# POTENTIAL ALTERATIONS

This chapter covers everything you need to know to implement a real-world e-commerce product catalog that has a decent amount of complexity. As always, there are many other ways you could present the catalog, other features you could implement, and other stylistic choices you could make. I'll highlight a few of those here; no doubt you can come up with several others on your own.

## Paginating Results

One feature I have not incorporated into this site is *pagination*: making a list of items appear in smaller groupings—like 10 to 20 at a time—over multiple pages. For starters, the page for listing every coffee product does so using a single drop-down menu, so no pagination is required there. As for the non-coffee products, the site would have to get fairly large for it to require pagination. But should you want to add that feature, it's quite simple to implement. If you don't already know how, I discuss the concept in detail in my book, *PHP and MySQL: Visual QuickStart Guide* (Peachpit Press), or you can simply ask how in my support forums (**www.DMCInsights.com/phorum/**).

With the *Coffee* site, there's one complication to using pagination: the pretty URLs. If you're using **mod_rewrite**, you'll need to modify the **.htaccess** rules so that it recognizes and handles the pagination variables that get passed along in the URL.

## Highlighting New Products

You may want to highlight new products to make the site look fresh and busy. You might do so by:

- Showing the most recent three or four products on the home page
- Displaying the most recent three or four products in a given category on the **shop.php** page
- Creating a separate "New Items" page that lists every product added within the past X days or weeks

The paradigm for implementing any of these has already been created: It's essentially the same code and queries as used to display sale items, just without using the **sales** table and ordering the results by the creation dates.

## Directly Linking to Product Categories

**tip**

This technique is known as a *Suckerfish Menu*, and is quickly demonstrated in Chapter 11.

One reasonable complaint about this site's design is that, except for the sale items, the customer must click through two pages to get to actual products they can purchase (first the shop page, then browse). An easy fix would be to change the navigation so that it allows for direct access to the categories.

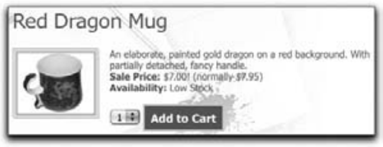
To do this, you'd convert the main navigation menu into a nested unordered list. You'd use PHP to generate the sublists, but the desired HTML would be:

```
<ul class="nav">
    <li><a href="/shop/coffee/">Coffee</a>
        <ul>
            <li><a href="/browse/coffee/Dark+Roast/2">Dark Roast</a></li>
            <li><a href="/browse/coffee/Kona/3">Kona</a></li>
            <li><a href="/browse/coffee/Original+Blend+1">Original
            ➥Blend</a></li>
        </ul>
    </li>
    <li><a href="/shop/goodies/">Goodies</a></li>
    <li><a href="/shop/sales/">Sales</a></li>
    <li><a href="/wishlist.php">Wish List</a></li>
    <li><a href="/cart.php">Cart</a></li>
</ul>
```

What you see for the specific coffee types would need to be applied to the categories of goodies, too. You would then apply a combination of CSS and/or JavaScript so that the sublists are only shown when the user mouses over the primary links. That specific code can be found by searching online for "Suckerfish Menu," although you'll need to tweak the offered CSS so it fits in with the design of the template.

## Adding Multiple Coffees or Quantities at Once

It's not a big deal, but the site as written will let the customer add only a single quantity of a single product to their cart at a time. As for the quantities, the customer will be able to easily update that in the cart itself. But if you wanted to allow for entering a quantity, you would have to turn the products listing into a form, as in the coffee listing, and add a **<SELECT>** menu or text box for the quantity (**Figure 8.20**). Most importantly, the form must also store the product's SKU in a hidden input:


Red Dragon Mug

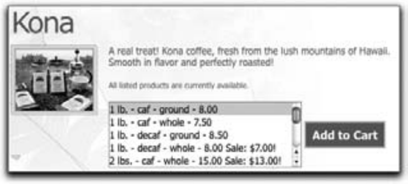
An elaborate, painted gold dragon on a red background. With partially detached, fancy handle.
**Sale Price:** $7.00! (normally $7.95)
**Availability:** Low Stock

Figure 8.20

```
echo '<form action="/cart.php" method="get">
<input type="hidden" name="action" value="add" />
<input type="hidden" name="sku" value="' . $row['sku'] . '" />
<h3>' . $row['name'] . '</h3>
<div class="img-box">
    <p><img alt="' . $row['name'] . '" src="/products/' . $row['image']
    ➥. '" />' . $row['description'] . '<br />' .
    get_price($type, $row['price'], $row['sale_price']) .
    '<strong>Availability:</strong> ' . get_stock_status($row['stock']) . '</p>
    <select name="qty"><option>1</option><option>2</option></select>
    <input type="submit" value="Add to Cart" class="button" />
</div></form>';
```

With the specific coffees, there are other ways you could go about presenting them. You could list the specific coffees as a **<SELECT>** menu that allows multiple selections. Here's what you would have in **list_coffees.html** (Figure 8.21):

```
<select name="sku[ ]" multiple="multiple" size="5">
```


Kona

A real treat! Kona coffee, fresh from the lush mountains of Hawaii. Smooth in flavor and perfectly roasted!

All listed products are currently available.

```
1 lb. - caf - ground - 8.00
1 lb. - caf - whole - 7.50
1 lb. - decaf - ground - 8.50
1 lb. - decaf - whole - 8.00 Sale: $7.00!
2 lbs. - caf - whole - 15.00 Sale: $13.00!
```

Figure 8.21

Note that I've changed the name of the item from *sku* to *sku[ ]*, so that an array of products will be sent to **cart.php**.

If you wanted, you could instead list the coffees as a series of check boxes or with a corresponding series of text boxes, into which the user enters the quantity desired. In either of these two cases, you would still have to use *sku[ ]* as the name for every form element.

As you'll see in the next chapter, you'd need to modify **cart.php** to handle an incoming array, if you were to apply any of these three modifications to how the coffee products are listed. If you were to add a quantity option to coffee or non-coffee products, that would need to be factored in an updated **cart.php**, too.

## Larger Images

Depending upon what your e-commerce site is selling, you may want to implement a feature so that the site supports product images in multiple sizes. This wasn't really necessary in this example—a large picture of coffee beans is no more likely to bag a sale.

At the very least, a site may want its product images to be available in three sizes:

- Thumbnail, for use in places like the home page in this example
- Regular, for use on product listing pages
- Large, which would appear above the page when the user clicks one of the smaller images

**tip**

You can also manipulate images in PHP by making system calls to the ImageMagick library, if your server supports it.

The administrative interface could be designed to handle three image sizes in one of two ways. First, it could be up to the administrator to provide all three images in set sizes. Second, the administrator could submit one large image, which the PHP script would then dynamically resize to create the smaller versions. The latter would be preferable, although this requires quite a bit more PHP code and support for the GD image manipulation library.

Second, you'll need to decide how to represent the various images on the server. You could use the same filename for each but store them in three separate folders: **/products**, **/products/thumbs/**, and **/products/large**. You could then use PHP to confirm the existence of the product image in a given size, before attempting to use that image in the HTML.

**tip**

Another option is to store images in **BLOB** columns in the database itself. There are arguments for and against this practice, though.

Alternatively, you could create a separate **images** table in the database:

```
CREATE TABLE `images` (
id INT UNSIGNED NOT NULL AUTO_INCREMENT,
product_type ENUM('coffee', 'other'),
product_id SMALLINT UNSIGNED NOT NULL,
image VARCHAR(45) NOT NULL,
image_type ENUM('thumb', 'regular', 'large', 'alternative'),
PRIMARY KEY (id),
KEY product (product_type, product_id),
KEY (image_type)
);
```

The table would have a many-to-many relationship with both the **non_coffee_products** and **specific_coffees** tables (just like the **sales** table). For each image type that exists for a given product, a new record would be created in this table. The structure also allows for the submission of alternative images, such as the same shirt in a different color or a view of a product from another perspective. Of course, by adding this table, you'll need to update the **select_products()** stored procedure so that it performs a **JOIN** across **images**, too. And you'd likely want to create a new product function that generates the HTML for showing images based upon what images exist for that product.

**tip**

Thanks to the MVC design, many features can be added by updating the model (the stored procedure) and the view (the HTML) without touching the base PHP script (the controller).