**Generating Reports**

One thing you can't get away from, even in a paperless society, is reports. There's always some report you'll need to generate from the data in your database. The trick is doing this as quickly and as painlessly as possible.

If you use codes to identify data uniquely in your tables, normally, you'll need to generate reports that include data from more than one table. If you want to produce a report showing what products are in what categories, you'll want to display the actual category name and not the category ID value that's stored in the products table. This requires creating more complex SQL queries than what we've been doing so far in the project.

When you need to retrieve data from more than one table, you need a *linking data field*. The linking data field links a record in one table to a specific record in another table. An example of this is the catid data field in the products table. If you want to display the full category name in a product report, you must first retrieve the catid value from the product record in the products table and then look up that value in the categories table for the associated category name.

You can do this in two separate SELECT queries, one to retrieve the products record, then another to retrieve the matching categories record, but there's an even easier way. In SQL, you can query data from multiple tables at the same time!

When you retrieve data from multiple tables, you must list all of the tables in your FROM clause, then reference the individual data fields relative to the table they come from. Here's an example of what I'm talking about:

SELECT products.prodid, products.description, categories.name FROM products, categories WHERE products.catid = categories.catid;

This query returns all of the products showing the product description and the full category name for the category they belong. Notice that every place where I reference a data field, I added the table name as well so the MySQL server knows which data field came from which table.

This format can get tedious, especially if you use long table and data field names. To help, SQL allows you to declare *aliases*. An alias is a shortcut name you can assign to a table name in a query. You can use the shortcut name anywhere in the SQL statement, even before you actually define it.

To define a shortcut, you use the AS clause within the FROM clause that specifies the tables. Here's an example of how that works:

SELECT t1.prodid, t1.description, t2.name FROM products as t1, categories as t2 WHERE t1.catid = t2.catid;

The FROM clause defines the alias *t1* to represent the products table name, and the alias *t2* to represent the categories table name. The aliases are then used everywhere in the SQL statement where the tables are referenced.

**Creating Food Store Reports**

The Food Store administration back-end application allows you to generate three different reports for the store manager:

* A list of pending orders.
* The customer and product information for a specific order.
* A spreadsheet of products purchased during a specified time period.

You'll need to use a few complex SQL queries to obtain the information necessary to generate these reports. Let's look at creating these reports one at a time.

**Listing Pending Orders**

As the application records a new order in the orders table, it assigns the order a default status value of *pending*. This means that the customer placed the order, but the store manager hasn't processed it. The Food Store back-end application provides a link in the navigation area for the manager to list pending orders. The link passes the manager to the process.inc.php file. The file builds a list of pending orders, allowing the manager to select one order for processing. Let's create that file now.

1. Create a file called *process.inc.php* in the admin folder under the store folder in your application area.
2. Open the file in a text editor, and add the following code:

<?php

echo "<h2>Pending Orders</h2><br>\n";

$query = "SELECT t1.orderid, t1.custid, t1.date, t2.lastname";

$query = $query . " FROM orders as t1, customers as t2";

$query = $query . " WHERE t1.status = 'pending' AND t1.custid = t2.custid";

$query = $query . " ORDER BY t1.date";

$result = mysql\_query($query);

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

echo "<tr><td>Order ID</td><td>Customer ID</td><td>Last Name</td><td>Date Submitted</td><td> </td></tr>\n";

while($row=mysql\_fetch\_array($result, MYSQL\_ASSOC))

{

$orderid = $row['orderid'];

$custid = $row['custid'];

$lastname = $row['lastname'];

$date = $row['date'];

echo "<tr><td>$orderid</td><td>$custid</td><td>$lastname</td><td>$date</td>\n";

echo "<td><a href=\"admin.php?content=shiporder&id;=$orderid\">Process</a></td></tr>\n";

}

echo "</table>\n";

?>

1. Save the file, and exit the editor.

You should be able to follow the complex SQL query now. It retrieves all of the orders records that have the status data field set to *pending*. It also displays the customer's last name by matching the custid data field in the orders table to the custid data field in the customers table. This adds a little personal touch to the report.

**Processing an Order**

When the manager selects a pending order from the list, the application continues to the shiporder.inc.php file, passing the orderid value. This code produces a complete bill, showing the customer's full information (including a shipping address), all of the products included in the order, and the total cost of the order. You know this report will require some fancy SQL!

Let's build that file now.

1. Create a file called *shiporder.inc.php* in the admin folder under the store folder in your application area.
2. Open the file in a text editor, and add the following code

<?php

$orderid = $\_GET['id'];

$query = "SELECT t1.orderid, t1.custid, t1.date,";

$query = $query . " t2.lastname, t2.firstname, t2.address,";

$query = $query . " t2.city, t2.state, t2.zip FROM";

$query = $query . " orders as t1, customers as t2 WHERE t1.orderid = $orderid AND";

$query = $query . " t1.custid = t2.custid";

$result = mysql\_query($query);

$row = mysql\_fetch\_array($result, MYSQL\_ASSOC);

$custid = $row['custid'];

$date = $row['date'];

$firstname = $row['firstname'];

$lastname = $row['lastname'];

$address = $row['address'];

$city = $row['city'];

$state = $row['state'];

$zip = $row['zip'];

echo "<h2>Order information for order #" . $orderid . "</h2><br>\n";

echo $firstname . " " . $lastname . "<br>\n";

echo $address . "<br>\n";

echo $city . ", " . $state . " " . $zip . "<br><br>\n";

echo "<h3>Items:</h3>\n";

echo "<table width=\"75%\" cellpadding=\"1\" border=\"1\">\n";

echo "<tr><td>Product ID</td><td>Description</td><td>Price</td><td>Quantity</td><td>Total</td></tr>\n";

$query = "SELECT t1.prodid, t1.quantity, t2.description, t2.price";

$query = $query . " FROM order\_items as t1, products as t2 WHERE t1.orderid = $orderid";

$query = $query . " AND t1.prodid = t2.prodid";

$result = mysql\_query($query);

$total = 0;

while($row = mysql\_fetch\_array($result, MYSQL\_ASSOC))

{

$prodid = $row['prodid'];

$quantity = $row['quantity'];

$description = $row['description'];

$price = $row['price'];

$subtotal = $price \* $quantity;

$total += $subtotal;

echo "<tr><td>$prodid</td><td>$description</td>\n";

printf("<td>%.2f</td><td>%d</td><td>%.2f</td></tr>\n", $price, $quantity, $subtotal);

}

echo "<tr><td colspan=\"4\"><b>Order Total</b></td>\n";

printf("<td>%.2f</td></tr>\n", $total);

echo "</table>\n";

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

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

echo "<input type=\"hidden\" name=\"orderid\" value=\"$orderid\">\n";

echo "<input type=\"submit\" name=\"button\" value=\"Post order\">\n";

echo "</form>\n";

?>

1. Save the file, and exit the editor.

This code builds two SQL queries, one to display the customer information for the order and another to retrieve the product information for the order. It also does some math, totaling the cost for each item and producing a total for the order.

When the manager selects the Post order button, the code passes control over to the postorder.inc.php file. In this file, you can generate a printed report or an e-mail message to send to someone to process the order. In our simple example, we'll just mark the order record as being processed and leave it at that. Let's build our simple version of the postorder.inc.php file.

1. Create a file called *postorder.inc.php* in the admin folder under the store folder in your application area.
2. Open the file in a text editor, and add the following code:

<?php

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

$query = "UPDATE orders SET status = 'shipped' WHERE orderid = $orderid";

$result = mysql\_query($query) or die(mysql\_error());

if ($result)

{

echo "<h2>Order processed.</h2>\n";

} else

{

echo "<h2>Unable to process order at this time.</h2>\n";

}

echo "<a href=\"admin.php?content=process\">Process more orders</a>\n";

?>

1. Save the file, and exit the text editor.

Obviously, if this were a real store you'd want to post the order information in some sort of tracking system. For our project, though, just a simple message that the order was processed suffices.

That's all there is to that report. In Chapter 4, you'll see how to generate a completely different type of report using your PHP skills.