POTENTIAL ALTERATIONS

As with several other chapters in this book, this one concludes with some variations on what you can do with this chapter's content. None of these alterations are obligatory or necessarily better than what's already been done, just worth considering.

Making Recommendations

Part of a good shopping cart, at least prior to the customer completing the sale, is recommending other products to purchase. Generously said, recommendations offer an additional benefit to the customer, because they truly may be interested in other items and appreciate those items being brought to their attention. Cynically, it could mean more money for the business.

There are two broad types of recommendations:

- *Upselling*: recommending similar products that are better and more expensive
- *Cross-selling*: recommending related products that the customer might want in addition

Upselling in this particular site is simple: If a customer has a one-pound bag of coffee in their cart, recommend the two-pound bag. Or if your site sells audio visual equipment, upselling might be recommending other models (like a receiver or DVD player) by the same manufacturer.

Cross-selling has a larger potential—not everything can be upsold—but requires more thought. When a site doesn't have that many products or that large of an order history, you can implement recommendations by creating a system whereby the administrator makes associations among products. If the site sells a lot of stuff, this might become impractical, so if you have a number of orders in the system, you can create automatic recommendations based on what other customers have purchased. The premise, and the underlying code, is simple: If a customer has, say, a bag of Kona coffee in their cart, recommended products would be those things that other customers purchased in addition to Kona coffee. You could even define the strength of a recommendation based on how often it is purchased along with the original product.

Creating Add to Wish List Links

A simple change you could make to the site would be to include *Add to Wish List* links beside products, just like the *Add to Cart* links. With the **wishlist.php** script as written, you would just need to create an "add" action conditional, like the one in **cart.php**.

Shipping Alternatives

Shipping, like the choice of payment processor itself, is such a big topic that I could arguably dedicate an entire chapter to the myriad of ways to handle this part of an order. The simplest, but clearly not the best, way to handle shipping is to not charge anything additional at all: Just factor enough profit into each item sold to cover the expense. The site that does this will run the risk of losing business to other sites that overtly charge less for the same item, even though those sites will later add in shipping charges. Also, this approach would not allow for different shipping options (such as the speed of delivery) or easy adjustments to the cost of shipping as they change over time.

The second simplest way to calculate shipping is implemented here: a proportional amount dictated by the order total. This approach is easy to manage, easy to change, and reasonable, both for the business and for the customer.

To calculate shipping based on the weight of the order, you'd need to modify the database so that the weight of items is recorded along with the other product details. Depending on what you're selling, you'd be best off representing all weights in the same unit: grams, kilograms, ounces, pounds, what have you. The shopping cart would then need to retrieve the weight for each product, generate a weight total, and then calculate the shipping using the total weight.

On a similar note, you could create an additional shipping cost representative column in the database. This could be a column added to the specific products tables (**non_coffee_products** and **specific_coffees**, accordingly), in which case there would expect to be a lot of **NULL** values, which is not ideal. Alternatively, you could create a new table that represents each product that has an additional shipping cost as one row:

```
CREATE TABLE `extra_shipping` (
    `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
    `product_type` ENUM('coffee','other') NOT NULL,
    `product_id` MEDIUMINT UNSIGNED NOT NULL,
    `extra_charge` DECIMAL(4,2) UNSIGNED NOT NULL,
    `date_created` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `date_modified` TIMESTAMP NOT NULL DEFAULT '0000-00-00 00:00:00',
    PRIMARY KEY (`id`),
    KEY `product_type` (`product_type`,`product_id`),
) ENGINE=MyISAM  DEFAULT CHARSET=utf8;
```

Unfortunately, this would mean another table joined into many of the **SELECT** queries.

The most complicated way of calculating shipping is based on the distance (and possibly the weight or size as well). To pull this off, you'd need the customer's postal code and, for international orders, country. You would then tie into the system developed by your shipping company of choice. For example, UPS and FedEx both have Application Programming Interfaces (APIs) available through which you can get exact prices on shipping based upon the distance, the weight, the size, and the delivery speed. These APIs work quite similarly to the payment gateway API. For more information, see the documentation for the shipping company of your choosing.

Improving the Cart Display

The success of the site will depend, in some small part, on the user's reaction to the shopping cart. If it's nice and inviting and makes the customer comfortable, they're more likely to complete the sale. With that in mind, you may want to put some effort into improving that interface. For example, you should probably consider creating links from the products in the cart to the product's image and description in the site. Customers often like being able to revisit what they're buying.

Second, you could add messages to the cart page to indicate the result of the latest action. The message could range from something as simple as "The cart has been updated." to something more specific like "Mugs::Red Dragon has been removed from your shopping cart." To do this, the HTML view file will need to check for and display a message:

```
if ($message) echo $message;
```

Then the PHP script would assign a value to **$message** for each action. If you want to refer to specific products by name, you'd also need to create a stored procedure that retrieved the product information for a given product type and ID. Such a procedure could then be called after an **INSERT**, **UPDATE**, or **DELETE** query is executed.

Tweaking the Database

tip

On *nix systems, **cron** can be used to execute a PHP script automatically at periodic intervals. Such a script can be used to perform maintenance.

The foundation of the Web site is the database, so I'd be remiss not to mention alternatives there. To start, the system as written will create a lot of flotsam: wish list and shopping cart items never to be purchased. You would likely want to create a PHP (or command-line) script that routinely rids the database of old stuff. A record is old if its modification date is more than, say, six months old or if its creation date is more than six months old and its modification date is still 0000-00-00 00:00:00, meaning the record had never been updated.

Second, if you're using the stored procedures and like how they work, you should probably read up on how to handle errors in stored procedures. While not hard to do, the topic is large and technical enough that I had to omit it from the book, lest I took away from the more important points.

Finally, you could get much better performance from the database by taking advantage of **VIEW** tables. A **VIEW** table is a memorized **SELECT** query that you can run other queries on as if it were a real table. The syntax for creating a **VIEW** is:

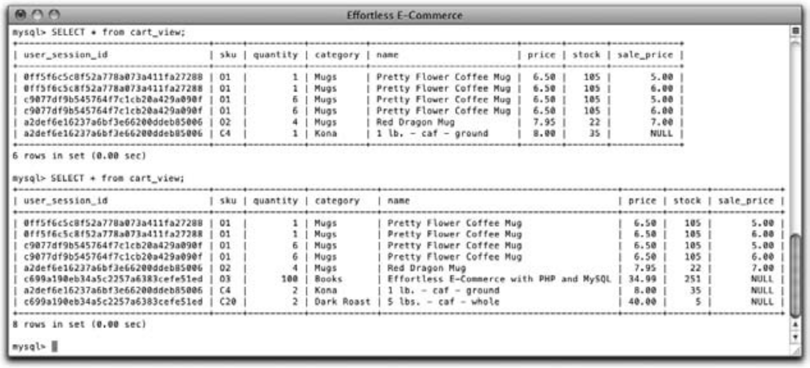
CREATE VIEW *view_name* AS *<SELECT QUERY>*

As an example, the **UNION** used in the procedures for retrieving every shopping cart or wish list item is quite demanding. You could create a **VIEW** that performs all the **JOIN**s, effectively replacing the **product_type** and **product_id** values from the database tables with the actual information you want to display in the Web browser. The view would also store the associated user session ID values.

```
CREATE VIEW cart_view AS
SELECT user_session_id, CONCAT("O", ncp.id) AS sku, c.quantity,
➥ncc.category, ncp.name, ncp.price, ncp.stock, sales.price AS sale_price
➥FROM carts AS c INNER JOIN non_coffee_products AS ncp ON
➥c.product_id=ncp.id INNER JOIN non_coffee_categories AS ncc ON
➥ncc.id=ncp.non_coffee_category_id LEFT OUTER JOIN sales ON (
➥sales.product_id=ncp.id AND sales.product_type='other' AND ((NOW()
➥BETWEEN sales.start_date AND sales.end_date) OR (NOW() >
➥sales.start_date AND sales.end_date IS NULL)) ) WHERE
➥c.product_type="other" UNION SELECT user_session_id,
➥CONCAT("C", sc.id), c.quantity, gc.category, CONCAT_WS(" - ", s.size,
➥sc.caf_decaf, sc.ground_whole), sc.price, sc.stock, sales.price FROM
➥carts AS c INNER JOIN specific_coffees AS sc ON c.product_id=sc.id
➥INNER JOIN sizes AS s ON s.id=sc.size_id INNER JOIN general_coffees
```

```
➥AS gc ON gc.id=sc.general_coffee_id LEFT OUTER JOIN sales ON
➥(sales.product_id=sc.id AND sales.product_type='coffee' AND ((NOW()
➥BETWEEN sales.start_date AND sales.end_date) OR (NOW() >
➥sales.start_date AND sales.end_date IS NULL)) ) WHERE
➥c.product_type="coffee";
```

To be clear, the **SELECT...UNION...SELECT** query is the same as the one in the **get_shopping_cart_contents()** stored procedure, except that the **user_session_id** value is now part of the selection, instead of part of the **WHERE** condition. **Figure 9.11** shows two **SELECT** queries run on this **VIEW** table, with the latter automatically reflecting changes in the **carts** table (due to customer actions).



```
mysql> SELECT * from cart_view;
+----------------------------------+-----+----------+----------+--------------------------+-------+-------+------------+
| user_session_id                  | sku | quantity | category | name                     | price | stock | sale_price |
+----------------------------------+-----+----------+----------+--------------------------+-------+-------+------------+
| 0ff5f6c5c8f52a778a073a411fa27288 | 01  |        1 | Mugs     | Pretty Flower Coffee Mug |  6.50 |   105 |       5.00 |
| 0ff5f6c5c8f52a778a073a411fa27288 | 01  |        1 | Mugs     | Pretty Flower Coffee Mug |  6.50 |   105 |       6.00 |
| c9077df9b545764f7c1cb20a429a090f | 01  |        6 | Mugs     | Pretty Flower Coffee Mug |  6.50 |   105 |       5.00 |
| c9077df9b545764f7c1cb20a429a090f | 01  |        6 | Mugs     | Pretty Flower Coffee Mug |  6.50 |   105 |       6.00 |
| a2def6e16237a6bf3e66200ddeb85006 | 02  |        4 | Mugs     | Red Dragon Mug           |  7.95 |    22 |       7.00 |
| a2def6e16237a6bf3e66200ddeb85006 | C4  |        1 | Kona     | 1 lb. - caf - ground     |  8.00 |    35 |       NULL |
+----------------------------------+-----+----------+----------+--------------------------+-------+-------+------------+
6 rows in set (0.00 sec)

mysql> SELECT * from cart_view;
+----------------------------------+-----+----------+------------+------------------------------------------+-------+-------+------------+
| user_session_id                  | sku | quantity | category   | name                                     | price | stock | sale_price |
+----------------------------------+-----+----------+------------+------------------------------------------+-------+-------+------------+
| 0ff5f6c5c8f52a778a073a411fa27288 | 01  |        1 | Mugs       | Pretty Flower Coffee Mug                 |  6.50 |   105 |       5.00 |
| 0ff5f6c5c8f52a778a073a411fa27288 | 01  |        1 | Mugs       | Pretty Flower Coffee Mug                 |  6.50 |   105 |       6.00 |
| c9077df9b545764f7c1cb20a429a090f | 01  |        6 | Mugs       | Pretty Flower Coffee Mug                 |  6.50 |   105 |       5.00 |
| c9077df9b545764f7c1cb20a429a090f | 01  |        6 | Mugs       | Pretty Flower Coffee Mug                 |  6.50 |   105 |       6.00 |
| a2def6e16237a6bf3e66200ddeb85006 | 02  |        4 | Mugs       | Red Dragon Mug                           |  7.95 |    22 |       7.00 |
| c699a190eb34a5c2257a6383cefe51ed | 03  |      100 | Books      | Effortless E-Commerce with PHP and MySQL | 34.99 |   251 |       NULL |
| a2def6e16237a6bf3e66200ddeb85006 | C4  |        2 | Kona       | 1 lb. - caf - ground                     |  8.00 |    35 |       NULL |
| c699a190eb34a5c2257a6383cefe51ed | C20 |        2 | Dark Roast | 5 lbs. - caf - whole                     | 40.00 |     5 |       NULL |
+----------------------------------+-----+----------+------------+------------------------------------------+-------+-------+------------+
8 rows in set (0.00 sec)

mysql>
```

Figure 9.11

Once this view is defined, the **get_shopping_cart_contents()** query would only need to do a **SELECT** on this one table, with a single condition: matching the user's session ID:

```
SELECT * FROM cart_views WHERE user_session_id=uid;
```

tip

VIEW tables were added to MySQL in version 5.0.