CHAPTER 1

Create and Test a Database and Table

This chapter introduces the concept of a database and a practical way of testing it. Using the examples, you will create a MariaDB or MySQL database and a table. As you work through the examples, you will become familiar with the database administration interface.

After completing this chapter, the student will be able to

- Define and design a database and table
- Install and use a WAMP package
- Use phpMyAdmin to create a database and table
- Secure phpMyAdmin and databases with a user ID and password
- Delete a database and/or table

Databases can be used to store products, details of customers, records of members of a society or a club, and much more. They can store names, passwords, addresses, e-mail addresses, registration dates, blog entries, and telephone numbers. Databases can be regarded as folders containing tables of data. The table of data has *columns* and *rows*; the rows in database tables are called *records*. Table 1-1 shows a typical database table.

Table 1-1. *A Typical Database*

user_id	first_name	last_name	email	password	phone
1	Kevin	Kettle	kev@kettle.com	K3ttl3fur	305 111 1111
2	Susan	Saucepan	sue@kitchen.org.uk	N@sus5	01111 222 1111
3	Oliver	Oven	oliver@cokker.co.uk	H0tst0v3	03333 111 4444

A. W. West and S. Prettyman, *Practical PHP 7, MySQL 8, and MariaDB Website Databases*,
https://doi.org/10.1007/978-1-4842-3843-1_1

Defining Developer, Administrator, and User

In this book, the term *developer* (aka *webmaster*) means the person who designs and produces the database; they will integrate the database into a website. Sometimes the term *webmaster* or *web designer* may be used. When it is used, it usually means the same thing as *developer*. The words *administrator* and *membership secretary* have the same meaning in some of the book's tutorials, which are based on building a database for a club. The word *administrator* means the person responsible for monitoring and maintaining the content of the database tables. Clearly, one person can be both a developer and an administrator. However, most developers will maintain the structure of a database but will not want the hassle of amending and deleting records; that should be the role of an administrator (such as a club or society's membership secretary).

The *user* is any member of the general public viewing and possibly interacting with a website database. For security reasons, users have extremely limited access to the database; however, they will be allowed to register for membership, log in to a special section, or change their password.

■ **Caution** The organization commissioning a database must conform to the rules and laws of the country in which the database is developed and resides. In the United Kingdom, the Data Protection Act for the territory in which the database was developed must be followed. This is especially important if that data is going to be used for profit. This may require obtaining a license. In addition, the developer and administrator must normally sign a document confirming that they will never disclose the details of persons recorded in the database. The UK Information Commissioner's Office (ICO) requires an annual license fee based on the revenues of the organization that owns the database. There is no equivalent law in the United States, but privacy laws can differ between states. It is essential that you understand and obey the data-protection laws for your client's territory.

Databases that are accessible to multiple countries will need to meet the requirements for each country. Sometimes this requires the creation of different versions of the database and/or website, when otherwise it would not be required. Many laws and regulations lag behind the overall need to provide the most secure database environment possible. The developer should always err on the side of the "most secure" when designing and using databases. Databases used for educational purposes only (such as the databases shown in this book) do not have to meet the requirements and regulations (such as licensing).

Defining Interactive Websites

Interactive websites are often called *dynamic* websites; however, this book uses the word *interactive* because *dynamic* can signify so many things. For instance, it can mean moving, powerful, eye-catching, flashy, exciting. To a beginner, none of those meanings defines a web page that interacts with a user.

Dynamic is often used to mean exciting, but there is little excitement to be seen in an interactive registration form. *Dynamic* is also a musical term meaning changes or variations in loudness or speed. If *dynamic* can refer to change, why were dynamic templates designed to provide consistency from one web page to another? The term *interactive* has one clear meaning and will be used from now on in this book.

MariaDB and MySQL (with PHP) allow users and administrators to interact with a database using website pages. For instance, users can register as members of an organization via a registration page on a website. Users will be able to supply their personal data for the membership tables. The database management system then enters the users' input into the administrator's tables automatically; this lightens the workload of the administrator. The website's registration page can be programmed to filter users' data input and verify it. From an interactive page, users may even be allowed to update their own records in a database.

Interactivity means that the administrator's workload is greatly reduced, but not completely. For instance, if the database is for a bookshop, the administrator will still have to enter any new titles and prices. On the other hand, an interactive database can be programmed to alert the administrator when the stock of a certain book needs replenishing.

In Chapter 2, you will learn to develop a simple interactive website.

Using MariaDB or MySQL Only for Interactive Database Tables

A noninteractive data table means that only the administrator can enter or amend the table's information. A noninteractive data table would be more easily created and administered using a spreadsheet, such as Microsoft Excel. However, website users cannot interact with such a data table. Employing a MariaDB or MySQL database management system (DBMS) to create a noninteractive (static) version of the data table would be like using a sledgehammer to crack a nut. Website users would have no input and cannot search or update data.

Using MariaDB or MySQL DBMS for a noninteractive version would not reduce the workload of an administrator; they would have to enter all the members' data and verify that the data is genuine.

■ **Note** A few interactive web pages do not need a database to function. For instance, a Contact Us form can be regarded as interactive because it takes a user's input and transmits it to the website's owner via a PHP form handler via an e-mail; this can be achieved easily without a database. In this book, the term *interactive* always means the user can interact with a database.

Methods for Developing and Maintaining Databases

The four methods for managing databases are as follows (with the easiest method first and hardest last):

- phpMyAdmin (or other administrative tool)
- PHP
- SQL scripts
- Command line

In this book, we will be mainly using the first two methods (phpMyAdmin and PHP). We will use SQL scripts and the command line to briefly demonstrate how to create, update, and distribute your database. For interactive databases, you will need some PHP files. You do not need an extensive knowledge of PHP before you can create interactive databases. We will introduce the PHP syntax you require in the appropriate place in each project—that is, in context. The step-by-step, fully worked examples will show you what MariaDB and MySQL can do and how to do it.

Because of their popularity, graphical user interfaces (GUIs) have been developed to facilitate the task of developing and maintaining databases. These administration tools are part of development packages that include web servers (Apache), databases (MariaDB or MySQL), and programming languages (PHP). In this book, we will introduce two of the most popular packages: XAMPP and EasyPHP.

A Brief Look Inside Web Server Communication

Databases need a server, a DBMS, and a PHP processor, as shown in Figure 1-1. These can be downloaded as an all-in-one, already configured package. The testing and development of the projects in this book are based on the free XAMPP and EasyPHP packages.

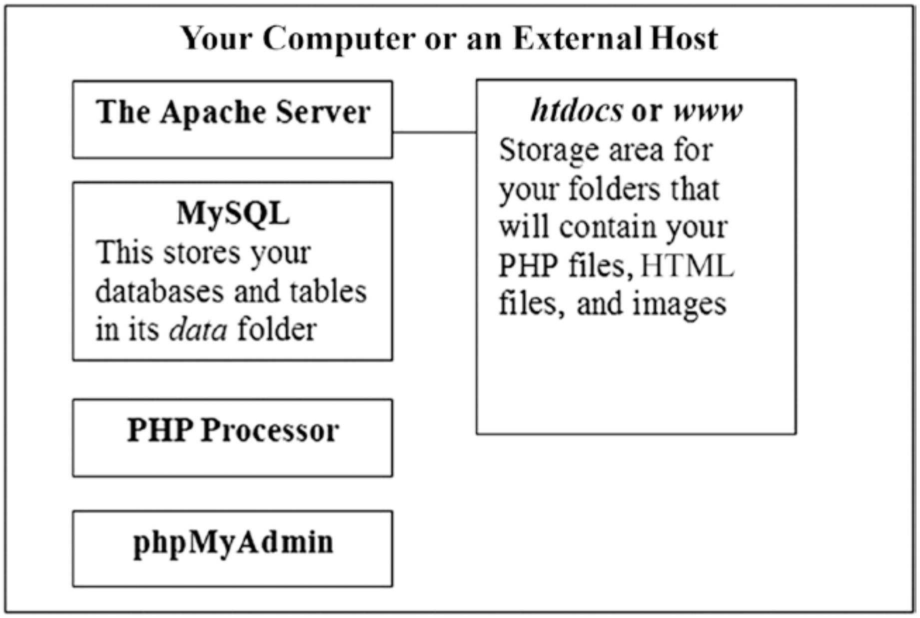

Figure 1-1. *Web server communication*

Figure 1-1 shows the main components built into the XAMPP and EasyPHP development platforms. They are as follows:

- Apache is the web page server used by the great majority of hosts and for web development on local hosts (user computers). Web servers determine whether web pages contain PHP code by looking at the file ending (*.php*). If PHP code exists, the code is passed to the PHP interpreter for execution. If the PHP interpreter determines database-related code or SQL exists within the PHP program, this information is passed to the MySQL or MariaDB database management system. The database management system executes the SQL statements and returns the results to the PHP interpreter. The PHP interpreter completes executing the PHP code and returns the results to the Apache server. The Apache server gathers the results of the PHP code, along with any HTML, CSS, and/or JavaScript, and sends this information to the web browser on the user's machine. The web browser then executes the HTML, CSS, and/or JavaScript code.
- MySQL and MariaDB are not just databases; they are also database management systems. These packages include tools to create, manage, and secure databases.
- The PHP processor interprets and executes any PHP code. It will throw errors or exceptions if there are syntax (coding errors), logical errors, or system errors (out of memory).
- phpMyAdmin is a GUI-based administration tool for creating and maintaining databases and their tables.

A single all-in-one package such as XAMPP or EasyPHP contains the four programs mentioned in Figure 1-1 and is referred to as a WAMP (Windows, Apache, MySQL/MariaDB, and PHP). In WAMPs, the main components are preconfigured so that they can talk to each other. The equivalent on an Apple OS is MAMP, and on a Linux computer it is LAMP.

The folder *htdocs* (or *eds-www*) is the default storage and executable area for your web pages. Apache, by default, looks in *htdocs* (or *eds-www*) for your web pages. These pages may be designed to allow users to interact with the database. Other pages will operate unseen by the user to transmit information back and forth between the browser and database. The pages are usually HTML and PHP files or a combination of both.

■ **Caution** Everything inside Figure 1-1 may be already installed on a remote host, but you should never use a remote host to create a database while you are learning. For security reasons, do not use a remote host until you have become proficient. We recommend you learn and develop a database using an all-in-one package on your own computer. Note that an all-in-one package installed on your own computer is purely a development tool. The database, when developed and thoroughly tested, can eventually be uploaded to a host to make it available to users.

A Free Development Platform for Testing

You will not be able to test your work in the normal way—that is, by using a browser to view a database and the PHP code located on your hard drive. When a web page containing PHP is requested by the browser, the PHP code is executed in the web server. The results of the executing of the code will display in the browser. If you view the browser code (right-click the web page and select View Source), you will not see the PHP code. You will see the results produced by the execution of the code. This confuses beginning developers who are used to seeing any HTML, CSS, or standard JavaScript code they have created in the browser. Remember, HTML, CSS, and standard JavaScript are executed in the browser. PHP code is executed in the web server, even when that server is on your own PC.

Using an all-in-one package on your computer will allow you to see all the code you create and the results of the execution of this code on your PC. This book assumes that you will use a package on your own computer while you are learning and for developing future database-driven websites.

■ **Note** In the current world of hackers who attempt to corrupt or gather personal, corporate, and government data, all web pages on Internet-hosted websites *must* be secured. The earliest projects in this book are necessarily simple and have some secure features. However, it is not recommended that you upload these pages to an Internet host. When you have gained experience and confidence, and you are sure that you understand how to secure websites and databases, you can adapt the book's later projects for use in your own websites and then upload them to a remote host.

Using XAMPP on Your Own Computer

The XAMPP package is free and is preconfigured so that the components will talk to each other. This eliminates the hassle of the usual practice of downloading several individual components and then configuring them to work together. XAMPP includes packages for Windows (WAMP), Linux (LAMP), and the Apple (Mac) OS (MAMP). The examples in this book are developed using a WAMP environment. However, these examples should also work in a LAMP or MAMP environment.

At the time of writing, the most recent version of XAMPP is version 7.2.4. This version is used throughout the book. It has component versions as follows: Apache 2.4.33, MariaDB 10.1.25, PHP 7.2.4, and phpMyAdmin 4.8.0, along with other tools. The examples in this book are not totally compatible with PHP versions before 7.0.

■ **Note** If you are installing a version that is newer than demonstrated in this book, the installation steps may be slightly different. You should refer to the XAMPP site (`www.apachefriends.org`) or search for installation instructions on the Internet (try `youtube.com`) for any version not demonstrated in this book. If the newer version includes a major upgrade (for example, PHP 8), some code changes may be required. Check the Apress website for code changes related to major version releases.

Before we give you the instructions for downloading XAMPP, we need to settle a question that bothers every beginner concerning the transferring of a developed database from XAMPP or EasyPHP to the remote host. If you use one of these packages on your own computer, a question will arise, as stated in the title of the next section:

Will I Be Able to Transfer the Database from XAMPP or EasyPHP to a Remote Host?

The main thought that haunts a beginner is this: “If I develop a database on a local WAMP, will I be able to move it easily to a remote host?” Beginners have every reason to be worried because most manuals rarely give even a hint on this topic. However, the answer is this: “Yes, you will be able to move the database.” You will find full instructions later in this book.

Now we will provide the information for downloading and installing XAMPP.

■ **Caution** Should you want to install multiple free WAMPs, it is possible to install both EASYPHP and XAMPP on the same computer. However, make sure one of them is shut down before opening the other; otherwise, they will fight for the same ports and cause annoying problems.

Downloading and Installing XAMPP

XAMPP needs minimal configuring. To download the package, go to `www.apachefriends.org/`.

Click Download from the menu at the top, as shown in Figure 1-2, of the main page. The download page will then display the current versions available. If it’s available, select the 7.2.4 version of XAMPP to ensure that all examples you use from this book will be compatible. If there is a minor release of version 7 (such as 7.3.0), it probably will still be compatible. If version 7 is not available, download the latest version. Then check the Apress website to see whether there are any coding changes required to the examples presented in this book. The download page varies from time to time, so you may have to explore the page to find the version shown.

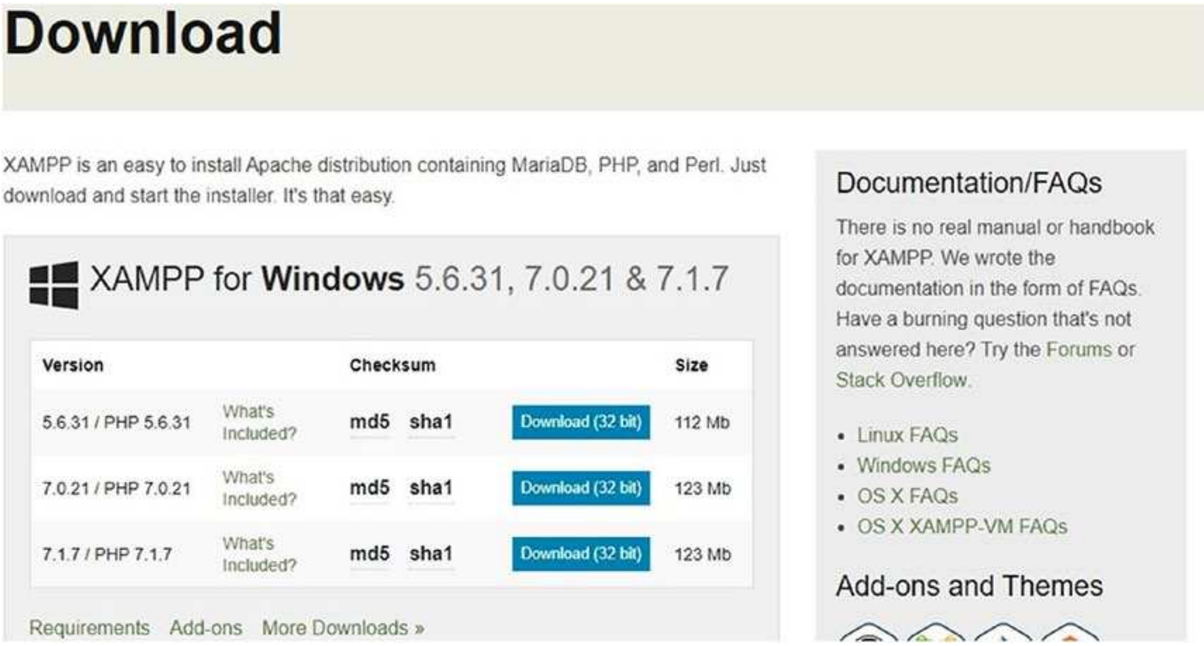

Figure 1-2. *Installing XAMPP*

■ **Note** The download page will state that you must have the C++ runtime libraries installed. XAMPP and its related tools are created using C++. If you are using a current version of Windows, you probably already have the correct version of C++. However, if you receive an error indicating that it is missing, follow the link the error provides to download the correct version.

Click the Download button next to the most current version. This will begin the download process. The file will automatically download to the download folder on your PC. Depending on which browser you are using, you could be asked if you want to run or save the program. Some browsers will automatically save the program and require you to click the file to run it. If you are unaware of how to find downloaded files for the browser you are using, search the Internet for directions. Once the file has downloaded, start the setup process by double-clicking the file (or clicking the Run option if prompted).

The environment will prompt you to ask permission to install the program. Click Yes to continue the process. The setup program will determine the security settings on your machine. It may provide a warning that your security might restrict some of the XAMPP options available. You can choose to either continue the installation or stop the current installation and temporarily turn off your security program. If you turn off your security, make sure your PC is no longer connected to the Internet. The setup program will also look at your User Account Control (UAC) settings. These settings limit access to folders and files based on the user ID in which you are signed into the system. If you are using a computer in which you do not have administrative rights, the setup routine will provide a warning message prompting you to install your program in a different location than the default (program files). Do as the prompt suggests and change the default settings when prompted to a different location, such as *c:/XAMPP*.

A Welcome to XAMPP Setup Wizard screen will appear. Just click Next to continue the setup. The next screen will display all the options available for installation. If you are a beginner, just accept the items already checked by clicking the Next button. Intermediate and advanced users might consider removing items that you are sure you will not use. You can always install them later if you need them. The next screen will show the installation location. Use *c:/XAMPP* to avoid any possible security access issues. Click Next.

The next screen will try to convince you to also look at installing CMS programs (such as WordPress). For now, uncheck the Learn More box. You can install it later if you are interested. Click the Next button. The next screen will indicate the setup is ready. Click Next. The installation will begin. By default, the Start Control Panel check box will be checked. Leave it checked. Click the Finish button. You may be prompted with a language selection prompt before the control panel displays.

If you have errors during installation, copy the error message and paste it into a search engine (like Google). Look at the listing of suggested solutions. Go to a trusted website and follow the directions to correct your error.

The items on the XAMPP control panel labeled Running usually appear automatically, and you will then be able to stop the various modules. If they do not start automatically, click the Start buttons on the XAMPP control panel for Apache and MySQL. If a button says Stop, that module is already running. What next? If you are asked about running the modules as services, choose to run Apache and MySQL as services, and then those modules will automatically start when you double-click the XAMPP desktop icon (Figure 1-3).

Figure 1-3. *The XAMPP icon*

Create a shortcut on your Desktop for XAMPP's *htdocs* folder, and place it alongside the XAMPP icon, as shown in Figure 1-4. Use this shortcut for loading your PHP files into the *C:\xampp\htdocs* folder.

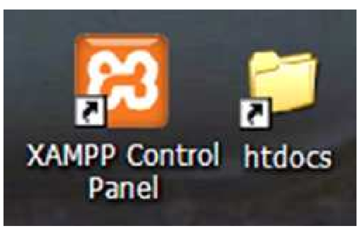

Figure 1-4. *Time-saving shortcuts*

If a desktop icon was not created during the installation, we recommend you go to the *C:\xampp* folder and then create a desktop shortcut for the *xampp-control.exe* file.

For maximum convenience, put the two desktop items side by side, as shown in Figure 1-4. One icon starts and stops XAMPP, and the other allows you to create and modify pages directly in the XAMPP *htdocs* folder.

One common problem is that Skype and other applications also use port 80, the default port for Apache. If another application is using this port, Apache won't start. The log (Figure 1-5) will indicate if Apache could not start because the port was in use. If this occurs, go to the XAMPP control panel and click the Netstat button. This window will list all applications running on your PC and what ports they are using. Determine an unused port (such as 8080) and close the Netstat window. Then click the Config button next to Apache. Select *httpd.conf*. The configuration file will open in Notepad (or your default text editor). Click Edit, click Search, and enter **80**. Click Find Next until you discover the following line:

```
Listen 80
```

Change this line to the following:

```
Listen 8080
```

Or, change it to whatever port you decided to use.

Now go back to search and click Find Next until you find the following line:

```
ServerName: localhost: 80
```

Change this line to the following:

```
ServerName: localhost: 8080
```

Or change it to the same port you decided to use.

Save the file, and close Notepad (or your text editor). Go to the XAMPP control panel and click the Config button at the top-right corner of the page. Click the Services and Port Settings button. Click the tab Apache. Enter the new port for Apache (8080 or whatever port you decided to use). Click Save. Close the Config window. Now click the Start button next to Apache in the control panel (Figure 1-5). After a few seconds, Apache should start. This will be indicated by a green background behind Apache. You will also need to start MySQL because it could not start without Apache running first.

Starting XAMPP

From here onward, to test your pages in XAMPP, double-click the desktop icon and check that Apache and MySQL/MariaDB have started. If they have not started, click the Start button for Apache. Once it starts, then click the Start button for MySQL/MariaDB and then minimize the control panel.

Figure 1-5 shows the XAMPP control panel.

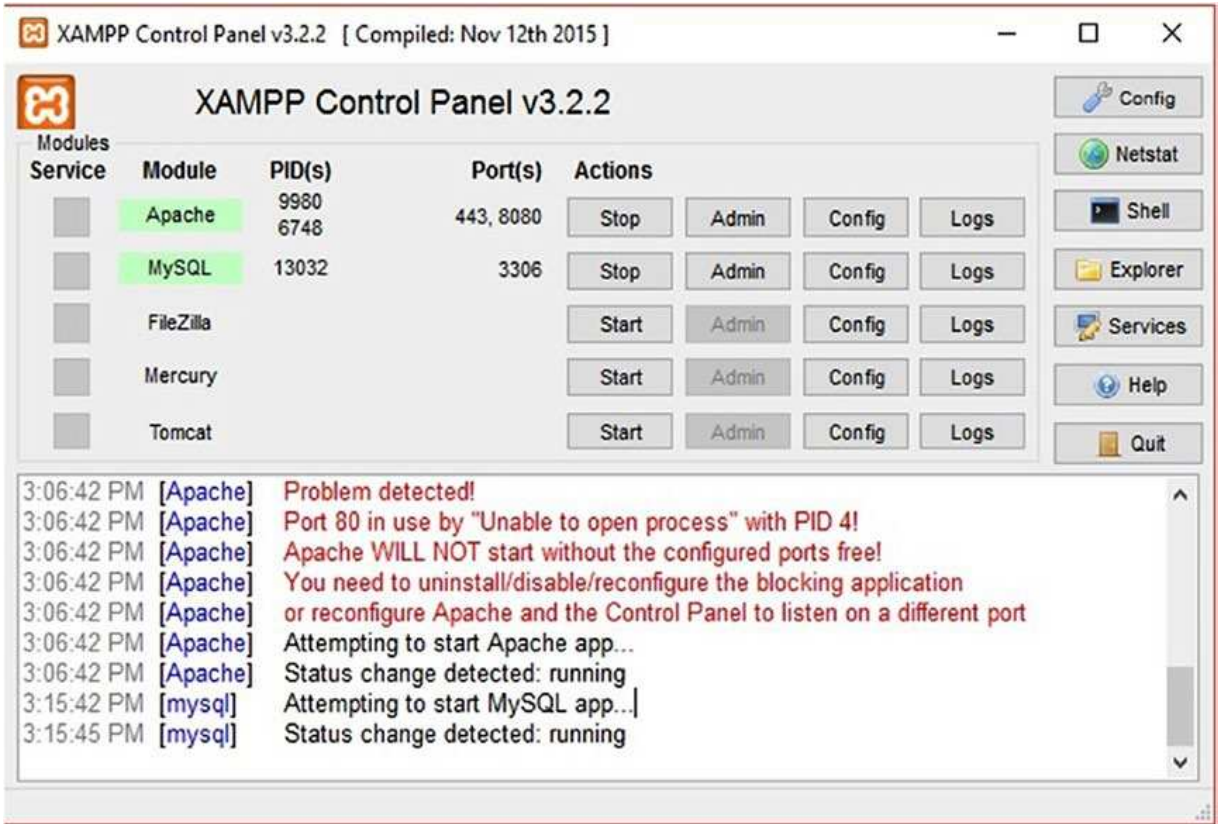

Figure 1-5. *The XAMPP control panel*

We suggest you always minimize the control panel so that you have a clear desktop for starting work on your databases.

After starting Apache and MySQL/MariaDB, you can test your installation and examine all the XAMPP examples and tools; to do this, enter one of the following addresses in your browser:

```
http://localhost/
http://127.0.0.1/
```

If you added a port number, be sure to include the same number in a format like one of the following examples:

```
http://127.0.0.1:8080/
http://localhost:8080/
```

Closing XAMPP

Close XAMPP when you have finished testing your database and PHP files. This will free up memory for tasks other than database development. To close, click the maximize XAMPP control panel on the taskbar and then click the Quit button on the control panel, as shown in Figure 1-6. Alternatively, you can right-click the icon in the Notification area and then click Quit.

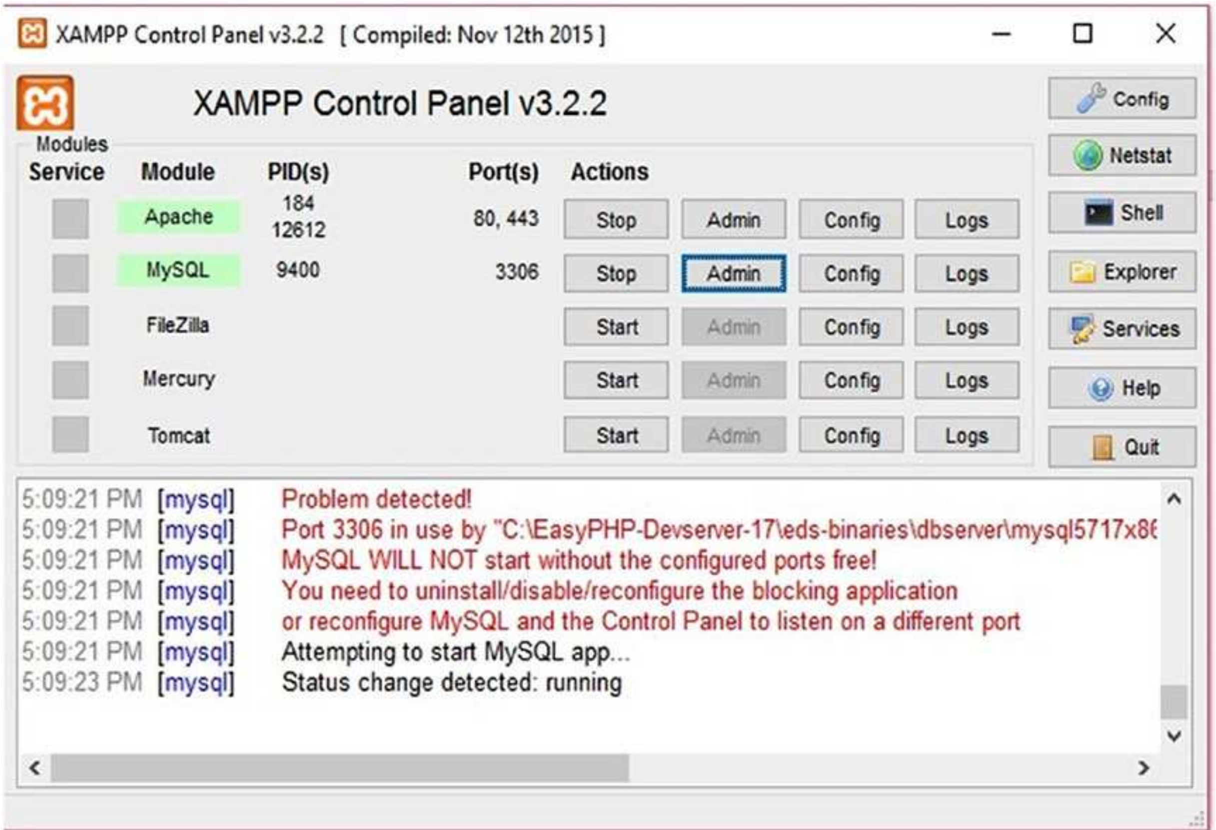

Figure 1-6. *Closing the XAMPP program*

The security of a database and its data is extremely important. XAMPP provides a method for making the database and tables on your computer safe from harmful interference, which is described later in this chapter.

Where Is MariaDB and MySQL 8?

You might be confused at this point about XAMPP and the location of the MariaDB databases. The creators of XAMPP did switch from using MySQL databases to using MariaDB databases; however, everything we just explained seems to indicate that MySQL is still part of XAMPP. However, this is not the case. When the creators decided to switch database systems, they had to make only minor configuration changes to use MariaDB. Since MariaDB does not operate any differently than the free version of MySQL, the creators decided to leave everything with the MySQL title. Since you might be new to this environment, this might be confusing. However, to current users of XAMPP, this allowed an easy transition between database environments. Just remember, wherever you see MySQL in XAMPP, it is MariaDB.

The title of this book includes *MySQL 8*. MySQL 8 is a full, professional version with the latest bells and whistles, including business analytics tools. However, MySQL 8 is not provided with XAMPP or EasyPHP, and the full version is not free. If you are developing sites for corporations or government agencies, you should consider using MySQL 8 (or the latest version). Any version of MySQL can be attached to XAMPP. You do need to read the MySQL documentation to make sure the version you are attaching is compatible with the Apache and PHP versions you have installed. Adding a new version of MySQL is an intermediate skill. We have provided directions in the last chapter to help when you are ready to convert.

Using EasyPHP on Your Own Computer

If you would prefer to use EasyPHP or you are a beginner and your fellow designers have convinced you that you must use an early version of MySQL (i.e., MariaDB will just not do), you can install EasyPHP, which is a WAMP package that includes the free version of MySQL. It also includes the MongoDB database system, which is a non-SQL database. MongoDB is not covered in this book.

There is no reason you need to install both packages. However, if you choose to do so, you may need to either run one at a time or adjust ports so both can run at the same time. Once installed, you will see that both packages work in similar ways. All examples have been tested in both environments.

Download and Install EasyPHP

To download the newest version of EasyPHP, go to the following site:

```
www.easyphp.org
```

Download a nonbeta version of easyPHP Development Server by clicking the Download button in the middle of the main page. The version that has been used to test and run the examples in this book is 17.0, which includes PHP 7.2, Apache 2.4, and MySQL 5.7 (the latest free edition). If you notice a new major release (such as 18.0), check the Apress website for this book to see whether there are any requirements for code changes in the examples shown.

The setup program should start up automatically. The Windows environment will require you to allow the installation to occur by clicking the Yes button. Once you click the button, the setup routine will ask for your language preference. Next it will allow you to change the location of your installed files. If you do not have administrative rights to your PC (such as a company-owned PC), you may not be able to install EasyPHP under *Program Files*. We suggest changing the location of your installed files to the following:

C:\EasyPHP-DevServer-xx (*xx* is the version you are installing such as 17)

Click Next. The installation process will then ask if you want a desktop icon created. We suggest you create one for easy access. Click Next. The current screen will verify the location of your install. Click Install. When the installation is completed, the last screen will ask if you want to launch the development server. Launch it by clicking Finish. If you have errors during install, copy the error into a search engine (like Google) to look for solutions. In the list that is provided, go to a well-known site and attempt to follow the directions to fix your error.

Starting EasyPHP

Go to the EasyPHP icon in the system tray, right-click, and launch the dashboard. If the dashboard fails to launch, close the development server (right-click the icon and select Exit), and try launching it from your desktop shortcut. The dashboard uses port 1111. If you still can't launch the dashboard, right-click the development server icon and select Tools and then Open Port Controller. Verify that another process is not using the same port. If another process is using that port, stop that process and repeat the previous steps.

Within the dashboard (Figure 1-7), click the start button below HTTP SERVER. Clicking this button should start both Apache and MySQL. The start button will change to a stop button if the server starts properly. At this point, both the HTTP SERVER and DATABASE SERVER should be displayed with stop buttons below the titles. If you encounter an error, try stopping the development server (right-click the system tray icon and select Quit) and restart it (click the icon on the desktop). If you still have errors, copy and paste the error in a search engine (such as Google) to see a list of possible solutions. Select a suggested solution from a known site and try to follow the directions.

Figure 1-7. *The EasyPHP DevServer*

Closing EasyPHP

Anytime you need to close EasyPHP, just close the development server by going to the system tray, right-clicking the icon, and selecting Quit. This will close all processes related to EasyPHP.

phpMyAdmin Security

The initial installation of phpMyAdmin within XAMPP and EasyPHP has the username *root* and no password. If you use those settings on your own computer, there is a security risk when connected to the Internet. If you are creating databases for more than just personal use or your own education, you should create a phpMyAdmin password. If you work in the same room with other people, the password will provide security if the password is not divulged to the other people. As a best practice, you should password-protect your development environment.

If you are not already in phpMyAdmin, go to the XAMPP control panel or the easyPHP DevServer page. In XAMPP click admin to the right of MySQL. In EasyPHP, click the open button to the right of the version of phpMyAdmin displayed on the page (Figure 1-8).

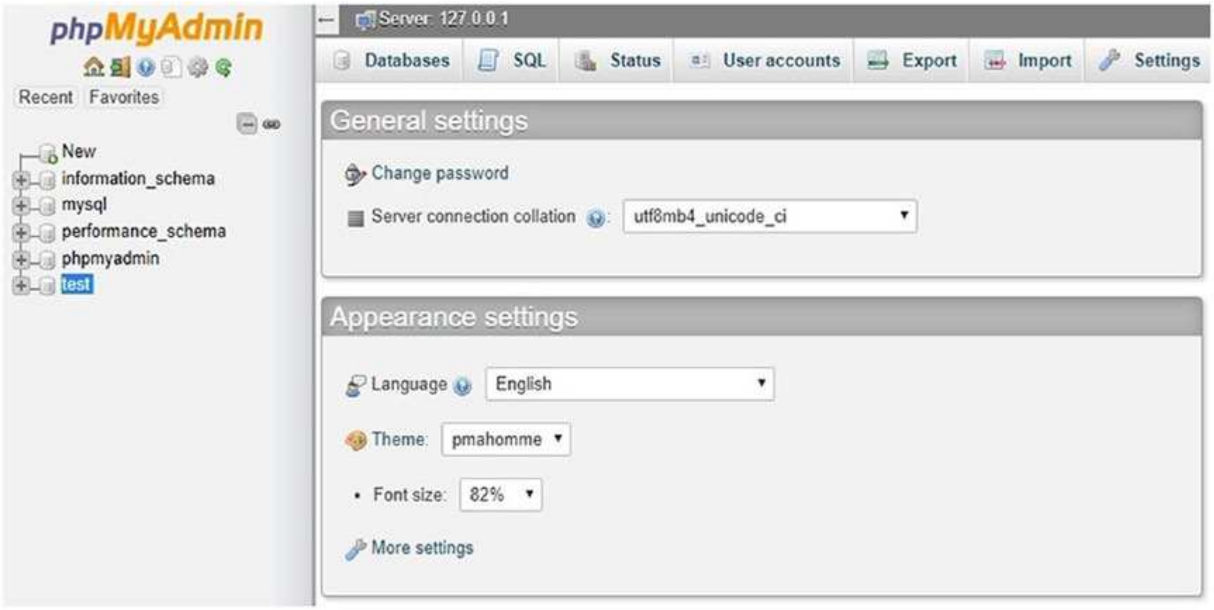

Figure 1-8. *The phpMyAdmin page*

Note that if you have changed the port from the default port, you may need to add the port number when bring up phpMyAdmin. For example, if you changed the port to 8080, the address for phpMyAdmin is now as follows:

```
http://localhost:8080/phpMyAdmin/
```

Click the User accounts tab in phpMyAdmin (Figure 1-9).

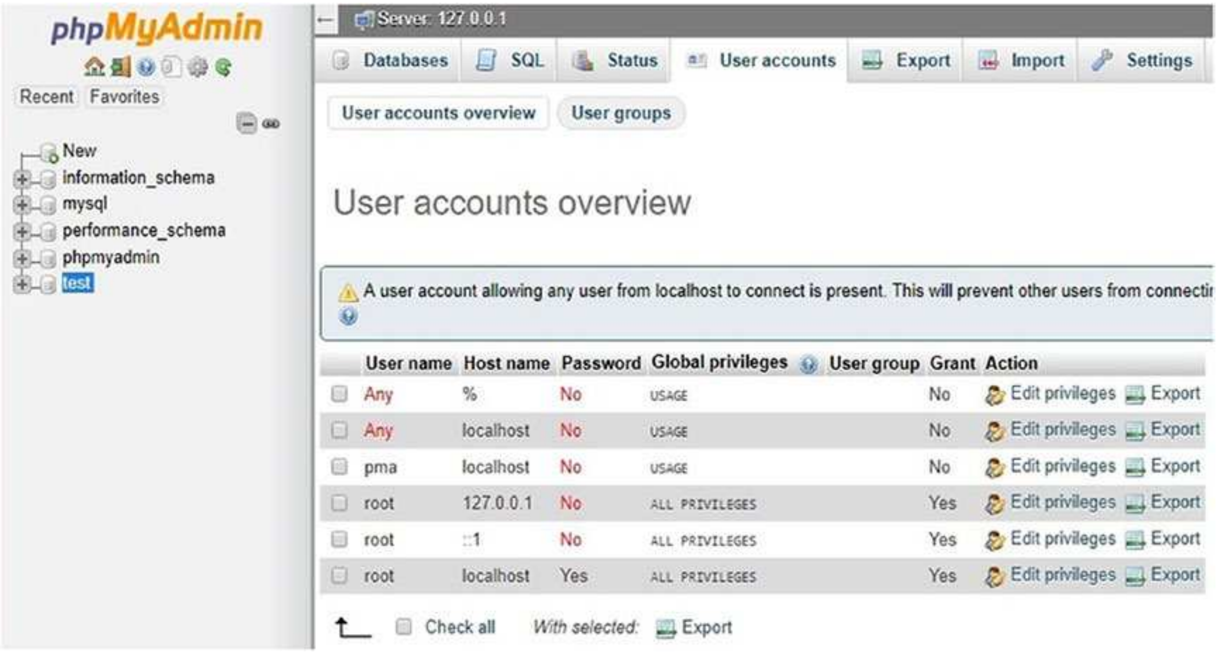

Figure 1-9. *The phpMyAdmin User accounts tab*

Click the Edit privileges link to the right of the last entry, root localhost. An Edit privileges page will display (Figure 1-10). Click the Change password link at the top of the page.

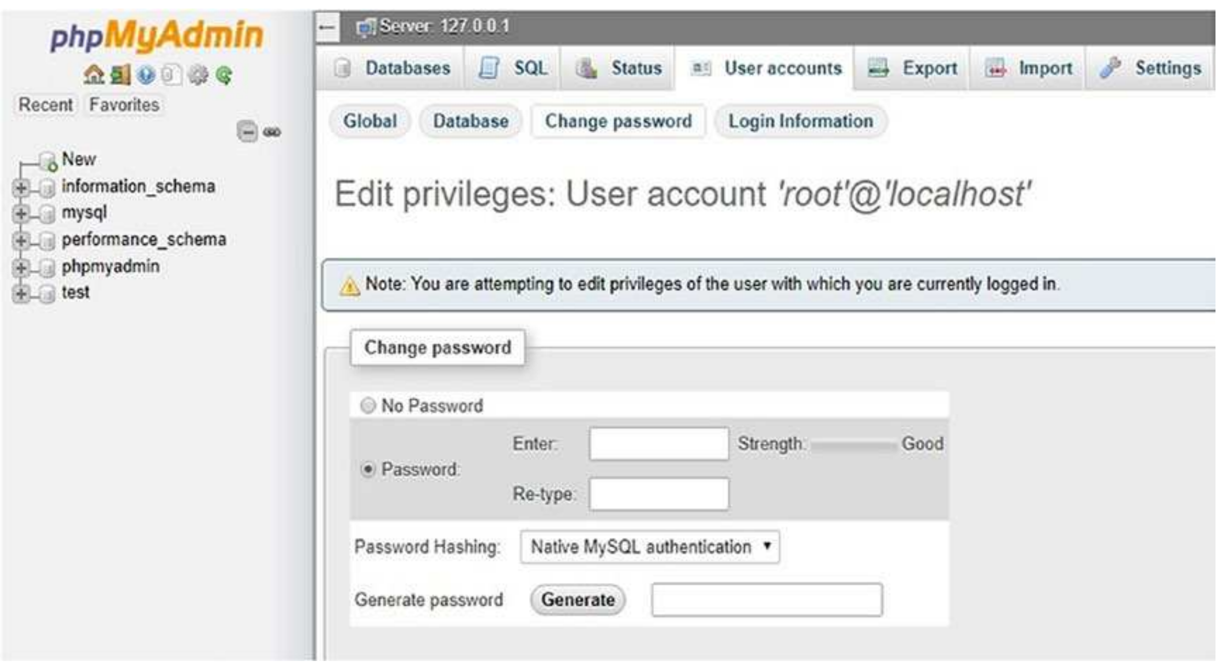

Figure 1-10. *phpMyAdmin Edit privileges page*

Enter your password in both boxes. Click the OK button near the bottom of the page. If the password changed, you will get a notification that it has changed. You now need to tell phpMyAdmin to use the new password.

In XAMPP, go to your XAMPP folder on your PC. Open the folder phpMyAdmin. Now open the file *config.inc.php* in Microsoft Notepad (or another text editor).

In easyPHP, go to the *EasyPHP-DevServer-xx* folder, go into *eds-modules*, and open the *phyMyAdmin* folder. Now open the file *config.inc.php* in Microsoft Notepad (or another text editor).

Search for the following:

```
['auth_type'] = 'config';
```

Change this line to the following:

```
['auth_type'] = 'cookie';
```

Save the file.

If you are using XAMPP, go back to the control panel. Stop and restart MySQL. Go back to the phpMyAdmin page (`http://localhost/phpMyAdmin/` or `http://localhost:8080/phpMyAdmin/`). The page should now require that you enter a user ID (root) and password (the one you just created).

If you are using EasyPHP, try going back into the DevServer page and restart the servers. Then click the open button for phpMyAdmin. Enter the user ID (root) and password (the one you just created). If after entering the user ID and password an error message indicating that mysql_native_password is missing, follow the next directions.

Go to the *EasyPHP-DevServer-xx* folder, go into *eds-binaries*, then into *dbserver*, into the *MySQL* folder, and finally into the *bin* folder. This folder contains several applications that can be used to run and administer MySQL and phpMyAdmin. However, these applications only run in the command-line interface. To make finding and using them easier, we are going to place this location in the Path variable for your PC. This allows you to just execute the program you need without entering the complete path name. Go to the URL line in your browser and double-click the address. The complete address should now be highlighted. Right-click the highlighted address and select Copy.

Go to the search icon (bottom right of your system tray), click it, and enter **system variables**. Click the Edit System Environmental Variables result. The System Properties window will appear. Click the Environmental Variables button. In the box provided (either the top box for all users of your machine or the bottom box for the user ID you are signed in with), select the PATH code. Then click the Edit button. Now click the New button. A space will now be provided in the box to enter the path that you copied. Click in the box, right-click, and select Paste (or hit Ctrl+V). The path you copied should now be in the box. Make sure it is a complete path to the bin location. When it is correct, click the OK button. Your bin is now added to the PATH variable. Click OK for each of the windows to close them and submit the change. Now, you will need to shut down all your processes and reboot your machine. This will cause Windows to use the new PATH.

Once your machine is rebooted, restart the EasyPHP DevServer and make sure that the HTTP server and mySQL server are both started. Go to the Window's search icon and enter **cmd**. Then hit Enter. This will open the command-line window. You are now finally ready to add the missing module to phpMyAdmin.

Enter **mysql -u root** and click the Enter key. You should now see a mysql prompt (mysql>). Enter **use mysql**; (don't forget the semicolon) and click the Enter key. This tells the server to use the MySQL database. Enter the following line to add the missing plugin:

```
update user set plugin='mysql_native_password' where user='root';
```

You should see a message “Query OK.” Now also change the password in the command line with the following line:

```
update user set Password=PASSWORD('12345') where user='root';
```

12345 should be replaced with the password you set previously. You should also see a message “Query OK.” All your changes should be complete. Close the mysql prompt by entering **exit**. Hit the X to close the command window. Go back into phpMyAdmin and click User accounts. You should now see that the Yes flag has been set indicating that root now has a password. Note EasyPHP still might automatically let you in to phpMyAdmin even with the password set.

This password situation occurs with the current (as of the publish date of this book) version of the free MySQL edition. As you noticed earlier, this does not occur in XAMPP because XAMPP is using MariaDB.

If you have any problems with these steps and want to back out, the easiest way to do so is to uninstall the package using the uninstall program under the main package folder and then reinstall XAMPP or EasyPHP. As said earlier, if you get error messages, copy them into a web search engine (Google) to look for suggested solutions.

Accessing phpMyAdmin Directly

If you have installed XAMPP and the servers have been started, you can access phpMyAdmin directly by entering the following URL in any browser:

```
http://localhost/phpMyAdmin/ or http://127.0.0.1/phpMyAdmin/
```

(Remember to include the final forward slash and the port number if you changed the default port of 80.)

If you have installed easyPHP and the servers have been started, you can access phpMyAdmin directly by entering one of the following URLs in any browser:

```
http://localhost:8080/eds-modules/phpmyadmin470x170718155219/
http://127.0.0.1:8080/eds-modules/phpmyadmin470x170718155219/
```

(Your actual phpmyadmin version/build might be different, which will create a different *phyMyAdmin* folder.)

Be sure to include the `http://`; otherwise, the browser will look for the location on the Internet instead of your PC.

■ **Note** The sections describing the use of phpMyAdmin apply to any of the development platforms: XAMPP, WAMPServer, or easyPHP.

You might have set the password in XAMPP or EasyPHP earlier, so whenever you access phpMyAdmin, you might need to log in using that password. This prevents Internet robots and human beings from interfering with your database. The latter case is important if you work in an office with others—you could have a spy or mischievous meddler in the place where you work. If you created a password for phpMyAdmin, a dialog box will appear, as shown in Figure 1-11.

Figure 1-11. *Enter the password in the dialog box to access phpMyAdmin*

Enter your username (root) and password and then click the Go button. phpMyAdmin loads rather slowly, but it will eventually appear.

Note that open source programs are continually being improved and upgraded, and you may find that you have a newer version of phpMyAdmin in your XAMPP or EasyPHP package than that used in this book. You may also see upgrade messages alerting you to a new version in the phpMyAdmin main window. Where personal data is concerned, security is paramount, so these incremental updates are a good thing for you, though they do mean that some of the screenshots in the book might no longer accurately reflect what you see on the screen. Don't worry if an interface looks a little different from the ones shown in this book; the usage will normally be similar.

The phpMyAdmin interface may look a little daunting at first, but we'll cover the relevant parts of it when we need to use them. For the moment, you can close the phpMyAdmin window.

You now know how to install and secure XAMPP and easyPHP, and you also learned how to start and stop the servers. Most of what you have just read will probably be very new, but there are some parts that you will recognize because they follow the normal Microsoft Windows organization of files and folders.

The Familiar Bits

Within the XAMPP package, the structure of the folders and files will be familiar to Windows users, although their names may not be recognizable.

Figure 1-12 shows the XAMPP folders.

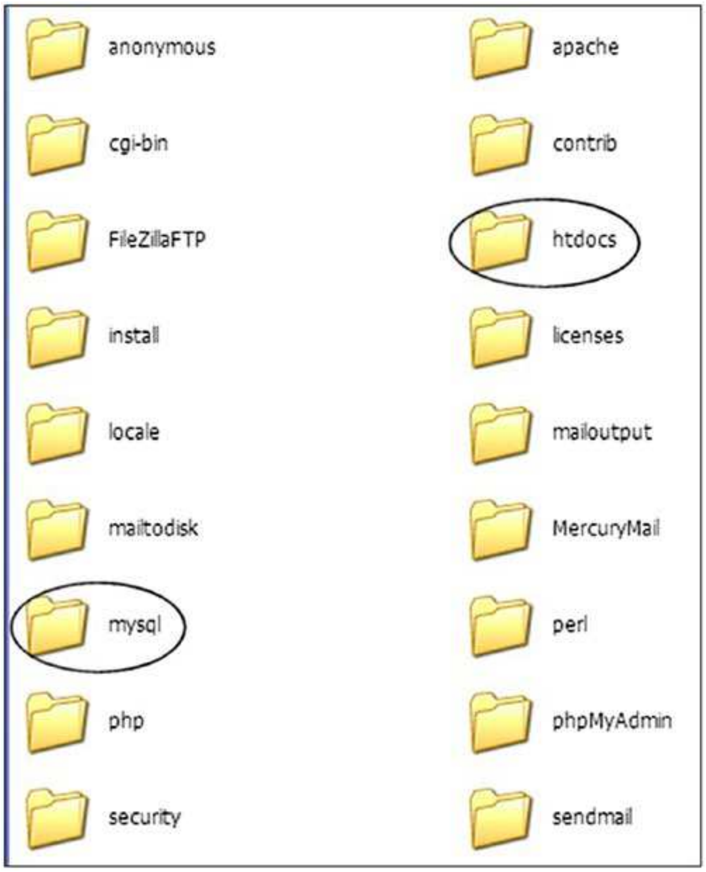

Figure 1-12. *The folders in the XAMPP package; your list might be slightly different*

In Figure 1-12, note the *htdocs* folder. This is where you will place all your PHP files and the HTML pages for your website and databases.

Within the XAMPP folder, you will find a folder called *MySQL*. This folder contains a folder called *data* where the databases and tables will reside. A database must have a unique name. A file within the data folder contains all the information about the database, and it has the file type **.opt*.

Tables are files; when you have created any tables, these will also live inside the folder named *data*, and they will have the file type **.frm*.

In easyPHP once you open the *EasyPHP--Devserver-xx* folder, you will discover an *eds-www* folder. This is where you place all your PHP files and HTML pages. The following file path is the location of the database files:

C:\EasyPHP-Devserver-17\eds-binaries\dbserver\mysql5717x86x170717151041\data

Intermediate or advanced developers should consider creating virtual directories, which are other locations for executable files. While this topic is not covered in this book, you can find additional information on the Internet by searching for *XAMPP Virtual Directory* or *easyPHP Virtual Directory*.

Now that you're familiar with the look and feel of the tools you'll be using, you're ready to move ahead. The next section will take you nearer to creating your first database and table.

Planning a Database: The Essential First Step

The first and most important stage is to plan the database so that you have something practical to use. Let's assume we need to plan a database for the membership of an organization.

First, decide on a name for the database. We will give this database the name *simpledb*. Remember that the database is like an empty folder that will eventually contain one or more tables.

Then, assemble the data items into a table. We have given this table the name *users*.

Next, decide what information you want in the table; your decision is not binding because you can change any part of the database during development. Let's suppose we need five pieces of information about the users. We set out some typical data in Table 1-1 earlier in the chapter and in Table 1-2.

Table 1-2. *Draft Plan for the users Table in the simpledb Database*

user_id	first_name	last_name	email	password	registration_date
	Kevin	*Kettle*	kev@kettle.co.uk	*K3ttl2fur*	
	Susan	*Saucepan*	sue@kitchen.org.uk	*n@Sus*5	
	Oliver	*Oven*	oliver@cooker.co.uk	*H*0*tsts*0*v*3	

Each row in a table is called a *record*, and each cell is called a *field*. A database can contain more than one table. We have used some fictitious names to help plan the table. The first column is labeled user_id, and this column is additional to the five columns of data. The column user_id will be explained later; just accept it for the moment and be sure to leave it empty. Also, leave the registration dates empty because this is an automatic entry; it does not need values entered, nor does it need allocated space.

Now we must allocate some space for the data. Table 1-3 shows the number of characters we have allocated for each item.

Table 1-3. *Number of Characters to Allow for Column in the users Table*

user_id	first_name	last_name	email	password	registration_date
6	30	40	50	60	

Write down or print the two tables, and keep them close at hand because you will be referring to them in the next stages.

Now, decide on a username and password for the database, and enter that information in your notebook. Four pieces of information are required: the name of the database, the host, the password, and the username. In this project, these are as follows:

- *Name*: simpledb
- *Host*: localhost
- *Password*: Hmsv1ct0ry1 (this comes from hmsvictory but with an uppercase first letter and by replacing *I* with 1 and *o* with 0 for a strong password)
- *User*: horatio

To generate a more secure password, phpMyAdmin includes a password generator.

Next, we will create our first database using phpMyAdmin.

Creating a Database Using phpMyAdmin

Open phpMyAdmin either directly as mentioned at the beginning of this section or through the XAMPP administration page or the easyPHP dashboard. Remember, the database server must be running to use phpMyAdmin.

Click the Databases tab in the top menu. You will then see the interface shown in Figure 1-13.

Figure 1-13. *The phpMyAdmin interface for creating the database*

Type a name for the database. For this example, it will be simpledb, all in lowercase. You should change Collation to utf8-general-ci; the default of latin1_swedish_ci causes confusion.

■ **Note** Collations determine the type of characters, sorting abilities, performance, and other characteristics of the data. The collation utf8-general-ci is an older standard that meets the needs of the data examples in this book. Your data requirements may require a different collation.

Then click the Create button (set the Collation field to utf8-general-ci). After you click the Create button, the page will switch to the create a table page. However, first we want to change the properties of the database. Check the left arrow at the top of the page (left side). This will return to the main page. Click the Databases tab again. Now the previous page will look similar to Figure 1-14.

Figure 1-14. *In the lower part of the page, select the box near the name of your database*

When you select the box next to your new database, as shown in Figure 1-14, click Check Privileges, and you will be taken to a screen where you will see a list of users who have access to the database. To make the database secure, you must add a username and password. Click Add user account, as shown in Figure 1-15.

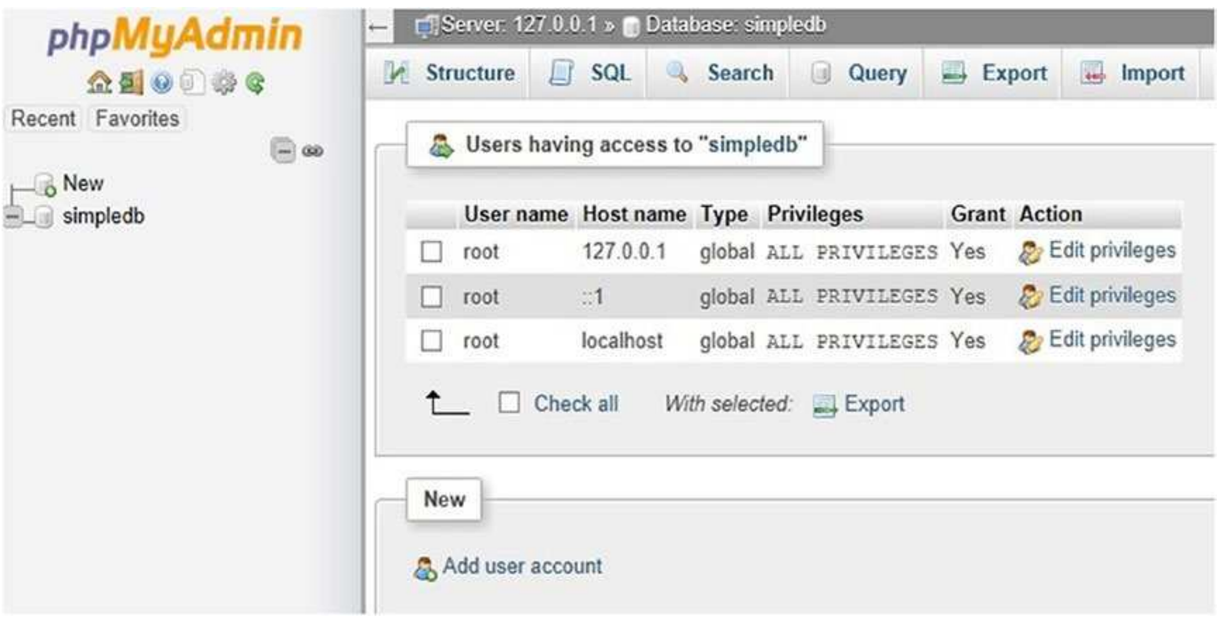

Figure 1-15. *The Add user account icon is at the bottom of this figure*

Clicking Add user account will load the Add user account screen, shown next in Figure 1-16.

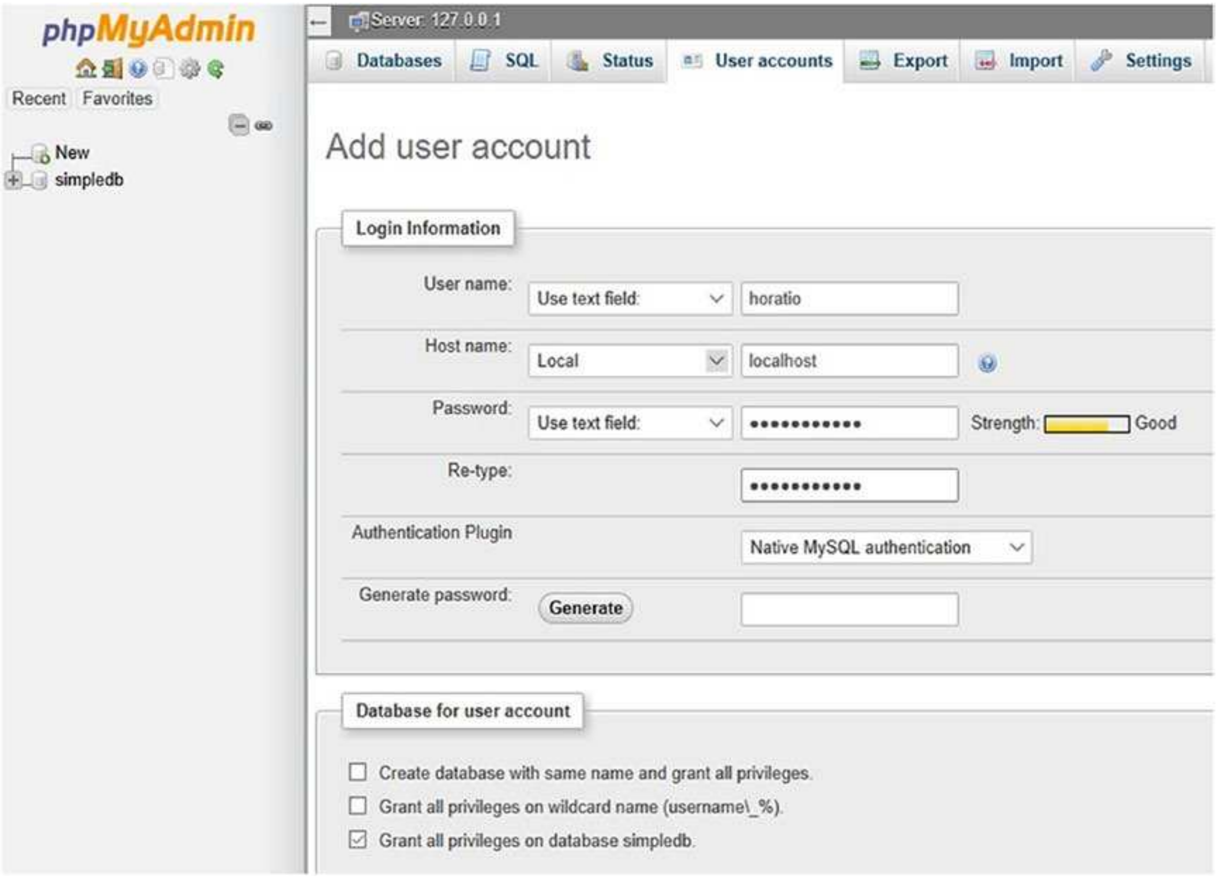

Figure 1-16. *This screen enables you to add a user and a password*

■ **Caution** Adding a username and password is essential; otherwise, your database will be insecure and vulnerable to attack by unscrupulous individuals or their robots. This is the most important habit to cultivate. Be sure to record the user and password details in your notebook. Make sure to create a strong password that includes a combination of lowercase and uppercase, numbers, and special characters. Keeping a detailed record will save you hours of frustration later.

Using the drop-down menus, accept the default Use text field in the first field and enter the username in the field to right of it. In the second field labeled Host, select local. The word *localhost* will appear in the field on the right. Localhost is the default name for the server on your computer. Enter a password in the third field, and confirm your password by retyping it in the lower field. The Generate Password button will create a random strong password if you want something unique and more secure.

Scroll down to Database for user account, and click Grant all privileges on database. Because you are the webmaster, you need to be able to deal with every aspect of the database; therefore, you need all the privileges. If you add other users, you need to restrict their privileges by deselecting boxes such as Drop, Delete, and Shutdown.

Scroll down to the bottom of the form, and click the OK or Go button. You have now created the database and secured it against attack. The database can be regarded as an empty folder that will eventually contain one or more tables.

■ **Note** If you get lost when using phpMyAdmin and can't see what you should do next, always click the little house at the top of the left panel. Hover over the icon to ensure that it is the Home button.

Now we will create our first table.

Create a Table Using phpMyAdmin

Now let's work with the phpMyAdmin web page to insert one or more data tables into a database. This page will give you complete control over your table(s), including troubleshooting and backing up.

Click the name of your new database; you will find it on the left panel. You will then see the screen shown in Figure 1-17.

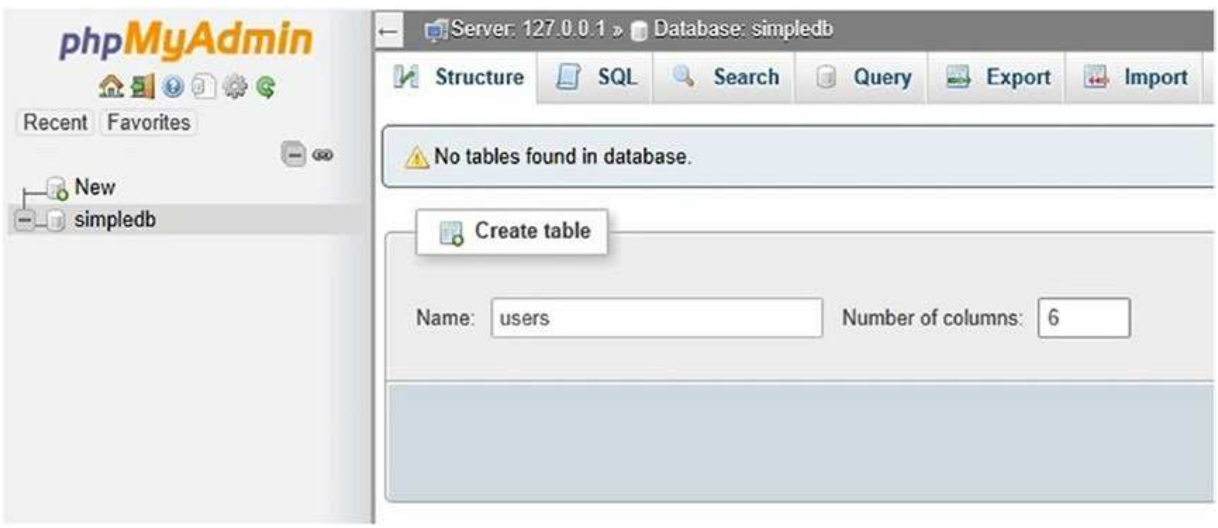

Figure 1-17. *Click the Go button at the bottom of the page to create the table*

Enter a name for the table, and specify the number of columns. Then click the Go button at the bottom of the page. You will be taken to a screen showing the columns flipped 90 degrees so that columns look like rows; this is shown in Figure 1-18. The fields are empty and waiting for you to define the table.

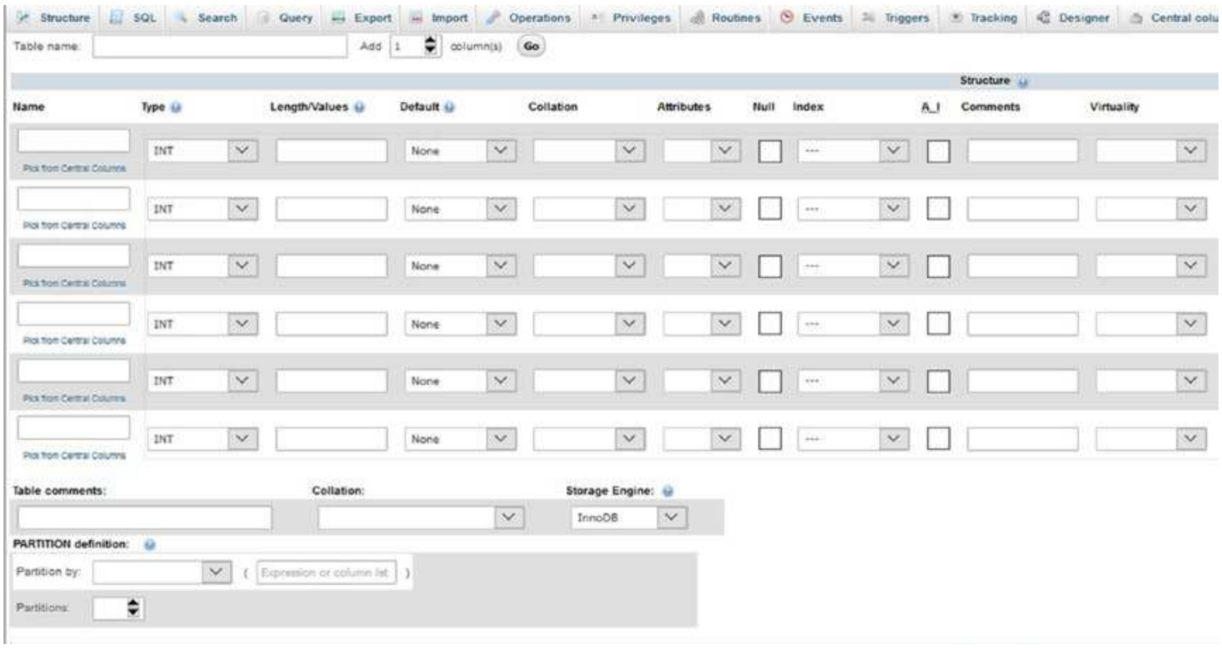

Figure 1-18. *The six rows represent six columns. The column titles will be entered in the fields on the left.*

Use the data from Tables 1-2 and 1-3 that we planned earlier, and enter the column name, data type, and number of characters. The details for creating the users table are given in Table 1-4.

Table 1-4. *The Attributes for the users Table*

Column name	Type	Length/Value	Default	Attributes	Null	Index	A_I
user_id	*MEDIUMINT*	6	None	*Unsigned*	☐	PRIMARY	☑
first_name	VARCHAR	30	None		☐		☐
last_name	VARCHAR	40	None		☐		☐
email	VARCHAR	50	None		☐		☐
password	CHAR	60	None		☐		☐
registration_date	DATETIME				☐		☐

Accept all the default settings for each item except for the user_id. Here, you will need to select UNSIGNED, PRIMARY, and the type; also select the A_I (auto increment) box. If you find entering the values of 6, UNSIGNED, and PRIMARY cause your version of phpMyAdmin to produce errors (or warnings), you can leave the default for those items. As mentioned in the later text, when you click A_I, the system will request you select PRIMARY. MEDIUMINT will default to size 6.

The various categories under the heading Type will be explained later; the heading Length/Value refers to the maximum number of characters. The Length/Value for the registration_date is left blank because the length is predetermined. Do not enter anything under the headings Default and NULL. The attribute UNSIGNED means that the user_id integer cannot be a negative quantity. The Index for the user_id is the primary index, and A_I refers to Automatic Incrementation of the id number; as each user is registered to the database, they are given a unique number. The number is increased by one as each new user is added. When selecting the A_I check box, a pop-up window may ask you to verify that it is the primary index and will request the size again. *Do not* put a size in the pop-up window. If you do, you will get an error. Figure 1-19 shows the screen for specifying the attributes.

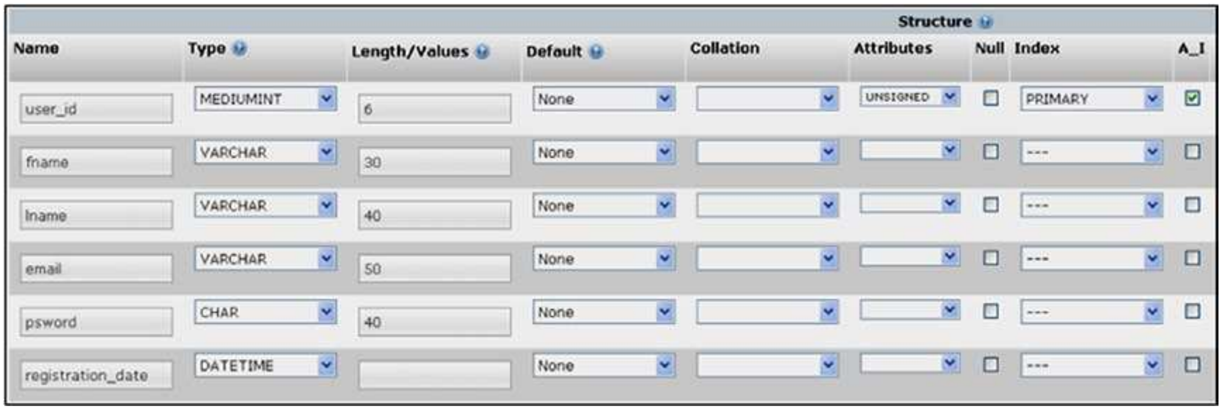

Figure 1-19. *This screen allows you to specify column titles and the type of content*

The rows represent columns, and they are very wide; you may have to scroll horizontally to enter some of the information. You will find more options as you scroll right, but we will not need them for this tutorial.

How do you fill out the fields? Enter the six column titles in the fields on the left under the heading Name. Select the type of column in the second column of fields under the heading Type. Select them from the drop-down menus. The types used in this table are as follows:

- MEDIUMINT can store integers ranging from minus 8,388,608 to plus 8,388,607. You could choose the next smallest category SMALLINT if the number of users will never exceed 65,535.
- VARCHAR specifies a variable-length string of characters from 0 to 255 long.
- CHAR is a string of characters traditionally used for passwords. Be sure to size this with 60 characters so that your PHP code can hash the passwords. The current PHP hashing algorithm will convert a password into a hashed string of 60 characters. A user's password can be, say, 6 to 12 characters long, but it will still be stored in the database as a hashed 60-character string. This will be discussed further in Chapter 2.

■ **Note** The length of hashed passwords may increase in newer editions of PHP. Make sure to research the current hashing size before setting the length of this field in the table.

- DATETIME stores the date and time in the format YYYY-MM-DD-HH:MM:SS.

Enter the number of characters in the third column of fields under the heading Length/Values. Refer to Table 1-4 for these numbers.

Under the heading Default, accept the default None. This field allows you to enter a default value if you want.

You might need to scroll to the right to complete filling in the fields. Under the heading Attributes, use the drop-down menu to select UNSIGNED for user_id. This ensures that the integer range becomes zero to 16,777,215. A negative quantity is not applicable for the user_id.

The next two entries concern only the user_id (Figure 1-20).

Figure 1-20. *Two extra entries for the user_id column*

For the user_id, under the heading Index, click the drop-down menu to select PRIMARY. The user_id should always be a primary index.

Under the heading A_I, select the topmost check box so that the user_id number is automatically incremented when each new record is added to the database. As stated earlier, you might see a pop-up window that will ask you to verify that it is primary and size. *Do not* enter a size in the pop-up window. Verify and/or add any missing information and click OK.

Enter the additional information shown in the tables and in Figure 1-20. Once you complete entering the information, scroll to the bottom and click the SAVE button.

■ **Caution** If you forget to select the A_I box for user_id, you will receive an error message when you later try to enter the second record. The message will say that you are trying to create a duplicate value 0 for user_id.

Some people prefer a blend of a GUI and the command line for programming a table; phpMyAdmin allows you to do this by using the SQL language. However, this book will mainly use the phpMyAdmin GUI. The SQL alternative is described next. You can skip this section if you want, but we recommend that you come back to it at some future date because you will undoubtedly come across SQL in other more advanced books and tutorials.

The SQL Alternative

The next section describes a slightly quicker way of using phpMyAdmin for creating a database and a table. SQL stands for Structured Query Language; it is the official language for MySQL, MariaDB, and many other databases. You will be pleased to see that it uses plain English commands. The only problem is that it is easier to create typographical errors or spelling mistakes in the SQL window than in the phpMyAdmin interface shown earlier in Figure 1-19.

Using SQL, a database can be created complete with a password and username. This saves several steps.

We assume you have created the database simpledb, so we cannot use that name again. Let's assume that an administrator (Adrian) wants to create a database called *members* using the following information:

- *Database name*: members
- *Privileges*: all
- *Username*: adrian
- *Password*: stapler12

Figure 1-21 shows the code to create the database in the SQL window.

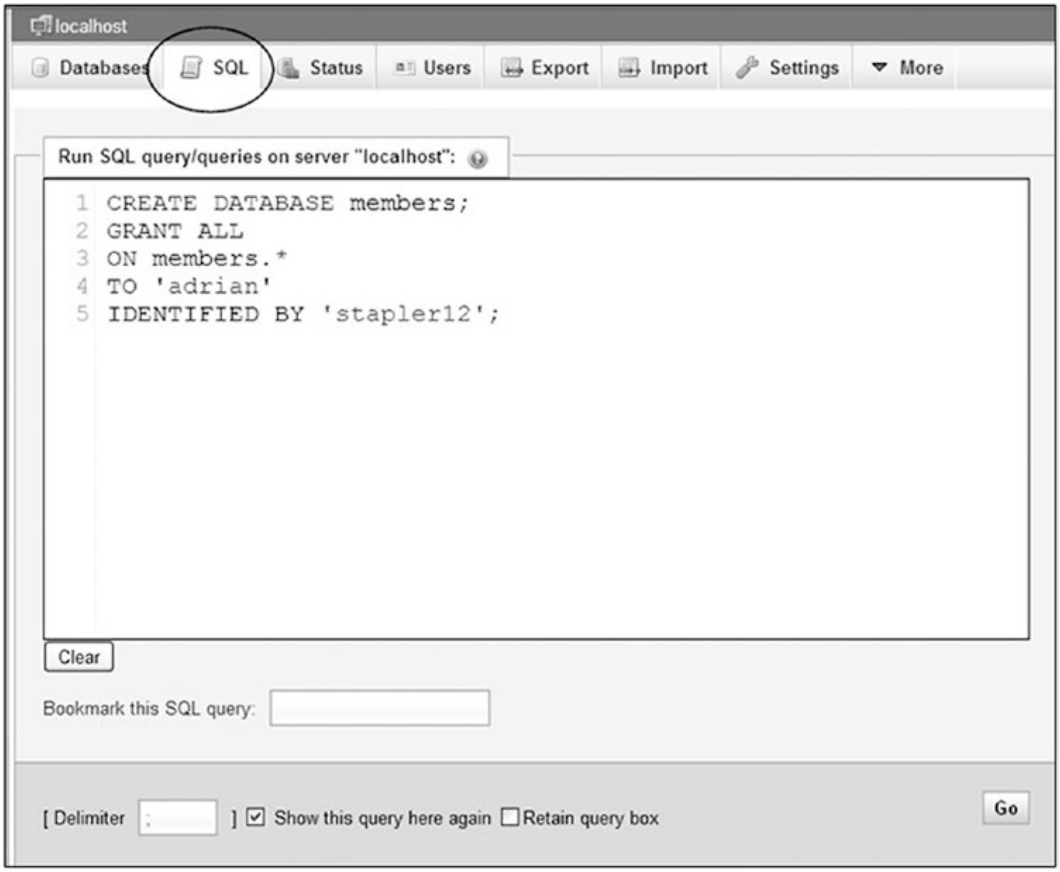

Figure 1-21. *The SQL window*

In phpMyAdmin, click New on the database "tree" on the left panel of the window to notify phpMyAdmin that you are no longer using the simpleDB database. Click the SQL tab (shown circled) to reveal the SQL window.

The details shown in Figure 1-21 must be entered in the following format:

```
CREATE DATABASE members;
GRANT ALL
ON members.*
TO 'adrian'
IDENTIFIED BY 'stapler12';
```

Enter each item is entered on a separate line by pressing Enter at the end of each line. The SQL keywords (like CREATE DATABASE) are traditionally in uppercase. Other items are normally entered in lowercase. Note the semicolons and the single quotes—these are important. When you are satisfied with the entries, click the Go button.

If you entered the code correctly, two query statements will display indicating that the database was created and adrian was attached as the user. They will indicate an empty result set. That is OK because nothing exists in the database.

Now we will create a table named *users* in the members database using the SQL window.

Click the members database in the database "tree" on the left panel of phpMyAdmin. If the members database does not appear, refresh the page so that it does appear. Open the SQL window, and enter this:

```
CREATE TABLE users (
user_id MEDIUMINT (6) UNSIGNED
AUTO_INCREMENT,
first_name VARCHAR(30) NOT NULL,
last_name VARCHAR(40) NOT NULL,
email VARCHAR(50) NOT NULL,
password CHAR(60) NOT NULL,
registration_date DATETIME,
PRIMARY KEY (user_id)
);
```

NOT NULL shown in the previous code will require that data be entered into the field. It is a required field. In this example, all the fields are actually required because registration_date is automatically populated.

Figure 1-22 shows the details in the SQL window.

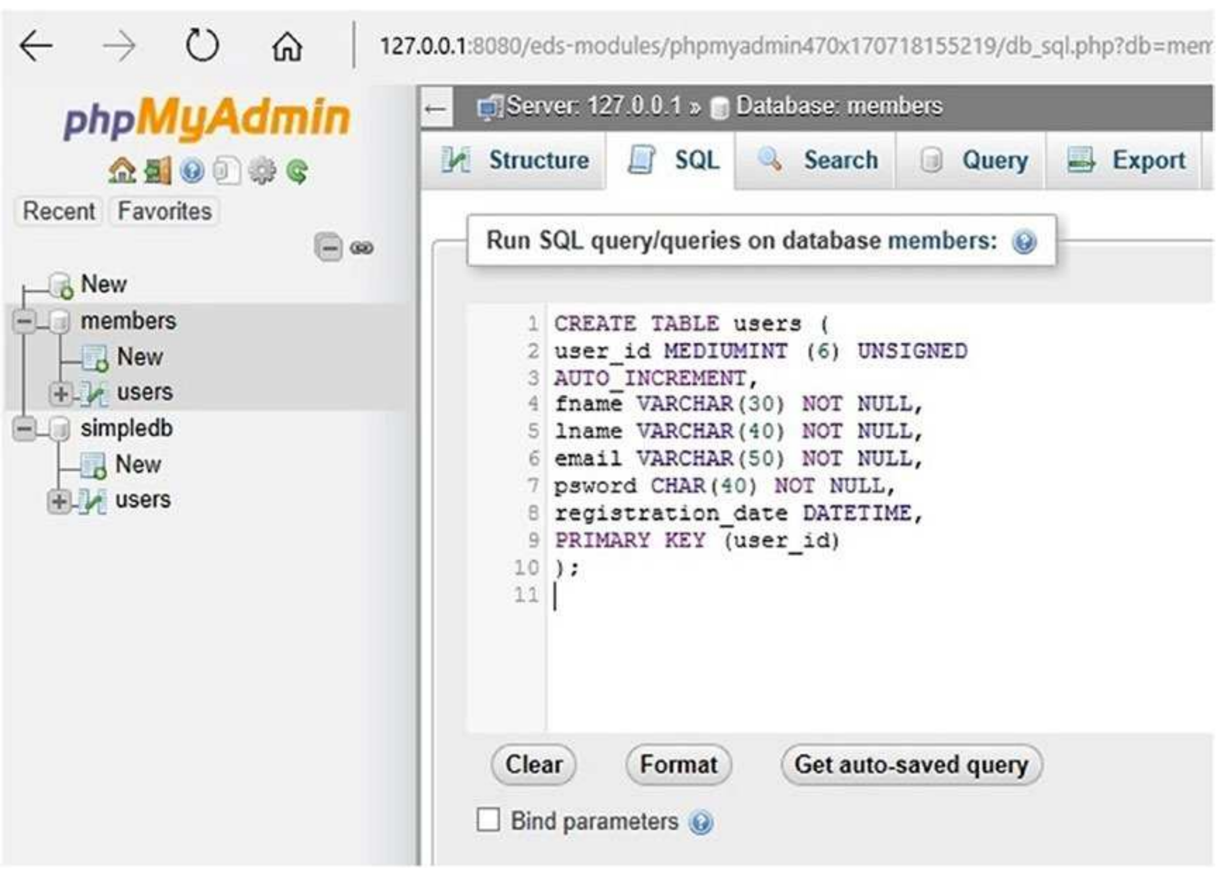

Figure 1-22. *Creating a table in the SQL window of phpMyAdmin*

Note that the brackets are all normal brackets, not curly brackets. Press the Enter key after each line, and remember to put the closing bracket and the semicolon at the end of the last line. Each item is separated by a closing comma (lines 3 through 8); if your table has six columns, you should have six commas. Click the Go button, and the table will be created.

■ **Tip** We encourage you to explore the SQL topic just described. The ability to work with SQL will be a useful alternative sometime in the future. You can find many free tutorials by searching the Internet.

Deleting Databases and Tables

When learning, beginners often need to start over after creating a database or a table and may want to delete earlier attempts. When you first use phpMyAdmin, you might get carried away and create several databases and tables. Then you might decide to clear up the mess and delete some of them.

Let's look at how to delete a database. If phpMyAdmin is not already open, start it by using one of the methods previously shown. Select the Databases tab (shown in Figure 1-23) and then select the box next to the database to be deleted.

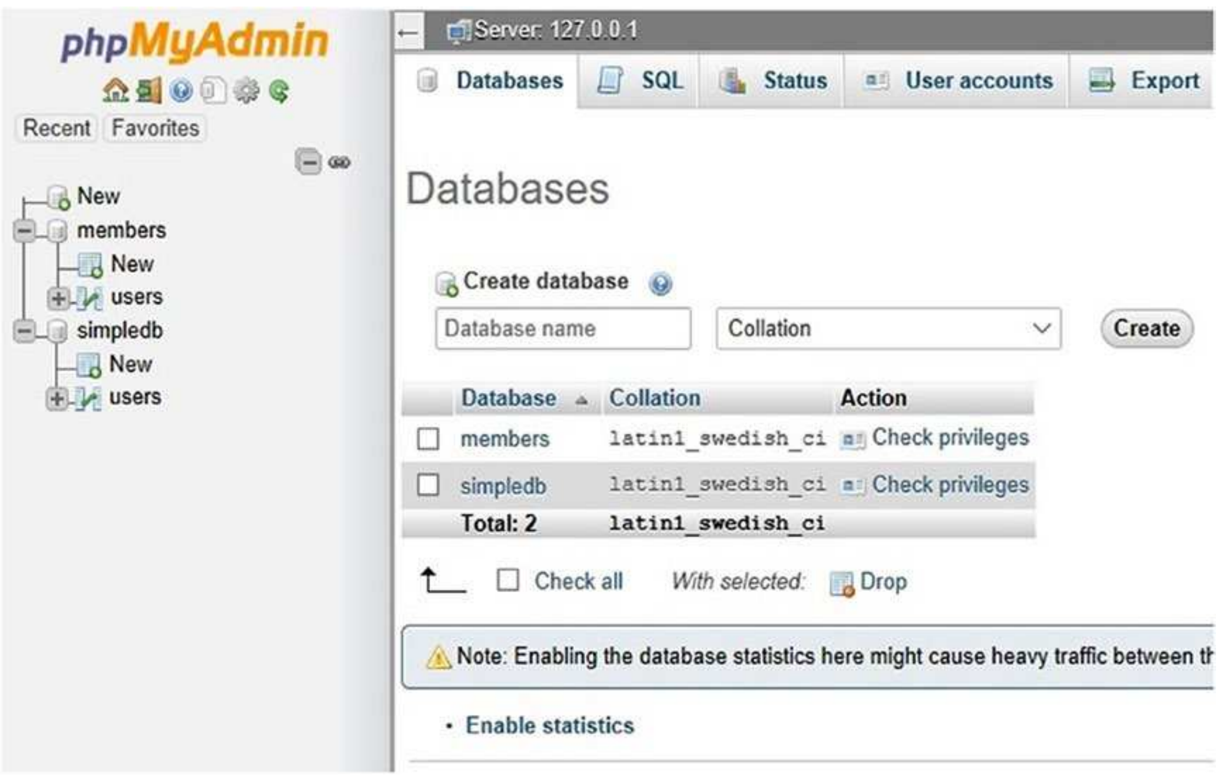

Figure 1-23. Deleting a database

When you have selected the database to be deleted, click the Drop icon (at the bottom right). You will be asked if you really want to delete the database; go ahead and complete the deletion. Everything associated with that database will be deleted, including its tables. If successful, a message will tell you that one database has been dropped.

You may want to preserve a database but delete all or one of its tables. In phpMyAdmin, select the database in the "tree" on the left panel. In the next screen, you will see the table(s); select the box next to the table(s) you want to delete. Figure 1-24 shows the users table of the simpledb database.

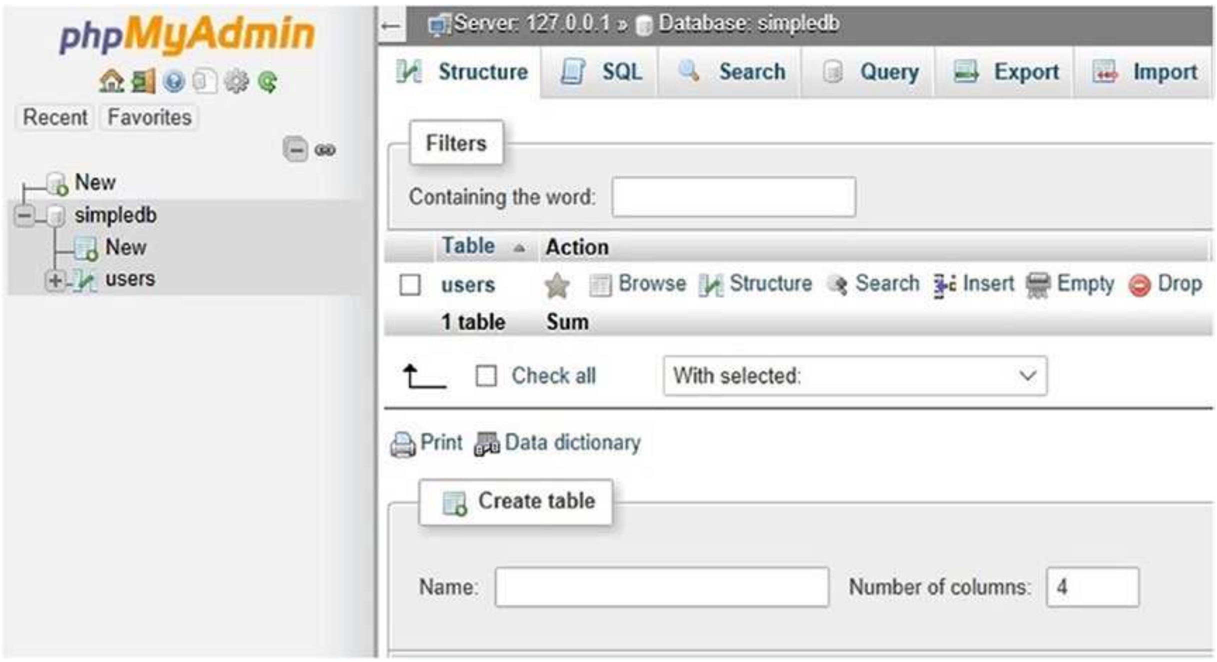

Figure 1-24. *Deleting a table in phpMyAdmin*

After selecting the table to delete, click the Drop icon (to the right of the table name). You will be asked if you really want to delete the table(s). You can choose between Drop and Cancel.

Summary

In this chapter, we defined a database and then looked at two free platforms for developing and testing databases and PHP files. We hope you were successful in downloading and installing XAMPP or easyPHP. We then explored phpMyAdmin and learned how to use it to create a database and a table. SQL was investigated as an alternative method for creating databases and tables. We learned about securing both the phpMyAdmin and databases with a user ID and password. We discovered how to delete databases and tables using phpMyAdmin. In the next chapter, we will create and test simple interactive web pages.

CHAPTER 2

Create Web Pages That Interact with Users

This chapter will demonstrate how a simple database can be linked to a web page so that the page can interact with users. The general public will have access to the website but will not have access to the database structure or the data contained in the tables. They will not be permitted to view the protected pages, such as a members-only page, because they have no password until they register. However, the web designer must provide a way for users to interact with the database to register as a member, search the database, and change a password. The PHP language and SQL code will provide the solution.

After completing this chapter, you will be able to

- Create a database and a template for a website
- Understand and use the PHP include function
- Understand how a server processes a PHP page
- Create an interactive template page
- Create mysqli code to connect to the database
- Create a registration page for members of an organization
- Understand and use the PHP echo() statement
- Create sticky forms
- Use simple arrays
- Create forms that display members' records
- Create code that hashes a password

We will use the simpledb database and the users table from the previous projects for our interactive web pages. Be aware that this tutorial is not practical or secure. It is a stepping stone to the more secure and ambitious projects described in subsequent chapters. In practice, you would never allow ordinary members to view a list of members. The interactive elements in this project are as follows:

- Users can register as members by inserting their details into a form displayed on the screen. The registration details would be entered into the database table and could be used by the administrator to send regular newsletters to members.
- Registered users can change their password.

A. W. West and S. Prettyman, *Practical PHP 7, MySQL 8, and MariaDB Website Databases*,
https://doi.org/10.1007/978-1-4842-3843-1_2

- A user can view the list of members (for this project only). In later chapters, this facility would be available only to the webmaster and (to a limited extent) the membership secretary.

The features that make this example unsuitable for the real-world are as follows:

- No provision is made for registered members to subsequently log in to access a special section or page. This will be dealt with in Chapter 3.
- Users should never be able to access a table of members' details.
- At this early stage, for simplicity, limited filtering of the users' information is provided. The table could therefore contain faulty data and bogus e-mail addresses.
- In this chapter only, any user who knows a member's e-mail address and old password could change the member's password.

All these security issues will be dealt with in subsequent chapters. Despite the drawbacks, the project will provide you with valuable practice in coding and testing the pages. You will also learn more database jargon and some basic PHP code.

Creating the Folder for Holding the Database Pages

Within XAMPP's *htdocs* folder or the easyPHP's *eds-www* folder, create a new subfolder named *simpledb*. All the pages created in this chapter will be placed within the simpledb folder. You have a choice between hand-coding the files from the listings supplied or loading the book's code into the simpledb folder. (You can download the code from Apress.com.) We recommend that you hand-code the programs for this chapter; the files are small and won't take too long to create. You will learn more and learn faster if you type and test the code, especially if you make mistakes and learn to correct them.

Creating the Temporary Template

Obviously, some aspects of an interactive database must be accessible to users. That means incorporating it into a real-world web page. We will name our web page *template.php* (shown in Figure 2-1). As you can see, there is a main header, some body text, a navigation sidebar on the left, an information column on the right, and a footer at the bottom of the page.

Figure 2-1. *The template*

In Listing 2-1, the head section contains the DOCTYPE, a page title, and a link to the Bootstrap style sheet. The body of the page contains some PHP code, and this will be explained step-by-step at the end of the listing.

Because the file contains PHP code (no matter how little), the file is saved with the file type *.php*.

Listing 2-1. Creating a Template for the Project (template.php)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Template for an interactive web page</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <!-- Bootstrap CSS File  -->
  <link rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
  integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4"
  crossorigin="anonymous">
</head>

<body>
<div class="container" style="margin-top:30px">

<!-- Header Section                                                                  #1-->
<header class="jumbotron text-center row"
style="margin-bottom:2px; background:linear-gradient(white, #0073e6);padding:20px;">
  <?php include('header-for-template.php'); ?>
</header>

<!-- Body Section                                                                    #2-->
  <div class="row" style="padding-left: 0px;">
<!-- Left-side Column Menu Section -->
  <nav class="col-sm-2">
      <ul class="nav nav-pills flex-column">
        <?php include('nav.php'); ?>
      </ul>
  </nav>

<!-- Center Column Content Section -->
  <div class="col-sm-8">
     <h2 class="text-center">This is the Home Page</h2>
        <p>The home page content. The home page content. The home page content. The home
        page content. <br>
        The home page content. The home page content. The home page content. The home page
        content. <br>
        The home page content. The home page content. <br>
        The home page content. The home page content. The home page content. </p>
  </div>
```

```
<!-- Right-side Column Content Section                                          #3-->
   <aside class="col-sm-2">
       <php include('info-col.php'); ?>
   </aside>
 </div>

<!-- Footer Content Section                                                     #4-->
<footer class="jumbotron text-center row"
style="padding-bottom:1px; padding-top:8px;">
  <?php include('footer.php'); ?>
</footer>
</div>
</body>
</html>
```

■ **Note** The <?php tag informs the server that PHP code follows. The ?> tag lets the server know where the PHP code ends. The server will pass any code between these tags to the PHP interpreter to be executed. Code outside the tags, HTML and JavaScript code, will eventually be sent to the user's browser without any processing.

The PHP code shown in Listing 2-1 contains the include() function, which will now be explained. (Strictly speaking, the include() function is a PHP language construct, but the difference is so small that we will continue to call it a function for simplicity.)

Introducing the PHP include() Function

You will have noted that there does not seem to be enough code in Listing 2-1 to create the page displayed in Figure 2-1. Here is the reason why.

The four pieces of PHP code shown between the <?php and ?> tags in Listing 2-1 have each pulled an additional file into the page. The page is a combination of five files: a main file plus four external files. The four external files are pulled into the template page using the PHP include() function.

For updating and maintaining a website, the include() function is a wonderful time-saver. Let's suppose you have a client website with 40 pages, and each page needs the same block of menu buttons. If your client asks you to add or delete one menu button, normally you would have to amend 40 pages to add or delete the button on each page. Using the include() function, you would design the website so that each page included this line of PHP code: <?php include('menu.php'); ?>. This would pull the block of menu buttons into each web page. You would design the block of buttons in another file named, say, *menu.php*. To add a new button to all 40 pages, you would have to add the new button only to the one file called *menu.php*.

■ **Note** A PHP function is a tiny program that will perform a particular task. The include() function takes whatever is inside the brackets and pulls it into the page at the place where the include() function is located. PHP has two similar functions: include() and require(). They both pull a file into a page so that the file is included in the displayed page. The difference is in the way they react to a missing or faulty file. If the file to be included is missing or corrupt, include() will not halt the execution of a page. A warning will occur, but the page will continue to load. In contrast, an error will occur if require() can't find the file or the file is faulty. In this case, the page will cease executing. Use include() for anything that is not absolutely necessary for the web page to process properly. In this example, the page could still function without a proper header. But use require() for loading the details of a database because the page will be of little use if the database can't be opened.
In addition, PHP also has include_once() and require_once() functions. Both functions will determine whether the file has already been included before. include_once() will not execute if the file has been included before, without complaint. require_once() will not load the file again if it has already been loaded.

The four elements to be pulled into the page are the header, the block of menu buttons, the info panel on the right side, and the footer. Included files can be any type—for instance, a *.txt* file, a *.php* file, or an *.html* file. Note, your choice of file type can affect whether the code from a file can be viewed externally. HTML files can easily be viewed within browsers. However, do not depend on a type of file ending to signal if your file should be secured. Make sure your code is in a secured folder. An include() statement can be placed anywhere within an HTML page as long as it is surrounded by the PHP tags; these start with the tag <?php and close with the tag ?>. The details of the four included external files are explained in the next section.

You might also be wondering why we don't just use a content management system (CMS), such as WordPress, to assemble and display our page. First, this is not a CMS book, and second, you are actually learning about how CMS systems work. They assemble pages in a similar way. Some CMS systems are created using PHP.

The Included Header File

Figure 2-2 shows a display of the included header file.

Figure 2-2. *The header for the template*

This header is temporary. When we create an interactive version of the template, the new header will contain an additional menu so that users can register and thereby insert their details into the simpledb database.

■ **Caution** Don't be tempted to create the four noninteractive pages yet (*page-2.php*, *page-3.php*, *page-4.php*, and *page-5.php*) because, later in the chapter, these will require a new version of the header.

Listing 2-2a. Header Section of Template File (template.php)

```
<!-- Header Section                                                              #1-->
<header class="jumbotron text-center row"
   style="margin-bottom:2px; background:linear-gradient(white, #0073e6); padding:20px;">
   <?php include('header-for-template.php'); ?>
</header>
```

The PHP statement in Listing 2-2a imports the contents of the *header-for-template* file into the page (code shown in Listing 2-2b). In addition, Bootstrap CSS code is imported in the head section of the template file (see the top of Listing 2-1). With Bootstrap we can reduce the amount of CSS we develop and automatically create pages that adjust to multiple screen sizes (including smartphones). The Bootstrap jumbotron class produces a large box for displaying important information (such as the header). We have also centered the text (text-center) and declared the header as a row. Bootstrap rows are controlled by grids. Each row is divided into cells to help lay out the page. This is similar to the cells and rows in a spreadsheet. We will determine the amount of cells used for each column of our header in Listing 2-2b. We have made a few adjustments to the default settings of the jumbotron by reducing the bottom margin (2px), changing the background to a gradient (white to blue), and changing the padding (20px). Listing 2-2b includes the code for the temporary header.

Listing 2-2b. Code for the Temporary Header (header-for-template.php)

```
<div class="col-sm-2">
<img class="img-fluid float-left" src="logo.jpg" alt="Logo">
</div>
<div class="col-sm-8">
 <h1 class="font-bold">Header Goes Here</h1>
</div>
```

In Listing 2-2a, the class row was referenced in the header tag. This allows us to style a Bootstrap row with columns. In Listing 2-2b. we create two columns: one that holds a logo and one that holds the header text. The logo column uses two small cells (col-sm-2), and the header text uses eight small cells (col-sm-8). The img tag also includes an img-fluid class, along with the float-left class, which allows the Bootstrap code to resize the image on smaller devices. The header text is also bolded (font-bold).

This book assumes you have some CSS and Bootstrap programming knowledge. We will provide some links in each chapter and in the appendix to provide additional resources. However, a great method to learn is to adjust existing code to see how it affects the layout. You may want to spend a few minutes adjusting this example and the following examples to see how they will change the layout of the web page.

■ **Note** We have chosen to use Bootstrap to reduce the amount of CSS code that we have to demonstrate. Bootstrap is an open source toolkit that includes HTML, CSS, and JavaScript. This toolkit provides a responsive grid system and prebuilt CSS/JS components. With Bootstrap, the developer can spend less time on HTML, CSS, and JavaScript coding and more time on programming (with PHP).

In this book we will use Bootstrap classes to provide multiplatform formatting of our web pages. The toolkits can be directly linked online (as we have done in our examples) or downloaded from the Bootstrap website. You can adjust the default settings of many of the toolkit components using CSS. If you are not familiar with Bootstrap, we suggest you review and reference the following sites for more information:

`www.getboostrap.com`

`https://www.w3schools.com/bootstrap/bootstrap_get_started.asp`

The Included Menu File

Figure 2-3 shows the included block of menu buttons.

Figure 2-3. *The included menu buttons*

Listing 2-3a gives the HTML code for the menu within the template page.

Listing 2-3a. Navigation Menu of Template File (template.php)

```
<!-- Body Section                                                                        #2-->
  <div class="row" style="padding-left: 0px;">
<!-- Left-side Column Menu Section -->
  <nav class="col-sm-2">
      <ul class="nav nav-pills flex-column">
       <?php include('nav.php'); ?>
      </ul>
  </nav>
```

Similar to the header, the body section of the page defines a row. With the row are three columns. The left column (col-sm-2) reserves two cells for the navigation menu. The menu will display in “pill” format (see Figure 2-3) and is flexible for different device sizes. The one PHP statement imports the *nav.php* file, which includes the actual code listing for the links (see Listing 2-3b).

Listing 2-3b. The Navigation Menu (nav.php)

```
  <li class="nav-item">
          <a class="nav-link active" href="index.php">Home</a>
  </li>
  <li class="nav-item">
          <a class="nav-link" href="page-2.php">Page 2</a>
   <li>
   <li class="nav-item">
          <a class="nav-link" href="page-3.php">Page 3</a>
   </li>
   <li class="nav-item">
          <a class="nav-link" href="page-4.php">Page 4</a>
   </li>
   <li class="nav-item">
          <a class="nav-link" href="page-5.php">Page 5</a>
     </li>
```

The code for the navigation section (menu) includes an unordered list with links to each of the future pages of the website. This is a common practice to create menus with unordered lists. Each list item is defined with a class nav-item, which is used by Bootstrap to format the menu. Each link (<a>) is defined with a class nav-link, which formats each button for the menu. The active class is used to visually indicate which page is currently displayed (see Figure 2-3). There is also a deactive class that will gray out a menu item.

The Included Information Column

The included information column sits on the right side of the page, as shown in Figure 2-4.

Figure 2-4. *Information column for inclusion in the template*

Listing 2-4a gives the HTML code for the information column within the template file.

Listing 2-4a. Information Column in Template File (template.php)

```
<!-- Right-side Column Content Section                                                    #3-->
   <aside class="col-sm-2">
         <php include('info-col.php'); ?>
   </aside>
```

The information column reserves two small cells for display (col-sm-2). The PHP include statement pulls in the details of the information column from *info-col.php* (see Listing 2-4b).

Listing 2-4b. The Information Column Details (info-col.css)

```
<div>
<h3>This is the information column</h3>
        <p>Web design by <br>A W West and<br>Steve Prettyman</p>
</div>
```

Listing 2-4b includes the HTML code that displays information on the right side of the page. The information moves to the right side because the menu and the main section of the page (discussed soon) are defined with columns and cells first before the information column is defined.

The Included Footer File

Figure 2-5 shows the included footer.

Figure 2-5. *The footer for inclusion in the template*

Listing 2-5a gives the HTML code for the footer.

Listing 2-5a. Footer code for Template File (.php)

```
<!-- Footer Content Section                                                    #4-->
<footer class="jumbotron text-center row"
style="padding-bottom:1px; padding-top:8px;">
  <?php include('footer.php'); ?>
</footer>
```

The footer is also declared as a jumbotron and a row. The text is centered (text-center). In addition, the default setting for the jumbotron is adjusted to reduce the block size to be more appropriate for a footer. The PHP code pulls in the HTML to be displayed (see Listing 2-5b).

Listing 2-5b. The Footer (footer.php)

```
<p class="h6 col-sm-12">Copyright &copy; Adrian West & Steve Prettyman 2018  Designed by
<a href="http://www.colycomputerhelp.co.uk/">Adrian West </a> and
<a href=http://www.littleoceanwaves.com>Steve Prettyman </a>  Valid
<a href="http://jigsaw.w3.org/css-validator/">CSS</a> &amp;
<a href="http://validator.w3.org/">HTML5</a></p>
```

Listing 2-5b uses the Bootstrap class h6 to determine the attributes of the font used to display the text. The paragraph tag also reserves 12 small columns (the complete width of the page) to display the footer information.

How Does the Server Process the Page?

When an HTML file is saved as a PHP file, the *.php* extension alerts the server to processes the HTML as normal but also looks out for any PHP code. The server is in HTML (copy) mode by default, and any HTML code is sent to the browser as normal. When it finds a <?php tag, the server switches to PHP (interpret) mode. Any PHP code is interpreted. The results of executing the interpreted PHP code are returned to the page, not the actual PHP code itself. It continues in this mode until it encounters the PHP closing tag ?>; it then switches back to HTML (copy) mode. This cyclic behavior continues until the end of the page of code.

The Interactive Version of the Template

In this version of the template, we will introduce interactivity by creating a new header file with an additional menu. This menu will allow users to register and enter their details into the simpledb database. It will also allow users to change their password and view the table of members.

Figure 2-6 shows the new header.

Figure 2-6. *A registration menu is added to the header*

The interactive element will be embedded in the header; therefore, the previous header is now modified to include a menu. The new header will be named *header.php*, and the code is shown in Listing 2-6.

Listing 2-6. Placing a Registration Menu in the New Header (header.php)

```
<div class="col-sm-2">
<img class="img-fluid float-left" src="logo.jpg" alt="Logo">
</div>
<div class="col-sm-8">
 <h1 class="blue-text mb-4 font-bold">Header Goes Here</h1>
 </div>
     <nav class="col-sm-2">
         <div class="btn-group-vertical btn-group-sm" role="group" aria-label="Button Group">
  <button type="button" class="btn btn-secondary" onclick="location.href = 'register-page.php'" >
                   Register</button>
  <button type="button" class="btn btn-secondary" onclick="location.href = 'register-view_
  users-page.php'">
                  View Users</button>
<button type="button" class="btn btn-secondary" onclick="location.href = 'register-password.
php'" >
                  New Password</button>
</div>
    </nav>
```

In this example, a Bootstrap vertical button group has been created (btn-group-vertical) with small buttons (btn-group-sm). Each button within the group is defined with the HTML button tag and the Bootstrap btn and btn-secondary classes. The on-click attribute is used to call the page requested by the user. In later chapters, we will also see examples of using the Bootstrap navbar to create a horizontal bar of buttons below a header.

In the template file, we will need to swap the previous included header for the new header. The new template page will be named *index.php*, and it will now have two blocks of menu buttons, as shown in Figure 2-7.

Figure 2-7. *The new home page template with two menus*

The only difference in the code between the old template and the new index file is the new header reference. The changes are shown in the following snippet of code in Listing 2-7. The complete code is contained in the *index.php* file that can be downloaded from the Apress.com website.

Listing 2-7. Including the New Header in the Home Page (index.php)

```
<!-- Header Section -->
<header class="jumbotron text-center row"
    style="margin-bottom:2px; background:linear-gradient(white, #0073e6); padding:20px;">
    <?php include('header-for-template.php'); ?>
</header>
```

Now you can create the four ordinary website pages, using the new template (*index.php*); save four copies of the file, naming the copies *page-2.php*, *page-3.php*, *page-4.php*, and *page-5.php*. Change the content of those pages a little so that they differ from *index.php* and to indicate which page the user is viewing. You might consider changing the active nav-item (button) to the page that is currently being viewed. The code shown previously sets the Home page navigation button to active.

Connecting to the Database

Before we can do anything in the database, we must connect to it. This is achieved by creating a connection file that we will call *mysqli_connect.php*. The code for the connection file is given in the next snippet.

Listing for the Snippet of Code That Connects to the Database (mysqli_connect.php)

```
<?php
// This file provides the information for accessing the database.and connecting to MySQL.
// First, we define the constants:                                             #1
Define ('DB_USER', 'horatio'); // or whatever userid you created
Define ('DB_PASSWORD', 'Hmsv1ct0ry'); // or whatever password you created
Define ('DB_HOST', 'localhost');
Define ('DB_NAME', 'simpledb');

// Next we assign the database connection to a variable that we will call $dbcon:      #2
try
{
      $dbcon = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
      mysqli_set_charset($dbcon, 'utf8'); //                                   #4
      // more code will go here later
}
catch(Exception $e) // We finally handle any problems here                     #3
{
     // print "An Exception occurred. Message: " . $e->getMessage();
     print "The system is busy please try later";
}
catch(Error $e)
{
      //print "An Error occurred. Message: " . $e->getMessage();
      print "The system is busy please try again later.";
}
?>
```

Save the file as *mysqli_connect.php*, and place it in the XAMPP *htdocs* or the easyPHP *eds-www* folder.

■ **Note** When a database is set up on a remote host, this file is placed one level above the root folder for security. Not protecting this file can lead to DOS attacks. Leaving this file located in a public area of the web server will allow anyone to use the code to access the database. Place the file one level above the program code, in an area secured by the web server (or in another secured folder). Use a require statement similar to require('../mysqlconnect.php'); to access your connection information. The examples in this book access the connection file from the same location as the code files. This is done for easy testing but should not be done in a live environment on a publicly accessible web server.

In this book the catch messages used are for debugging. Live sites should not display error messages to the user. A generic message (as shown in the previous listing) should be displayed to the user. Users will be much more willing to come back to a site that is currently "busy" than a site that is displaying error messages. These error messages can also be a security breach by possibility displaying code lines that may cause errors.

Explanation of the Code

The *mysqli_connect.php* file contains some code conventions that you may not be familiar with, so let's briefly run through them here.

Single-line comments begin with a double forward slash or with a hash symbol—for example:

```
// more code will go here later
# more code will go here later
```

Constants are fixed definitions created by using the keyword `DEFINE`.

```
DEFINE ('DB_USER', 'horatio');
```

Variables like `$dbcon` are memory storage devices for containing information that can be made to vary; they are preceded by a dollar symbol. Variables are created using the following format: $var = some_information. The equal sign is called an *assignment operator*; it assigns what is on the right side of the equal sign to the variable on the left side. The example assigns some information to the variable $var.

We will now examine the code using the line numbers as references.

```
// First, we define the constants:                                        #1
Define ('DB_USER', 'horatio'); // or whatever userid you created
Define ('DB_PASSWORD', 'Hmsv1ct0ry'); // or whatever password you created
Define ('DB_HOST', 'localhost');
Define ('DB_NAME', 'simpledb');
```

As a convention, not a rule, programmers create constant names with all capital letters. This helps to identify which items are constants (which cannot be changed while the program is running) and which items are variables (can be changed while the program is running). In PHP, we can also tell that an item is a variable if it has the dollar sign as the first character. It is good practice to place your DEFINE code lines near the top of your program code. This makes it easy to find when you must change a value. For example, if you need to change a tax rate.

■ **Note** If you did not create a user ID and password for the simpledb database, the user ID will be root, and the password will be "" (two quotes with no spaces in between them).

```
// Next we assign the database connection to a variable that we will call $dbcon:      #2
try
{
      $dbcon = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
      // more code will go here later
}
```

The text following the dollar sign can be anything that is relevant to the information held in the memory. For instance, the variable for connecting to our database in the listing is named $dbcon. This was chosen to signify a connection to a database; it could be $connect_db or any other text that indicates a connection to the database. Some programmers prefer to use camel case ($dbCon), where the first word is lowercase and any words that follow begin with an uppercase letter. Others keep all words lowercase, and some even use an underscore (_) between the words ($db_con). It does not matter what style you use. Just be consistent and name your variables in the same style.

We use the host, username, password, and database name that we defined earlier to connect to the database.

The try block (everything between the { } is considered to be in a block) will “throw” any problems (errors or exceptions) into a catch block for the program to handle.

```
catch(Exception $e) // We finally handle any problems here                              #3
{
      //print "An Exception occurred. Message: " . $e->getMessage();
      print "The system is busy please try later";
}
catch(Error $e)
{
      //print "An Error occurred. Message: " . $e->getMessage();
      print "The system is busy please try again later.";
}
```

For testing purposes, sometimes we will display the error and exception messages that may occur. In production, we would replace the print statements with a generic message to the user, such as “System currently not available, please try later.” Allowing users to see error messages gives them a feeling that the design of your website is poor, is vulnerable, and can crash frequently. Users who see error messages on websites usually leave the site and do not return. However, in today’s environment, it is not unusual to see a generic “System currently busy or not available” type message. Many sites now limit the number of users to keep from hackers trying to flood and crash the site.

It is also a major security breach to let users see any error messages. These messages may reveal code that provides actual locations of items, such as the database itself. This information, along with a timestamp, should be passed to a log file to assist the system administrator.

You may see example code on the Internet or older books with the PHP database code that uses OR die. This has been a PHP standard since the beginning of time. However, with the release of PHP 7, the designers are encouraging all developers to use the industry-standard try/catch format. Also, when using OR die, you cannot capture all exceptions and errors. With try/catch, you can now do so. In PHP 7, the developer is encouraged to use the mysqli (instead of mysql) method to connect and access databases. It is considered the most secure method of access.

We have changed the HTML-encoded language from the default to utf-8 as suggested in Listing 2-8. The connection to the database must also include code that indicates the correct encoded language. The type of encoded language used on the HTML page must match the encoded language used in the database table. Inconsistent matches can cause problems, including invalid search results.

```
mysqli_set_charset($dbcon, 'utf8');                                    #4
```

(Note that the format is different from the code set in an HTML document because it does not include the hyphen as in utf-8.)

Next, we need some pages for the header's new menu to call. These pages will contain the interactive features. The pages will allow users to register, allow users to view a table of the registered persons, and permit a password to be changed.

The Registration Page

■ **Caution** When you finally migrate a database to a remote host, be sure to comply with the Data Protection Act for your territory or country, and state clearly on the registration page that the user's personal details will not be shared or sold to other organizations. The rules covering the protection of data vary from country to country, and it is essential that you read them and comply with them. Usually, any person within an organization who can access users' details must sign a document agreeing never to share personal information. An annual registration fee might be required to the government organization that regulates the data protection law. These precautions do not apply to experimental databases using fictitious data such as those described in this book.

The registration page allows users to enter their personal details directly into a table in the database. Figure 2-8 shows the interface.

***Figure 2-8.** The registration page*

When the user clicks the Register menu button on the new header, the page shown in Figure 2-8 is displayed. If the user fills out the form correctly and then clicks the Register button (below the entry fields), the user's entries are entered in the users table in the database. A "thank you" page is then displayed.

Note that the Register and New Password buttons on the registration page are now redundant because the user has already accessed the registration page. Obviously, the user does not yet have a password to change. The redundant buttons will be left in place for all the examples in this chapter to avoid complicating the instructions. The redundant buttons will be removed or changed in the next two chapters.

Now we will examine the code for the entire registration page.

Listing 2-8a. Creating the Registration Page (register-page.php)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Template for an interactive web page</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <!-- Bootstrap CSS File -->
  <link rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
  integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4"
  crossorigin="anonymous">
<script src="verify.js"></script>
</head>
<body>
    <div class="container" style="margin-top:30px">
    <!-- Header Section -->
    <header class="jumbotron text-center row"
         style="margin-bottom:2px; background:linear-gradient(white, #0073e6); padding:20px;">
         <?php include('header.php'); ?>
    </header>
    <!-- Body Section -->
        <div class="row" style="padding-left: 0px;">
    <!-- Left-side Column Menu Section -->
     <nav class="col-sm-2">
      <ul class="nav nav-pills flex-column">
          <?php include('nav.php'); ?>
      </ul>
    </nav>

    <!-- Validate Input
   <?php
    if ($_SERVER['REQUEST_METHOD'] == 'POST') {                                       //#1
     require('process-register-page.php');
    } // End of the main Submit conditional.
   ?>
   <div class="col-sm-8">
      <h2 class="h2 text-center">Register</h2>
```

```
  <form action="register-page.php" method="post" onsubmit="return checked(); >     <!-- #2 -->
     <div class="form-group row">

<label for="first_name" class="col-sm-4 col-form-label">First Name:</label>
    <div class="col-sm-8">
      <input type="text" class="form-control" id="first_name" name="first_name"
          placeholder="First Name" maxlength="30" required
          value="<?php if (isset($_POST['first_name'])) echo $_POST['first_name']; ?>" >
    </div>
  </div>

  <div class="form-group row">
    <label for="last_name" class="col-sm-4 col-form-label">Last Name:</label>
    <div class="col-sm-8">
      <input type="text" class="form-control" id="last_name" name="last_name"
          placeholder="Last Name" maxlength="40" required
          value="<?php if (isset($_POST['last_name'])) echo $_POST['last_name']; ?>">
    </div>
  </div>

<div class="form-group row">
    <label for="email" class="col-sm-4 col-form-label">E-mail:</label>
    <div class="col-sm-8">
      <input type="email" class="form-control" id="email" name="email"
          placeholder="E-mail" maxlength="60" required
          value="<?php if (isset($_POST['email'])) echo $_POST['email']; ?>">
    </div>
  </div>

<div class="form-group row">
    <label for="password1" class="col-sm-4 col-form-label">Password:</label>
    <div class="col-sm-8">
      <input type="password" class="form-control" id="password1" name="password1"
          placeholder="Password" minlength="8" maxlength="12"
          required value="<?php if (isset($_POST['password1'])) echo $_POST['password1']; ?>">
          <span id='message'>Between 8 and 12 characters.</span>
  </div>

<div class="form-group row">
    <label for="password2" class="col-sm-4 col-form-label">Confirm Password:</label>
    <div class="col-sm-8">
      <input type="password" class="form-control" id="password2" name="password2"
          placeholder="Confirm Password" minlength="8" maxlength="12" required
          value="<?php if (isset($_POST['password2'])) echo $_POST['password2']; ?>">
    </div>
  </div>
```

```
<div class="form-group row">
    <div class="col-sm-12">
       <input id="submit" class="btn btn-primary" type="submit" name="submit"
       value="Register">
    </div>
 </div>
</form>
</div>

!-- Right-side Column Content Section                                            #3 -->
<?php
if(!isset($errorstring)) {
        echo '<aside class="col-sm-2">';
        include('info-col.php');
        echo '</aside>';
        echo '</div>';
        echo '<footer class="jumbotron text-center row col-sm-14"
                style="padding-bottom:1px; padding-top:8px;">';
 }
 else
 {
        echo '<footer class="jumbotron text-center col-sm-12"
        style="padding-bottom:1px; padding-top:8px;">';
 }
  include('footer.php');
 ?>
</div>
</body>
</html>
```

Explanation of the Code

The basic structure of the code is the same as the index file. The differences occur in the body, specifically in the center column that now contains the form for the user to register.

```
<?php
 if ($_SERVER['REQUEST_METHOD'] == 'POST') {                                    //#1
  require('process-register-page.php');
 } // End of the main Submit conditional.
?>
```

In PHP, forms can be processed either by the form submit attribute passing the information to a separate PHP program or by including PHP code within the same file as the form. In this example, we technically have included the PHP code in the same file, because we are pulling in the code from the process-register-page file. We also know this because the action attribute of the form is calling the file itself (form action="register-page.php") instead of another file (see partial listing next). If the submit button has been clicked, the PHP code in the process register file is executed. If the user has not clicked the submit button (probably the first time the page has been displayed), the form is shown. Let's take a look at the form itself.

```
<form action="register-page.php" method="post" onsubmit="return checked();>     <!-- #2 -->
     <div class="form-group row">

<label for="first_name" class="col-sm-4 col-form-label">First Name:</label>
    <div class="col-sm-8">
      <input type="text" class="form-control" id="first_name" name="first_name"
          placeholder="First Name" maxlength="30" required
          value="<?php if (isset($_POST['first_name'])) echo $_POST['first_name']; ?>" >
    </div>
  </div>
```

The register-page submits to itself when the user clicks the submit button. When submitted, the form calls the JavaScript function checked() to verify the passwords. The code for this function is listed here:

```
function checked() {
  if (document.getElementById('password1').value ==
    document.getElementById('password2').value) {
    document.getElementById('message').style.color = 'green';
    document.getElementById('message').innerHTML = 'Passwords match';
        return true;
  } else {
    document.getElementById('message').style.color = 'red';
    document.getElementById('message').innerHTML = 'Passwords do not match';
        return false;
  }
}
```

This code was imported from the verify.js program along with the Bootstrap CSS code. If the passwords match, the code is submitted. If the passwords do not match, an error message is displayed, allowing the user to correct the problem.

The PHP code, which we will see next, will verify the information the user has provided and insert it into the database. There is a lot that happens within each element (such as the first_name textbox) of the form. The first_name textbox has a label attached that uses four small Boostrap grid cells. The textbox uses eight small grid cells. The form and textboxes are formatted by Boostrap using the form-control class. The maximum length of a first name that can be entered by the user is 30 characters. The required attribute tells HTML that the user must enter information into the textbox before the information will be submitted.

The one line of PHP code, marked as #2, creates a sticky element that saves any entries made by the user. This allows the page to retain any entries when it is reloaded. Every time the user clicks the submit button, the page is reloaded. Normally this would wipe out any entries since HTML pages do not retain information (disconnected data). When the user submits the form, the information is passed to the server via the POST method. We can retrieve any element that has been passed by the form using the $_POST method and the name of the element ($_POST['first_name']). The PHP code uses isset to determine whether there was an entry in the first_name textbox. If there was an entry, the value that was passed is glued back into the textbox. This allows the user to see what they entered originally and to respond to any error messages and correct problems without having to completely retype the entries. The other textboxes work in a similar way.

■ **Note** Displaying information from $_POST directly to a web page can open a risk that hackers can use to manipulate web pages, attempt CSRF/XSS attacks, and manipulate the data in databases. As the chapters progress, the data sanitation and security will increase to reduce the chances of security breaches. The code in this chapter is not meant to be used on a live site.

```
!-- Right-side Column Content Section                                                   #3 -->
<?php
if(!isset($errorstring)) {
        echo '<aside class="col-sm-2">';
        include('info-col.php');
        echo '</aside>';
        echo '</div>';
        echo '<footer class="jumbotron text-center row col-sm-14"
                style="padding-bottom:1px; padding-top:8px;">';
 }
 else
 {
        echo '<footer class="jumbotron text-center col-sm-12"
        style="padding-bottom:1px; padding-top:8px;">';
 }
  include('footer.php');
```

Error message(s) will be produced if the information entered does not validate. As you will see in the PHP code explained soon, most errors messages are placed in the variable $errorstring. If this variable exists, one or more errors exist. When this occurs, the right column (see #3 in the previous code) is not displayed to provide space for the error message(s). The column size of the footer is also slightly adjusted.

Now let's look at the PHP code that validates the information the user provided.

Listing 2-8b. Validating the Registration Page (process-register-page.php)

```
<?php
// This script is a query that INSERTs a record in the users table.
// Check that form has been submitted:
        $errors = array(); // Initialize an error array.                                 #1
       // Check for a first name:                                                        #2
       $first_name = trim($_POST['first_name']);
       if (empty($first_name)) {
              $errors[] = 'You forgot to enter your first name.';
       }
                // Check for a last name:
       $last_name = trim($_POST['last_name']);
       if (empty($last_name)) {
              $errors[] = 'You forgot to enter your last name.';
       }
       // Check for an email address:
       $email = trim($_POST['email']);
       if (empty($email)) {
              $errors[] = 'You forgot to enter your email address.';
       }
```

```
        // Check for a password and match against the confirmed password:              #3
        $password1 = trim($_POST['password1']);
        $password2 = trim($_POST['password2']);
        if (!empty($password1)) {
               if ($password1 !== $password2) {
                        $errors[] = 'Your two passwords did not match.';
                }
         } else {
                $errors[] = 'You forgot to enter your password.';
         }       if (empty($errors)) { // If everything's OK.                              #4
try {
         // Register the user in the database...
         // Hash password current 60 characters but can increase
             $hashed_passcode = password_hash($password1, PASSWORD_DEFAULT);            //#5
              require ('mysqli_connect.php'); // Connect to the db.                     //#6
         // Make the query:                                                              #7
         $query = "INSERT INTO users (userid, first_name, last_name, email, password,
         registration_date) ";
                 $query .="VALUES(' ', ?, ?, ?, ?, NOW() )";                            //#8
         $q = mysqli_stmt_init($dbcon);                                                 //#9
         mysqli_stmt_prepare($q, $query);
         // use prepared statement to ensure that only text is inserted
         // bind fields to SQL Statement
         mysqli_stmt_bind_param($q, 'ssss', $first_name, $last_name, $email, $hashed_passcode);
      // execute query
         mysqli_stmt_execute($q);
         if (mysqli_stmt_affected_rows($q) == 1) { // One record inserted               #10
                 header ("location: register-thanks.php");
                 exit();
                 } else { // If it did not run OK.
                 // Public message:
                                 $errorstring = "<p class='text-center col-sm-8'
                                 style='color:red'>";
                         $errorstring .= "System Error<br />You could not be registered due ";
                         $errorstring .= "to a system error. We apologize for any
                         inconvenience.</p>";
                         echo "<p class=' text-center col-sm-2'
                         style='color:red'>$errorstring</p>";
                 // Debugging message below do not use in production
                 //echo '<p>' . mysqli_error($dbcon) . '<br><br>Query: ' . $query . '</p>';
                                mysqli_close($dbcon); // Close the database connection.
                 // include footer then close program to stop execution
                         echo '<footer class="jumbotron text-center col-sm-12"
                                         style="padding-bottom:1px; padding-top:8px;">
                                                 include("footer.php");
                                                </footer>';
                              exit();
        }
    }
```

```
   catch(Exception $e) // We finally handle any problems here                          #11
   {
     // print "An Exception occurred. Message: " . $e->getMessage();
     print "The system is busy please try later";
   }
   catch(Error $e)
   {
      //print "An Error occurred. Message: " . $e->getMessage();
      print "The system is busy please try again later.";
   }
        } else { // Report the errors.                                                 #12
               $errorstring = "Error! The following error(s) occurred:<br>";
               foreach ($errors as $msg) { // Print each error.
                        $errorstring .= " - $msg<br>\n";
               }
               $errorstring .= "Please try again.<br>";
               echo "<p class=' text-center col-sm-2' style='color:red'>$errorstring</p>";
               }// End of if (empty($errors)) IF.
?>
```

■ **Note** At this point, beginners may be mystified because they are more familiar with code that steps through a page from top to bottom. In the preceding example of interactive code, the beginner would expect the form fields to be at the top of the page of code. In fact, they come last in the listing (after the PHP code that has been imported via the require method). There is a main if statement (if ($_SERVER['REQUEST_METHOD'] == 'POST') {) that controls the flow of the program. If the user has already been to the page and clicked the submit button, the if statement is true, and the PHP code is executed (from the included file; process_register_page.php in the previous example). If this is the first time the user has visited the page, the if statement is false. Thus, the HTML code within the else part of the statement is executed. HTML5 code will also do some limited verification of information entered by the user before the form submitted and the PHP code can be executed. In addition, the PHP code will verify that what was received from the web page is “good,” verified, and sanitized information.

It may seem that we are duplicating effort by verifying the information twice. However, using HTML5 (and maybe some JavaScript) to do a first attempt to verify information reduces server calls and speeds up the verification process. We still need to verify the information again on the server because it can be manipulated, intentionally or not, before it arrives.

Explanation of the Code

Many of the PHP statements were explained by comments in the listings, but some items need further comment. The PHP code might look horribly complicated, but we will break it apart into smaller pieces to help you understand it. Let’s begin now.

```
$errors = array(); // Initialize an error array.                                       #1
```

An array is a method to store multiple items of similar information into one storage location. In our example, we are using the array to store any error messages. The previous line initiates the array and gives it the name $errors. Technically, $errors is now pointing to an area in memory that has been reserved for the array. Some additional information on arrays and their use is given at the end of this chapter and in the appendix.

```
// Check for a first name:                                                  #2
$first_name = trim($first_name);
if (empty($first_name)) {
       $errors[] = 'You forgot to enter your first name.';
}
```

An element in a global variable such as $_POST['first_name'] is always enclosed in square brackets. This is a feature of super global variables. Technically everything that is passed from our form to the PHP code is passed into a post array (the reason we used square brackets instead of round ones). We then use the POST super global statements to retrieve what we want from the array.

The code beginning with $first_name = trim is an instruction to trim (remove) any spaces or other white spaces from the beginning and end of the user's first_name entry. Spaces at the beginning and end of input data are unwanted characters. The trimmed first_name is then assigned to the variable $first_name. We should do additional removal of invalid characters for strong security. However, for now, let's keep it as simple as possible. The format shown earlier is also used for the last name and e-mail.

■ **Note** Any variable taking the $_POST[ ] format, such as $_POST['first_name'], is a global variable—it is accessible to that page on the website. Ordinary variables have the format $first_name, and they can be accessed only by loading the page in which they appear. $_POST will pull the information passed from the HTML form via a post array that has been automatically populated when the user clicked the submit button. The variable name contained in the square brackets (first_name) must match the NAME parameter of the HTML input statement exactly.

```
// Check for a password and match against the confirmed password:           #3
$password1 = trim($_POST['password1']);
$password2 = trim($_POST['password2']);
if (!empty($password1)) {
        if ($password1 !== $password2) {
                $errors[] = 'Your two password did not match.';
        }
} else {
        $errors[] = 'You forgot to enter your password.';
}
```

Both of the new passwords are trimmed of unnecessary spaces. If password1 is not empty (!empty), then the comparison of the two new passwords is performed. The !== (one ! and two =) symbols determine whether the passwords entered in the form are the same. This format of the not equals ensures that the uppercase and lowercase are the same. It also ensures that the data types (string compared to integer) are the same. If the passwords are not the same, an error message is placed in the error array to be displayed later.

```
if (empty($errors)) { // If everything's OK.                                #4
```

This block of code is in the successful section. This section starts the registration process when all the fields have been filled out correctly—in which case the $errors array is empty. The line is saying "Because there are no errors, we can connect to the database and insert the user's data."

```
$hashed_passcode = password_hash($password, PASSWORD_DEFAULT);                        //#5
```

We must make sure that any passwords are as secure as possible. The previous code uses the PHP password hashing method to convert the password entered by the user into a secure password. PASSWORD_DEFAULT will always contain the most current PHP hashing code available. In the database table we have set the password field to hold 60 characters. In the future, new hashing techniques might require you to increase the size of this field.

```
require ('mysqli_connect.php'); // Connect to the db.                                 //#6
```

In this line, the database connection is made with require() instead of include() so that the script will run if the connection is made, but it will not run if no connection is available. This is because there is no point continuing if the database is not accessible because data cannot be inserted into its table. We use require() when something vital is to be included. We could have used require_once here since we want to pull in this code only one time to establish the connection to the database.

```
// Make the query:                                                                     #7
$query = "INSERT INTO users (userid, first_name, last_name, email, password,
registration_date) ";
```

If a successful connection to the database is achieved, this line prepares the data for entry into the table called *users*. The word *query* sometimes means "query," but most often it means "Do something." In this case, it means "Insert a new record using the following data...." The line is saying "Insert into the table named *users*, under the column headings labeled `userid`, `first_name`, and so on...." The query is assigned to a variable `$query`. Line #7 and line #8 are, in fact, SQL prepared queries. This demonstrates how well HTML, PHP, and SQL work together.

```
$query .="VALUES(' ', ?, ?, ?, ?, NOW() )";                                           //#8
```

This piece of SQL code provides a holding place for the data to be entered. We are using a prepared statement because it will never execute data that has been bound to the locations (where the ? marks exist). This keeps hackers from being able to insert additional SQL statements into our SQL string in an attempt to destroy our data. Note that the first value is deliberately empty (quotes with an empty space in between) because it is the field that is automatically incremented and entered by the MySQL or MariaDB database management system, not by the user. The NOW function automatically places the data and time into the registration_date column in the database table.

```
$q = mysqli_stmt_init($dbcon);                                                        //#9
mysqli_stmt_prepare($q, $query);
// use prepared statement to ensure that only text is inserted
// bind fields to SQL Statement
mysqli_stmt_bind_param($q, 'ssss', $first_name, $last_name, $email, $hashed_passcode);
// param must include four variables to match the four s parameters
// execute query
mysqli_stmt_execute($q);
```

The connection string from the previously included file (*mysqli_connect.php*) is attached to mysqli and can be referenced with the $q variable. The $query string is then declared as a prepared statement (which indicates that variables will be inserted into the code) and is associated with the database connection string. The variables ($first_name, $last_name, $email, $hashed_passcode) are attached to the prepared SQL statement. Note the s symbol indicates each individual variable contains a string. To indicate numeric information, you can use i (integer) or d (double). b is also available for BLOB. The statement is then executed against the database table.

```
if (mysqli_stmt_affected_rows($q) == 1) { // One record inserted                    #10
       header ("location: register-thanks.php");
       exit();
```

If the operation was a success, the database will be closed. Then the header method will replace the current registration page with a “thank you” page. The header method produces an HTML Get message to request the page. The exit command will close down the current page since it is no longer in use.

```
catch(Exception $e) // We finally handle any problems here                          #11
   {
     // print "An Exception occurred. Message: " . $e->getMessage();
     print "The system is busy please try later";
   }
   catch(Error $e)
   {
      //print "An Error occurred. Message: " . $e->getMessage();
      print "The system is busy please try again later.";
   }
```

If there is an error in executing the SQL commands or some other error or exception, the flow of the code will jump into the catch blocks to handle the problem. As mentioned, for testing we can display our errors, but when we switch to production, we will display a generic “System currently not available” message and log the problems in an error log.

```
} else { // Report the errors.                                                      #12

               $errorstring = "Error! The following error(s) occurred:<br>";
               foreach ($errors as $msg) { // Print each error.
                       $errorstring .= " - $msg<br>\n";
               }
               $errorstring .= "Please try again.<br>";
               echo "<p class='text-center col-sm-2' style='color:red'>$errorstring;</p>";
               }// End of if (empty($errors)) IF.
```

If there are user errors (invalid entries), messages will exist in the $error array. The else structure is implemented when there are entries in the array. The foreach loop cycles through the $errors array, and if it finds any error messages, it is then displayed (echoed) on the screen (foreach is one word without a space).

The $errorstring variable is checked in the HTML code to determine whether to display the right column. If there are errors, the right column is not displayed to make room for the error messages.

You might be wondering why we are creating PHP code that does a similar validation to the HTML5 code shown. PHP code is interpreted and executed on the server. HTML code is executed in the browser. Even if the code is validated in the browser, it might get corrupted on its path to the server PHP program. You should validate with HTML (and JavaScript) for quicker validation and response to the user. It also cuts down on server calls.

The PHP Keyword echo

The keyword echo appears in many places in the previous code. It is the PHP way of telling a browser to “Display something on the screen.” Some designers use the alternative keyword print, but this book will usually use echo because we have found that beginners confuse this with the command for sending something to an inkjet or laser printer; hard-copy printing is something that PHP cannot do.

■ **Note** Any HTML code that you want can be placed in an echo statement. The echo writes the content into the HTML document (virtually, not literally) so that it can be displayed in a browser.

Beginners can be puzzled by the behavior of echo when a line break is required. For instance, the following code displays no line space:

```
echo "I found that PHP ";
echo "was much easier to learn than Perl";
```

Browsers display this as follows:
I found that PHP was much easier to learn than Perl
To push the second line down, you must insert the line-break tag, <br>, as follows:

```
echo "I found that PHP<br>";
echo "was much easier to learn than Perl";
```

Browsers display this as follows:
I found that PHP was much easier to learn than Perl

The “Thank You” Page

Figure 2-9 shows the “thank you” page.

Figure 2-9. *The “thank you” page*

The registration page calls up the “thank you” page with the PHP header statement. You will find this in the successful section of the registration page’s code. For convenience, we repeat it here:

```
$dbcon->close();
header ("location: register-thanks.php");
exit();
```

Listing 2-9 shows the entire “thank you” page code.

Listing 2-9. Creating the “Thank You” Page (register-thanks.php)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Template for an interactive web page</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <!-- Bootstrap CSS File -->
  <link rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
  integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4"
  crossorigin="anonymous">
</head>
<body>
<div class="container" style="margin-top:30px">
<!-- Header Section -->
<header class="jumbotron text-center row"
style="margin-bottom:2px; background:linear-gradient(white, #0073e6); padding:20px;">
  <?php include('header.php'); ?>
</header>
<!-- Body Section -->
  <div class="row" style="padding-left: 0px;">
<!-- Left-side Column Menu Section -->
  <nav class="col-sm-2">
      <ul class="nav nav-pills flex-column">
         <?php include('nav.php'); ?>
      </ul>
  </nav>
<!-- Center Column Content Section -->
  <div class="col-sm-8 text-center">
      <h2>Thank you for registering</h2>
      On the Home Page, you will now be able to login and add new quotes to the message board.
<!-- login does not yet work, nut will in the next chapter -->
  </div>
<!-- Right-side Column Content Section -->
      <aside class="col-sm-2">
           <?php include('info-col.php'); ?>
      </aside>
  </div>
<!-- Footer Content Section -->
     <footer class="jumbotron text-center row"
          style="padding-bottom:1px; padding-top:8px;">
```

```
        <?php include('footer.php'); ?>
    </footer>
</div>
</body>
</html>
```

As you can see, the page was created from the template with just a few changes. This keeps the consistency that should occur with a website. We will now look at what happens when invalid information is received at the server. We will use an array to display error messages.

Displaying Error Messages That Are Collected in an Array

If invalid information (empty values) are received in all the fields on the server, multiple errors will be created. The corresponding error messages are inserted into the errors array. These are then displayed so that the user is aware of the situation. It would be annoying if only the first error was shown and then, after clicking the Register button a second time, the second error was displayed, and so on.

Figure 2-10 shows an example of an error that would be displayed if this situation occurs.

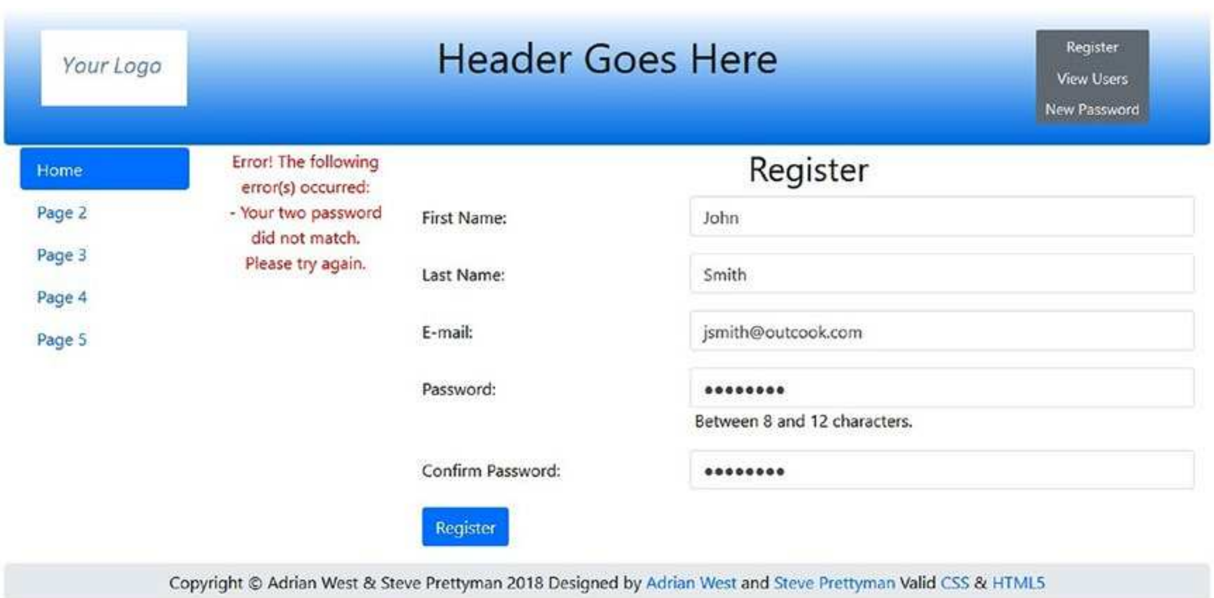

Figure 2-10. *The errors displayed if the user failed to enter any data*

The errors are displayed on the Registration page. As mentioned previously, the right column is removed when user errors occur to provide an area to display the messages. Errors might indicate that somehow the data was corrupted between the browser and the server. If the user enters incorrect information and hits the submit button, the HTML5 and JavaScript code will handle most of the validation situations and display error messages. If you want to test more server error messages, remove the required attributes from each of the HTML textboxes.

Hashing the Password

All passwords must be hashed to keep hackers from discovering them. Starting with PHP 5.5, the developers have created a new password hashing (encoding) method. This method is deemed safe and is one of the best ways to hash passwords. The format is as follows:

```
$hashedPassword = password_hash($password1, PASSWORD_DEFAULT);
```

In this example, $password1 is the unhashed version of the password. PASSWORD_DEFAULT is a PHP constant. As security changes, the PHP developers plan to keep the coding format the same and change the PHP constant to represent any new hashing schemes. The password_hash method will convert the password entered by the user into a secured hashed password that can then be saved in a database table. We will discover later that we must use the password_verify method to determine whether a user entered password will match a hashed password stored in the database.

Viewing Members' Records

When a user has registered, the website administrator can use phpMyAdmin to view the table and its entries. Let's suppose a user registered on April 26, 2018, and they entered the following information:

First name: Steve Last name: Johnson, Email: sjohnson@sjohnson.com, and Password: aaaaaaaa

phpMyAdmin will allow you to view each record in the user table. Access phpMyAdmin, select the simpledb database, click the users table, and then click the Browse tab. Figure 2-11 shows Steve Johnson's entries.

Figure 2-11. *Viewing a record in phpMyAdmin*

Note that the password aaaaaaaa is hashed as $2y$10$lEmRKPYfu/Nb6ECtbmp7YOuIZeZDYuCn ZKRmEBnQ6nRHDKJHdEgMK.

Often website designers and database administrators who do not have access to phpMyAdmin may also need to view a table of database entries created by all the registered users. The next example provides the information for doing this.

The View Users Page

When the View Users button is clicked, a table of registered users is displayed, as shown in Figure 2-12.

Figure 2-12. *A table of users is displayed*

The table is displayed when the user clicks the View Users button on the top-right menu. Listing 2-10 shows the code for displaying the table of users.

Listing 2-12. Displaying a Table of Registered Members on the Screen (register-view-users.php)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Template for an interactive web page</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <!-- Bootstrap CSS File -->
  <link rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
  integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4"
  crossorigin="anonymous">
</head>
<body>
<div class="container" style="margin-top:30px">
<!-- Header Section -->
<header class="jumbotron text-center row"
style="margin-bottom:2px; background:linear-gradient(white, #0073e6); padding:20px;">
  <?php include('header.php'); ?>
</header>
<!-- Body Section -->
  <div class="row" style="padding-left: 0px;">
<!-- Left-side Column Menu Section -->
  <nav class="col-sm-2">
      <ul class="nav nav-pills flex-column">
              <?php include('nav.php'); ?>
```

```
      </ul>
  </nav>
<!-- Center Column Content Section -->
  <div class="col-sm-8">
  <h2 class="text-center">These are the registered users</h2>
<p>
<?php
try {
// This script retrieves all the records from the users table.
require('mysqli_connect.php'); // Connect to the database.
// Make the query:
// Nothing passed from user safe query                                          #1
$query = "SELECT CONCAT(last_name, ', ', first_name) AS name, ";
$query .= "DATE_FORMAT(registration_date, '%M %d, %Y') AS ";
$query .= "regdat FROM users ORDER BY registration_date ASC";
$result = mysqli_query ($dbcon, $query); // Run the query.
if ($result) { // If it ran OK, display the records.
// Table header.                                                                #2
echo '<table class="table table-striped">
<tr><th scope="col">Name</th><th scope="col">Date Registered</th></tr>';
// Fetch and print all the records:                                             #3
while ($row = mysqli_fetch_array($result, MYSQLI_ASSOC)) {
echo '<tr><td>' . $row['name'] . '</td><td>' . $row['regdat'] . '</td></tr>'; }
    echo '</table>'; // Close the table so that it is ready for displaying.
    mysqli_free_result ($result); // Free up the resources.
} else { // If it did not run OK.
// Error message:
echo '<p class="error">The current users could not be retrieved. We apologize';
echo ' for any inconvenience.</p>';
// Debug message:
// echo '<p>' . mysqli_error($dbcon) . '<br><br>Query: ' . $q . '</p>';
exit;
} // End of if ($result)
mysqli_close($dbcon); // Close the database connection.
}
catch(Exception $e) // We finally handle any problems here
   {
     // print "An Exception occurred. Message: " . $e->getMessage();
     print "The system is busy please try later";
   }
   catch(Error $e)
   {
      //print "An Error occurred. Message: " . $e->getMessage();
      print "The system is busy please try again later.";
   }
?>

  </div>
<!-- Right-side Column Content Section -->
        <aside class="col-sm-2">
      <?php include('info-col.php'); ?>
```

```
        </aside>
  </div>
<!-- Footer Content Section -->
<footer class="jumbotron text-center row"
style="padding-bottom:1px; padding-top:8px;">
  <?php include('footer.php'); ?>
</footer>
</div>
</body>
</html>
```

Explanation of the Code

You will have seen most of the code before, but here is an explanation of the new items:

```
// Make the query:
// Nothing passed from user safe query                                              #1
$query = "SELECT CONCAT(last_name, ', ', first_name) AS name, ";
$query .= "DATE_FORMAT(registration_date, '%M %d, %Y') AS ";
$query .= "regdat FROM users ORDER BY registration_date ASC";
$result = mysqli_query ($dbcon, $query); // Run the query.
if ($result) { // If it ran OK, display the records.
```

The SQL query selects and strings together (concatenates) the last name, then a comma, then the first name, as well as selecting the registration data. It sets the temporary headings for these as name and regdat. It rearranges the registration dates in the format month-day-year. It uses the select query to extract the information from the users table. Finally, it requests that each row of information be displayed in ascending (ASC) date order (oldest first). To display the records with the latest registrations first (descending order), you use DESC instead of ASC. Since the query is not gathering any information from the user, we do not need to create a prepared statement, passing in variables as shown in the registration page. This query string cannot be hacked.

```
// Table header.                                                                    #2
echo '<table class="table table-striped">
<tr><th scope="col">Name</th><th scope="col">Date Registered</th></tr>';
```

This block of code displays (echoes) the table using the Bootstrap classes table and table-stripped to format the display. The column headers, Name and Date Registered, are also declared as the first row of the table displayed.

```
// Fetch and print all the records:                                                 #3
while ($row = mysqli_fetch_array($result, MYSQLI_ASSOC)) {
echo '<tr><td>' . $row['name'] . '</td><td>' . $row['regdat'] . '</td></tr>'; }
```

This block of code loops through the rows and displays them while row data from the database table is available. MYSQLI_ASSOC creates an associative array. A PHP associative array uses keys (words) for indexes. For example, $row['name'] contains the concatenated first name and last name pulled from the users table (see #1).

Users sometimes want to change their password. The next section demonstrates how this is done.

The Change Password Page

When the user clicks the New Password button on the menu at the top right of the header, the form shown in Figure 2-13 appears.

Figure 2-13. *The change password form*

The form is suitable only if the user knows their current password and e-mail address. If they have forgotten their password, a different approach is needed that will be discussed in a later chapter. However, this “new password” page is useful if the user does not like the password they originally chose. Listing 2-13a gives the HTML code for the change password form.

Listing 2-13a. Creating a Page to Allow Users to Change a Password (change-password.php)

```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Template for an interactive web page</title>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
 <!-- Bootstrap CSS File -->
 <link rel="stylesheet"
 href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css"
 integrity="sha384-9gVQ4dYFwwWSjIDZnLEWnxCjeSWFphJiwGPXr1jddIhOegiu1FwO5qRGvFXOdJZ4"
 crossorigin="anonymous">
<script src="verify.js"></script>
</head>
<body>
<div class="container" style="margin-top:30px">
<!-- Header Section -->
<header class="jumbotron text-center row col-sm-14"
style="margin-bottom:2px; background:linear-gradient(white, #0073e6); padding:20px;">
 <?php include('header.php'); ?>
```

```
</header>
<!-- Body Section -->
 <div class="row" style="padding-left: 0px;">
<!-- Left-side Column Menu Section -->
  <nav class="col-sm-2">
     <ul class="nav nav-pills flex-column">
        <?php include('nav.php'); ?>
      </ul>
  </nav>
 <!-- Validate Input -->
   <?php
  if ($_SERVER['REQUEST_METHOD'] == 'POST') {
   require('process-change-password.php');                                      //#1
  } // End of the main Submit conditional.
  ?>
<div class="col-sm-8">
<h2 class="h2 text-center">Change Password</h2>
<form action="change-password.php" method="post" name="regform"
 id="regform" onsubmit="return checked();">
<div class="form-group row">
    <label for="email" class="col-sm-4 col-form-label">E-mail:</label>
        <div class="col-sm-8">
        <input type="email" class="form-control" id="email" name="email"
        placeholder="E-mail" maxlength="60" required
        value="<?php if (isset($_POST['email'])) echo $_POST['email']; ?>">
        </div>
 </div>
 <div class="form-group row">
        <label for="password" class="col-sm-4 col-form-label">Current Password:</label>
        <div class="col-sm-8">
        <input type="password" class="form-control" id="password" name="password"
        placeholder="Password" minlength="8" maxlength="12"
        required value="<?php if (isset($_POST['password'])) echo $_POST['password']; ?>">
        </div>
 </div>
<div class="form-group row">
        <label for="password1" class="col-sm-4 col-form-label">New Password:</label>
        <div class="col-sm-8">
         <input type="password" class="form-control" id="password1" name="password1"
          placeholder="Password" minlength="8" maxlength="12"
          required value="<?php if (isset($_POST['password1'])) echo $_POST['password1']; ?>">
          <span id='message'>Between 8 and 12 characters.</span>
</div>
<div class="form-group row">
         <label for="password2" class="col-sm-4 col-form-label">Confirm Password:</label>
         <div class="col-sm-8">
                 <input type="password" class="form-control" id="password2" name="password2"
          placeholder="Confirm Password" minlength="8" maxlength="12" required
          value="<?php if (isset($_POST['password2'])) echo $_POST['password2']; ?>">
        </div>
 </div>
```

```
<div class="form-group row">
        <div class="col-sm-12">
        <input id="submit" class="btn btn-primary" type="submit" name="submit"
        value="Change Password">
        </div>
</div>
</form>
</div>
<!-- Right-side Column Content Section -->
<?php
 if(isset($errorstring)) {
         echo '<footer class="jumbotron text-center col-sm-12"
        style="padding-bottom:1px; padding-top:8px;">';
 }
 else
 {
        echo '<aside class="col-sm-2">';
        include('info-col.php');
        echo '</aside>';
        echo '</div>';
        echo '<footer class="jumbotron text-center row col-sm-14"
        style="padding-bottom:1px; padding-top:8px;">';
 }
  include('footer.php');
 ?>
</footer>
</div>
</body>
</html>
```

Explanation of the Code

This section gives an explanation of the code.

```
  if ($_SERVER['REQUEST_METHOD'] == 'POST') {
   require('process-change-password.php');                                    //#1
  } // End of the main Submit conditional.
```

The HTML for the change password page is similar to the HTML for the register page. The main difference is the PHP code is now imported from the process-change-password file. Let's take a look at the contents of this file in Listing 2-13b.

Listing 2-13b. Processing the Changed Password (process-change-password.php)

```
<?php
// This script is a query that UPDATES the password in the users table.
// Check that form has been submitted:
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
require ('mysqli_connect.php'); // Connect to the db.
$errors = array(); // Initialize an error array.
// Check for an email address:
```

```
$email = trim($_POST['email']);
if (empty($email)) {
        $errors[] = 'You forgot to enter your email address.';
}
// Check for a password and match against the confirmed password:
$password = trim($_POST['password']);
if (empty($password)) {
        $errors[] = 'You forgot to enter your old password.';
}
// Prepare and check new password                                               #1
$new_password = trim($_POST['password1']);
$verify_password = trim($_POST['password2']);
if (!empty($new_password)) {
        if (($new_password != $verify_password) ||
        ( $password == $new_password ))
        {
$errors[] = 'Your new password did not match the confirmed password and/or ';
$errors[] = 'Your old password is the same as your new password.';
        }
} else {
        $errors[] = 'You did not enter a new password.';
}if (empty($errors)) { // If everything's OK.
try {
 // Check that the user has entered the right email address/password combination:      #2
 $query = "SELECT userid, password FROM users WHERE ( email=? )";
$q = mysqli_stmt_init($dbcon);
mysqli_stmt_prepare($q, $query);
 // use prepared statement to ensure that only text is inserted
 // bind fields to SQL Statement
mysqli_stmt_bind_param($q, 's', $email);
 // execute query
 mysqli_stmt_execute($q);
$result = mysqli_stmt_get_result($q);
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
if ((mysqli_num_rows($result) == 1)                                            //#3
          && (password_verify($password, $row['password'])))
          {    // Found one record
         // Change the password in the database...
         // Hash password current 60 characters but can increase
         $hashed_passcode = password_hash($new_password, PASSWORD_DEFAULT);
         // Make the query:
         $query = "UPDATE users SET password=? WHERE email=?";
         $q = mysqli_stmt_init($dbcon);
         mysqli_stmt_prepare($q, $query);
         // use prepared statement to ensure that only text is inserted
         // bind fields to SQL Statement
         mysqli_stmt_bind_param($q, 'ss', $hashed_passcode, $email);
         // execute query
         mysqli_stmt_execute($q);
         if (mysqli_stmt_affected_rows($q) == 1) {   // one row updated             #4
                       // Thank you
```

```
                header ("location: password-thanks.php");
                            exit();
        } else { // If it did not run OK.                                                  #5
                // Public message:
                $errorstring = "System Error! <br /> You could not change password due ";
                $errorstring .= "to a system error. We apologize for any inconvenience.</p>";
                echo "<p class='text-center col-sm-2' style='color:red'>$errorstring</p>";
                // Debugging message below do not use in production
                //echo '<p>' . mysqli_error($dbcon) . '<br><br>Query: ' . $query . '</p>';
                // include footer then close program to stop execution
                echo '<footer class="jumbotron text-center col-sm-12"
                        style="padding-bottom:1px; padding-top:8px;">
                        include("footer.php");
                        </footer>';
                exit();
        }
         } else { // Invalid email address/password combination.
                $errorstring = 'Error! <br /> ';
                        $errorstring .= 'The email address and/or password do not match those
                        on file.';
                $errorstring .= " Please try again.";
                echo "<p class='text-center col-sm-2' style='color:red'>$errorstring</p>";
} }
   catch(Exception $e) // We finally handle any problems here
   {
     // print "An Exception occurred. Message: " . $e->getMessage();
     print "The system is busy please try later";
   }
   catch(Error $e)
   {
      //print "An Error occurred. Message: " . $e->getMessage();
      print "The system is busy please try again later.";
   }
   } else { // Report the errors.                                                         #6
        //header ("location: register-page.php");
        $errorstring = "Error! The following error(s) occurred:<br>";
        foreach ($errors as $msg) { // Print each error.
                $errorstring .= " - $msg<br>\n";
        }
        $errorstring .= "Please try again.<br>";
        echo "<p class=' text-center col-sm-2' style='color:red'>$errorstring</p>";
        }// End of if (empty($errors)) IF.
} // End of the main Submit conditional.
?>
```

Explanation of the Code

This section explains the code.

```
// Prepare and check new password                                                         #1
$new_password = trim($_POST['password1']);
```

```
$verify_password = trim($_POST['password2']);
if (!empty($new_password)) {
        if (($new_password != $verify_password) ||
        ( $password == $new_password ))
        {
$errors[] = 'Your new password did not match the confirmed password and/or ';
$errors[] = 'Your old password is the same as your new password.';
       }
} else {
       $errors[] = 'You did not enter a new password.';
```

The first two lines of code trim any blank spaces from the strings. In later chapters, we will verify that the password is in a proper format. However, for now, we are using prepared statements; thus, any coding the user attempts to insert into the textboxes will not be executed. If the two new passwords match, the new password is placed in $password. If they do not match, an error is sent back to the web page.

```
(
// Check that the user has entered the right email address/password combination:       #2
 $query = "SELECT userid, password FROM users WHERE ( email=? )";
$q = mysqli_stmt_init($dbcon);
mysqli_stmt_prepare($q, $query);
 // use prepared statement to ensure that only text is inserted
 // bind fields to SQL Statement
mysqli_stmt_bind_param($q, 's', $email);
 // execute query
 mysqli_stmt_execute($q);
$result = mysqli_stmt_get_result($q);
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
```

If all verifications are successful, we must make sure that the user provided an existing e-mail and password. The #2 line in the previous code creates a prepared select statement that pulls the user IS and password from the users table using the e-mail provided. If the SQL statement returns a result, we have verified that the e-mail exists in the database.

```
if ((mysqli_num_rows($result) == 1)                                                  //#3
          && (password_verify($password, $row['password'])))
          {    // Found one record
        // Change the password in the database...
         // Hash password current 60 characters but can increase
         $hashed_passcode = password_hash($new_password, PASSWORD_DEFAULT);
        // Make the query:
        $query = "UPDATE users SET password=? WHERE email=?";
        $q = mysqli_stmt_init($dbcon);
        mysqli_stmt_prepare($q, $query);
        // use prepared statement to ensure that only text is inserted
        // bind fields to SQL Statement
        mysqli_stmt_bind_param($q, 'ss', $hashed_passcode, $email);
        // execute query
        mysqli_stmt_execute($q);
```

If we have gotten a result from the database table, we also need to verify that the password is correct. The if statement uses the function password_verify to compare the old password with the password in the table. This is necessary because the password stored in the table is hashed and the password provided by the user is not hashed. If they match, the new password is hashed. A prepared SQL update query is created to change the password within the users table. This query is then executed.

```
if (mysqli_stmt_affected_rows($q) == 1) {    // one row updated                                #4
                        // Thank you
                        header ("location: password-thanks.php");
                                 exit();
```

If the SQL update query is successful, the “thank you” page is displayed. The header function calls the password thanks page using an HTML Get command. The current page is closed using the exit method.

```
} else { // If it did not run OK.                                                               #5
               // Public message:
               $errorstring = "System Error! <br /> You could not change password due ";
               $errorstring .= "to a system error. We apologize for any inconvenience.</p>";
               echo "<p class='text-center col-sm-2' style='color:red'>$errorstring</p>";
               // Debugging message below do not use in production
               //echo '<p>' . mysqli_error($dbcon) . '<br><br>Query: ' . $query . '</p>';
               // include footer then close program to stop execution
               echo '<footer class="jumbotron text-center col-sm-12"
                       style="padding-bottom:1px; padding-top:8px;">
                       include("footer.php");
                       </footer>';
               exit();
```

If no rows were changed, the query did not execute successfully. The else statement captures this situation and displays a System Error statement. Since this is a serious error, the page is provided with a footer and the code is closed (exit()). This will close any access to the database and will not let any other code execute. Debugging code that will provide more information on the problem is listed in comments. This code should never be activated in live code. The error information provided might display code and database connection information, which would be a major security violation.

```
} else { // Report the errors.                                                                  #6
        //header ("location: register-page.php");
        $errorstring = "Error! The following error(s) occurred:<br>";
        foreach ($errors as $msg) { // Print each error.
                $errorstring .= " - $msg<br>\n";
        }
        $errorstring .= "Please try again.<br>";
        echo "<p class=' text-center col-sm-2' style='color:red'>$errorstring</p>";
        }// End of if (empty($errors)) IF.
```

The last section of code is similar to the registration page. It displays any error messages caused when the user entered invalid information. These errors are provided on the same page as the form to allow the user to make corrections and resubmit the information.

Confirming a Successful Password Change

If the password change is successful, the page shown in Figure 2-14 is displayed.

The page shown in Figure 2-14 is similar to the registration "thank you" page. The code can be viewed from your downloaded files for this chapter.

Figure 2-14. *The password has been changed*

Testing the Tutorial's Pages

To see the interactive pages working, double-click the XAMPP or easyPHP icon on your desktop. When the control panel appears, check that Apache and MySQL/MariaDB are running.

Type `http://localhost/simpledb/index.php` into the address field of a browser and then click the Register button so that you can enter some users.

To save you time and the effort of dreaming up fictitious people, use the suggestions provided in Table 2-1.

Table 2-1. *Suggestions for Entering Members' Details*

Name	E-mail	Password
Mike Rosoft	miker@myisp.com	W1llgat3s
Olive Branch	obranch@myisp.com.uk	Ano1ly0n3
Frank Insence	finsence@myisp.net	P3rfum300
Annie Versary	aversary@myisp.com	B1rthd3yg1rl
Terry Fide	tfide@myisp.de	Scar3dst1ff
Rose Bush	rbush@myisp.co.uk	R3dbl00ms

Now that you have some more data in your users table, run XAMPP or EasyPHP and use a browser to click the menu item to display the table of registered users.

■ **Tip** No doubt you will encounter error messages as you create code and test it. For help, refer to the troubleshooting tips in Chapter 12.

Earlier in this chapter, we promised more information on the use of PHP arrays. The next section will help you understand additional aspects of this most useful feature.

More About Arrays

Arrays are used throughout this book, and if you access forums for advice on the use of PHP, you will be confronted with many arrays.

An array is a holding place in memory for multiple similar items (such as cereals shown next). The items are grouped together with the same variable name ($cereals). To refer to an individual element in an array, we must use a key (subscript). In PHP, the subscript can be a number ($cereals[1]) or a string (such as $row['name']). We will be using both formats throughout the book.

Arrays can be populated with elements in several ways. We create an array named $cereals, as follows:

```
<?php
$cereals = array();
?>
```

A few elements can be inserted into the array as follows (note that the first array element is usually zero):

```
<?php
$cereals = array[];
$cereals[0] = "oats";
$cereals[1] = "barley";
$cereals[2] = "corn";
$cereals[3] = "wheat";
?>
```

To display the elements of this array, insert the following code just before the closing tag ?>:

```
echo "$cereals[0] " . "$cereals[1]  " . "$cereals[2]  " .  "$cereals[3]";?>
```

The display will produce the following:
Oats barley corn wheat
The echo statement concatenates the string together using the concatenation symbol (.). However, in PHP we can also place program variables (such as $name) within quotations, as shown here:

```
$first_name = "Fred";
$middle_name = "Adam";
$last_name = "Smith";

echo "$first_name $middle_name $last_name";
```

PHP will also allow us to insert items into a numeric array without providing the index number.

```
$error[] = "One error";
$error[] = "Another error";
```

PHP will automatically index the first item with 0 and the second with 1. Thus, we can display the values as follows:

```
echo "$error[0] $error[1]";
```

There is much more to learn about arrays. However, just this small amount of knowledge should help you understand how we are using arrays when pulling information from a database table and when we are loading and displaying error messages.

■ **Caution** The data entered into the database tables in this chapter are not completely filtered. They have been deliberately stripped of most filters and cleaners so that the essential processes are uncluttered and clearly visible. An increasing number of security and filtering features will be added in later chapters.

Summary

In this chapter, you created your first interactive pages and tested them. You learned a little more about PHP—in particular, you should have grasped the idea of looking for and recognizing logical patterns in the code. You learned how to use several PHP functions: include(), required(), and echo(). You also learned many mysqli functions. You discovered the importance of PHP conditionals for creating interactive database pages. You also learned how to check whether the user has entered information before sending the data to the database table. In the next chapter, we will add some extra security and demonstrate how users and an administrator can log in and log out of private pages on a website. You will also learn how to remove or replace the redundant links in the header menus.