**Building the Back End**

Before we can start working on our storefront Web pages, we need to work on the *back-end applications*. This is the common term for Web programs that control the behind-the-scenes operations involved in providing data for the main Web application.

Your customers can't start ordering products unless you do some work entering and arranging new products in your database. All of these things happen in the back-end application.

We'll work on five elements to the back-end application in this course:

* **Password protection**—ensures that only authorized managers can access the back-end application.
* **Category management**—allows a manager to create the layout for the store products.
* **Product management**—provides an interface so the store manager can easily create and modify product information.
* **Order processing**—lets the store manager view and process customer orders.
* **Generating reports**—generates management reports showing the store activity.

We'll only focus on the first three elements of the back-end application in this lesson. We'll save processing customer orders and generating reports for a later lesson after we have some customer orders to process.

**The Back-End Layout**

Since my goal for this course is to teach you new and exciting PHP and MySQL things, we won't spend a lot of time burying ourselves in HTML and CSS code to create the course project. We'll obviously need to use some to make our site functional, but it won't be quite as exotic as some online retail stores you've seen. Once you have the basic store template, you'll be able to spruce it up to your own liking after you finish the course.

The core of our back-end application is the *admin.php* program. It creates a standard Web page format for the application by using an HTML table to divide the Web page into five sections that remain throughout the application.

* A *header* area that contains a standard banner for each Web page.
* A *footer* area that contains contact information that appears on each Web page.
* A *navigation* area that contains links to quickly traverse the Web site.
* A *status* area that provides real-time information about the store at a glance.
* The *main* area where individual Web page content appears.

If you took the *Introduction to PHP and MySQL* course with me, you'll recognize most of this code. If you didn't take the first course, or you need a refresher on what we did in that course, don't worry. In the Supplementary Materials section, you'll find a link to a complete explanation of what each code sections does, along with an explanation of how to use a CSS style sheet in the application.

The application code uses the PHP *include()* function to include PHP code files to provide unique content for each section. Here's how to create the admin.php file:

1. Create a folder called *store* under the www folder in the c:\wamp folder.
2. Create another folder called *admin* under the newly created store folder.
3. Create a text file called *admin.php* in the admin folder, and enter the following code:

<?php

session\_start();

?>

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<link rel="stylesheet" type="text/css" href="mystyle.css" />

<title>The Food Store - Admin Console</title>

</head>

<?php

include("/mylibrary/login.php");

login();

?>

<body>

<table width="100%" border="0">

<tr>

<td id="header" height="90" colspan="3">

<?php include("header.inc.php"); ?></td>

</tr>

<tr>

<td id="nav" width="20%" valign="top">

<?php include("adminnav.inc.php"); ?></td>

<td id="main" width="50%" valign="top">

<?php

if (!isset($\_REQUEST['content']))

{

if (!isset($\_SESSION['store\_admin']))

include("adminlogin.html");

else

include("adminmain.inc.php");

}

else

{

$content = $\_REQUEST['content'];

$nextpage = $content . ".inc.php";

include($nextpage);

} ?></td>

<td id="status" width="30%" valign="top">

<?php include("adminstatus.inc.php"); ?></td>

</tr>

<tr>

<td id="footer" colspan="3">

<div align="center">

<?php include("footer.inc.php"); ?>

</div></td>

</tr>

</table>

</body>

</html>

1. Save the file (remember to use double quotes around the filename if you're using Microsoft Notepad).

The admin.php code starts by using the PHP *session\_start()* function. This function ensures that a session cookie will be started for each client session. This is important, as that's how we identify the administrator after he or she logs in.

After the standard HTML stuff, there's an interesting PHP section:

include("/mylibrary/login.php");

login();

The code uses another include() function to incorporate PHP code from another file located in a folder called *mylibary*. We'll be writing a few functions for our application, and this is the first one. Each function will be stored in the folder mylibary, under the store folder.

Following that, we call a function—*login()*. This isn't a standard PHP function. It's one that we create in the login.php code. This function will handle the work of logging into the MySQL server and connecting to a default database for us.

Since all of the application code is contained within the admin.php file (via the include() statements), we only need to log into the MySQL database server once. We can then use that one connection throughout the application.

**Hiding Connect Information**

One trick that's becoming more popular in MySQL programming is using the include() function to hide files that contain sensitive information. The most sensitive information you have in your application is the login information to your MySQL database server, contained in the *mysql\_connect()* function. If hackers got a hold of this file, they'd know the user name and password for your MySQL database. That could be potentially bad.

To prevent that problem, we'll create a separate PHP function called login() and save it in an alternative location. To keep things simple, I'm just including it in the mylibrary folder under our store folder. In real practice, it would be best to place it in a folder out of the Apache Web server's path so it's not accessible via the Web.

Here's how to create the login() function:

1. Create a folder called *mylibrary* under the store folder.
2. Use Notepad to create a new file called *login.php*.
3. In the new file, enter the following code:

<?php

function login()

{

$con = mysql\_connect("localhost", "test", "test") or die('Could not connect to server');

mysql\_select\_db("store", $con) or die('Could not connect to database');

}

?>

1. Save the file, and exit Notepad.

The login() function uses the standard mysql\_connect() and mysql\_select\_db() functions to connect to our store database. When you use the login() function in the admin.php file, it establishes a connection to the MySQL database server. This connection stays active throughout the life of the Web page. This feature is called *persistent*. You can process multiple mysql\_query() functions on a single mysql\_connect() connection. Since the code defines the login() function at the top of the admin.php file, all SQL queries for any file included in the application file will use this database connection.

This is a great performance boost! Instead of making multiple database connections throughout the application page, it'll just use a single connection. When the Web page finishes loading, PHP closes the connection so it isn't valid across Web pages (it isn't persistent between Web pages).

That's enough for this chapter. In Chapter 3, we'll look at the login Web pages required to allow our store managers to log into the back-end application.