**Programming the Transactions**

With all the database theory stuff out of the way, we can get back to coding. When the customer confirms the final order, they're sent to the finishorder.inc.php file. This file updates all of the database tables with the information contained in the customer's shopping cart.

As you saw earlier in this lesson, you need to modify three tables:

* A new record added to the *orders* table.
* A new record added to the *order\_items* table for each product purchased.
* The quantity value updated in the *products* table for each product purchased.

This lets you put your newly learned transaction skills to work. You'll need to create a transaction that performs each of these database actions. If all of the database statements succeed, you can commit the transaction. If any one of the database statements fails, you must roll back the transaction and inform the customer that you can't process the order.

The finishorder.inc.php file does all this. Let's create the finishorder.inc.php file. Follow these steps:

1. Create a file called *finishorder.inc.php* in the store folder in the application area.
2. Open the file with a text editor and enter the following code:

<?php

echo "<h2><u>Finalizing Order</u></h2><br>\n";

echo "Creating order.\n";

$custid = $\_SESSION['cust'];

$date = date("Y-m-d G:i:s");

$status = "pending";

$result1 = @mysql\_query("Set autocommit=0");

$result2 = @mysql\_query("set sql\_mode = 'STRICT\_ALL\_TABLES'");

$result3 = @mysql\_query("START TRANSACTION");

$query = "INSERT INTO orders (custid, date, status) VALUES " .

" ($custid, '$date', '$status')";

$result4 = @mysql\_query($query);

$query = "SELECT LAST\_INSERT\_ID() from orders";

$result5 = @mysql\_query($query);

$row = mysql\_fetch\_array($result5);

$orderid = $row[0];

foreach($\_SESSION['cart'] as $prodid => $quantity)

{

$query = "SELECT price FROM products where prodid = $prodid";

$result6i = @mysql\_query($query);

$row = mysql\_fetch\_array($result6i);

$price = $row[0];

$query = "INSERT into order\_items VALUES ($orderid, $prodid, $quantity, $price)";

$result6a = @mysql\_query($query);

$query = "UPDATE products set quantity = quantity - $quantity WHERE prodid = $prodid";

$result6b = @mysql\_query($query);

if ($result6a && $result6b)

{

$result6 = true;

} else

{

$result6 = false;

break;

}

}

if ($result1 && $result2 && $result3 && $result4 && $result5 && $result6)

{

$result = @mysql\_query("COMMIT");

if ($result)

{

echo "Your order has been placed.<br><br>\n";

echo "<h2>Your order number is #$orderid.<br>\n";

echo "<h2>Please save this number for future reference.<br>Thank you!</h2>\n";

unset($\_SESSION['cart']);

} else

{

echo "<h2>Sorry, we are unable to create your order at this time.</h2>\n";

echo "<h2>Please double check product availability.</h2>\n";

}

} else

{

$result = @mysql\_query("ROLLBACK");

echo "<h2>Sorry, we are unable to create your order at this time.</h2>\n";

}

?>

1. Save the file and exit the text editor.

That's a lot of code, but you should be able to follow along with what's happening. First, the program disables the autocommit feature and sets the sql\_mode to STRICT\_ALL\_TABLES so the updates to the products table will work properly.

You'll notice something odd with the way I performed the queries:

$result1 = @mysql\_query("Set autocommit=0");

The @ symbol in front of the mysql\_query() function is an interesting feature. This suppresses any error messages that may result from the query.

You've seen before how to use the *or die()* feature to suppress error messages. However, the PHP preprocessor stops processing statements and automatically exits when you use this technique. You don't want that to happen in this case, as you have a transaction that you started. If a query fails, you just want to flag that it failed so you know to use the ROLLBACK statement at the end of the transaction.

You do that by assigning unique PHP variable names to each query. When you've completed all of the queries in the transaction, all we need to do is check if any query failed by logically ANDing the results. The *&&* symbol performs the logical AND function. All of the values in the operation must be TRUE for the result to be TRUE. If a query failed, its result will be FALSE, causing the result of the AND statement to be FALSE. This is what enables you to detect if a query fails.

After preparing the transaction, you can start inserting and updating records. First, the code inserts a new record in the orders table. Next, you need to use our friend the LAST\_INSERT\_ID() function to retrieve just what orderid the server assigned to the new order so you can use it for the records in the order\_items table.

The code uses the foreach statement to iterate through the products in the customer's shopping cart, adding each product to the order\_items table. Each product relates to the order using the orderid retrieved.

Finally, the code uses the UPDATE statement for each product to update the inventory total for each product with the amount purchased by the customer. If any of the inserts or updates fails, the result variable is set to false, and the code uses the break statement to jump out of the foreach loop to stop processing products.

At the end of the process, if all of the SQL statements succeed, the code sends the COMMIT statement to the server, committing the data to the database, and displays a message with the orderid value for the customer to record.

The final checkout customer receipt

Also, if the checkout process is complete, the code removes the entire shopping cart session cookie to prevent duplicate checkout orders.

That completes the checkout process for the customer. We've covered quite a bit of ground in the last two lessons just trying to present a pleasant checkout experience for our customers. Let's go on to Chapter 5 and wrap up this lesson.