**Creating Special Reports**

In the real world, not all data resides in Web pages. There are times when you need to generate reports in a special format, such as exporting data into the accountants' favorite tool—a spreadsheet. You do this by using a little bit of HTTP trickery along with your PHP code.

Another feature in this report you'll need to use a date to filter out data in the database tables. Unfortunately, MySQL uses a somewhat odd way of storing date information. Usually it's a good idea to allow your customers to enter dates in a format they're used to using, then convert the information into what MySQL wants. Let's look at doing this first.

**Working With Dates**

SQL allows you to use dates in queries. This provides an easy way to extract information based on dates. MySQL's peculiar format for storing dates is

YYYY-MM-DD

where *YYYY* is the four-digit year, *MM* is the two-digit month, and *DD* is the two-digit day. Almost everyone else in the world displays dates differently. The trick is to create an HTML form that offers the date elements individually, and then combine those elements into the format required by MySQL. This lets your manager enter a date in a format such as 10/25/2007, then convert that into the standard 2007-10-25 MySQL format.

The report.inc.php file provides an HTML form for entering a start and end date to generate a report. We'll use those dates later to create a spreadsheet report for the manager. First, let's create the HTML form. Just follow these steps:

1. Create a file called report.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:

<h2>Calculate Report for period: </h2>

<form action="displayreport.php" method="post">

<input type="hidden" name="content" value="quarterly">

<table width="100%" cellpadding="1" border="1">

<tr><td>Enter start date (mm/dd/yyyy):</td>

<td><input type="text" name="startmonth" size="2">

<input type="text" name="startday" size="2">

<input type="text" name="startyear" size="4"</td></tr>

<tr><td>Enter end date (mm/dd/yyyy):</td>

<td><input type="text" name="endmonth" size="2">

<input type="text" name="endday" size="2">

<input type="text" name="endyear" size="4"</td></tr>

</table>

<input type="submit" name="button" value="Get report">

</form>

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

There's nothing too fancy here. The code uses individual textboxes to allow the manager to enter the day, month, and year values individually and in whatever format they're accustom to (the order shown is in U.S. format; if you prefer European format, you can easily reverse the month and day textboxes).

This code passes the start and end date information to a separate program called displayreport.php. There's a reason why it does this instead of using admin.php.

**Creating Spreadsheets**

By default, the client's browser attempts to display all data within the browser window. The HTTP protocol provides a special header that identifies data types. If the browser recognizes a data type that an external application can handle, it has the ability to redirect the data automatically to that application.

The HTTP header specifies the data type using the *Content-type* label. The label that identifies an Excel spreadsheet looks like this:

Content-type: application/vnd.ms-excel

The first part of the content type defines the general MIME media type of the data. The second part of the content type defines the specific media type. When the client browser sees this header, it knows that the Excel spreadsheet application is to use the data.

You can exploit that feature by using PHP to create a special HTTP header for your data, identifying the data as a spreadsheet. If you have a spreadsheet application on your PC that can read Excel spreadsheets, this HTTP header will start your application automatically.

This technique uses our friend the header() function to create the special HTTP header. If you remember, the header() function must be the first item sent to the client's browser. Because of this, we need to create a separate PHP program instead of using an include file in the admin.php program.

Let's build the code first and then walk through it.

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

<?php

header("Content-type: application/vnd.ms-excel");

header("Content-Disposition: attachment; filename=report.xls");

header("Pragma: no-cache");

header("Expires: 0");

include("/mylibrary/login.php");

login();

$startday = $\_REQUEST['startday'];

$startmonth = $\_REQUEST['startmonth'];

$startyear = $\_REQUEST['startyear'];

$dbstartdate = $startyear . "-" . $startmonth . "-" . $startday;

$startdate = $startmonth . "/" . $startday . "/" . $startyear;

$endday = $\_REQUEST['endday'];

$endmonth = $\_REQUEST['endmonth'];

$endyear = $\_REQUEST['endyear'];

$dbenddate = $endyear . "-" . $endmonth . "-" . $endday;

$enddate = $endmonth . "/" . $endday . "/" . $endyear;

$query = "SELECT products.description, sum(order\_items.quantity) as total, products.price ";

$query .= " FROM orders, order\_items, products";

$query .= " WHERE orders.orderid = order\_items.orderid";

$query .= " AND order\_items.prodid = products.prodid";

$query .= " AND orders.status = 'shipped'";

$query .= " AND orders.date >= '$dbstartdate' AND orders.date <= '$dbenddate'";

$query .= " GROUP BY products.description";

$result = mysql\_query($query);

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

echo "<tr><td colspan=\"4\"><b>Products sold between " . $startdate . " and " . $enddate . "</b></td></tr>\n";

echo "<tr><td><b>Product</b></td><td><b>Quantity Sold</b></td><td><b>Unit price</b></td><td><b>Total</b></td></tr>\n";

$count = 3;

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

{

$product = $row['description'];

$quantity = $row['total'];

$price = $row['price'];

$total = $quantity \* $price;

echo "<tr><td>$product</td><td>$quantity</td>\n";

printf("<td>%.2f</td><td>=B%s \* C%s</tr>\n", $price, $count, $count);

$count++;

}

$count--;

echo "<tr><td><b>Total</b></td><td>=SUM(B3:B" . $count . ")</td><td> </td>\n";

echo "<td>=SUM(D3:D" . $count . ")</td></tr>\n";

echo "</table>\n";

?>

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

The first few lines use the header() function to define the spreadsheet. They tell the client browser what type of file you're generating using the *Content-type* header, and the name of the file (make sure you use the .xls extension) using the *Content-disposition* header. It then uses another header to tell the client browser not to cache the data (sometimes this causes problems when using external applications).

The next section retrieves the start and end dates from the HTML form, and it creates two forms for each date: a standard form to use in the spreadsheet and a MySQL form to use in our query.

Next, the code builds the SELECT query. This is the most complicated query we've done yet in the application! Let's break it apart into pieces to discuss:

SELECT products.description, sum(order\_items.quantity) as total, products.price  
FROM orders, order\_items, products

This part retrieves the product description and price from the products table, and uses the SUM() function in MySQL to add the quantities for all the occurrences of products purchased within the start and end date.

Now, the next section:

WHERE orders.orderid = order\_items.orderid  
AND order\_items.prodid = products.prodid  
AND orders.status = 'shipped'  
AND orders.date >= '$dbstartdate'  
AND orders.date <= '$dbenddate'

Wow, that's a mouthful! This WHERE clause looks for orders with a date greater or equal to the start date, but less than or equal to the end date. Notice that you can use standard mathematical operators with dates in MySQL! Also, it‘s looking only for orders with the status *shipped*, which means the manager has processed the order.

The first two items in the WHERE clause link our three data tables together, matching the linking data field between the orders and order\_items tables, and the order\_items and products table. This allows us to find all of the order\_items records for the order and then retrieve the proper product descriptions from the products table.

Finally, the code uses another special SELECT statement clause:

GROUP BY products.description

Whenever you use the SUM() function, you must use the GROUP BY clause. This clause groups data records in the result set. Instead of placing the records in the result set in the order they appear in the table, MySQL will group the records together. In this example, you group the records based on the product description. With the products grouped, you can then use the SUM() function to total the quantity data fields for each product.

This is about as complicated of a query as you'll ever want to generate, but it creates an impressive report! The next (and final) step is to build the actual spreadsheet.

To build a spreadsheet, you must create an HTML table. Each cell in the table becomes a cell in the final spreadsheet. The code places the individual data records retrieved from the SQL query into the table cells where you want them to reside. The code also performs some basic calculations from the retrieved database data to enter into the spreadsheet. However, you may notice something odd with the code.

The code produces some odd text in some of the table cells. The text it produces is standard Excel spreadsheet equations. Since Excel (or another application similar to it) is processing data, you can exploit it by using the processing power of the application to do some of the work for you. By using the SUM() Excel function in the table cells, you're forcing Excel to process the data you're placing in the spreadsheet! This is a great technique to use if you're working with complex data and don't want to perform the calculations on the Web server.

Viewing the resulting spreadsheet report in Excel

That completes the code for our Food Store application. In just a few short weeks, you've completed an entire Web-based storefront application! Let's move on to Chapter 5 and wrap things up.