CHAPTER 22

Creating an Online Storefront

In this chapter, you learn the following:

- **Create relational tables for an online store**
- **Create scripts to display store categories**
- **Create scripts to display individual items**

In this short chapter, you create a generic online storefront. As with the previous project chapters, you learn how to create the relevant database tables as well as the scripts for displaying the information to the user. The examples used in this chapter represent one of an infinite number of possibilities to complete these tasks and are meant to provide a foundation of knowledge rather than a definitive method for completing this task.

Planning and Creating the Database Tables

Before you tackle the process of creating database tables for an online store, think about the real-life shopping process. When you walk into a store, items are ordered in some fashion: The hardware and the baby clothes aren't mixed together, the electronics and the laundry detergent aren't side by side, and so on. Applying that knowledge to database normalization, already you can see that you need a table to hold categories and a table to hold items. In this simple store, each item belongs to one category.

Next, think about the items themselves. Depending on the type of store you have, your items might or might not have colors, and might or might not have sizes. But all your items will have a name, a description, and a price. Again, thinking in terms of normalization, you can imagine that you might have one general items table and two additional tables that relate to the general items table.

Table 22.1 shows sample table and field names to use for your online storefront. In a minute, you create the actual SQL statements, but first you should look at this information and try to see the relationships. Ask yourself which of the fields should be primary or unique keys.

TABLE 22.1 Storefront Table and Field Names

Table Name	Field Names
`store_categories`	`id, cat_title, cat_desc`
`store_items`	`id, cat_id, item_title, item_price, item_desc, item_image`
`store_item_size`	`item_id, item_size`
`store_item_color`	`item_id, item_color`

As you can see in the following SQL statements, the `store_categories` table has two fields besides the `id` field: `cat_title` and `cat_desc`, for title and description, respectively. The `id` field is the primary key, and `cat_title` is a unique field because there's no reason you would have two identical categories:

```
CREATE TABLE store_categories (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    cat_title VARCHAR (50) UNIQUE,
    cat_desc TEXT
);
```

Next we tackle the `store_items` table, which has five fields besides the `id` field—none of which are unique keys. The lengths specified in the field definitions are arbitrary; you should use whatever best fits your store.

The `cat_id` field relates the item to a particular category in the `store_categories` table. This field is not unique because you will want more than one item in each category. The `item_title`, `item_price`, and `item_desc` (for description) fields are self-explanatory. The `item_image` field holds a filename (in this case, the file is assumed to be local to your server) that you use to build an HTML <img> tag when it is time to display your item information:

```
CREATE TABLE store_items (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    cat_id INT NOT NULL,
    item_title VARCHAR (75),
    item_price FLOAT (8,2),
    item_desc TEXT,
    item_image VARCHAR (50)
);
```

Both the store_item_size and store_item_color tables contain optional information: If you sell books, they won't have sizes or colors, but if you sell shirts, they will. For each of these tables, the item_id and item_size fields are not unique keys because you can associate as many colors and sizes with a particular item as you want:

```
CREATE TABLE store_item_size (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    item_id INT NOT NULL,
    item_size VARCHAR (25)
);
CREATE TABLE store_item_color (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    item_id INT NOT NULL,
    item_color VARCHAR (25)
);
```

These are all the tables necessary for a basic storefront—that is, for displaying the items you have for sale. Chapter 23, "Creating a Shopping Cart Mechanism," integrates the user experience into the mix. For now, just concentrate on your inventory.

In Chapter 20, "Creating an Online Address Book," you learned how to use PHP forms and scripts to add or delete records in your tables. If you apply the same principles to this set of tables, you can easily create an administrative front end to your storefront. We do not go through that process in this book, but feel free to do it on your own. (If you understood what was going on in Chapter 20, you know enough about PHP and MySQL to complete the tasks.)

For now, you can simply issue MySQL queries, via the MySQL monitor or other interface, to add information to your tables. Following are some examples, if you want to follow along with sample data.

Inserting Records into the store_categories Table

The following queries create three categories in your store_categories table: hats, shirts, and books.

```
INSERT INTO store_categories VALUES
 ('1', 'Hats', 'Funky hats in all shapes and sizes!');

INSERT INTO store_categories VALUES ('2', 'Shirts', 'From t-shirts to
sweatshirts to polo shirts and beyond.');

INSERT INTO store_categories VALUES ('3', 'Books', 'Paperback, hardback,
books for school or play.');
```

In the next section, we add some items to the categories.

Inserting Records into the `store_items` Table

The following queries add three item records to each category. Feel free to add many more.

```
INSERT INTO store_items VALUES ('1', '1', 'Baseball Hat', '12.00',
'Fancy, low-profile baseball hat.', 'baseballhat.gif');

INSERT INTO store_items VALUES ('2', '1', 'Cowboy Hat', '52.00',
'10 gallon variety', 'cowboyhat.gif');

INSERT INTO store_items VALUES ('3', '1', 'Top Hat', '102.00',
'Good for costumes.', 'tophat.gif');

INSERT INTO store_items VALUES ('4', '2', 'Short-Sleeved T-Shirt',
'12.00', '100% cotton, pre-shrunk.', 'sstshirt.gif');

INSERT INTO store_items VALUES ('5', '2', 'Long-Sleeved T-Shirt',
'15.00', 'Just like the short-sleeved shirt, with longer sleeves.',
'lstshirt.gif');

INSERT INTO store_items VALUES ('6', '2', 'Sweatshirt', '22.00',
'Heavy and warm.', 'sweatshirt.gif');

INSERT INTO store_items VALUES ('7', '3', 'Jane\'s Self-Help Book',
'12.00', 'Jane gives advice.', 'selfhelpbook.gif');

INSERT INTO store_items VALUES ('8', '3', 'Generic Academic Book',
'35.00', 'Some required reading for school, will put you to sleep.',
'boringbook.gif');

INSERT INTO store_items VALUES ('9', '3', 'Chicago Manual of Style',
'9.99', 'Good for copywriters.', 'chicagostyle.gif');
```

NOTE

The preceding queries refer to various graphics, such as `baseballhat.gif`, which are not included in the code. You can find sample images or make some placeholder graphics of your own.

Inserting Records into the `store_item_size` Table

The following queries associate sizes with one of the three items in the `shirts` category and a generic "one size fits all" size to each of the hats (assume that they're strange hats). On your own, insert the same set of size associations for the remaining items in the `shirts` category:

```
INSERT INTO store_item_size (item_id, item_size) VALUES (1,'One Size Fits All');
INSERT INTO store_item_size (item_id, item_size) VALUES (2,'One Size Fits All');
INSERT INTO store_item_size (item_id, item_size) VALUES (3,'One Size Fits All');
INSERT INTO store_item_size (item_id, item_size) VALUES (4,'S');
INSERT INTO store_item_size (item_id, item_size) VALUES (4,'M');
INSERT INTO store_item_size (item_id, item_size) VALUES (4,'L');
INSERT INTO store_item_size (item_id, item_size) VALUES (4,'XL');
```

Inserting Records into the `store_item_color` Table

The following queries associate colors with one of the three items in the `shirts` category. On your own, insert color records for the remaining shirts and hats.

```
INSERT INTO store_item_color (item_id, item_color) VALUES (1,'red');
INSERT INTO store_item_color (item_id, item_color) VALUES (1,'black');
INSERT INTO store_item_color (item_id, item_color) VALUES (1,'blue');
```

Displaying Categories of Items

Believe it or not, the most difficult task in this project is now complete. Compared to thinking up categories and items, creating the scripts used to display the information is easy!

The first script you make is one that lists categories and items. Obviously, you do not want to list all categories and all items all at once as soon as the user walks in the door, but you do want to give the user the option of immediately picking a category, seeing its items, and then picking another category. In other words, this script serves two purposes: It shows the categories, and then if a user clicks a category link, it shows the items in that category.

Listing 22.1 shows the full code for `seestore.php`. If you have worked through this book sequentially, you will notice a lot of the same basic construction you saw in previous chapters; as I mentioned earlier in this book, these projects are all examples of foundational CRUD (create, read, update, delete) applications. Even so, the code is still explained in detail after the listing.

LISTING 22.1 Script to View Categories

```
<?php
//connect to database
$mysqli = mysqli_connect("localhost", "joeuser", "somepass", "testDB");

$display_block = "<h1>My Categories</h1>
<p>Select a category to see its items.</p>";

//show categories first
$get_cats_sql = "SELECT id, cat_title, cat_desc FROM
                 store_categories ORDER BY cat_title";
$get_cats_res =  mysqli_query($mysqli, $get_cats_sql)
                 or die(mysqli_error($mysqli));

if (mysqli_num_rows($get_cats_res) < 1) {
     $display_block = "<p><em>Sorry, no categories to browse.</em></p>";
} else {
    while ($cats = mysqli_fetch_array($get_cats_res)) {
        $cat_id  = $cats['id'];
        $cat_title = strtoupper(stripslashes($cats['cat_title']));
```

LISTING 22.1 Continued

```
        $cat_desc = stripslashes($cats['cat_desc']);

        $display_block .= "<p><strong><a href=\"".$_SERVER['PHP_SELF'].
        "?cat_id=".$cat_id."\">".$cat_title."</a></strong><br/>"
        .$cat_desc."</p>";

        if (isset($_GET['cat_id']) && ($_GET['cat_id'] == $cat_id)) {
           //create safe value for use
           $safe_cat_id = mysqli_real_escape_string($mysqli,
                 $_GET['cat_id']);

           //get items
           $get_items_sql = "SELECT id, item_title, item_price
                             FROM store_items WHERE
                             cat_id = '".$cat_id."' ORDER BY item_title";
           $get_items_res = mysqli_query($mysqli, $get_items_sql)
                            or die(mysqli_error($mysqli));

           if (mysqli_num_rows($get_items_res) < 1) {
                 $display_block = "<p><em>Sorry, no items in this
                 category.</em></p>";
           } else {
                 $display_block .= "<ul>";
                 while ($items = mysqli_fetch_array($get_items_res)) {
                     $item_id  = $items['id'];
                     $item_title = stripslashes($items['item_title']);
                     $item_price = $items['item_price'];

                     $display_block .= "<li><a
                       href=\"showitem.php?item_id=".
                     $item_id."\">".$item_title."</a>
                     (\$".$item_price.")</li>";
                 }

                 $display_block .= "</ul>";
            }
            //free results
            mysqli_free_result($get_items_res);
        }
     }
  }
}
//free results
mysqli_free_result($get_cats_res);
//close connection to MySQL
mysqli_close($mysqli);
?>
<!DOCTYPE html>
<html>
<head>
```

```
69: <title>My Categories</title>
70: </head>
71: <body>
72: <?php echo $display_block; ?>
73: </body>
74: </html>
```

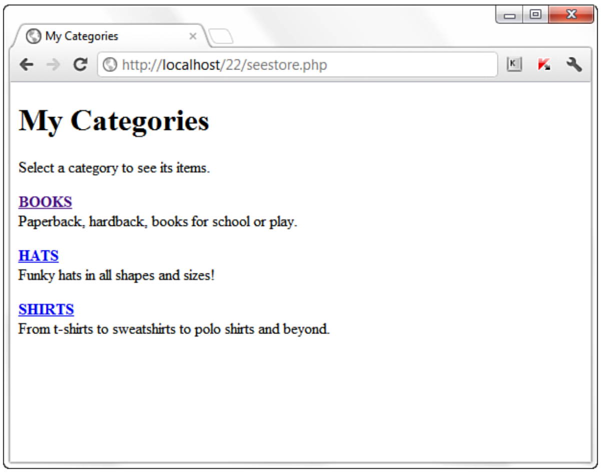
Given the length of scripts you saw in Chapter 20, these 74 fully functional lines should be a welcome change. Line 3 opens the database connection because regardless of which action the script is taking—showing categories or showing items in categories—the database is necessary.

Line 5 starts the `$display_block` string, with some basic page title information added to it. Lines 9–12 create and issue the query to retrieve the category information. Line 14 checks for categories; if none are in the table, a message is stored in the `$display` block variable for display to the user, and that's all this script does. (It jumps to the HTML in line 66 and prints to the screen after freeing up some database results.) However, if categories are found, the script moves on to line 17, which begins a `while` loop to extract the information.

In the `while` loop, lines 18–20 retrieve the ID, title, and description of the category. String operations are performed to ensure that no slashes are in the text and that the category title is in uppercase for display purposes. Lines 22–24 place the category information, including a self-referential page link, in the `$display_block` string. If a user clicks the link displayed by that string, she returns to this same script, except with a category ID passed in the query string. The script checks for this value in line 26.

If a `$_GET['cat_id']` value has been passed to the script (and has been verified as a valid ID) because the user clicked on a category link in hopes of seeing listed items, the script builds and issues another query using a safe version of that value (lines 32–36) to retrieve the items in the category. Lines 38–51 check for items and then build an item string as part of `$display_block`. Part of the information in the string is a link to a script called `showitem.php`, which you create in the next section.

After reaching that point, the script has nothing left to do besides free up some resources, and it prints the HTML and value of `$display_block`. Figure 22.1 shows the outcome of the script when accessed directly; only the category information shows.



My Categories

http://localhost/22/seestore.php

My Categories

Select a category to see its items.

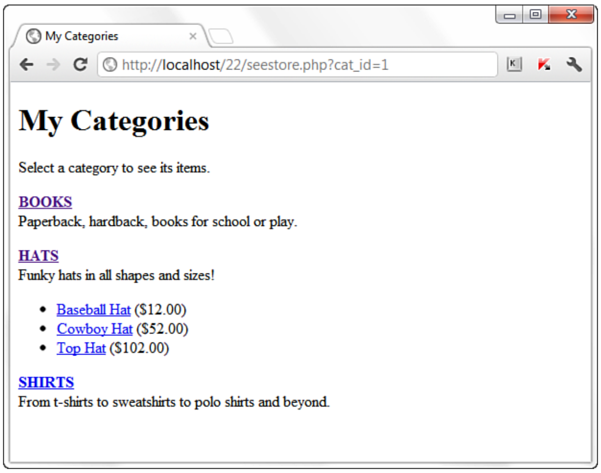
BOOKS
Paperback, hardback, books for school or play.

HATS
Funky hats in all shapes and sizes!

SHIRTS
From t-shirts to sweatshirts to polo shirts and beyond.

FIGURE 22.1
Categories in the store.

In Figure 22.2, you see what happens when the user clicks on the HATS link: The script gathers all the items associated with the category and prints them on the screen. The user can still jump to another category on this same page, and the script will gather the items for that category.



My Categories

http://localhost/22/seestore.php?cat_id=1

My Categories

Select a category to see its items.

BOOKS
Paperback, hardback, books for school or play.

HATS
Funky hats in all shapes and sizes!

- Baseball Hat ($12.00)
- Cowboy Hat ($52.00)
- Top Hat ($102.00)

SHIRTS
From t-shirts to sweatshirts to polo shirts and beyond.

FIGURE 22.2
Items within a category in the store.

The last piece of the puzzle for this chapter is the creation of the item display page.

Displaying Items

The item display page in this chapter simply shows all the item information. In Chapter 23, you add a few lines to it to make it function with an Add to Cart button, but for now we focus on the display. Listing 22.2 shows the code for `showitem.php`.

LISTING 22.2 Script to View Item Information

```
<?php
//connect to database
$mysqli = mysqli_connect("localhost", "joeuser", "somepass", "testDB");

$display_block = "<h1>My Store - Item Detail</h1>";

//create safe values for use
$safe_item_id = mysqli_real_escape_string($mysqli, $_GET['item_id']);

//validate item
$get_item_sql = "SELECT c.id as cat_id, c.cat_title, si.item_title,
                 si.item_price, si.item_desc, si.item_image FROM store_items
                 AS si LEFT JOIN store_categories AS c on c.id = si.cat_id
                 WHERE si.id = '".$safe_item_id."'";
$get_item_res = mysqli_query($mysqli, $get_item_sql)
                or die(mysqli_error($mysqli));

if (mysqli_num_rows($get_item_res) < 1) {
    //invalid item
    $display_block .= "<p><em>Invalid item selection.</em></p>";
} else {
    //valid item, get info
    while ($item_info = mysqli_fetch_array($get_item_res)) {
        $cat_id = $item_info['cat_id'];
        $cat_title = strtoupper(stripslashes($item_info['cat_title']));
        $item_title = stripslashes($item_info['item_title']);
        $item_price = $item_info['item_price'];
        $item_desc = stripslashes($item_info['item_desc']);
        $item_image = $item_info['item_image'];
    }

   //make breadcrumb trail & display of item
   $display_block .= <<<END_OF_TEXT
   <p><em>You are viewing:</em><br/>
   <strong><a href="seestore.php?cat_id=$cat_id">$cat_title</a> &gt;
     $item_title</strong></p>
   <div style="float: left;"><img src="$item_image" alt="$item_title"
     /></div>
   <div style="float: left; padding-left: 12px">
   <p><strong>Description:</strong><br/>$item_desc</p>
   <p><strong>Price:</strong> \$$item_price</p>
END_OF_TEXT;

   //free result
   mysqli_free_result($get_item_res);

```

LISTING 22.2 Continued

```
45:    //get colors
46:    $get_colors_sql = "SELECT item_color FROM store_item_color WHERE
47:                       item_id = '".$safe_item_id."' ORDER BY item_color";
48:    $get_colors_res = mysqli_query($mysqli, $get_colors_sql)
49:                      or die(mysqli_error($mysqli));
50:
51:    if (mysqli_num_rows($get_colors_res) > 0) {
52:        $display_block .= "<p><strong>Available Colors:</strong><br/>";
53:        while ($colors = mysqli_fetch_array($get_colors_res)) {
54:            item_color = $colors['item_color'];
55:            $display_block .= $item_color."<br/>";
56:        }
57:    }
58:    //free result
59:    mysqli_free_result($get_colors_res);
60:
61:    //get sizes
62:    $get_sizes_sql = "SELECT item_size FROM store_item_size WHERE
63:                     item_id = ".$safe_item_id." ORDER BY item_size";
64:    $get_sizes_res = mysqli_query($mysqli, $get_sizes_sql)
65:                     or die(mysqli_error($mysqli));
66:
67:    if (mysqli_num_rows($get_sizes_res) > 0) {
68:       $display_block .= "<p><strong>Available Sizes:</strong><br/>";
69:       while ($sizes = mysqli_fetch_array($get_sizes_res)) {
70:           $item_size = $sizes['item_size'];
71:           $display_block .= $item_size."<br/>";
72:       }
73:    }
74:    //free result
75:    mysqli_free_result($get_sizes_res);
76:
77:    $display_block .= "</div>";
78: }
79: //close connection to MySQL
80: mysqli_close($mysqli);
81: ?>
82: <!DOCTYPE html>
83: <html>
84: <head>
85: <title>My Store</title>
86: </head>
87: <body>
88: <?php echo $display_block; ?>
89: </body>
90: </html>
```

Line 3 makes the database connection because information in the database forms all the content of this page. Line 5 starts the `$display_block` string, with some basic page title information.

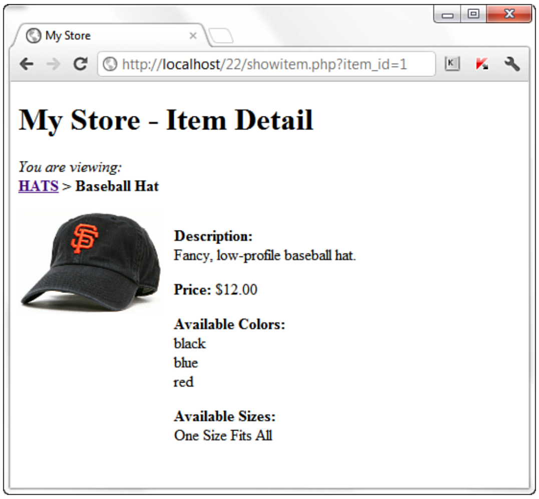
Lines 11–13 create and issue the query to retrieve the category and item information, using the safe value created in line 8. This particular query is a table join. Instead of selecting the item information from one table and then issuing a second query to find the name of the category, this query simply joins the table on the category ID to find the category name.

Line 15 checks for a result; if there is no matching item in the table, a message is printed to the user and that's all this script does. However, if item information is found, the script moves on and gathers the information in lines 23–30.

In lines 34–35, you first create what's known as a *breadcrumb trail*. This is simply a navigational device used to get back to the top-level item in the architecture. In other words, you're going to print a link so the user can get back to the category. The category ID, retrieved from the master query in this script, is appended to the link in the breadcrumb trail.

In lines 36–39, you continue to add to the `$display_block`, setting up a display of information about the item. You use the values gathered in lines 23–30 to create an image link, print the description, and print the price. What's missing are the colors and sizes, so lines 46–57 select and print any colors associated with this item, and lines 62–73 gather the sizes associated with the item.

Lines 77–78 wrap up the `$display_block` string and the master `if...else` statement. Because the script has nothing left to do after closing the connection to MySQL, it prints the HTML (lines 82–90) including the value of `$display_block`. Figure 22.3 shows the outcome of the script when selecting the baseball hat from the hats category. Of course, your display will differ from mine because you won't have the same images I used, but you get the idea.



My Store - Item Detail

You are viewing:
HATS > **Baseball Hat**

Description:
Fancy, low-profile baseball hat.

Price: $12.00

Available Colors:
black
blue
red

Available Sizes:
One Size Fits All

FIGURE 22.3
The baseball hat item page.

That's all there is to creating a simple item display. In Chapter 23, you modify this script so that it can add the item to a shopping cart.

Summary

In this chapter, you applied your basic PHP and MySQL knowledge to the creation of a storefront display. You learned how to create the database table and scripts for viewing categories, item lists, and single items.

Q&A

Q. ***In the item detail record, you use single filenames in the `item_image` field. What if I want to link to items on a different server?***

A. You can enter a URL in the `item_image` field as long as you define the field to hold a long string such as a URL.

Workshop

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

Quiz

1. Which PHP function was used to uppercase the category title strings?
2. Why don't the `store_item_size` and `store_item_color` tables contain any unique keys?
3. Why do you continue to use `mysqli_real_escape_string()` on values that will be used in database queries?

Answers

1. `strtoupper()`
2. Presumably, you will have items with more than one color and more than one size. Therefore, `item_id` is not a unique key. Also, items may have the same colors or sizes, so the `item_color` and `item_size` fields must not be primary or unique either.
3. You should use `mysqli_real_escape_string()` to ensure values from the user, which will be used in database queries, are safe to use, no matter if you've created one script, ten scripts, or one hundred.

Activities

1. Create three more categories, with an item or two in each, by issuing queries of your own in MySQL.
2. Make some images (or use Creative Commons licensed images) for each of the items in your store, and put them in an `images` directory on your server. Doing so necessitates one change to the `showitem.php` script: adding the `image` directory to the file path of the generated `<img/>` tag.