31

Building a Shopping Cart

In this chapter, you learn how to build a basic shopping cart. You add this on top of the Book-O-Rama database implemented in Part II, “Using MySQL.” You also explore another option: setting up and using an existing open source PHP shopping cart.

In case you have not heard it before, the term *shopping cart* (sometimes also called a *shopping basket*) is used to describe a specific online shopping mechanism. As you browse an online catalog, you can add items to your shopping cart. After you’ve finished browsing, you check out of the online store—that is, purchase the items in your cart.

To implement the shopping cart for this project, you need to implement the following functionality:

- A database of the products you want to sell online
- An online catalog of products, listed by category
- A shopping cart to track the items a user wants to buy
- A checkout script that processes payment and shipping details
- An administration interface

Solution Components

You probably remember the Book-O-Rama database developed in Part II. In this project, you get Book-O-Rama’s online store up and going. The solution components fall under these general goals:

- You need to find a way of connecting the database to users’ browsers. Users should be able to browse items by category.
- Users should also be able to select items from the catalog for later purchase. You need to be able to track which items they have selected.
- After users have finished shopping, you need to be able to total their order, take their delivery details, and process their payment.
- You should also build an administration interface to Book-O-Rama’s site so that the administrator can add and edit books and categories on the site.

Now that you know the idea behind the project, you can begin designing the solution and its components.

Building an Online Catalog

You already have a database for the Book-O-Rama catalog. However, it probably needs some alterations and additions for this application. One of these is to add categories of books, as stated in the requirements.

You also need to add some information to the existing database about shipping addresses, payment details, and so on. You already know how to build an interface to a MySQL database using PHP, so this part of the solution should be pretty easy.

You should also use transactions while completing customers' orders. To do this, you need to convert your Book-O-Rama tables to use the InnoDB storage engine. This process is also reasonably straightforward.

Tracking Users' Purchases While They Shop

There are two basic ways you can track users' purchases while they shop. One is to put their selections into the database, and the other is to use a session variable.

Using a session variable to track selections from page to page is easier to write because it does not require you to constantly query the database for this information. By using this approach, you also avoid the situation in which you end up with a lot of junk data in the database from users who are just browsing and change their minds.

You need, therefore, to design a session variable or set of variables to store a user's selections. When a user finishes shopping and pays for her purchases, you will put this information in the database as a record of the transaction.

You can also use this data to give a summary of the current state of the cart in one corner of the page so that a user knows at any given time how much she is planning to spend.

Implementing a Payment System

In this project, you add the user's order and take the delivery details but do not actually process payments. Many, many payment systems are available, and the implementation for each one is different. For this project, you write a dummy function that can be replaced with an interface to your chosen system.

Although there are several different payment gateways you can use, and many different interfaces to these gateways, the functionality behind real-time credit card processing interfaces is generally similar. You need to open a merchant account with a bank for the cards you want to accept—and typically your bank will have a list of recommended providers for the payment system itself. Your payment system provider will specify what parameters you need to pass to its system, and how. Many payment systems have sample code already available for use with PHP, which you could easily use to replace the dummy function created in this chapter.

When in use, the payment system transmits your data to a bank and returns a success code or one of many different types of error codes. In exchange for passing on your data, the payment gateway charges you a setup or annual fee, as well as a fee based on the number or value of your transactions. Some providers even charge for declined transactions.

At the minimum, your payment system needs information from the customer (such as a credit card number), identifying information from you (to specify which merchant account is to be credited), and the total amount of the transaction.

You can work out the total of an order from a user's shopping cart session variable. You then record the final order details in the database and get rid of the session variable at that time.

Building an Administration Interface

In addition to the payment system and so on, you also need to build an administration interface that lets you add, delete, and edit books and categories from the database.

One common edit that you might make is to alter the price of an item (for example, for a special offer or sale). This means that when you store a customer's order, you should also store the price she paid for an item. It would make for an accounting nightmare if the only records you had were the items each customer ordered and the current price of each one. This also means that if the customer has to return or exchange the item, you will give her the right amount of credit.

You are not going to build a fulfillment and order tracking interface for this example. However, you can add one onto this base system to suit your needs.

Solution Overview

Let's put all the pieces together now. There are two basic views of the system: the user view and the administrator view. After considering the functionality required, we came up with two system flow designs you can use, one for each view. They are shown in Figures 31.1 and 31.2, respectively.

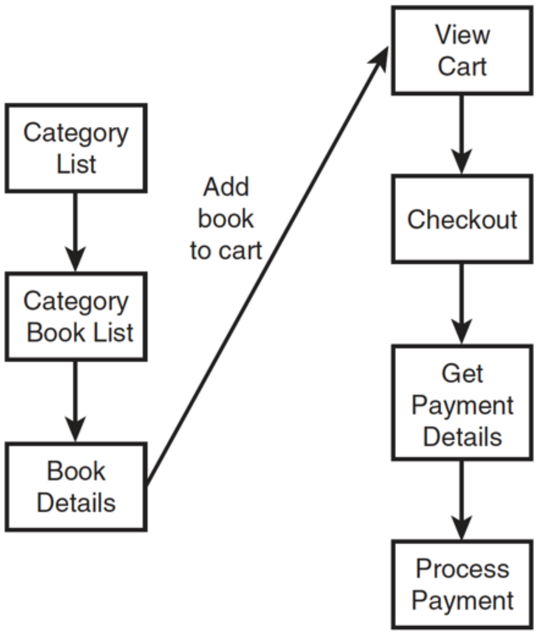


Figure 31.1 The user view of the Book-O-Rama system lets users browse books by category, view book details, add books to their cart, and purchase them.

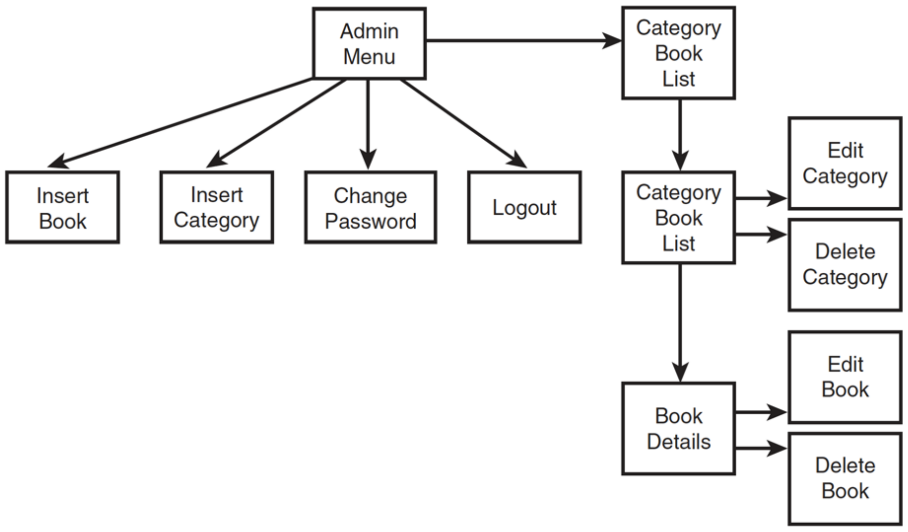


Figure 31.2 The administrator view of the Book-O-Rama system allows insertion, editing, and deletion of books and categories.

Figure 31.1 shows the main links between scripts in the user part of the site. A customer comes first to the main page, which lists all the categories of books in the site. From there, she can go to a particular category of books, and from there to an individual book's details.

You give the user a link to add a particular book to her cart. From the cart, she can check out of the online store.

Figure 31.2 shows the administration interface, which has more scripts but not much new code. These scripts let an administrator log in and insert books and categories.

The easiest way to implement editing and deletion of books and categories is to show the administrator a slightly different version of the user interface to the site. The administrator can still browse categories and books, but instead of having access to the shopping cart, he can go to a particular book or category and edit or delete that book or category. By making the same scripts suit both normal and administrator users, you can save yourself time and effort.

The three main code modules for this application are as follows:

- Catalog
- Shopping cart and order processing (We bundled them together because they are strongly related.)
- Administration

As is often the case with a project such as this, you will need to build and use a set of function libraries. For this project, you use a function API similar to other projects in this text. Try to confine the parts of the code that output HTML to a single library to support the principle of separating logic and content and, more importantly, to make the code easier to read and maintain.

You also need to make some minor changes to the Book-O-Rama database for this project. We renamed the database `book_sc` (Shopping Cart) to distinguish the shopping cart database from the one built in Part II.

A summary of the files in the application is shown in Table 31.1.

Table 31.1 Files in the Shopping Cart Application

Name	Module	Description
`index.php`	Catalog	Main front page of site for users. Shows the users a list of categories in the system.
`show_cat.php`	Catalog	Page that shows the users all the books in a particular category.
`show_book.php`	Catalog	Page that shows the users details of a particular book.
`show_cart.php`	Shopping cart	Page that shows the users the contents of their shopping carts. Also used to add items to the cart.
`checkout.php`	Shopping cart	Page that presents the users with complete order details. Gets shipping details.
`purchase.php`	Shopping cart	Page that gets payment details from users.
`process.php`	Shopping cart	Script that processes payment details and adds the order to the database.
`login.php`	Administration	Script that allows the administrator to log in to make changes.
`logout.php`	Administration	Script that logs out the admin user.
`admin.php`	Administration	Main administration menu.
`change_password_form.php`	Administration	Form to let administrators change their log passwords.
`change_password.php`	Administration	Script that changes the administrator password.
`insert_category_form.php`	Administration	Form to let administrators add a new category to the

		database.
`insert_category.php`	Administration	Script that inserts a new category into the database.
`insert_book_form.php`	Administration	Form to let administrators add a new book to the system.
`insert_book.php`	Administration	Script that inserts a new book into the database.
`edit_category_form.php`	Administration	Form to let administrators edit a category.
`edit_category.php`	Administration	Script that updates a category in the database.
`edit_book_form.php`	Administration	Form to let administrators edit a book's details.
`edit_book.php`	Administration	Script that updates a book in the database.
`delete_category.php`	Administration	Script that deletes a category from the database.
`delete_book.php`	Administration	Script that deletes a book from the database.
`book_sc_fns.php`	Functions	Collection of include files for this application.
`admin_fns.php`	Functions	Collection of functions used by administrative scripts.
`book_fns.php`	Functions	Collection of functions for storing and retrieving book data.
`order_fns.php`	Functions	Collection of functions for storing and retrieving order data.
`output_fns.php`	Functions	Collection of functions for outputting HTML.
`data_valid_fns.php`	Functions	Collection of functions for validating input data.
`db_fns.php`	Functions	Collection of functions for connecting to the `book_sc` database.
`user_auth_fns.php`	Functions	Collection of functions for authenticating administrative users.
`book_sc.sql`	SQL	SQL to set up the `book_sc` database.
`populate.sql`	SQL	SQL to insert some sample data into the `book_sc` database.

Now, let's look at the implementation of each of the modules.

Note

This application contains a lot of code. Much of it implements functionality you have looked at already throughout the book, such as storing data to and retrieving it from the database, and authenticating the administrative user. We look briefly at this code but spend most of our time on the shopping cart functions.

Implementing the Database

As we mentioned earlier, we made some minor modifications to the Book-O-Rama database presented in Part II. The SQL to create the `book_sc` database is shown in Listing 31.1.

Listing 31.1 `book_sc.sql`—SQL to Create the `book_sc` Database

```
create database book_sc;

use book_sc;

create table customers
(
  customerid int unsigned not null auto_increment primary key,
  name char(60) not null,
  address char(80) not null,
  city char(30) not null,
  state char(20),
  zip char(10),
  country char(20) not null
) type=InnoDB;

create table orders
(
  orderid int unsigned not null auto_increment primary key,
  customerid int unsigned not null references customers(customerid),
  amount float(6,2),
  date date not null,
  order_status char(10),
  ship_name char(60) not null,
  ship_address char(80) not null,
  ship_city char(30) not null,
  ship_state char(20),
  ship_zip char(10),
  ship_country char(20) not null
) type=InnoDB;

create table books
(
   isbn char(13) not null primary key,
   author char(100),
   title char(100),
   catid int unsigned,
   price float(4,2) not null,
   description varchar(255)
) type=InnoDB;

create table categories
(
  catid int unsigned not null auto_increment primary key,
  catname char(60) not null
) type=InnoDB;

create table order_items
(
  orderid int unsigned not null references orders(orderid),
  isbn char(13) not null references books(isbn),
  item_price float(4,2) not null,
  quantity tinyint unsigned not null,
  primary key (orderid, isbn)
) type=InnoDB;

create table admin
 (
  username char(16) not null primary key,
  password char(40) not null
);

grant select, insert, update, delete
on book_sc.*
to book_sc@localhost identified by 'password';
```

Although nothing was wrong with the original Book-O-Rama interface, you must address a few other requirements now that you are going to make it available online.

The changes made to the original database are as follows:

- The addition of more address fields for customers. Having additional fields is more important now that you are building a more realistic application.
- The addition of a shipping address to an order. A customer's contact address might not be the same as the shipping address, particularly if she is using the site to buy a gift.
- The addition of a `categories` table and a `catid to books` table. Sorting books into categories makes the site easier to browse.
- The addition of `item_price` to the `order_items` table to recognize the fact that an item's price might change. You want to know how much the item cost when the customer ordered it.
- The addition of an `admin` table to store administrator login and password details.
- The removal of the reviews table. You could add reviews as an extension to this project. Instead, each book has a description field containing a brief blurb about the book.
- The change in storage engines to InnoDB. You do this so that you can use foreign keys and also so you can use transactions when entering customer order information.

To set up this database on your system, run the `book_sc.sql` script through MySQL as the root user, as follows:

```
mysql -u root -p < book_sc.sql
```

(You need to supply your root password.)

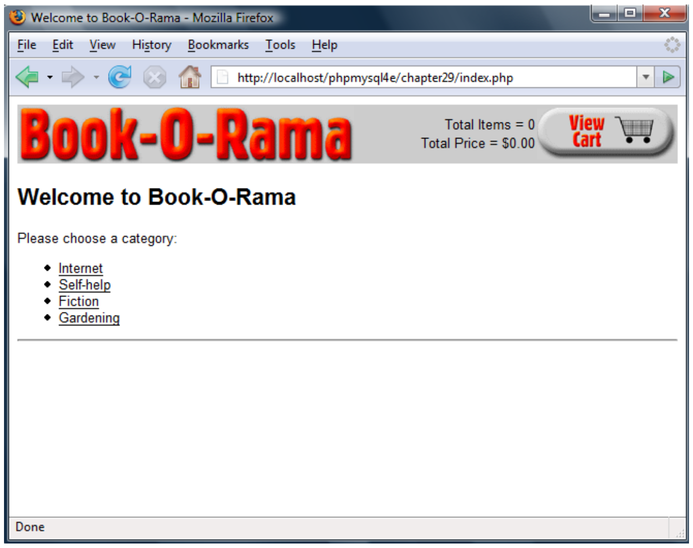
Beforehand, you should change the password for the `book_sc` user to something better than `'password'`. Note that if you change the password in `book_sc.sql`, you will also need to change it in `db_fns.php`. (You'll see where shortly.)

We also included a file of sample data called `populate.sql`. You can put the sample data into the database by running it through MySQL in this same way.

Implementing the Online Catalog

Three catalog scripts are used in this application: the main page, category page, and book details page.

The front page of the site is produced by the script called `index.php`. The output of this script is shown in Figure 31.3.


Welcome to Book-O-Rama - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/index.php
Book-O-Rama
View Cart
Done

Total Items = 0
Total Price = $0.00

Welcome to Book-O-Rama

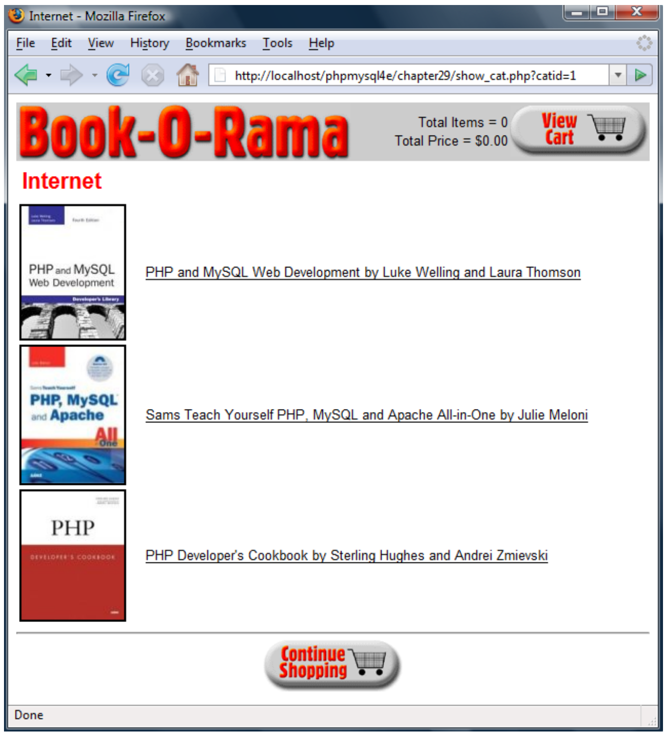
Please choose a category:

- Internet
- Self-help
- Fiction
- Gardening

Figure 31.3 The front page of the site lists the categories of books available for purchase.

Notice that, in addition to the list of categories on the site, it has a link to the shopping cart in the top-right corner of the screen and some summary information about what's in the cart. These elements appear on every page while a user browses and shops.

If a user clicks one of the categories, she'll be taken to the category page, produced by the script `show_cat.php`. The category page for the Internet books section is shown in Figure 31.4.


Internet - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/show_cat.php?catid=1
Book-O-Rama
View Cart
Continue Shopping
Done

Total Items = 0
Total Price = $0.00

Internet

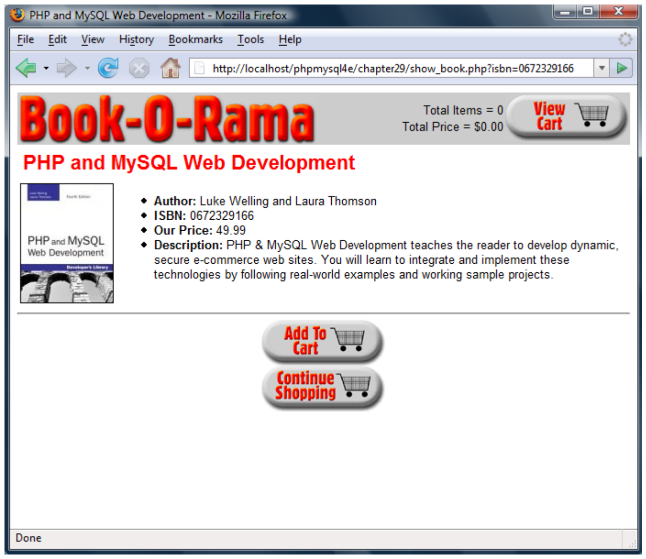
PHP and MySQL Web Development by Luke Welling and Laura Thomson

Sams Teach Yourself PHP, MySQL and Apache All-in-One by Julie Meloni

PHP Developer's Cookbook by Sterling Hughes and Andrei Zmievski

Figure 31.4 Each book in the category is listed with a photo.

All the books in the Internet category are listed as links. If a user clicks one of these links, she will be taken to the book details page. The book details page for one book is shown in Figure 31.5.


PHP and MySQL Web Development - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/show_book.php?isbn=0672329166
Book-O-Rama
View Cart
Add To Cart
Continue Shopping
Done

Total Items = 0
Total Price = $0.00

PHP and MySQL Web Development

- **Author:** Luke Welling and Laura Thomson
- **ISBN:** 0672329166
- **Our Price:** 49.99
- **Description:** PHP & MySQL Web Development teaches the reader to develop dynamic, secure e-commerce web sites. You will learn to integrate and implement these technologies by following real-world examples and working sample projects.

Figure 31.5 Each book has a details page that shows more information, including a long description.

On this page, as well as the View Cart link, an Add to Cart link enables the user to select an item for purchase. We return to this feature when we look at how to build the shopping cart later.

Let's look at each of these three scripts.

Listing Categories

The first script used in this project, `index.php`, lists all the categories in the database. It is shown in Listing 31.2.

Listing 31.2 **`index.php`—Script to Produce the Front Page of the Site**

```
<?php
  include_once 'book_sc_fns.php';
  // The shopping cart needs sessions, so start one
  session_start();
  do_html_header("Welcome to Book-O-Rama");

  echo "<p>Please choose a category:</p>";

  // get categories out of database
  $cat_array = get_categories();

  // display as links to cat pages
  display_categories($cat_array);

  // if logged in as admin, show add, delete, edit cat links
  if(isset($_SESSION['admin_user'])) {
    display_button("admin.php", "admin-menu", "Admin Menu");
  }
  do_html_footer();
```

```
?>
```

This script begins by including `book_sc_fns.php`, the file that includes all the function libraries for this application.

After that, you must begin a session. This is required for the shopping cart functionality to work. Every page in the site will use the session.

The `index.php` script also contains some calls to HTML output functions such as `do_html_header()` and `do_html_footer()` (both contained in `output_fns.php`). It also contains some code that checks whether the user is logged in as an administrator and gives her some different navigation options if she is; we return to this feature in the section on the administration functions.

The most important part of this script is

```
// get categories out of database
$cat_array = get_categories();

// display as links to cat pages
display_categories($cat_array);
```

The functions `get_categories()` and `display_categories()` are in the function libraries `book_fns.php` and `output_fns.php`, respectively. The function `get_categories()` returns an array of the categories in the system, which you then pass to `display_categories()`. Let's look at the code for `get_categories()`, shown in Listing 31.3.

Listing 31.3 `get_categories()` Function from `book_fns.php`—Function That Retrieves a Category List from the Database

```
function get_categories() {
   // query database for a list of categories
   $conn = db_connect();
   $query = "select catid, catname from categories";
   $result = @$conn->query($query);
   if (!$result) {
     return false;
   }
   $num_cats = @$result->num_rows;
   if ($num_cats == 0) {
      return false;
   }
   $result = db_result_to_array($result);
   return $result;
}
```

As you can see, the `get_categories()` function connects to the database and retrieves a list of all the category IDs and names. We wrote and used a function called `db_result_to_array()`, located in `db_fns.php`. This function is shown in Listing 31.4. It takes a MySQL result identifier and returns a numerically indexed array of rows, where each row is an associative array.

Listing 31.4 `db_result_to_array()` Function from `db_fns.php`—Function That Converts a MySQL Result Identifier into an Array of Results

```
function db_result_to_array($result) {
   $res_array = array();

   for ($count=0; $row = $result->fetch_assoc(); $count++) {
     $res_array[$count] = $row;
   }

   return $res_array;
}
```

In this case, you return this array back all the way to `index.php`, where you pass it to the `display_categories()` function from `output_fns.php`. This function displays each category as a link to the page containing the books in that category. The code for this function is shown in Listing 31.5.

Listing 31.5 `display_categories()` Function from `output_fns.php`—Function That Displays an Array of Categories as a List of Links to Those Categories

```
function display_categories($cat_array) {
  if (!is_array($cat_array)) {
     echo "<p>No categories currently available</p>";
     return;
  }
  echo "<ul>";
  foreach ($cat_array as $row)  {
    $url = "show_cat.php?catid=".urlencode($row['catid']);
    $title = $row['catname'];
    echo "<li>";
    do_html_url($url, $title);
    echo "</li>";
  }
  echo "</ul>";
  echo "<hr />";
}
```

The `display_categories()` function converts each category from the database into a link. Each link goes to the next script—`show_cat.php`—but each has a different parameter, the category ID or `catid`. (This unique number, generated by MySQL, is used to identify the category.)

This parameter to the next script determines which category you end up looking at.

Listing Books in a Category

The process for listing books in a category is similar. The script that does this, called `show_cat.php`, is shown in Listing 31.6.

Listing 31.6 `show_cat.php`—Script That Shows the Books in a Particular Category

```
<?php
  include ('book_sc_fns.php');
  // The shopping cart needs sessions, so start one
  session_start();

  $catid = $_GET['catid'];
  $name = get_category_name($catid);

  do_html_header($name);

  // get the book info out from db
  $book_array = get_books($catid);
  display_books($book_array);


  // if logged in as admin, show add, delete book links
  if(isset($_SESSION['admin_user'])) {
    display_button("index.php", "continue", "Continue Shopping");
    display_button("admin.php", "admin-menu", "Admin Menu");
    display_button("edit_category_form.php?catid=". urlencode($catid),
                   "edit-category", "Edit Category");
  } else {
    display_button("index.php", "continue-shopping", "Continue Shopping");
  }

  do_html_footer();
?>
```

This script is similar in structure to the index page, except that you retrieve books instead of categories.

You start with `session_start()` as usual and then convert the category ID you have been passed into a category name by using the `get_category_name()` function as follows:

```
$name = get_category_name($catid);
```

This function, shown in Listing 31.7, looks up the category name in the database.

Listing 31.7 `get_category_name()` Function from `book_fns.php`—Function That Converts a Category ID to a Category Name

```
function get_category_name($catid) {
   // query database for the name for a category id
   $conn = db_connect();
   $query = "select catname from categories
             where catid = '".$conn->real_escape_string($catid)."'";
   $result = @$conn->query($query);
   if (!$result) {
     return false;
   }
   $num_cats = @$result->num_rows;
   if ($num_cats == 0) {
      return false;
   }
   $row = $result->fetch_object();
   return $row->catname;
}
```

After you have retrieved the category name, you can render an HTML header and proceed to retrieve and list the books from the database that fall into your chosen category, as follows:

```
$book_array = get_books($catid);
display_books($book_array);
```

The functions `get_books()` and `display_books()` are extremely similar to the `get_categories()` and `display_categories()` functions, so we do not go into them here. The only difference is that you retrieve information from the `books` table rather than the `categories` table.

The `display_books()` function provides a link to each book in the category via the `show_book.php` script. Again, each link is suffixed with a parameter. This time around, it's the ISBN for the book in question.

At the bottom of the `show_cat.php` script, there is some code to display additional functions if an administrator is logged in. We look at these functions in the section on administrative functions.

Showing Book Details

The `show_book.php` script takes an ISBN as a parameter and retrieves and displays the details of that book. The code for this script is shown in Listing 31.8.

Listing 31.8 **`show_book.php`— Script That Shows the Details of a Particular Book**

```
<?php
  include ('book_sc_fns.php');
  // The shopping cart needs sessions, so start one
  session_start();

  $isbn = $_GET['isbn'];

  // get this book out of database
  $book = get_book_details($isbn);
  do_html_header($book['title']);
  display_book_details($book);

  // set url for "continue button"
  $target = "index.php";
  if($book['catid']) {
    $target = "show_cat.php?catid=". urlencode($book['catid']);
  }

  // if logged in as admin, show edit book links
  if(check_admin_user()) {
    display_button("edit_book_form.php?isbn=". urlencode($isbn), "edit-item", "Edit Item");
    display_button("admin.php", "admin-menu", "Admin Menu");
    display_button($target, "continue", "Continue");
  } else {
    display_button("show_cart.php?new=". urlencode($isbn), "add-to-cart",
                   "Add ". htmlspecialchars($book['title']) ." To My Shopping Cart");
    display_button($target, "continue-shopping", "Continue Shopping");
  }

  do_html_footer();
?>
```

Again, with this script you do similar things as in the previous two pages. You begin by starting the session and then use

```
$book = get_book_details($isbn);
```

to get the book information out of the database. Next, you use

```
display_book_details($book);
```

to output the data in HTML.

Note that `display_book_details()` looks for an image file for the book as `images/{$book['isbn']}.jpg`, in which the name of the file is the book's ISBN plus the .jpg extension. If this file does not exist in the images subdirectory, no image will be displayed. The remainder of the `show_book.php` script sets up navigation. A normal user has the choices Continue Shopping, which takes her back to the category page, and Add to Cart, which adds the book to her shopping cart. If a user is logged in as an administrator, she will get some different options, which we look at in the section on administration.

We've completed the basics of the catalog system. Now let's look at the code for the shopping cart functionality.

Implementing the Shopping Cart

The shopping cart functionality all revolves around a session variable called `cart`. It is an associative array that has ISBNs as keys and quantities as values. For example, if you add a single copy of this book to your shopping cart, the array would contain

```
0672329166=> 1
```

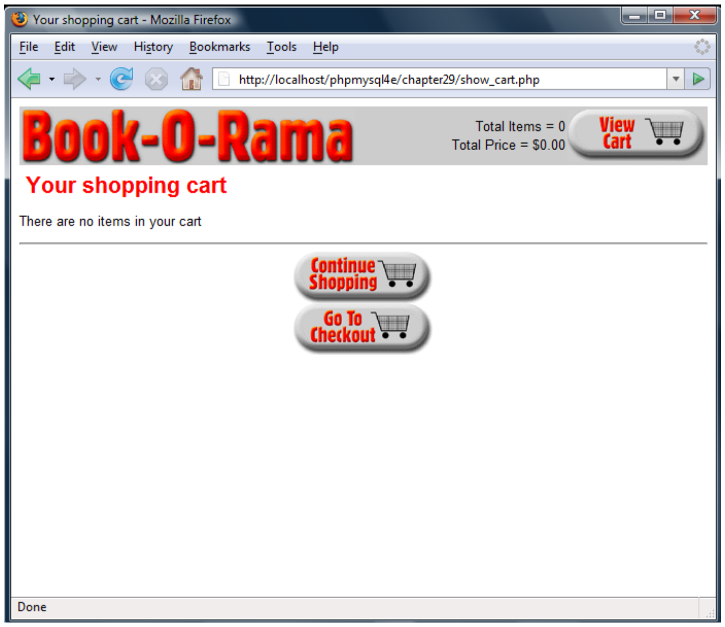
That is, the array would contain one copy of the book with the ISBN 0672329166. When you add items to the cart, they are added to the array. When you view the cart, you use the `cart` array to look up the full details of the items in the database.

You also use two other session variables to control the display in the header that shows Total Items and Total Price. These variables are called `items` and `total_price`, respectively.

Using the `show_cart.php` Script

Let's examine how the shopping cart code is implemented by looking at the `show_cart.php` script. This script displays the page you will visit if you click on any View Cart or Add to Cart links. If you call `show_cart.php` without any parameters, you will get to see the contents of it. If you call it with an ISBN as a parameter, the item with that ISBN will be added to the cart.

To understand fully how this script operates, look first at Figure 31.6.


Your shopping cart - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/show_cart.php
Book-O-Rama
Total Items = 0
Total Price = $0.00
View Cart
Continue Shopping
Go To Checkout
Done

Your shopping cart

There are no items in your cart

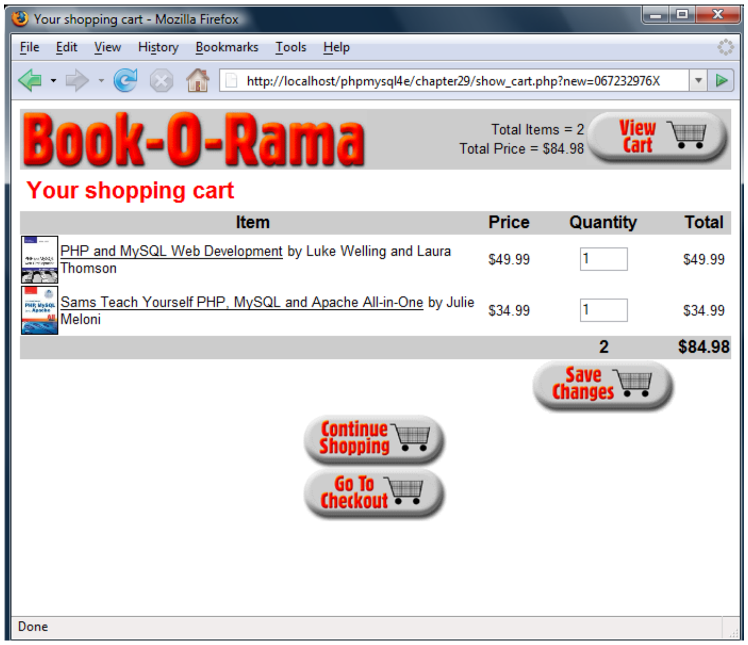
Figure 31.6 The `show_cart.php` script with no parameters just shows the contents of the cart.

In this case, we clicked the View Cart link when our cart was empty; that is, we had not yet selected any items to purchase.

Figure 31.7 shows the cart a bit further down the track after we selected two books to buy. In this case, we got to this page by clicking the Add to Cart link on the `show_book.php` page for this book, *PHP and MySQL Web Development*. If you look closely at the URL bar, you will see that we called the script with a parameter this time. The parameter is called `new` and has the value `067232976X`—that is, the ISBN for the book just added to the cart.

From this page, you can see that you have two other options. The Save Changes button can be used to change the quantity of items in the cart. To do this, the user can alter the quantities directly and click Save Changes. This is actually a submit button that takes the user back to the `show_cart.php` script again to update the cart.

In addition, the user can click the Go to Checkout button when she is ready to leave. We come back to that shortly.


Your shopping cart - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/show_cart.php?new=067232976X
Book-O-Rama
Total Items = 2
Total Price = $84.98
View Cart
Save Changes
Continue Shopping
Go To Checkout
Done

Your shopping cart

Item	Price	Quantity	Total
PHP and MySQL Web Development by Luke Welling and Laura Thomson	$49.99	1	$49.99
Sams Teach Yourself PHP, MySQL and Apache All-in-One by Julie Meloni	$34.99	1	$34.99
		2	$84.98

Figure 31.7 The `show_cart.php` script with the new parameter adds a new item to the cart.

For now, let’s look at the code for the `show_cart.php` script. This code is shown in Listing 31.9.

Listing 31.9 **`show_cart.php`— Script That Controls the Shopping Cart**

```
<?php
  include ('book_sc_fns.php');
  // The shopping cart needs sessions, so start one
  session_start();

  @$new = $_GET['new'];

  if($new) {
    //new item selected
    if(!isset($_SESSION['cart'])) {
      $_SESSION['cart'] = array();
      $_SESSION['items'] = 0;
      $_SESSION['total_price'] ='0.00';
    }

    if(isset($_SESSION['cart'][$new])) {
      $_SESSION['cart'][$new]++;
    } else {
      $_SESSION['cart'][$new] = 1;
    }
    $_SESSION['total_price'] = calculate_price($_SESSION['cart']);
    $_SESSION['items'] = calculate_items($_SESSION['cart']);
  }

  if(isset($_POST['save'])) {
    foreach ($_SESSION['cart'] as $isbn => $qty) {
      if($_POST[$isbn] == '0') {
        unset($_SESSION['cart'][$isbn]);
      } else {
        $_SESSION['cart'][$isbn] = $_POST[$isbn];
      }
    }

    $_SESSION['total_price'] = calculate_price($_SESSION['cart']);
    $_SESSION['items'] = calculate_items($_SESSION['cart']);
  }

  do_html_header("Your shopping cart");

  if(($_SESSION['cart']) && (array_count_values($_SESSION['cart']))) {
    display_cart($_SESSION['cart']);
  } else {
    echo "<p>There are no items in your cart</p><hr/>";
  }

  $target = "index.php";

  // if we have just added an item to the cart, continue shopping in that category
  if($new)   {
    $details =  get_book_details($new);
    if($details['catid']) {
      $target = "show_cat.php?catid=".urlencode($details['catid']);
    }
  }
  display_button($target, "continue-shopping", "Continue Shopping");

  // use this if SSL is set up
  // $path = $_SERVER['PHP_SELF'];
  // $server = $_SERVER['SERVER_NAME'];
  // $path = str_replace('show_cart.php', '', $path);
  // display_button("https://".$server.$path."checkout.php",
  //                 "go-to-checkout", "Go To Checkout");

  // if no SSL use below code
  display_button("checkout.php", "go-to-checkout", "Go To Checkout");

  do_html_footer();
?>
```

This script has three main parts: displaying the cart, adding items to the cart, and saving changes to the cart. We cover these parts in the next three sections.

Viewing the Cart

No matter which page you come from, you display the contents of the cart. In the base case, when a user has just clicked View Cart, the only part of the code that will be executed follows:

```
if(($_SESSION['cart']) && (array_count_values($_SESSION['cart']))) {
  display_cart($_SESSION['cart']);
} else {
  echo "<p>There are no items in your cart</p><hr/>";
}
```

As you can see from this code, if you have a cart with some contents, you will call the `display_cart()` function. If the cart is empty, you'll give the user a message to that effect.

The `display_cart()` function just prints the contents of the cart as a readable HTML format, as you can see in Figures 31.6 and 31.7. The code for this function can be found in `output_fns.php`, which is included here as Listing 31.10. Although it is a display function, it is reasonably complex, so we chose to include it here.

Listing 31.10 **`display_cart()` Function from `output_fns.php`—Function That Formats and Prints the Contents of the Shopping Cart**

```
function display_cart($cart, $change = true, $images = 1) {
  // display items in shopping cart
  // optionally allow changes (true or false)
  // optionally include images (1 - yes, 0 - no)

   echo "<table border=\"0\" width=\"100%\" cellspacing=\"0\">
         <form action=\"show_cart.php\" method=\"post\">
         <tr><th colspan=\"".(1 + $images)."\" bgcolor=\"#cccccc\">Item</th>
         <th bgcolor=\"#cccccc\">Price</th>
         <th bgcolor=\"#cccccc\">Quantity</th>
         <th bgcolor=\"#cccccc\">Total</th>
         </tr>";

  //display each item as a table row
  foreach ($cart as $isbn => $qty)  {
    $book = get_book_details($isbn);
    echo "<tr>";
    if($images == true) {
      echo "<td align=\"left\">";
      if (file_exists("images/{$isbn}.jpg")) {
         $size = GetImageSize("images/{$isbn}.jpg");
         if(($size[0] > 0) && ($size[1] > 0)) {
           echo "<img src=\"images/".htmlspecialchars($isbn).".jpg\"
                  style=\"border: 1px solid black\"
                  width=\"".($size[0]/3)."\"
                  height=\"".($size[1]/3)."\"/>";
         }
      } else {
         echo "&nbsp;";
      }
      echo "</td>";
    }
    echo "<td align=\"left\">
          <a
href=\"show_book.php?isbn=".urlencode($isbn)."\">".htmlspecialchars($book['title'])."</a>
          by ".htmlspecialchars($book['author'])."</td>
          <td align=\"center\">\$".number_format($book['price'], 2)."</td>
          <td align=\"center\">";

    // if we allow changes, quantities are in text boxes
    if ($change == true) {
      echo "<input type=\"text\" name=\"".htmlspecialchars($isbn)."\"
value=\"".htmlspecialchars($qty)."\" size=\"3\">";
    } else {
      echo $qty;
    }
    echo "</td><td align=\"center\">\$".number_format($book['price']*$qty,2)."</td></tr>\n";
  }
  // display total row
  echo "<tr>
        <th colspan=\"".(2+$images)."\" bgcolor=\"#cccccc\">&nbsp;</td>
        <th align=\"center\" bgcolor=\"#cccccc\">".htmlspecialchars($_SESSION['items'])."</th>
        <th align=\"center\" bgcolor=\"#cccccc\">
            \$".number_format($_SESSION['total_price'], 2)."
        </th>
        </tr>";

  // display save change button
  if($change == true) {
    echo "<tr>
          <td colspan=\"".(2+$images)."\">&nbsp;</td>
          <td align=\"center\">
             <input type=\"hidden\" name=\"save\" value=\"true\"/>
             <input type=\"image\" src=\"images/save-changes.gif\"
                    border=\"0\" alt=\"Save Changes\"/>
          </td>
          <td>&nbsp;</td>
          </tr>";
  }
  echo "</form></table>";
}
```

The basic flow of this function is as follows:

1. Loop through each item in the cart and pass the ISBN of each item to `get_book_details()` so that you can summarize the details of each book.
2. Provide an image for each book, if one exists. Use the HTML image height and width tags to resize the image a little smaller here. This means that the images will be a little distorted, but they are small enough that this isn't much of a problem. (If the distortion bothers you, you can always resize the images using the gd library discussed in Chapter 21, "Generating Images," or manually generate different-size images for each product.)
3. Make each cart entry a link to the appropriate book—that is, to `show_book.php` with the ISBN as a parameter.
4. If you are calling the function with the `change` parameter set to `true` (or not set—it defaults to `true`), show the boxes with the quantities in them as a form with the Save Changes button at the end. (When you reuse this function after checking out, you don't want the user to be able to change her order.)

Nothing is terribly complicated in this function, but it does quite a lot of work, so you might find reading it through carefully to be useful.

Adding Items to the Cart

If a user has come to the `show_cart.php` page by clicking an Add to Cart button, you have to do some work before you can show her the contents of her cart. Specifically, you need to add the appropriate item to the cart, as follows.

First, if the user has not put any items in her cart before, she will not have a cart, so you need to create one:

```
if(!isset($_SESSION['cart'])) {
  $_SESSION['cart'] = array();
  $_SESSION['items'] = 0;
  $_SESSION['total_price'] ='0.00';
}
```

To begin with, the cart is empty.

Second, after you know that a cart is set up, you can add the item to it:

```
if(isset($_SESSION['cart'][$new])) {
  $_SESSION['cart'][$new]++;
} else {
  $_SESSION['cart'][$new] = 1;
}
```

Here, you check whether the item is already in the cart. If it is, you increment the quantity of that item in the cart by one. If not, you add the new item to the cart.

Third, you need to work out the total price and number of items in the cart. For this, you use the `calculate_price()` and `calculate_items()` functions, as follows:

```
$_SESSION['total_price'] = calculate_price($_SESSION['cart']);
$_SESSION['items'] = calculate_items($_SESSION['cart']);
```

These functions are located in the `book_fns.php` function library. The code for them is shown in Listings 31.11 and 31.12, respectively.

Listing 31.11 `calculate_price()` Function from `book_fns.php`— Function That Calculates and Returns the Total Price of the Contents of the Shopping Cart

```
function calculate_price($cart) {
  // sum total price for all items in shopping cart
  $price = 0.0;
  if(is_array($cart)) {
    $conn = db_connect();
    foreach($cart as $isbn => $qty) {
      $query = "select price from books where isbn='".$conn->real_escape_string($isbn)."'";
      $result = $conn->query($query);
      if ($result) {
        $item = $result->fetch_object();
        $item_price = $item->price;
        $price +=$item_price*$qty;
      }
    }
  }
  return $price;
}
```

As you can see, the `calculate_price()` function works by looking up the price of each item in the cart in the database. This process is somewhat slow, so to avoid doing this more often than you need to, you store the price (and the total number of items, as well) as session variables and recalculate only when the cart changes.

Listing 31.12 calculate_items() Function from book_fns.php—Function That Calculates and Returns the Total Number of Items in the Shopping Cart

```
function calculate_items($cart) {
  // sum total items in shopping cart
  $items = 0;
  if(is_array($cart))   {
    foreach($cart as $isbn => $qty) {
      $items += $qty;
    }
  }
  return $items;
}
```

The `calculate_items()` function is simpler; it just goes through the cart and adds the quantities of each item to get the total number of items using the `array_sum()` function. If there's not yet an array (if the cart is empty), it just returns 0 (zero).

Saving the Updated Cart

If the user comes to the `show_cart.php` script by clicking the Save Changes button, the process is a little different. In this case, the user has arrived via a form submission. If you look closely at the code, you will see that the Save Changes button is the submit button for a form. This form contains the hidden variable `save`. If this variable is set, you know that you have come to this script from the Save Changes button. This means that the user has presumably edited the quantity values in the cart, and you need to update them.

If you look back at the text boxes in the Save Changes form part of the script, found in the `display_cart()` function in `output_fns.php`, you will see that they are named after the ISBN of the item that they represent, as follows:

```
echo "<input type=\"text\" name=\"".htmlspecialchars($isbn)."\"
value=\"".htmlspecialchars($qty)."\" size=\"3\">";
```

Now look at the part of the script that saves the changes:

```
if(isset($_POST['save'])) {
  foreach ($_SESSION['cart'] as $isbn => $qty) {
    if($_POST[$isbn] == '0') {
      unset($_SESSION['cart'][$isbn]);
    } else {
      $_SESSION['cart'][$isbn] = $_POST[$isbn];
    }
  }

  $_SESSION['total_price'] = calculate_price($_SESSION['cart']);
  $_SESSION['items'] = calculate_items($_SESSION['cart']);
}
```

Here, you work your way through the shopping cart, and for each `isbn` in the cart, you check the `POST` variable with that name. These variables are the form fields from the Save Changes form.

If any of the fields are set to zero, you remove that item from the shopping cart altogether, using `unset()`. Otherwise, you update the cart to match the form fields, as follows:

```
if($_POST[$isbn] == '0') {
  unset($_SESSION['cart'][$isbn]);
} else {
  $_SESSION['cart'][$isbn] = $_POST[$isbn];
}
```

After these updates, you again use `calculate_price()` and `calculate_items()` to work out the new values of the `total_price` and `items` session variables.

Printing a Header Bar Summary

In the header bar of each page in the site, a summary of what's in the shopping cart is presented. This summary is obtained by printing out the values of the session variables `total_price` and `items`. This is done in the `do_html_header()` function.

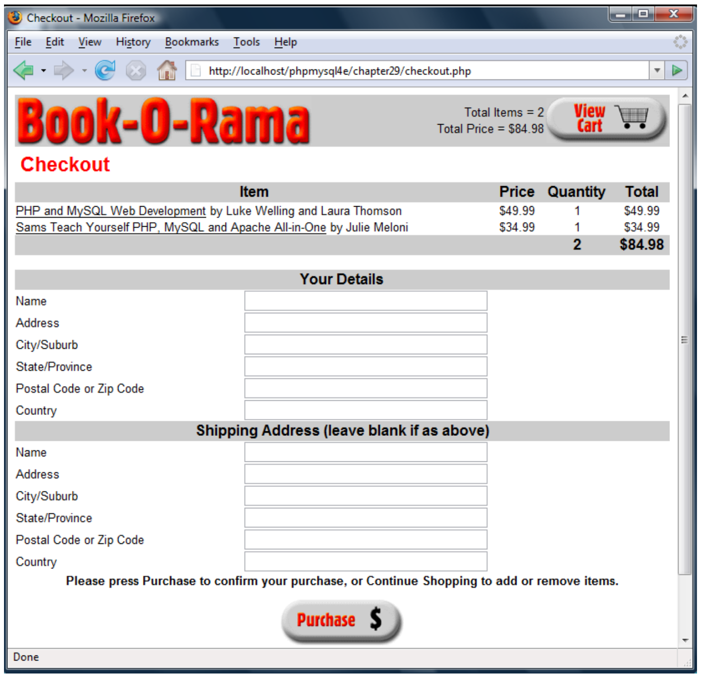
These variables are registered when the user first visits the `show_cart.php` page. You also need some logic to deal with the cases in which a user has not yet visited that page. This logic is also included in the `do_html_heaader()` function:

```
if (!$_SESSION['items']) {
  $_SESSION['items'] = '0';
}
if (!$_SESSION['total_price']) {
  $_SESSION['total_price'] = '0.00';
}
```

Checking Out

When the user clicks the Go to Checkout button from the shopping cart, this action activates the `checkout.php` script. The checkout page and the pages behind it should be accessed via the Secure Sockets Layer (SSL), but the sample application does not force you to do this. (To read more about SSL, review Chapter 15, "Building a Secure Web Application")

The checkout page is shown in Figure 31.8.


Checkout - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/checkout.php
Book-O-Rama
View Cart
Purchase $
Done

Total Items = 2
Total Price = $84.98

Checkout

Item	Price	Quantity	Total
PHP and MySQL Web Development by Luke Welling and Laura Thomson	$49.99	1	$49.99
Sams Teach Yourself PHP, MySQL and Apache All-in-One by Julie Meloni	$34.99	1	$34.99
		2	$84.98

Your Details

Name	
Address	
City/Suburb	
State/Province	
Postal Code or Zip Code	
Country	

Shipping Address (leave blank if as above)

Name	
Address	
City/Suburb	
State/Province	
Postal Code or Zip Code	
Country	

Please press Purchase to confirm your purchase, or Continue Shopping to add or remove items.

Figure 31.8 The `checkout.php` script gets the customer's details.

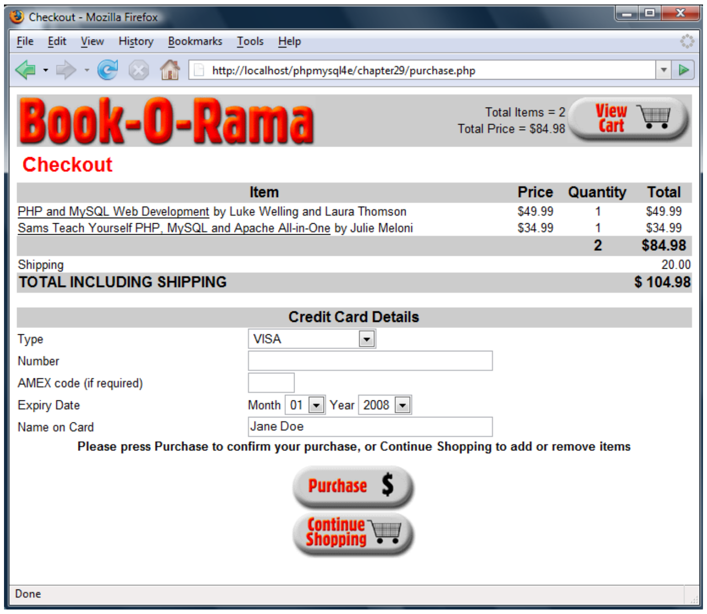
This script requires the customer to enter her address (and shipping address if it is different). It is quite a simple script, which you can see by looking at the code in Listing 31.13.

Listing 31.13 **`checkout.php`— Script That Gets the Customer Details**

```
<?php
  //include our function set
  include ('book_sc_fns.php');
  // The shopping cart needs sessions, so start one
  session_start();

  do_html_header("Checkout");

  if(($_SESSION['cart']) && (array_count_values($_SESSION['cart']))) {
    display_cart($_SESSION['cart'], false, 0);
    display_checkout_form();
  } else {
    echo "<p>There are no items in your cart</p>";
  }

  display_button("show_cart.php", "continue-shopping", "Continue Shopping");

  do_html_footer();
?>
```

There are no great surprises in this script. If the cart is empty, the script will notify the customer; otherwise, it will display the form shown in Figure 31.8.

If a user continues by clicking the Purchase button at the bottom of the form, she will be taken to the `purchase.php` script. You can see the output of this script in Figure 31.9.


Checkout - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/purchase.php
Book-O-Rama
View Cart
Purchase $
Continue Shopping
Done

Total Items = 2
Total Price = $84.98

Checkout

Item	Price	Quantity	Total
PHP and MySQL Web Development by Luke Welling and Laura Thomson	$49.99	1	$49.99
Sams Teach Yourself PHP, MySQL and Apache All-in-One by Julie Meloni	$34.99	1	$34.99
		2	$84.98
Shipping			20.00
TOTAL INCLUDING SHIPPING			$ 104.98

Credit Card Details

Type	VISA
Number	
AMEX code (if required)	
Expiry Date	Month 01 Year 2008
Name on Card	Jane Doe

Please press Purchase to confirm your purchase, or Continue Shopping to add or remove items

Figure 31.9 The `purchase.php` script calculates shipping and the final order total and gets the customer's payment details.

The code for the `purchase.php` script is slightly more complicated than the code for `checkout.php`. It is shown in Listing 31.14.

Listing 31.14 **`purchase.php`—Script That Stores the Order Details in the Database and Gets the Payment Details**

```
<?php

  include ('book_sc_fns.php');
  // The shopping cart needs sessions, so start one
  session_start();

  do_html_header("Checkout");

  // create short variable names
  $name = $_POST['name'];
  $address = $_POST['address'];
  $city = $_POST['city'];
  $zip = $_POST['zip'];
  $country = $_POST['country'];

  // if filled out
  if (($_SESSION['cart']) && ($name) && ($address) && ($city)
           && ($zip) && ($country)) {
    // able to insert into database
    if(insert_order($_POST) != false ) {
      //display cart, not allowing changes and without pictures
      display_cart($_SESSION['cart'], false, 0);

      display_shipping(calculate_shipping_cost());

      //get credit card details
      display_card_form($name);

      display_button("show_cart.php", "continue-shopping", "Continue Shopping");
    } else {
      echo "<p>Could not store data, please try again.</p>";
      display_button('checkout.php', 'back', 'Back');
    }
  } else {
    echo "<p>You did not fill in all the fields, please try again.</p><hr />";
    display_button('checkout.php', 'back', 'Back');
  }

  do_html_footer();
?>
```

The logic here is straightforward: You check that the user filled out the form and inserted details into the database using a function called `insert_order()`. This simple function pops the customer details into the database. The code for it is shown in Listing 31.15.

Listing 31.15 `insert_order()` Function from `order_fns.php`—Function That Inserts All the Details of the Customer's Order into the Database

```
function insert_order($order_details) {
  // extract order_details out as variables
  extract($order_details);

  // set shipping address same as address
  if((!$ship_name) && (!$ship_address) && (!$ship_city)
    && (!$ship_state) && (!$ship_zip) && (!$ship_country)) {
    $ship_name = $name;
    $ship_address = $address;
    $ship_city = $city;
    $ship_state = $state;
    $ship_zip = $zip;
    $ship_country = $country;
  }

  $conn = db_connect();

  // we want to insert the order as a transaction
  // start one by turning off autocommit
  $conn->autocommit(FALSE);

  // insert customer address
  $query = "select customerid from customers where
            name = '".$conn->real_escape_string($name) .
           "' and address = '". $conn->real_escape_string($address)."'
            and city = '".$conn->real_escape_string($city) .
           "' and state = '".$conn->real_escape_string($state)."'
            and zip = '".$conn->real_escape_string($zip) .
         "' and country = '".$conn->real_escape_string($country)."'";

  $result = $conn->query($query);

  if($result->num_rows>0) {
    $customer = $result->fetch_object();
    $customerid = $customer->customerid;
  } else {
    $query = "insert into customers values
            ('', '" . $conn->real_escape_string($name) ."','" .
            $conn->real_escape_string($address) .
            "','". $conn->real_escape_string($city) ."','" .
            $conn->real_escape_string($state) .
            "','". $conn->real_escape_string($zip) ."','" .
            $conn->real_escape_string($country)."')";
    $result = $conn->query($query);

    if (!$result) {
       return false;
    }
  }

  $customerid = $conn->insert_id;

  $date = date("Y-m-d");

  $query = "insert into orders values
            ('', '". $conn->real_escape_string($customerid) . "', '" .
            $conn->real_escape_string($_SESSION['total_price']) .
             "', '". $conn->real_escape_string($date) ."', 'PARTIAL',
             '" . $conn->real_escape_string($ship_name) . "', '" .
            $conn->real_escape_string($ship_address) .
             "', '". $conn->real_escape_string($ship_city)."', '" .
            $conn->real_escape_string($ship_state) ."',
             '". $conn->real_escape_string($ship_zip) . "', '".
            $conn->real_escape_string($ship_country)."')";

  $result = $conn->query($query);
  if (!$result) {
    return false;
  }
```

```
  $query = "select orderid from orders where
               customerid = '". $conn->real_escape_string($customerid)."' and
               amount > (".(float)$_SESSION['total_price'] ."-.001) and
               amount < (". (float)$_SESSION['total_price']."+.001) and
               date = '".$conn->real_escape_string($date)."' and
               order_status = 'PARTIAL' and
               ship_name = '".$conn->real_escape_string($ship_name)."' and
               ship_address = '".$conn->real_escape_string($ship_address)."' and
               ship_city = '".$conn->real_escape_string($ship_city)."' and
               ship_state = '".$conn->real_escape_string($ship_state)."' and
               ship_zip = '".$conn->real_escape_string($ship_zip)."' and
               ship_country = '".$conn->real_escape_string($ship_country)."'";

  $result = $conn->query($query);

  if($result->num_rows>0) {
    $order = $result->fetch_object();
    $orderid = $order->orderid;
  } else {
    return false;
  }

  // insert each book
  foreach($_SESSION['cart'] as $isbn => $quantity) {
    $detail = get_book_details($isbn);
    $query = "delete from order_items where
              orderid = '". $conn->real_escape_string($orderid)."' and isbn = '" .
              $conn->real_escape_string($isbn)."'";
    $result = $conn->query($query);
    $query = "insert into order_items values
              ('". $conn->real_escape_string($orderid) ."', '" .
              $conn->real_escape_string($isbn) .
              "', ". $conn->real_escape_string($detail['price']) .", " .
              $conn->real_escape_string($quantity). ")";
    $result = $conn->query($query);
    if(!$result) {
      return false;
    }
  }

  // end transaction
  $conn->commit();
  $conn->autocommit(TRUE);

  return $orderid;
}
```

The `insert_order()` function is rather long because you need to insert the customer's details, order details, and details of each book she wants to buy.

You will note that the different parts of the insert are enclosed in a transaction, beginning with

```
$conn->autocommit(FALSE);
```

and ending with

```
$conn->commit();
$conn->autocommit(TRUE);
```

This is the only place in this application where you need to use a transaction. How do you avoid having to do it elsewhere? Look at the code in the `db_connect()` function:

```
function db_connect() {
   $result = new mysqli('localhost', 'book_sc', 'password', 'book_sc');
   if (!$result) {
      return false;
   }
   $result->autocommit(TRUE);
   return $result;
}
```

Obviously, this is slightly different from the code used for this function in other chapters. After creating the connection to MySQL, you should turn on auto-commit mode. This ensures that each SQL statement is automatically committed, as we have previously discussed. When you actually want to use a multi-statement transaction, you turn off auto-commit, perform a series of inserts, commit the data, and then re-enable auto-commit mode.

You then work out the shipping costs to the customer's address and tell her how much it will be with the following line of code:

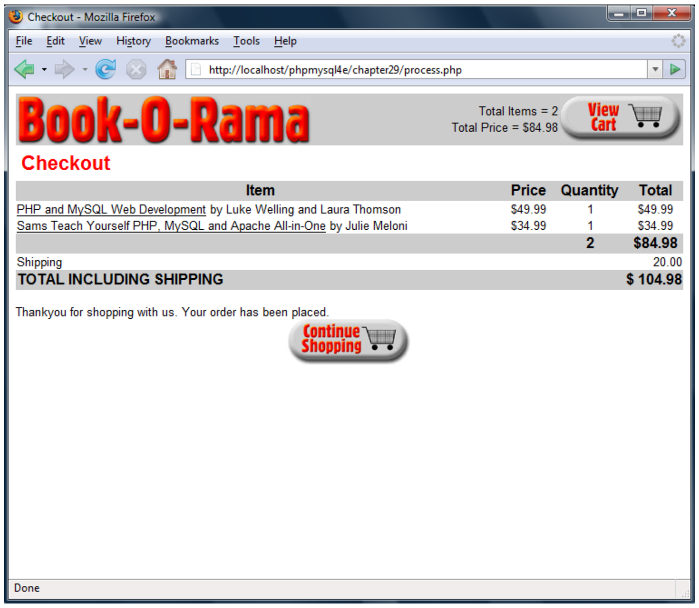
```
display_shipping(calculate_shipping_cost());
```

The code used here for `calculate_shipping_cost()` always returns $20. When you actually set up a shopping site, you must choose a delivery method, find out how much shipping costs for different destinations, and calculate those costs accordingly.

You then display a form for the user to fill in her credit card details by using the `display_card_form()` function from the `output_fns.php` library.

Implementing Payment

When the user clicks the Purchase button, you process her payment details using the `process.php` script. You can see the results of a successful payment in Figure 31.10.


Checkout - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/process.php
Book-O-Rama
View Cart
Continue Shopping
Done

Total Items = 2
Total Price = $84.98

Checkout

Item	Price	Quantity	Total
PHP and MySQL Web Development by Luke Welling and Laura Thomson	$49.99	1	$49.99
Sams Teach Yourself PHP, MySQL and Apache All-in-One by Julie Meloni	$34.99	1	$34.99
		2	$84.98
Shipping			20.00
TOTAL INCLUDING SHIPPING			$ 104.98

Thankyou for shopping with us. Your order has been placed.

Figure 31.10 This transaction was successful, and the items will now be shipped.

The code for `process.php` can be found in Listing 31.16.

Listing 31.16 **`process.php`— Script That Processes the Customer's Payment and Tells Her the Result**

```
<?php
  include ('book_sc_fns.php');
  // The shopping cart needs sessions, so start one
  session_start();

  do_html_header('Checkout');

  $card_type = $_POST['card_type'];
  $card_number = $_POST['card_number'];
  $card_month = $_POST['card_month'];
  $card_year = $_POST['card_year'];
  $card_name = $_POST['card_name'];

  if(($_SESSION['cart']) && ($card_type) && ($card_number) &&
     ($card_month) && ($card_year) && ($card_name)) {
    //display cart, not allowing changes and without pictures
    display_cart($_SESSION['cart'], false, 0);

    display_shipping(calculate_shipping_cost());

    if(process_card($_POST)) {
      //empty shopping cart
      session_destroy();
      echo "<p>Thank you for shopping with us. Your order has been placed.</p>";
      display_button("index.php", "continue-shopping", "Continue Shopping");
    } else {
      echo "<p>Could not process your card. Please contact the card
            issuer or try again.</p>";
      display_button("purchase.php", "back", "Back");
    }
  } else {
    echo "<p>You did not fill in all the fields, please try again.</p><hr />";
    display_button("purchase.php", "back", "Back");
  }

  do_html_footer();
?>
```

You process the user's card and, if all is successful, destroy her session.

The card processing function as it is written simply returns `true`. If you were actually implementing it, you would need to perform some validation (checking that the expiry date was valid and the card number well formed) and then process the actual payment.

Implementing an Administration Interface

The administration interface we implemented is very simple. We just built a Web interface to the database with some front-end authentication based on similar examples elsewhere in the book. We included it here for completeness, but with little discussion.

The administration interface requires a user to log in via the `login.php` file, which then takes him to the administration menu, `admin.php`. The login page is shown in Figure 31.11. The administration menu is shown in Figure 31.12.

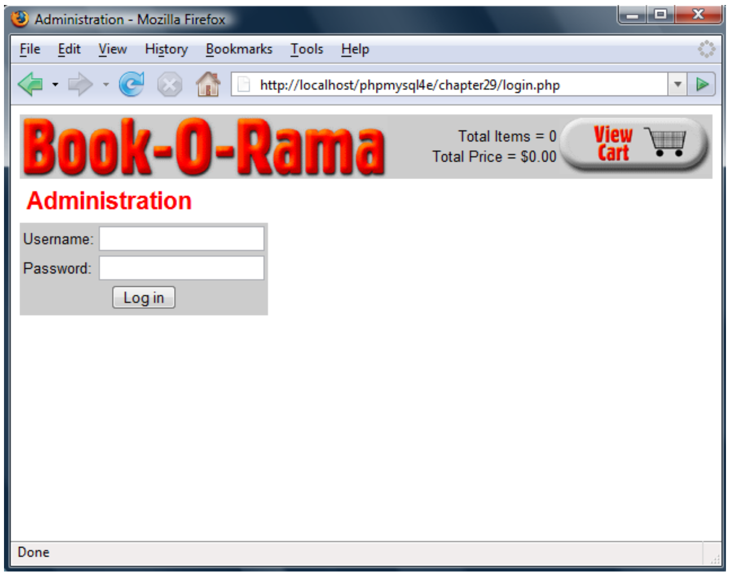

Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/login.php
Book-O-Rama
Total Items = 0
Total Price = $0.00
View Cart
Administration
Username:
Password:
Log in
Done

Figure 31.11 Users must pass through the login page to access the administration functions.

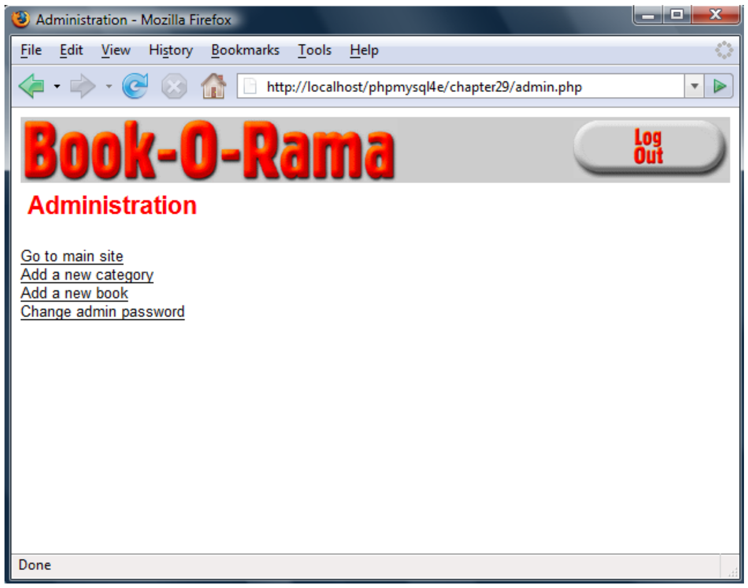

Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/admin.php
Book-O-Rama
Log Out
Administration
Go to main site
Add a new category
Add a new book
Change admin password
Done

Figure 31.12 The administration menu allows access to the administration functions.

The code for the admin menu is shown in Listing 31.17.

Listing 31.17 `admin.php`—Script That Authenticates the Administrator and Lets Him Access the Administration Functions

```
<?php
```

```
// include function files for this application
require_once('book_sc_fns.php');
session_start();


if (($_POST['username']) && ($_POST['passwd'])) {
    // they have just tried logging in

    $username = $_POST['username'];
    $passwd = $_POST['passwd'];

    if (login($username, $passwd)) {
      // if they are in the database register the user id
      $_SESSION['admin_user'] = $username;

   } else {
      // unsuccessful login
      do_html_header("Problem:");
      echo "<p>You could not be logged in.<br/>
            You must be logged in to view this page.</p>";
      do_html_url('login.php', 'Login');
      do_html_footer();
      exit;
    }
}

do_html_header("Administration");
if (check_admin_user()) {
  display_admin_menu();
} else {
  echo "<p>You are not authorized to enter the administration area.</p>";
}
do_html_footer();
?>
```

After the administrator reaches this point, he can change his password or log out as demonstrated in previous chapters.

You identify the administration user after login by means of the `admin_user` session variable and the `check_admin_user()` function. This function and the others used by the administrative scripts can be found in the function library `admin_fns.php`.

If the administrator chooses to add a new category or book, he will go to either `insert_category_form.php` or `insert_book_form.php`, as appropriate. Each of these scripts presents the administrator with a form to fill in. Each is processed by a corresponding script (`insert_category.php` and `insert_book.php`), which verifies that the form is filled out and inserts the new data into the database. Here, we look at the book versions of the scripts only because they are similar to one another.

The output of `insert_book_form.php` is shown in Figure 31.13.

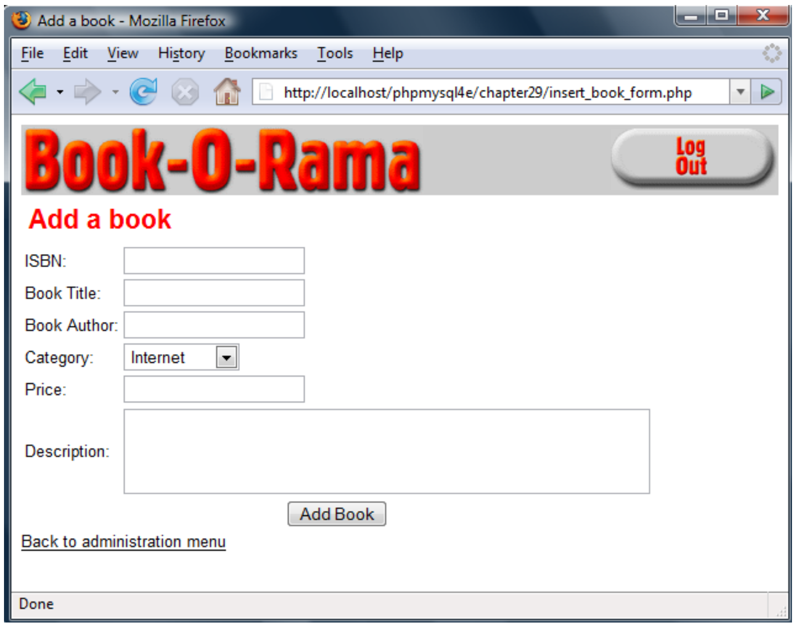

Add a book - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/insert_book_form.php
Book-O-Rama
Log Out
Add a book
ISBN:
Book Title:
Book Author:
Category:
Internet
Price:
Description:
Add Book
Back to administration menu
Done

Figure 31.13 This form allows the administrator to enter new books into the online catalog.

Notice that the `Category` field for books is an HTML `SELECT` element. The options for this `SELECT` come from a call to the `get_categories()` function you looked at previously.

When the Add Book button is clicked, the `insert_book.php` script is activated. The code for this script is shown in Listing 31.18.

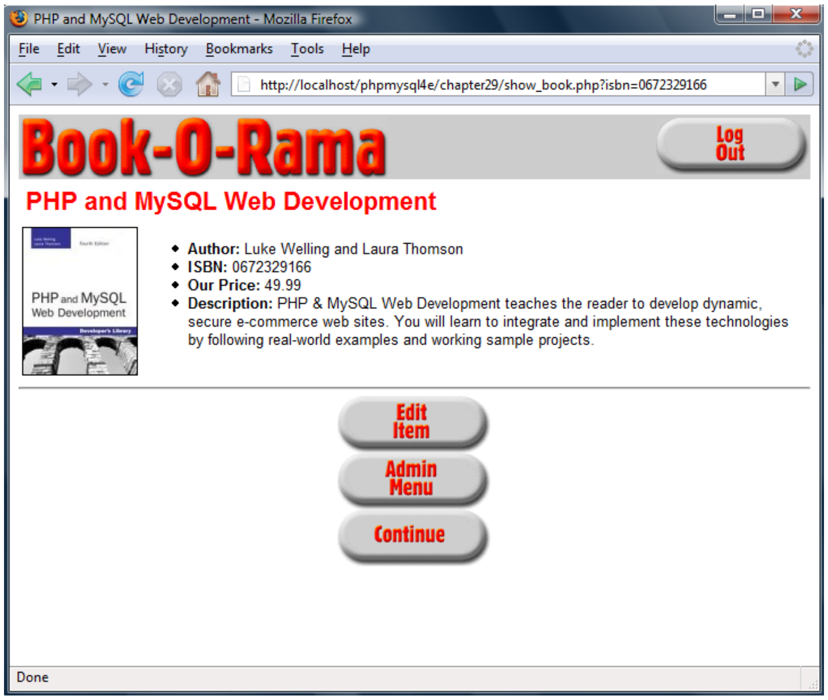
Listing 31.18 `insert_book.php`**—Script That Validates the New Book Data and Puts It into the Database**

```
<?php

// include function files for this application
require_once('book_sc_fns.php');
session_start();

do_html_header("Adding a book");
if (check_admin_user()) {
  if (filled_out($_POST)) {
    $isbn = $_POST['isbn'];
    $title = $_POST['title'];
    $author = $_POST['author'];
    $catid = $_POST['catid'];
    $price = $_POST['price'];
    $description = $_POST['description'];

    if(insert_book($isbn, $title, $author, $catid, $price, $description)) {
      echo "<p>Book <em>".htmlspecialchars($title)."</em> was added to the database.</p>";
    } else {
      echo "<p>Book <em>".htmlspecialchars($title)."</em> could not be added to the database.</p>";
    }
  } else {
    echo "<p>You have not filled out the form.  Please try again.</p>";
  }

  do_html_url("admin.php", "Back to administration menu");
} else {
  echo "<p>You are not authorised to view this page.</p>";
}

do_html_footer();

?>
```

You can see that this script calls the function `insert_book()`. This function and the others used by the administrative scripts can be found in the function library `admin_fns.php`.

In addition to adding new categories and books, the administrative user can edit and delete these items. We implemented this capability by reusing as much code as possible. When the administrator clicks the Go to Main site link in the administration menu, he goes to the category index at `index.php` and can navigate the site in the same way as a regular user, using the same scripts.

There is a difference in the administrative navigation, however: Administrators see different options based on the fact that they have the registered session variable `admin_user`. For example, if you look at the `show_book.php` page that you looked at previously in the chapter, you will see the different menu options shown in Figure 31.14.


PHP and MySQL Web Development - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/show_book.php?isbn=0672329166
Book-O-Rama
Log Out
Edit Item
Admin Menu
Continue
Done

PHP and MySQL Web Development

- **Author:** Luke Welling and Laura Thomson
- **ISBN:** 0672329166
- **Our Price:** 49.99
- **Description:** PHP & MySQL Web Development teaches the reader to develop dynamic, secure e-commerce web sites. You will learn to integrate and implement these technologies by following real-world examples and working sample projects.

Figure 31.14 The `show_book.php` script produces different output for an administrative user.

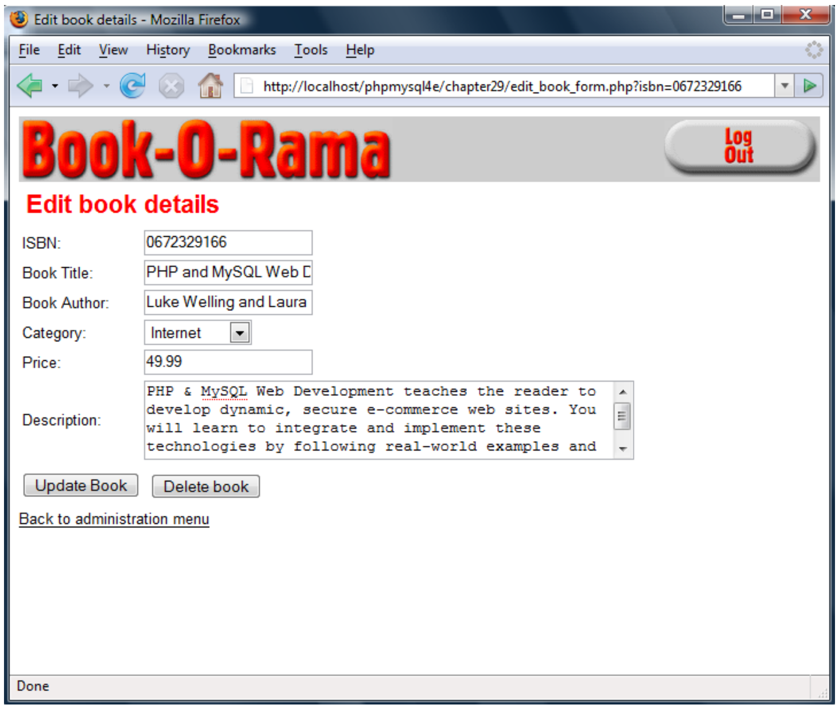
The administrator has access to two new options on this page: Edit Item and Admin Menu. Notice that the shopping cart does not appear in the upper-right corner; instead, this page has a Log Out button.

The code for this page is all there, back in Listing 31.8, as follows:

```
if(check_admin_user()) {
    display_button("edit_book_form.php?isbn=". urlencode($isbn),
    "edit-item", "Edit Item");
    display_button("admin.php", "admin-menu", "Admin Menu");
    display_button($target, "continue", "Continue");
  }
```

If you look back at the `show_cat.php` script, you will see that it also has these options built into it.

If the administrator clicks the Edit Item button, he will go to the `edit_book_form.php` script. The output of this script is shown in Figure 31.15.


Edit book details - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/phpmysql4e/chapter29/edit_book_form.php?isbn=0672329166
Book-O-Rama
Log Out
Update Book
Delete book
Back to administration menu
Done

Edit book details

ISBN:	0672329166
Book Title:	PHP and MySQL Web D
Book Author:	Luke Welling and Laura
Category:	Internet
Price:	49.99
Description:	PHP & MySQL Web Development teaches the reader to develop dynamic, secure e-commerce web sites. You will learn to integrate and implement these technologies by following real-world examples and

Figure 31.15 The `edit_book_form.php` script gives the administrator access to edit book details or delete a book.

This form is, in fact, the same one used to get the book's details in the first place. We built an option into that form to pass in and display existing book data. We did the same thing with the category form. To see what we mean, look at Listing 31.19.

Listing 31.19 **`display_book_form()` Function from `admin_fns.php`—Form That Does Double Duty as an Insertion and Editing Form**

```
function display_book_form($book = '') {
// This displays the book form.
// It is very similar to the category form.
// This form can be used for inserting or editing books.
// To insert, don't pass any parameters.  This will set $edit
// to false, and the form will go to insert_book.php.
// To update, pass an array containing a book.  The
// form will be displayed with the old data and point to update_book.php.
// It will also add a "Delete book" button.


  // if passed an existing book, proceed in "edit mode"
  $edit = is_array($book);

  // most of the form is in plain HTML with some
  // optional PHP bits throughout
?>
  <form method="post"
        action="<?php echo $edit ? 'edit_book.php' : 'insert_book.php';?>">
  <table border="0">
  <tr>
    <td>ISBN:</td>
    <td><input type="text" name="isbn"
         value="<?php echo htmlspecialchars($edit ? $book['isbn'] : ''); ?>" /></td>
  </tr>
  <tr>
    <td>Book Title:</td>
    <td><input type="text" name="title"
         value="<?php echo htmlspecialchars($edit ? $book['title'] : ''); ?>" /></td>
  </tr>
  <tr>
    <td>Book Author:</td>
    <td><input type="text" name="author"
         value="<?php echo htmlspecialchars($edit ? $book['author'] : ''); ?>" /></td>
   </tr>
   <tr>
      <td>Category:</td>
      <td><select name="catid">
      <?php
          // list of possible categories comes from database
          $cat_array=get_categories();
          foreach ($cat_array as $thiscat) {
               echo "<option value=\"".htmlspecialchars($thiscat['catid'])."\"";
               // if existing book, put in current catgory
               if (($edit) && ($thiscat['catid'] == $book['catid'])) {
                   echo " selected";
               }
               echo ">".htmlspecialchars($thiscat['catname'])."</option>";
          }
          ?>
          </select>
        </td>
   </tr>
   <tr>
    <td>Price:</td>
    <td><input type="text" name="price"
               value="<?php echo htmlspecialchars($edit ? $book['price'] : ''); ?>" /></td>
   </tr>
   <tr>
     <td>Description:</td>
     <td><textarea rows="3" cols="50"
          name="description"><?php echo htmlspecialchars($edit ? $book['description'] : '');
?></textarea></td>
    </tr>
    <tr>
      <td <?php if (!$edit) { echo "colspan=2"; }?> align="center">
         <?php
            if ($edit)
             // we need the old isbn to find book in database
             // if the isbn is being updated
             echo "<input type=\"hidden\" name=\"oldisbn\"
                    value=\"".htmlspecialchars($book['isbn'])."\" />";
         ?>
        <input type="submit"
               value="<?php echo $edit ? 'Update' : 'Add'; ?> Book" />
        </form></td>
        <?php
```

```
          if ($edit) {
            echo "<td>
                  <form method=\"post\" action=\"delete_book.php\">
                  <input type=\"hidden\" name=\"isbn\"
                   value=\"".htmlspecialchars($book['isbn'])."\" />
                  <input type=\"submit\" value=\"Delete book\"/>
                  </form></td>";
           }
          ?>
         </td>
      </tr>
  </table>
  </form>
<?php
}
```

If you pass in an array containing the book data, the form will be rendered in edit mode and will fill in the fields with the existing data:

```
<input type="text" name="price"
              value="<?php echo htmlspecialchars($edit ? $book['price'] : ''); ?>" />
```

You even get a different submit button. In fact, for the edit form, you get two—one to update the book and one to delete it. These buttons call the scripts `edit_book.php` and `delete_book.php`, which update the database accordingly.

The category versions of these scripts work in much the same way, except for one thing. When an administrator tries to delete a category, it will not be deleted if any books are still in it. (This is checked with a database query.) This approach prevents any problems you might get with deletion anomalies. We discussed these anomalies in Chapter 8, "Designing Your Web Database." In this case, if a category that still had books in it was deleted, these books would become orphans. You wouldn't know what category they were in, and you would have no way of navigating to them!

That's the overview of the administration interface.

Extending the Project

If you followed along with this project, you have built a fairly simple shopping cart system. There are many additions and enhancements you could make:

- In a real online store, you would need to build some kind of order tracking and fulfillment system. At the moment, you have no way of seeing the orders that have been placed.
- Customers want to be able to check the progress of their orders without having to contact you. We feel that it is important that a customer does not have to log in to browse. However, providing existing customers a way to authenticate themselves enables them to see past orders and enables you to tie behaviors together into a profile.
- At present, the images for books have to be transferred via FTP to the image directory and given the correct name. You could add file upload to the book insertion page to make this process easier.
- You could add user login, personalization, and book recommendations; online reviews; affiliate programs; stock level checking; and so on. The possibilities are endless.