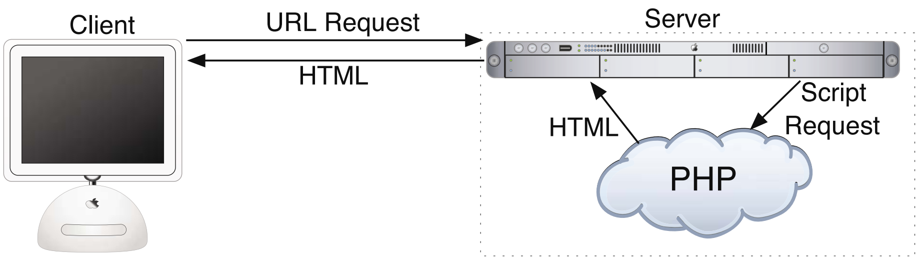
How PHP works

As previously stated, PHP is a server-side language. This means that the code you write in PHP sits on a host computer called a *server*. The server sends web pages to the requesting visitors (you, the client, with your browser).

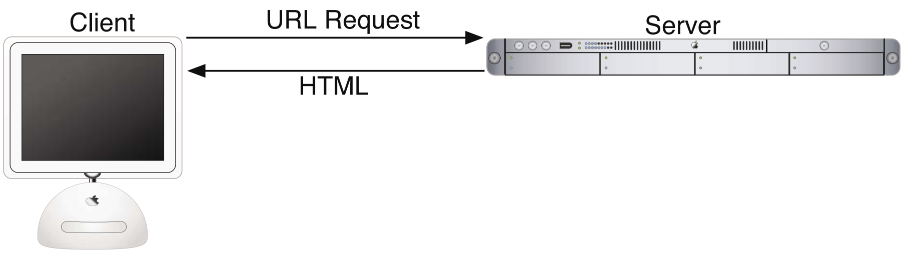
When a visitor goes to a site written in PHP, the server reads the PHP code and then processes it according to its scripted directions. In the example shown in **D**, the PHP code tells the server to send the appropriate data—HTML code—to the browser, which treats the received code as it would a standard HTML page.

This differs from a static HTML site where, when a request is made, the server merely sends the HTML data to the browser and there is no server-side interpretation occurring **E**. Because no server-side action is required, you can run HTML pages in your browser without using a server at all.

continues on next page



D How PHP fits into the client/server model when a user requests a page.



E The client/server process when a request for a static HTML page is made.

To the end user and the browser there is no perceptible difference between what **`home.html`** and **`home.php`** may look like, but how that page's content was created will be significantly different.

What is MySQL?

MySQL (**`www.mysql.com`**) **F** is the world's most popular open source database. In fact, today MySQL is a viable competitor to pricey goliaths such as Oracle and Microsoft's SQL Server (and, ironically, MySQL is owned by Oracle). Like PHP, MySQL offers excellent performance, portability, and reliability, with a moderate learning curve and little to no cost.

MySQL is a database management system (DBMS) for relational databases (therefore, MySQL is an RDBMS). A database, in the simplest terms, is a collection of data, be it text, numbers, or binary files, stored and kept organized by the DBMS.

There are many types of databases, from the simple flat-file to relational to object-oriented to NoSQL. A relational database uses multiple tables to store information in its most discernible parts. Although relational databases may involve more thought in the design and programming stages, they offer improved reliability and data integrity that more than make up for the extra effort required. Further, relational databases are more searchable and allow for concurrent users.

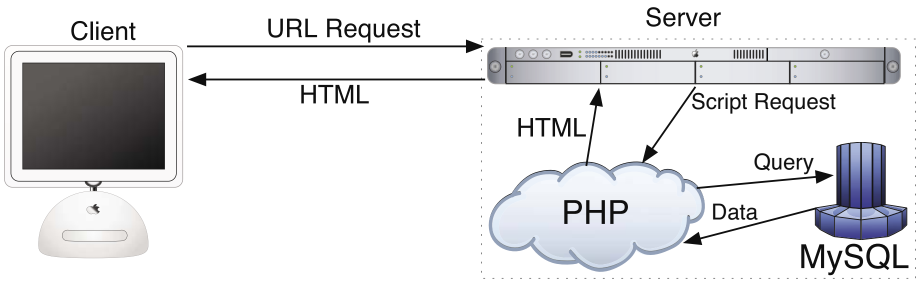
F The home page for the MySQL database application.

By incorporating a database into a web application, some of the data generated by PHP can be retrieved from MySQL **G**. This further moves the site's content from a static (hard-coded) basis to a flexible one, flexibility being the key to a dynamic web site.

MySQL is an open source application, like PHP, meaning that it is free to use or even modify (the source code itself is downloadable). There are occasions when you should pay for a MySQL license, especially if you are making money from the sales or incorporation of the MySQL product. Check MySQL's licensing policy for more information on this.

The MySQL software consists of several pieces, including the MySQL server (*mysqld*, which runs and manages the databases), the MySQL client (*mysql*, which gives you an interface to the server), and numerous utilities for maintenance and other purposes. PHP has always had good support for MySQL, and that is even truer in the most recent versions of the language.

continues on next page



G How most of the dynamic applications in this book will work, using both PHP and MySQL.

MySQL has been known to handle databases as large as 60,000 tables with more than several billion rows. MySQL can work with tables as large as thousands of terabytes on some operating systems, generally a healthy 4 GB otherwise. MySQL is used by NASA and the U.S. Census Bureau, among many others.

As of this writing, MySQL is on version 5.7.18. The version of MySQL you have affects what features you can use, so it’s important that you know what you’re working with. For this book, MySQL 5.7.14 was used, although you should be able to do everything in this book as long as you’re using a version of MySQL greater than 5.0.

Pronunciation Guide

Trivial as it may be, I should clarify up front that MySQL is technically pronounced “My Ess Cue Ell,” just as SQL should be said “Ess Cue Ell.” This is a question many people have when first working with these technologies. Though not a critical issue, it’s always best to pronounce acronyms correctly.

What You'll Need

To follow the examples in this book, you'll need the following tools:

- A web server application (for example, Apache, Nginx, or IIS)
- PHP
- MySQL
- A browser (Microsoft's Internet Explorer or Edge, Mozilla's Firefox, Apple's Safari, Google's Chrome, etc.)
- A text editor, PHP-capable WYSIWYG application (Adobe's Dreamweaver qualifies), or IDE (integrated development environment)
- An FTP application, if using a remote server

One of the great things about developing dynamic web sites with PHP and MySQL is that all of the requirements can be met at no cost whatsoever, regardless of your operating system! Apache, PHP, and MySQL are each free, browsers can be had without cost, and many good text editors are available for nothing.

The appendix discusses the installation process on the Windows and macOS operating systems. If you have a computer, you are only a couple of downloads away from being able to create dynamic web sites (in that case, your computer would represent both the client and the server in D and E). Conversely, you could purchase web hosting for only dollars per month that will provide you with a PHP- and MySQL-enabled environment already online.

About This Book

This book teaches you how to develop dynamic web sites with PHP and MySQL, covering the knowledge that most developers might require. In keeping with the format of the Visual QuickPro series, the information is discussed using a step-by-step approach with corresponding images. The focus has been kept on real-world, practical examples, avoiding “here’s something you could do but never would” scenarios. As a practicing web developer myself, I wrote about the information that I use and avoided those topics immaterial to the task at hand. As a practicing writer, I made certain to include topics and techniques that I know readers are asking about.

The structure of the book is linear, and the intention is that you’ll read it in order. It begins with three chapters covering the fundamentals of PHP (by the second chapter, you will have already developed your first dynamic web page). After that, there are four chapters on SQL (Structured Query Language, which is used to interact with all databases) and MySQL. Those chapters teach the basics of SQL, database design, and the MySQL application in particular. Then there’s one chapter on debugging and error management, information everyone needs. This is followed by a chapter introducing how to use PHP and MySQL together, a remarkably easy thing to do.

The following five chapters teach more application techniques to round out your knowledge. Security, in particular, is repeatedly addressed in those pages. The next two chapters expand your newfound knowledge into subjects that, though not critical, are ones you’ll want to pick up in time regardless. Finally, I’ve included two example chapters, in which the heart of different web applications are developed, with instructions.

Is this book for you?

This book was written for a wide range of people within the beginner-to-intermediate range. The book makes use of HTML5, so solid experience with HTML is a must. Although this book covers many things, it does not formally teach HTML or web design. Some CSS is sprinkled about these pages but also not taught.

Second, this book expects that you have one of the following:

- The drive and ability to learn without much hand holding, or...
- Familiarity with another programming language (even solid JavaScript skills would qualify), or...
- A cursory knowledge of PHP

Make no mistake: This book covers PHP and MySQL from A to Z, teaching everything you'll need to know to develop real-world web sites, but the early chapters in particular cover PHP at a quick pace. For this reason I recommend either some programming experience or a curious and independent spirit when it comes to learning new things. If you find that the material goes too quickly, you should probably start off with the latest edition of my book *PHP for the World Wide Web: Visual QuickStart Guide*, which goes at a much more tempered pace.

No database experience is required, since SQL and MySQL are discussed starting at a more basic level.

What's new in this edition

The first four editions of this book have been very popular, and I've received a lot of positive feedback on them (thanks!). In writing this new edition, I focused on ensuring the material is accurate, up to date, and in keeping with today's standards and best practices. The changes in this edition include

- Updating all the code to use HTML5
- Use of more modern HTML design techniques, including multiple examples of the Twitter Bootstrap framework
- Updating everything for the latest versions of PHP and MySQL
- Additional PHP and MySQL examples, such as performing transactions from a PHP script
- Even more information and examples for improving the security of your scripts and sites
- Removal of outdated content (e.g., things used in older versions of PHP or no longer applicable)
- Return of the installation appendix to the printed book (in the fourth edition, the appendix was freely available online instead)

For those of you that also own a previous edition (thanks, thanks, thanks!), I hope you find this to be a fresh and sharp update to an already excellent resource.

How this book compares to my other books

This is my fourth PHP and/or MySQL title, after (in order)

- *PHP for the World Wide Web: Visual QuickStart Guide*
- *PHP Advanced and Object-Oriented Programming: Visual QuickPro Guide*
- *MySQL: Visual QuickStart Guide*

I hope this résumé implies a certain level of qualification to write this book, but how do you, as a reader standing in a bookstore, decide which title is for you? Of course, you are more than welcome to splurge and buy the whole set, earning my eternal gratitude, but…

The *PHP for the World Wide Web: Visual QuickStart Guide* book is very much a beginner's guide to PHP. This title overlaps it some, mostly in the first three chapters, but uses new examples so as not to be redundant. For novices, this book acts as a follow-up to that one. The advanced book is really a sequel to this one, as it assumes a fair amount of knowledge and builds on many things taught here. The MySQL book focuses almost exclusively on MySQL (there are but two chapters that use PHP).

continues on next page

With that in mind, read the section "Is this book for you?" and see if the requirements apply. If you have no programming experience at all and would prefer to be taught PHP more gingerly, my first book would be better. If you are already very comfortable with PHP and want to learn more of its advanced capabilities, pick up *PHP Advanced and Object-Oriented Programming: Visual QuickPro Guide*. If you are most interested in MySQL and are not concerned with learning much about PHP, check out *MySQL: Visual QuickStart Guide*.

That being said, if you want to learn everything you need to know to begin developing dynamic web sites with PHP and MySQL today, then this is the book for you! It references the most current versions of both technologies, uses techniques not previously discussed in other books, and contains its own unique examples.

And whatever book you do choose, make sure you're getting the most recent edition or, barring that, the edition that best matches the versions of the technologies you'll be using.

Companion Web Site

I have developed a companion web site specifically for this book, which you may reach at `LarryUllman.com`. There you will find every script from this book, a text file containing lengthy SQL commands, and a list of errata that occurred during publication. (If you have problems with a command or script, and you are following the book exactly, check the errata to ensure there is not a printing error before driving yourself absolutely mad.) At this web site you will also find a popular forum where readers can ask and answer each other's questions (I answer many of them myself), and more!

Questions, comments, or suggestions?

If you have any questions on PHP or MySQL, you can turn to one of the many web sites, mailing lists, newsgroups, and FAQ repositories already in existence. A quick search online will turn up virtually unlimited resources. For that matter, if you need an immediate answer, those sources or a quick online search will most assuredly serve your needs (in all likelihood, someone else has already seen and solved your exact problem).

You can also direct your questions, comments, and suggestions to me. You'll get the fastest reply using the book's corresponding forum (I always answer those questions first). If you'd rather email me, my contact information is available on my site. I do try to answer every email I receive, although I cannot guarantee a quick reply.

Accessing the free Web Edition

Your purchase of this book in any format includes access to the corresponding Web Edition, which provides several special online-only features:

- The complete text of the book, with all the figures and in full color
- Updates and corrections as they become available

The Web Edition can be viewed on all types of computers and mobile devices with any modern web browser that supports HTML5. To get access to the Web Edition of *PHP and MySQL for Dynamic Web Sites: Visual QuickPro Guide* all you need to do is register this book:

1. Go to **`www.peachpit.com/register`**.
2. Sign in or create a new account.
3. Enter ISBN: 9780134301846.
4. Answer the questions as proof of purchase.

The Web Edition will appear under the Digital Purchases tab on your Account page. Click the Launch link to access the product.

4

Introduction to MySQL

Because this book discusses how to integrate several technologies—primarily PHP, SQL, and MySQL—a solid understanding of each is important before you begin writing PHP scripts that use SQL to interact with MySQL. This chapter is a departure from its predecessors in that it temporarily leaves PHP behind to delve into MySQL.

MySQL is the world's most popular open source database application (according to MySQL's web site, `www.mysql.com`) and is commonly used with PHP. The MySQL software comes with the database server that stores the actual data, different client applications for interacting with the database server, and several utilities. In this chapter, you'll see how to define a simple table using MySQL's allowed data types and other properties. Then you'll learn how to interact with the MySQL server using two different client applications. This information will be the foundation for the SQL taught in the next chapter.

In This Chapter

Naming Database Elements

Before you start working with databases, you have to identify your needs. The purpose of the application (or web site, in this case) dictates how the database should be designed. With that in mind, the examples in this chapter and the next will use a database that stores some user registration information.

When creating databases and tables, you should come up with names (formally called *identifiers*) that are clear, meaningful, and easy to type. Also, identifiers

- Should only contain letters, numbers, and the underscore (no spaces)
- Should not be the same as an existing keyword (like an SQL term or a function name)
- Should be treated as case-sensitive
- Cannot be longer than 64 characters (approximately)
- Must be unique within its realm

This last rule means that a table cannot have two columns with the same name and a database cannot have two tables with the same name. You can, however, use the same column name in two different tables in the same database; in fact, you often will do this.

As for the first three rules, I use the word *should*, as these are good policies more than exact requirements. Exceptions can be made to these rules, but the syntax for doing so can be complicated. Abiding by these suggestions is a reasonable limitation and will help avoid complications.

To name a database's elements:

1. Determine the database's name.

   This is the easiest and, arguably, least important step. Just make sure that the database name is unique for that MySQL server. If you're using a hosted server, your web host will likely provide a database name that may or may not include your account or domain name.

   For this first example, the database will be called *sitename*, since the information and techniques could apply to any generic site.

2. Determine the table names.

   The table names just need to be unique within this database, which shouldn't be a problem. For this example, which stores user registration information, the only table will be called *users*.

TABLE 4.1 users Table

Column Name	Example
`user_id`	834
`first_name`	Larry
`last_name`	David
`email`	ld@example.com
`pass`	emily07
`registration_date`	2017-08-31 19:21:03

3. Determine the column names for each table.

   The *users* table will have columns to store a user ID, a first name, a last name, an email address, a password, and the registration date. **Table 4.1** shows these columns, with sample data, using proper identifiers. Because MySQL has a function called *password*, I've changed the name of that column to just *pass*. This isn't strictly necessary but is really a good idea.

   For the *user_id* column, there are two common approaches. Some use simply *id* as the identifying column name in any table so that all tables have an *id* column. Others use a variation on *tablename_id*: *user_id* or *users_id*.

TIP Chapter 6, "Database Design," discusses database design in more detail, using more complex examples.

TIP To be precise, the length limit for the names of databases, tables, and columns is actually 64 bytes, not characters. While most characters in many languages require 1 byte apiece, it's possible to use a multibyte character in an identifier. But 64 bytes is still a lot of space, so this probably won't be an issue for you.

TIP Whether or not an identifier in MySQL is case-sensitive actually depends on many things, because each database is actually a folder on the server and each table is actually one or more files. On Windows and normally on macOS, database and table names are generally case-insensitive. On Unix and some macOS setups, they are case-sensitive. Column names are always case-insensitive. It's really best, in my opinion, to always use all lowercase letters and work as if case-sensitivity applied.

Choosing Your Column Types

Once you have identified all of the tables and columns that the database will need, you should determine each column's data type. When you're creating a table, MySQL requires that you explicitly state what sort of information each column will contain. There are three primary types, which is true for almost every database application:

- Text (aka *strings*)
- Numbers
- Dates and times

Within each of these, there are many variants—some of which are MySQL specific. Choosing your column types correctly not only dictates what information can be stored and how, but also affects the database's overall performance. **Table 4.2** lists most of the available types for MySQL, how much space they take up, and brief descriptions of each type. Note that some of these limits may change in different versions of MySQL, and the character set (to be discussed in Chapter 6, "Database Design") may also impact the size of the text types.

Many of the types can take an optional *Length* attribute, limiting their size. (The brackets, [ ], indicate an optional parameter to be put in parentheses.) For performance purposes, you should place some restrictions on how much data can be stored in any column. But understand that attempting to insert a string five characters long into a **`CHAR(2)`** column will result in truncation of the final three characters. Only the first two characters would be stored; the rest would be lost forever. This is true for any field in which the size is set (**`CHAR`**, **`VARCHAR`**, **`INT`**, etc.). Thus, your length should always correspond to the maximum possible value—as a number—or the longest possible string—as text—that might be stored.

The various date types have all sorts of unique behaviors, the most important of which you'll learn about in this book. All the behaviors are documented in the MySQL manual. You'll use the **`DATE`** and **`TIME`** fields primarily without modification, so you do not have to worry too much about their intricacies.

There are also two special types—**`ENUM`** and **`SET`**—that allow you to define a series of acceptable values for that column. An **`ENUM`** column can store only one value of a possible several thousand, whereas **`SET`** allows for several of up to 64 possible values. These are available in MySQL but aren't present in every database application.

TABLE 4.2 MySQL Data Types

Type	Size	Description
`CHAR[Length]`	Length bytes	A fixed-length field from 0 to 255 characters long
`VARCHAR[Length]`	String length + 1 or 2 bytes	A variable-length field from 0 to 65,535 characters long
`TINYTEXT`	String length + 1 bytes	A string with a maximum length of 255 characters
`TEXT`	String length + 2 bytes	A string with a maximum length of 65,535 characters
`MEDIUMTEXT`	String length + 3 bytes	A string with a maximum length of 16,777,215 characters
`LONGTEXT`	String length + 4 bytes	A string with a maximum length of 4,294,967,295 characters
`TINYINT[Length]`	1 byte	Range of –128 to 127 or 0 to 255 unsigned
`SMALLINT[Length]`	2 bytes	Range of –32,768 to 32,767 or 0 to 65,535 unsigned
`MEDIUMINT[Length]`	3 bytes	Range of –8,388,608 to 8,388,607 or 0 to 16,777,215 unsigned
`INT[Length]`	4 bytes	Range of –2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 unsigned
`BIGINT[Length]`	8 bytes	Range of –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 or 0 to 18,446,744,073,709,551,615 unsigned
`FLOAT[Length, Decimals]`	4 bytes	A small number with a floating decimal point
`DOUBLE[Length, Decimals]`	8 bytes	A large number with a floating decimal point
`DECIMAL[Length, Decimals]`	Length + 1 or 2 bytes	A `DOUBLE` stored as a string, allowing for a fixed decimal point
`DATE`	3 bytes	In the format YYYY-MM-DD
`DATETIME`	8 bytes	In the format YYYY-MM-DD HH:MM:SS
`TIMESTAMP`	4 bytes	In the format YYYYMMDDHHMMSS; acceptable range starts in 1970 and ends in the year 2038
`TIME`	3 bytes	In the format of HH:MM:SS
`ENUM`	1 or 2 bytes	Short for enumeration, which means that each column can have one of several possible values
`SET`	1, 2, 3, 4, or 8 bytes	Like `ENUM` except that each column can have more than one of several possible values

To select the column types:

1. Identify whether a column should be a text, number, or date/time type (**Table 4.3**).

   This is normally an easy and obvious step, but you want to be as specific as possible. For example, the date *2006-08-02* (MySQL format) could be stored as a string—*August 2, 2006*. But if you use the proper date format, you'll have a more useful database (and, as you'll see, there are functions that can turn *2006-08-02* into *August 2, 2006*).

2. Choose the most appropriate subtype for each column (**Table 4.4**).

   For this example, *user_id* is set as a **MEDIUMINT**, allowing for up to nearly 17 million values (as an *unsigned*, or non-negative, number). *registration_date* will be a **DATETIME**. It can store both the date and the specific time a user registered. When deciding among the date types, consider whether you'll want to access just the date, the time, or possibly both.

   When choosing a subtype, err on the side of storing too much information.

   The other fields will be mostly **VARCHAR**, since their lengths will differ from record to record. The only exception is the password column, which will be a fixed-length **CHAR** (you'll see why when inserting records in the next chapter). See the sidebar "**CHAR** vs. **VARCHAR**" for more information on these two types.

CHAR vs. VARCHAR

Both of these types store strings and can be set with a maximum length. The primary difference between the two is that anything stored as a **CHAR** will always be stored as a string the length of the column (using spaces to pad it; these spaces will be removed when you retrieve the stored value from the database). Conversely, strings stored in a **VARCHAR** column will require only as much space as the string itself. So the word *cat* in a **VARCHAR(10)** column requires 4 bytes of space (the length of the string plus 1), but in a **CHAR(10)** column, that same word requires 10 bytes of space. Hence, generally speaking, **VARCHAR** columns tend to require less disk space than **CHAR** columns.

However, databases are normally faster when working with fixed-size columns, which is an argument in favor of **CHAR**. And that same three-letter word—*cat*—in a **CHAR(3)** uses only 3 bytes but in a **VARCHAR(10)** requires 4. So how do you decide which to use?

If a string field will *always* be of a set length (e.g., a state abbreviation), use **CHAR**; otherwise, use **VARCHAR**. You may notice, though, that in some cases MySQL defines a column as the one type—like **CHAR**—even though you created it as the other: **VARCHAR**. This is perfectly normal and is MySQL's way of improving performance.

TABLE 4.3 users Table

Column Name	Type
`user_id`	number
`first_name`	text
`last_name`	text
`email`	text
`pass`	text
`registration_date`	date/time

TABLE 4.4 users Table

Column Name	Type
`user_id`	`MEDIUMINT`
`first_name`	`VARCHAR`
`last_name`	`VARCHAR`
`email`	`VARCHAR`
`pass`	`CHAR`
`registration_date`	`DATETIME`

TABLE 4.5 `users` Table

Column Name	Type
`user_id`	`MEDIUMINT`
`first_name`	`VARCHAR(20)`
`last_name`	`VARCHAR(40)`
`email`	`VARCHAR(60)`
`pass`	`CHAR(128)`
`registration_date`	`DATETIME`

3. Set the maximum length for text columns (**Table 4.5**).

   The size of any field should be restricted to the smallest possible value, based on the largest possible input. For example, if a column stores a state abbreviation, it would be defined as a `CHAR(2)`. Other times you might have to guess: I can't think of any first names longer than about 10 characters, but just to be safe I'll allow for up to 20.

TIP The length attribute for numeric types does not affect the range of values that can be stored in the column. Columns defined as `TINYINT(1)` or `TINYINT(20)` can store the exact same values. Instead, for integers, the length dictates the display width; for decimals, the length is the total number of digits that can be stored.

TIP If you need absolute precision when using non-integers, `DECIMAL` is preferred over `FLOAT` or `DOUBLE`.

TIP MySQL has a `BOOLEAN` type, which is just a `TINYINT(1)`, with 0 meaning FALSE and 1 meaning TRUE.

TIP Many of the data types have synonymous names: `INT` and `INTEGER`, `DEC` and `DECIMAL`, and so on.

TIP Depending on the version of MySQL in use, the `TIMESTAMP` field type is automatically set as the current date and time when an `INSERT` or `UPDATE` occurs, even if no value is specified for that particular field. If a table has multiple `TIMESTAMP` columns, only the first one will be updated when an `INSERT` or `UPDATE` is performed.

TIP MySQL also has several variants on the text types that allow for storing binary data. These types are `BINARY`, `VARBINARY`, `TINYBLOB`, `MEDIUMBLOB`, and `LONGBLOB`. Such types can be used for storing files or encrypted data.

Choosing Other Column Properties

Besides deciding what data types and sizes you should use for your columns, consider a handful of other properties.

First, every column, regardless of type, can be defined as **`NOT NULL`**. The **`NULL`** value, in databases and programming, is equivalent to saying that the field has no known value. Ideally, in a properly designed database, every column of every row in every table should have a value, but that isn't always the case. To force a field to have a value, add the **`NOT NULL`** description to its column type. For example, a required dollar amount can be described as

```
cost DECIMAL(5,2) NOT NULL.
```

Indexes, Keys, and AUTO_INCREMENT

Two concepts closely related to database design are indexes and keys. An *index* in a database is a way of requesting that the database keep an eye on the values of a specific column or combination of columns (loosely stated). The benefit of an index is improved performance when retrieving records but marginally hindered performance when inserting records or updating them.

A *key* in a database table is integral to the "normalization" process used for designing more complicated databases (see Chapter 6). There are two types of keys: *primary* and *foreign*. Each table should have exactly one primary key, and the primary key in one table is often linked as a foreign key in another.

A table's primary key is an artificial way to refer to a record and must abide by three rules:

1. It must always have a value.
2. That value must never change.
3. That value must be unique for each record in the table.

In the *users* table, *user_id* will be designated as a **`PRIMARY KEY`**, which is both a description of the column and a directive to MySQL to index it. Since *user_id* is a number—which primary keys almost always will be, the **`AUTO_INCREMENT`** description is also added to the column, which tells MySQL to use the next-highest number as the *user_id* value for each added record. You'll see what this means in practice when you begin inserting records.

When creating a table, you can also specify a default value for any column, regardless of type. In cases where a majority of the records will have the same value for a column, presetting a default will save you from having to specify a value when inserting new rows (unless that row's value for that column is different from the norm).

```
subscribe ENUM('Yes', 'No') default 'No'
```

With the *subscribe* column, if no value is specified when adding a record, the default will be used.

If a column cannot be **`NULL`** and does not have a default value, and no value is specified for a new record, that field will be given a default value based on its type. For numeric types, the default value is 0. For most date and time types, the type's version of "zero" will be the default (e.g., *0000-00-00*). The first **`TIMESTAMP`** column in a table will have a default value of the current date and time. String types use an empty string (**`''`**) as the default value, except for **`ENUM`**, whose default value—again, if not otherwise specified—is the first possible enumerated value (*Yes* in the previous example).

The number types can be marked as **`UNSIGNED`**, which limits the stored data to positive numbers and zero. This also effectively doubles the range of positive numbers that can be stored because no negative numbers will be kept (see Table 4.2). You can also flag the number types as **`ZEROFILL`**, which means that any extra room will be padded with zeros. **`ZEROFILL`**s are also automatically **`UNSIGNED`**.

Finally, when designing a database, you'll need to consider creating indexes, adding keys, and using the **`AUTO_INCREMENT`** property. Chapter 6 discusses these concepts in greater detail, but in the meantime, check out the sidebar "Indexes, Keys, and **`AUTO_INCREMENT`**" to learn how they affect the *users* table.

To finish defining your columns:

1. Identify your primary key.

   The primary key is quixotically both arbitrary and critically important. Almost always a number value, the primary key is a unique way to refer to a particular record. For example, your phone number has no inherent value but is unique to you (your home or mobile phone).

   In the *users* table, *user_id* will be the primary key: an arbitrary number used to refer to a row of data. Again, Chapter 6 will go into the concept of primary keys in more detail.

2. Identify which columns cannot have a **`NULL`** value.

   In this example, every field is required (cannot be **`NULL`**). As an example of a column that could have **`NULL`** values, if you stored people's addresses, you might have *address_line1* and *address_line2*, with the latter one being optional. In general, tables that have a lot of **`NULL`** values suggest a poor design (more on this in…you guessed it…Chapter 6).

continues on next page

3. Make any numeric type **UNSIGNED** if it won't ever store negative numbers.

   user_id, which will be a number, should be **UNSIGNED** so that it's always positive. As a rule, primary keys should always be unsigned. Other examples of **UNSIGNED** numbers would be the price of items in an e-commerce example, a telephone extension for a business, or a zip code.

4. Establish the default value for any column.

   None of the columns here logically implies a default value.

5. Confirm the final column definitions (**Table 4.6**).

   Before creating the tables, you should revisit the type and range of data you'll store to make sure that your database effectively accounts for everything.

TIP Text columns can also have defined character sets and collations. This will mean more...in Chapter 6.

TIP Default values must always be a static value, not the result of executing a function, with one exception: the default value for a TIMESTAMP column can be assigned as CURRENT_TIMESTAMP.

TIP TEXT columns cannot be assigned default values.

TABLE 4.6 users Table

Column Name	Type
user_id	MEDIUMINT UNSIGNED NOT NULL
first_name	VARCHAR(20) NOT NULL
last_name	VARCHAR(40) NOT NULL
email	VARCHAR(60) NOT NULL
pass	CHAR(128) NOT NULL
registration_date	DATETIME NOT NULL

Accessing MySQL

To create tables, add records, and request information from a database, you need some sort of *client* to communicate with the MySQL server. Later in the book, PHP scripts will act in this role, but being able to use another interface is necessary.

Although oodles of client applications are available, I'll focus on two: the *mysql client* and the web-based phpMyAdmin. A third option, the MySQL Workbench, is not discussed in this book but can be found at the MySQL web site (**`https://dev.mysql.com/downloads/workbench/`**), should you not be satisfied with these two choices.

The rest of this chapter assumes you have access to a running MySQL server. If you are working on your own computer, see Appendix A, "Installation," for instructions on installing MySQL, starting MySQL, and creating MySQL users, all of which must already be done in order to finish this chapter. If you are using a hosted server, your web host should provide you with the database access. Depending on the hosting, you may be provided with phpMyAdmin but not be able to use the command-line mysql client.

Using the mysql client

The mysql client is normally installed with the rest of the MySQL software. Although the mysql client does not have a pretty graphical interface, it's a reliable, standard tool that's easy to use and behaves consistently on many different operating systems.

The mysql client is accessed from a command-line interface, be it the Terminal application in Linux or macOS **A**, or a DOS prompt in Windows **B**. If you're not comfortable with command-line interactions, you might find this interface to be challenging, but it becomes easy to use in no time.

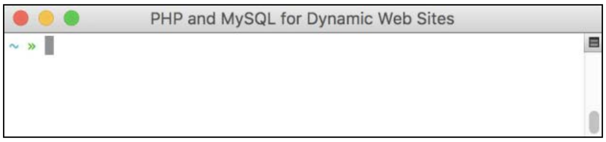
To start the application from the command line, type its name and press Return or Enter:

```
mysql
```

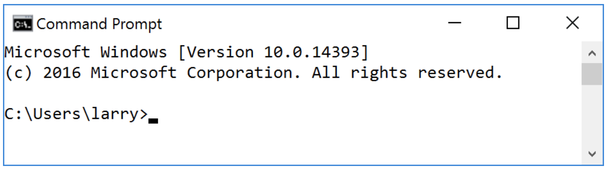
Depending on the server (or your computer), you may need to enter the full path to start the application. For example:

- **`/Applications/MAMP/Library/bin/`** → **`mysql`** (macOS, using MAMP)
- **`C:\xampp\mysql\bin\mysql`** (Windows, using XAMPP)

continues on next page



A A Terminal window in macOS.



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\larry>_
```

B A Windows DOS prompt or console (although the default is for white text on a black background).

When invoking this application, you can add arguments to affect how it runs. The most common arguments are the username, password, and hostname (computer name, URL, or IP address) you want to use to connect. You establish these arguments like so:

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

The **-p** option will cause the client to prompt you for the password. You can also specify the password on this line if you prefer—by typing it directly after the **-p** prompt—but it will be visible, which is insecure. The **-h hostname** argument is optional, and you can leave it off unless you cannot connect to the MySQL server without it.

Within the mysql client, every statement (SQL command) needs to be terminated by a semicolon. These semicolons are an indication to the client that the query is complete and should be run. The semicolons, a common point of confusion, are not part of the SQL itself. What this also means is that you can continue the same SQL statement over several lines within the mysql client, which makes it easy to read and to edit, should that be necessary.

As a quick demonstration of accessing and using the mysql client, these next steps will show you how to start the mysql client, select a database to use, and quit the client. Before following these steps,

- The MySQL server must be running.
- You must have a username and password with proper access.

Both are explained in Appendix A.

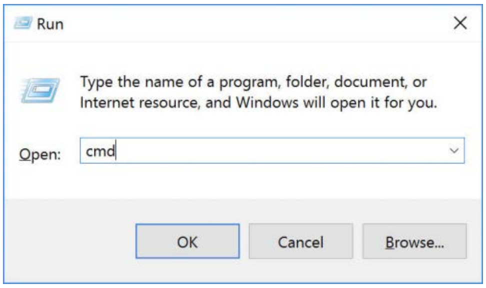
As a side note, in the following steps and throughout the rest of the book, I will continue to provide images using the mysql client on both Windows and macOS. Although the appearance differs, the steps and results will be identical. So in short, don't be concerned about why one image shows the DOS prompt and the next a Terminal.

To use the mysql client:

1. Access your system from a command-line interface.

   On Unix systems and macOS, this is just a matter of bringing up the Terminal or a similar application.

   If you are using Windows and you have installed MySQL on your computer, or press Windows Key+R, type **cmd** in the window **C**, and press Enter (or click OK) to bring up a DOS prompt.


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
cmd
OK
Cancel
Browse...

C Executing **cmd** within the Run prompt in Windows is one way to access a DOS prompt interface.

2. Invoke the mysql client, using the appropriate command **D**.

   ```
   /path/to/mysql/bin/mysql -u
   ➝ username -p
   ```

   The ***/path/to/mysql*** part of this step will be largely dictated by the operating system you are running and where MySQL was installed. I've already provided two options, based on installations of MAMP on macOS or XAMPP on Windows (both are installed in Appendix A).

   The basic premise is that you are running the mysql client, connecting as *username*, and requesting to be prompted for the password. Not to overstate the point, but the username and password values that you use must already be established in MySQL as valid (see Appendix A).

3. Enter the password at the prompt and press Return/Enter.

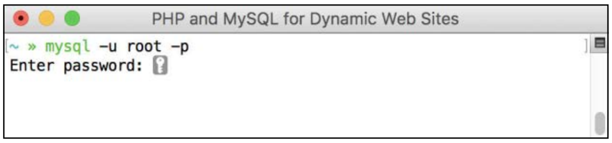
   The password you use here should be for the user you specified in the preceding step. If you used the proper username/password combination (i.e., someone with valid access), you should be greeted as shown in **E**. If access is denied, you're probably not using the correct values (see Appendix A for instructions on creating users).

4. Select the database you want to use **F**.

   ```
   USE test;
   ```

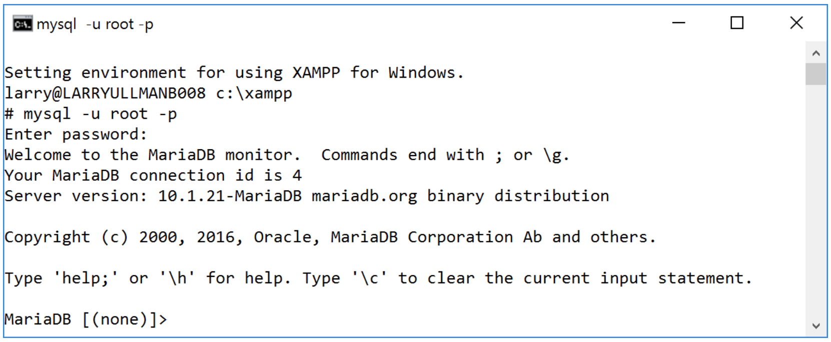
   The **USE** command selects the database to be used for every subsequent command. The *test* database is one that MySQL installs by default. Assuming it exists on your server, all users should be able to access it.

continues on next page



```
~ » mysql -u root -p
Enter password:
```

D Access the mysql client by entering the full path to the utility, along with the proper arguments.



```
Setting environment for using XAMPP for Windows.
larry@LARRYULLMANB008 c:\xampp
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.1.21-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

E If you are successfully able to log in, you'll see a welcome message like this.

5. Quit out of mysql G.

   ```
   exit
   ```

   You can also use the command **quit** to leave the client. This step—unlike most other commands you enter in the mysql client—does not require a semicolon at the end.

6. Quit the Terminal or DOS console session.

   ```
   exit
   ```

   The command **exit** will terminate the current session. On Windows, it will also close the DOS prompt window.

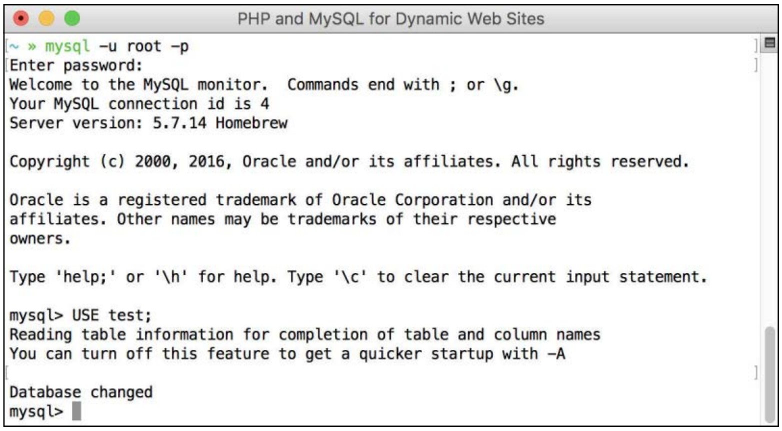
TIP If you know in advance which database you will want to use, you can simplify matters by starting mysql with

```
/path/to/mysql/bin/mysql -u username
-p databasename
```

TIP To see what else you can do with the mysql client, type

```
/path/to/mysql/bin/mysql --help
```

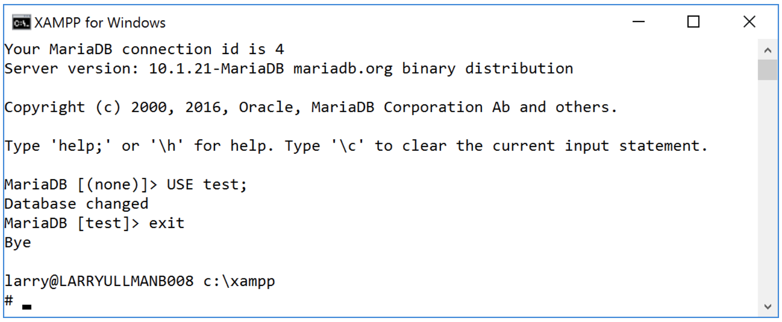
TIP The mysql client on most systems allows you to use the up and down arrows to scroll through previously entered commands. If you make a mistake in typing a query, you can scroll up to find it, and then correct the error.



PHP and MySQL for Dynamic Web Sites

```
~ » mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.14 Homebrew

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> 
```

F After getting into the mysql client, run a **USE** command to choose the database with which you want to work.



XAMPP for Windows

```
Your MariaDB connection id is 4
Server version: 10.1.21-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE test;
Database changed
MariaDB [test]> exit
Bye

larry@LARRYULLMANB008 c:\xampp
# _
```

G Type either **exit** or **quit** to terminate your MySQL session and leave the mysql client.

TIP In the mysql client, you can also terminate SQL commands using `\G` instead of the semicolon. For queries that return results, using `\G` displays those results as a vertical list, as opposed to a horizontal table, which is sometimes easier to peruse.

TIP If you are in a long statement and make a mistake, cancel the current operation by typing `\c` and pressing Return or Enter. If mysql thinks a closing single or double quotation mark is missing (as indicated by the `'>` and `">` prompts), you'll need to enter the appropriate quotation mark first.

Using phpMyAdmin

phpMyAdmin (**`www.phpmyadmin.net`**) is one of the best and most popular applications written in PHP. Its sole purpose is to provide an interface to a MySQL server. It is somewhat easier and more natural to use than the mysql client but requires a PHP installation and must be accessed through a web browser. If you're running MySQL on your own computer, you might find that using the mysql client makes more sense, because installing and configuring phpMyAdmin constitutes unnecessary extra work (although all-in-one PHP and MySQL installers may do this for you). If you are using a hosted server, your web host is virtually guaranteed to provide phpMyAdmin as the primary way to work with MySQL and the mysql client may not be an option.

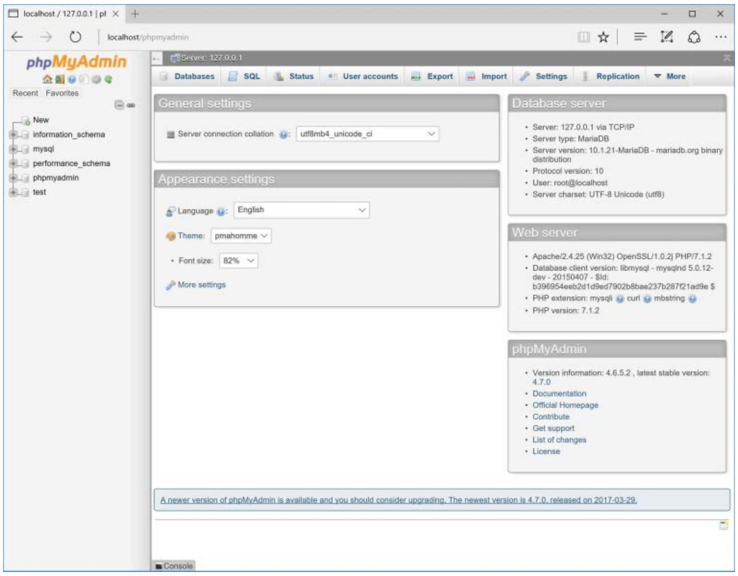
Using phpMyAdmin isn't hard, but the next steps run through the basics so that you'll know what to do in the following chapters.

To use phpMyAdmin:

1. Access phpMyAdmin through your web browser **H**.

   The URL you use will depend on your situation. If running web sites on your own computer, this might be **`http://localhost/phpMyAdmin/`**. If running on a hosted site, your web host will provide you with the proper URL. Likely, phpMyAdmin would be available through the site's control panel (should one exist).

continues on next page



Server connection collation: utf8mb4_unicode_ci

Language: English
Theme: pmahomme
Font size: 82%

- Server: 127.0.0.1 via TCP/IP
- Server type: MariaDB
- Server version: 10.1.21-MariaDB - mariadb.org binary distribution
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

- Apache/2.4.25 (Win32) OpenSSL/1.0.2j PHP/7.1.2
- Database client version: libmysql - mysqlnd 5.0.12-dev - 20150407 - $Id: b396954eeb2d1d9ed7902b8bae237b287f21ad9e $
- PHP extension: mysqli curl mbstring
- PHP version: 7.1.2

- Version information: 4.6.5.2 , latest stable version: 4.7.0
- Documentation
- Official Homepage
- Contribute
- Get support
- List of changes
- License

A newer version of phpMyAdmin is available and you should consider upgrading. The newest version is 4.7.0, released on 2017-03-29.

H The first phpMyAdmin page (when connected as a MySQL user who can access multiple databases).

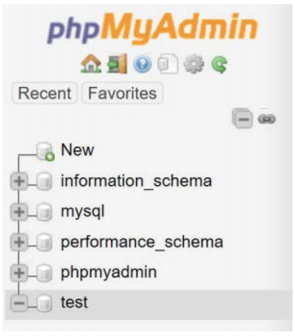
Note that phpMyAdmin will only work if it's been properly configured to connect to MySQL with a valid username/password/hostname combination.

2. If possible and necessary, use the list on the left to select a database to use I.

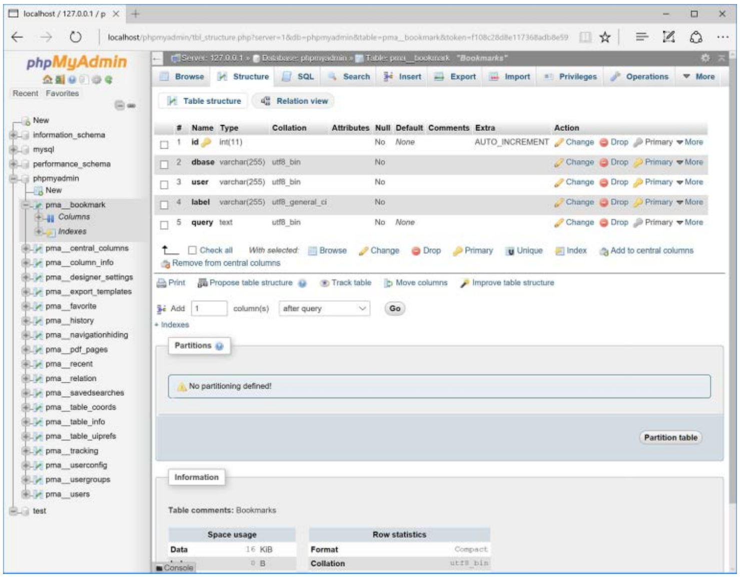
   What options you have here will vary depending on what MySQL user phpMyAdmin is connecting as. That user might have access to one database, several databases, or every database. On a hosted site where you have just one database, that database will probably already be selected for you. On your own computer, with phpMyAdmin connecting as the MySQL root user, you would see a pull-down menu or a simple list of available databases J.

3. Click on a table name in the left column to select that table J.

   You don't always have to select a table—in fact, you never will if you just use the SQL commands in this book, but doing so can often simplify some tasks.



I Use the list of databases on the left side of the window to choose with which database you want to work. This is the equivalent of running a **USE *databasename*** query within the mysql client.



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop Primary More
☐	2	dbase	varchar(255)	utf8_bin		No				Change Drop Primary More
☐	3	user	varchar(255)	utf8_bin		No				Change Drop Primary More
☐	4	label	varchar(255)	utf8_general_ci		No				Change Drop Primary More
☐	5	query	text	utf8_bin		No	None			Change Drop Primary More

No partitioning defined!

Table comments: Bookmarks

Space usage		Row statistics	
Data	16 KiB	Format	Compact
	0 B	Collation	utf8_bin

J Selecting a table from the left column changes the options on the right side of the page.

4. Use the tabs and links (on the right side of the page) to perform common tasks.

   For the most part, the tabs and links are shortcuts to common SQL commands. For example, you can use options on the Browse tab to perform a **SELECT** query and options on the Insert tab to add new records.

5. Use the SQL tab **K** to enter SQL commands.

   The next three chapters, and a couple more later in the book, will provide SQL commands that must be run to create, populate, and manipulate tables. These might look like

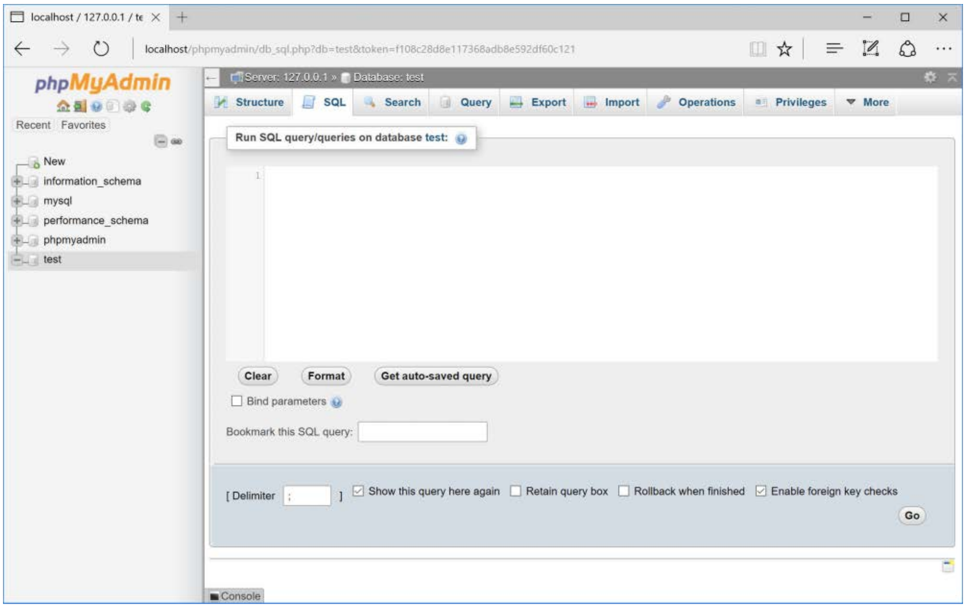
```
INSERT INTO tablename (col1, col2)
VALUES (x, y)
```

   These commands can be run using the mysql client, phpMyAdmin, or any other interface. To run them within phpMyAdmin, just enter them into the SQL tab and click Go.

TIP **There's a lot more that can be done with phpMyAdmin, but full coverage would require a chapter in its own right (and a long chapter at that). The information presented here will be enough for you to follow any of the examples in the book, should you not want to use the mysql client.**

TIP **phpMyAdmin can be configured to use a special database that will record your query history, allow you to bookmark queries, and more. See the phpMyAdmin documentation for details.**

TIP **One of the best reasons to use phpMyAdmin is to transfer a database from one computer to another. Use options on the Export tab in phpMyAdmin connected to the source computer to create a file of data. Then, on the destination computer, use the Import tab in phpMyAdmin (connected to that MySQL server) to complete the transfer.**



K The SQL tab, in the main part of the window, can be used to run any SQL command.

Review and Pursue

If you have any problems with the review questions or the pursue prompts, turn to the book's supporting forum (`LarryUllman.com/forums/`).

Review

- What version of MySQL are you using? If you don't know, find out now!
- What characters can be used in database, table, and column names?
- Should you treat database, table, and column names as case-sensitive or case-insensitive?
- What are the three general column types?
- What are the differences between **CHAR** and **VARCHAR**?
- How do you determine what size (in terms of subtype or length) a column should be?
- What are some of the other properties that can be assigned to columns?
- What is a primary key?
- If you're using the command-line mysql client to connect to MySQL, what username and password combination is required?

Pursue

- Find the online MySQL manual for your version of MySQL. Bookmark it!
- Start thinking about what databases you may need for your projects.
- If you haven't yet changed the MySQL root user password (assuming you've installed MySQL on your own computer), use the instructions in Appendix A to do so now.

Index