**Creating Product Categories**

Before your manager can enter new products in the store, you'll need to create Web pages that allow your store managers to create new categories to classify the products. Your Food Store application uses the standard HTML form for this. This method of allowing Web page visitors to post data has been around for as long as the World Wide Web.

Once a Web site visitor fills in the form and clicks a Submit button, the HTML code sends the data to another page for processing. The PHP language provides three methods for extracting data from HTML forms:

* The $\_GET[] array variable for forms using the HTML GET method.
* The $\_POST[] array variable for forms using the HTML POST method.
* The $\_REQUEST[] array variable for forms using either method.

When the PHP code retrieves the new category name from the HTML form, it'll need to insert it into the *categories* table.

To get to the HTML form Web page, you have a link in the navigation area (the *Add New Category* link). It uses the URL *admin.php?content=newcat*, which points to the admin.php file, using a content value of *newcat*.

The *newcat.inc.php* creates a simple HTML form that allows the manager to enter the name of a new category. To create the code file, just follow these steps:

1. Create a file called *newcat.inc.php* in the *admin* folder under the *store* folder in your WampServer *www* folder area.
2. Enter the following code into the new file:

<?php

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

{

echo "<h2>Sorry, you have not logged into the system</h2>\n";

echo "<a href=\"admin.php\">Please login</a>\n";

} else

{

echo "<h2>Add a new food category</h2>\n";

echo "<table width=\"100%\" cellpadding=\"1\" border=\"1\">\n";

echo "<form action=\"admin.php\" method=\"post\">\n";

echo "<tr><td>New category</td><td><input type=\"text\" name=\"catname\" size=\"40\"></td></tr>\n";

echo "</table>\n";

echo "<input type=\"hidden\" name=\"content\" value=\"addcat\">\n";

echo "<input type=\"submit\" value=\"Submit\">\n";

echo "</form>\n";

}

?>

1. Save the file and exit the editor.

First, the code checks to see if the manager is really logged into the system by checking for the *store\_admin* session cookie (you wouldn't want anyone bypassing your security system). Since you need to use PHP code to check for the session cookie, you must stay in PHP mode to create the HTML form. You need to use PHP *echo* statements to produce the required HTML code for this.

If the manager is logged in, the PHP code produces a simple HTML form by echoing the required HTML code to the client's browser. This code file also demonstrates how you use an HTML table to make your HTML forms look a little more organized. The table has two columns. One contains a text label for the data entry, and the other contains the data entry textbox.

The code uses a simple textbox to input the name of the new category. The action attribute for the form points to the admin.php file (the main administration page). It also uses a hidden input field to pass along a value for the *content* HTML variable, which the admin.php code checks and uses for the next Web page. The hidden value points to the *addcat* PHP file, which will do all the work of inserting the new category into the database table.

The Add Category Web page

When the manager clicks the Submit button in the form, the addcat.inc.php include file receives the form information and must process it. Here's where things get interesting.

**Processing Input Data**

The PHP code in the addcat.inc.php file must retrieve the data that the form passes to it and then insert the new data into the categories table to create the new category. It sounds simple, but there's a catch. The first thing to do is retrieve the data that the form passes using the *catname* form data field:

$catname = $\_POST['catname'];

This should look familiar to you. The challenge comes when you try to put the data into the MySQL table, because you don't really know what type of information the manager put in the form.

It's not that you don't trust the manager. You just don't know what the manager might innocently try to do. For example, look what happens if you're the manager and you try to insert the category name *Produce - farmer's market* using the INSERT SQL statement:

mysql>INSERT INTO categories (name) VALUES ('Produce - farmer's market');

'>

Oops, something went wrong. The MySQL console is looking for more data. Notice that the single quote you used in the data confused MySQL into thinking that the data string ended. When it saw the single quote at the end of the string, it thought you were starting another string. When you hit the ENTER key, it gave you another prompt to finish the string.

MySQL SQL statements are very picky about the format of the data. Certain characters—such as single and double quotes, backslashes, and the NUL character—cause problems in SQL statements. To solve this, you must use the MySQL *escape character* (the backslash) to *escape* (identify) these potentially dangerous characters:

mysql>INSERT INTO categories (name) VALUES ('Produce - farmer\'s market');

Query OK, 1 row affected (0.03 sec)

mysql>

That's much better. It's your job to manage and control the data used in your program. You always need to be on the lookout for characters that could break (either intentionally or by accident) your SQL statements. Fortunately, PHP provides a few tools to help you with this task.

**Escaping MySQL Data in PHP**

The PHP configuration file includes a controversial feature that automatically adds the escape character to all form data characters that can cause trouble in your SQL statements. The *magic\_quotes\_gpc* configuration value determines if PHP automatically adds the backslash to data received via the GET, POST, or cookie methods. If this feature is set, you don't have to worry about using the form data directly in an SQL statement.

This feature is controversial because you don't always need to have it turned on. The only time you need it is when you're pushing data into a MySQL table. For all other uses, it needlessly adds the backslash escape characters.

Because of this, there's no guarantee that your ISP has this feature turned on. This makes things somewhat confusing. You may or may not have to add the escape characters yourself in any form data, or conversely, you may or may not have to remove them if you just want to display the form data. It's up to your program to determine when you need to add the MySQL escape character or not.

This is quite a dilemma, but it has a simple solution. Follow these steps to create the *addcat.inc.php* code file to solve the issue.

1. Create the file *addcat.inc.php* in the *admin* folder of your store application.
2. Enter the following code into the new file:

<?php

$catname = $\_POST['catname'];

if (get\_magic\_quotes\_gpc())

{

$catname = stripslashes($catname);

}

$catnameval = mysql\_real\_escape\_string($catname);

$query="INSERT INTO categories (name) VALUES ('$catnameval')";

$result = mysql\_query($query);

if ($result)

echo "<h2>New category '$catname' added</h2>\n";

else

echo "<h2>Sorry, unable to add new category</h2>\n";

?>

1. Save the file and exit.

The *get\_magic\_quotes\_gpc()* function checks to see if the magic\_quotes\_gpc PHP setting is turned on for the server. If it's on, the code uses the *stripshlashes()* PHP function to actually remove any backslashes that the magic\_quotes\_gpc feature inserted into the data. This ensures that the data doesn't contain escape characters. So it can be used as-is in the application (except not for SQL queries). While this technique sounds backward, there's a reason for it.

The official PHP way to prepare data for MySQL queries is to use the *mysql\_real\_escape\_string()* function. This function guarantees that the data is properly escaped and will work properly in your SQL statement. To use this feature, though, you have to be sure that the magic\_quotes\_gpc feature in PHP hasn't already manipulated the original data. If you determine that this feature is enabled, you must include the stripslashes() to remove the slashes added by PHP.

This is the standard technique for handling form data in PHP. Using this technique, you have access to both the clean version of the data (called *$catname* in the program) and the escaped version (called *$catnameval*). You can then use whichever version you need for your purposes. In the addcat.inc.php code, I used both versions to show you what I'm talking about.

If you add a category that includes a single quote, you'll see the single quote just fine in the Web page display, plus it'll get added to the categories table just fine. It's the best of both worlds.

In the next chapter, we'll walk through the code you need to add a new product.