12 CREATING A DYNAMIC ONLINE GALLERY

What this chapter covers:

- Why storing images in a database is a bad idea, and what you should do instead
- Planning the layout of a dynamic gallery
- Displaying a fixed number of results in a row
- Limiting the number of records retrieved at a time
- Paging through a long set of results

In the last chapter, I showed you how to display the contents of the images table in a web page. It didn't look very attractive—text in an unadorned table. However, I hope you will have realized by now that all you need to do to display the images themselves is add `<img>` tags to the underlying XHTML, and you'll end up with something far more impressive. In fact, by the end of this chapter, you will have created the mini photo gallery shown in Figure 12-1.

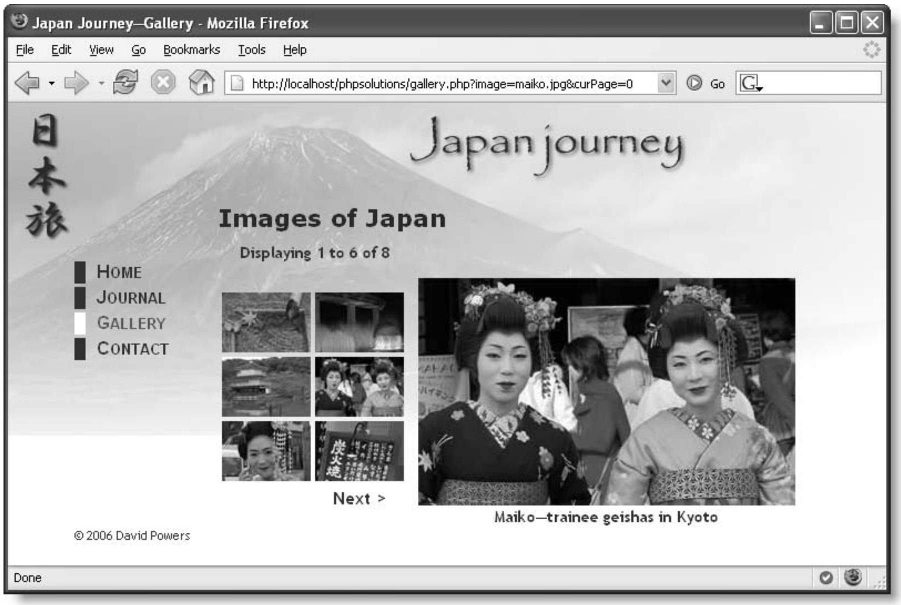

Japan Journey—Gallery - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/phpsolutions/gallery.php?image=maiko.jpg&curPage=0
Go
日本旅
Japan journey
Images of Japan
Displaying 1 to 6 of 8
HOME
JOURNAL
GALLERY
CONTACT
Next >
Maiko—trainee geishas in Kyoto
© 2006 David Powers
Done

Figure 12-1. The mini photo gallery is driven entirely by drawing information from a database.

Although it uses images, the gallery demonstrates some cool features that you will want to incorporate into text-driven pages, too. For instance, the grid of thumbnail images on the left displays two images per row. Just by changing two numbers, you can make the grid as many columns wide and as many rows deep as you like. Clicking one of the thumbnails replaces the main image and caption. It's the same page that reloads, but exactly the same technique is used to create online catalogs that take you to another page with more details about a product. The Next link at the foot of the thumbnails grid shows you the next set of photographs, using exactly the same technique as you use to page through a long set of search results. This gallery isn't just a pretty face or two . . .

First of all, a word about images and databases.

Why not store images in a database?

The images table contains only filenames and captions, but not the images themselves. Even though I said in the last chapter that you can always add new columns or tables to a database when new requirements arise, I'm not going to add anything to the images table. Instead, I intend to leave the images in their original location within the website—for the simple reason that storing images in a database is usually more trouble than it's worth. The main problems are as follows:

- Images cannot be indexed or searched without storing textual information separately.
- Images are usually large, bloating the size of tables.
- Table fragmentation affects performance if images are deleted frequently.
- Retrieving images from a database involves passing the image to a separate script, slowing down display in a web page.

Storing images in a database is messy, and involves more scripting. It's much more efficient to store images in an ordinary folder on your website and use the database for information about the images. You need just two pieces of information in the database—the filename and a caption that can also be used as alt text. You could also store the image's height and width, but it's not absolutely necessary. As you saw in Chapter 4, you can generate that information dynamically.

Planning the gallery

Unless you're good at visualizing how a page will look simply by reading its source code, I find that the best way to design a database-driven site is to start with a static page and fill it with placeholder text and images. I then create my CSS style rules to get the page looking the way I want, and finally replace each placeholder element with PHP code. Each time I replace something, I check the page in a browser to make sure that everything is still holding together.

Figure 12-2 shows the static mockup that I made of the gallery and points out the elements that need to be converted to dynamic code. The images are the same as those used for the random image generator in Chapter 4 and are all different sizes. I experimented by scaling the images to create the thumbnails, but decided that the result looked too untidy, so I made the thumbnails a standard size (80 × 54 pixels). Also, to make life easy, I gave each thumbnail the same name as the larger version and stored them in a separate folder called thumbs.

As you saw in the previous chapter, displaying the contents of the entire images table was easy. You created a single table row, with the contents of each field in a separate table cell. By looping through the result set, each record displayed on a row of its own, simulating the column structure of the database table. This time, the two-column structure of the thumbnail grid no longer matches the database structure. This means that you need to count how many thumbnails have been inserted in each row before triggering the creation of the next row.

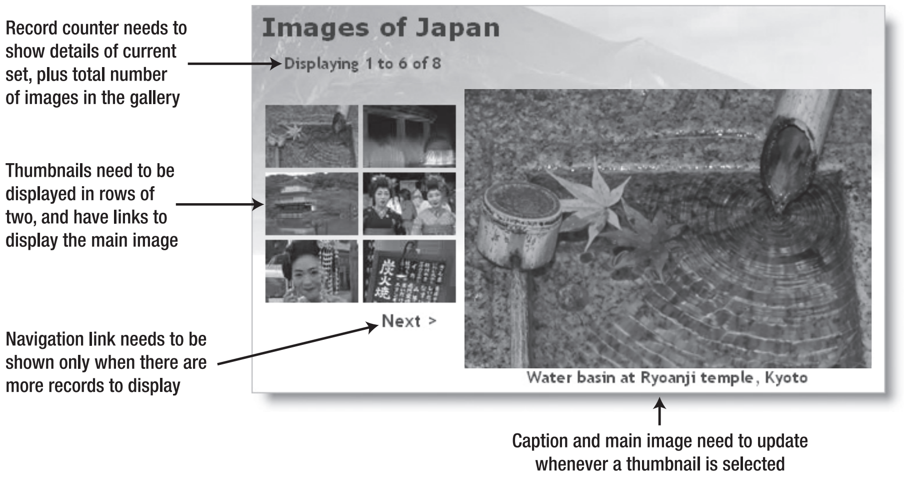

Record counter needs to show details of current set, plus total number of images in the gallery
Thumbnails need to be displayed in rows of two, and have links to display the main image
Navigation link needs to be shown only when there are more records to display
Caption and main image need to update whenever a thumbnail is selected
Images of Japan
Displaying 1 to 6 of 8
Next >
Water basin at Ryoanji temple, Kyoto

Figure 12-2. Working out what needs to be done to convert a static gallery to a dynamic one

Figure 12-3 shows the framework I created to hold the gallery together. The table of thumbnails and the `main_image <div>` are floated left and right respectively in a fixed-width wrapper `<div>` called `gallery`. I don't intend to go into the details of the CSS, but you may study that at your leisure.

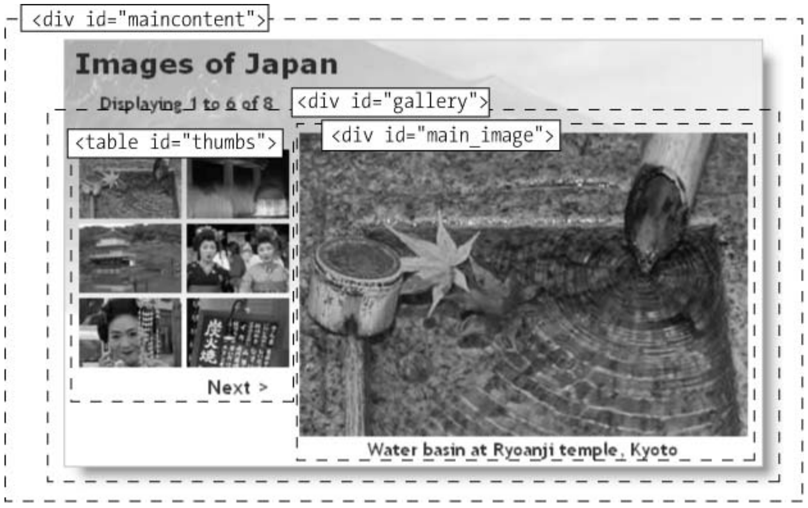


Figure 12-3. The underlying structure of the image gallery

Once I had worked out what needed to be done, I stripped out the code for thumbnails 2 to 6, and for the navigation link (which is nested in the final row of the `thumbs` table). The following listing shows what was left in the `maincontent <div>` of `gallery.php`, with the elements that need to be converted to PHP code highlighted in bold (you can find the code in gallery01.php in the download files for this chapter):

```
<div id="maincontent">
  <h1>Images of Japan</h1>
  <p id="picCount">Displaying 1 to 6 of 8</p>
  <div id="gallery">
    <table id="thumbs">
      <tr>
        <!-- This row needs to be repeated -->
        <td><a href="gallery.php"><img src="images/thumbs/basin.jpg" ➥
alt="" width="80" height="54" /></a></td>
      </tr>
      <!-- Navigation link needs to go here -->
    </table>
    <div id="main_image">
      <p><img src="images/basin.jpg" alt="" width="350" height="237" ➥
/></p>
      <p>Water basin at Ryoanji temple, Kyoto</p>
    </div>
  </div>
</div>
```

Converting the gallery elements to PHP

Before you can display the contents of the gallery, you need to connect to the phpsolutions database and retrieve all the records stored in the images table. The procedure for doing so is the same as in the previous chapter, using the following simple SQL query:

```
SELECT * FROM images
```

You can then use the first record to display the first image and its associated caption and thumbnail.

PHP Solution 12-1: Displaying the first image

If you set up the Japan Journey website in Chapter 4, you can work directly with the original gallery.php. Alternatively, use gallery01.php from the download files for this chapter. You also need to copy title.inc.php, menu.inc.php, and footer.inc.php to the includes folder of the phpsolutions site.

1. Load gallery.php into a browser to make sure that it displays correctly. The maincontent part of the page should look like Figure 12-4, with one thumbnail image and a larger version of the same image.

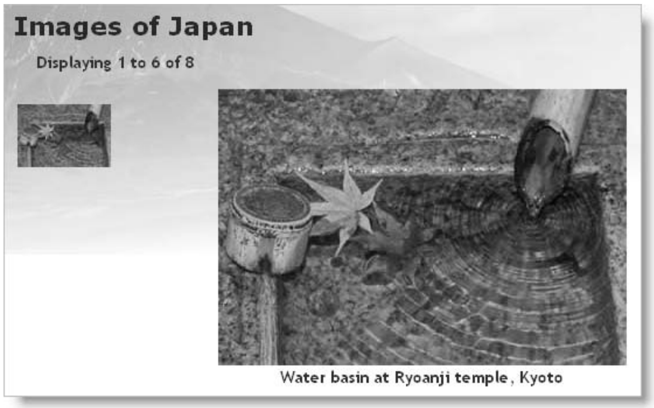


Figure 12-4. The stripped-down version of the static gallery ready for conversion

2. The gallery depends entirely on a successful connection to the database, so the first thing you need to do is include `connection.inc.php`. Add the following code just before the closing PHP tag above the `DOCTYPE` declaration in `gallery.php` (new code is highlighted in bold):

```
<?php
include('includes/title.inc.php');
// include MySQL connector function
if (! @include('includes/connection.inc.php'))
  echo 'Sorry, database unavailable';
  exit;
  }
?>
```

   Remember, `connection.inc.php` needs to be the correct version for the way you plan to connect to MySQL—using the original MySQL extension, the MySQL Improved object-oriented interface, or PDO. The `include` command for the connection script is used as the condition for an `if` statement. The condition also uses the negative operator (an exclamation mark) and the error control operator (`@`). If the connection script is included successfully, the code inside the `if` statement is ignored; but if the file can't be found, a custom error message is displayed, and the rest of the script is abandoned. In a live application, you would probably redirect visitors to a custom error page.

3. Connect to the database by calling the `dbConnect()` function in the include file, and prepare the SQL query ready to submit it. The gallery needs only `SELECT` privileges for the database, so pass `query` as the argument to `dbConnect()` like this (the code for steps 3 to 5 goes immediately before the closing PHP tag):

```
// create a connection to MySQL
$conn = dbConnect('query');
// prepare SQL to retrieve image details
$sql = 'SELECT * FROM images';
```

4. The code for submitting the query and extracting the first record from the result depends on which method of connection you are using. For the original MySQL functions, use this:

```
// submit the query
$result = mysql_query($sql) or die(mysql_error());
// extract the first record as an array
$row = mysql_fetch_assoc($result);
```

For MySQL Improved, use this:

```
// submit the query
$result = $conn->query($sql) or die(mysqli_error());
// extract the first record as an array
$row = $result->fetch_assoc();
```

For PDO, use this:

```
// submit the query
$result = $conn->query($sql);
// get any error messages
$error = $conn->errorInfo();
if (isset($error[2])) die($error[2]);
// extract the first record as an array
$row = $result->fetch();
```

The code for the original MySQL extension and MySQL Improved is exactly the same as you used in the previous chapter.

The PDO code, however, introduces a new method, `fetch()`, which gets the next record from the result set. You can't use a `foreach` loop like in the previous chapter, because you need to get the first result on its own.

5. All three methods now have the first record from the result set stored as an array in `$row`. This means that `$row['image_id']` contains the primary key of the first record, `$row['filename']` contains its filename, and `$row['caption']` contains its caption. You need the filename, caption, and the dimensions of the large version so that you can display the images in the main body of the page. Add the following code:

```
// get the name and caption for the main image
$mainImage = $row['filename'];
$caption = $row['caption'];
// get the dimensions of the main image
$imageSize = getimagesize('images/'.$mainImage);
```

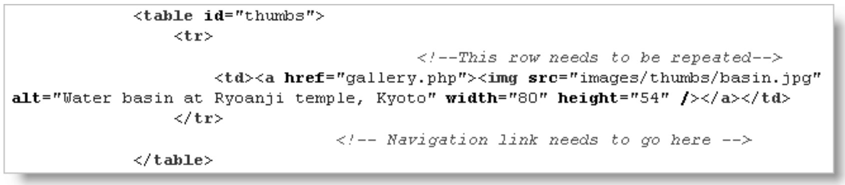
The `getimagesize()` function was described in Chapters 4 and 8.

6. You can now use this information to display the thumbnail, main image, and its caption dynamically. The main image and thumbnail have the same name, but you eventually want to display all thumbnails by looping through the full result set, so the dynamic code that needs to go in the table cell needs to refer to the current record—in other words, `$row['filename']` and `$row['caption']`, rather than to

$mainImage and $caption. You'll see later why I've assigned the values from the first record to separate variables. Amend the code in the table like this:

```
<td><a href="gallery.php"> ➥
<img src="images/thumbs/<?php echo $row['filename']; ?>" ➥
alt="<?php echo $row['caption']; ?>" width="80" height="54" /> ➥
</a></td>
```

7. Save gallery.php and view it in a browser. It should look the same as Figure 12-4. The only difference is that the thumbnail and its alt text are dynamically generated. You can verify this by looking at the source code. The original static version had an empty alt attribute, but as the following screenshot shows, it now contains the caption from the first record:



```
            <table id="thumbs">
                <tr>
                                        <!--This row needs to be repeated-->
                    <td><a href="gallery.php"><img src="images/thumbs/basin.jpg"
alt="Water basin at Ryoanji temple, Kyoto" width="80" height="54" /></a></td>
                </tr>
                                <!-- Navigation link needs to go here -->
            </table>
```

If things go wrong, make sure there's no gap between the static and dynamically generated text in the image's src attribute. Also check that you're using the right code for the type of connection you have created with MySQL. You can check your code against gallery_mysql02.php, gallery_mysqli02.php, or gallery_pdo02.php.

8. Once you have confirmed that you're picking up the details from the database, you can convert the code for the main image. Amend it like this (new code is in bold):

```
<div id="main_image">
  <p><img src="images/<?php echo $mainImage; ?>" ➥
alt="<?php echo $caption; ?>" <?php echo $imageSize[3]; ?> /></p>
  <p><?php echo $caption; ?></p>
</div>
```

As explained in Chapter 4, getimagesize() returns an array, the fourth element of which contains a string with the width and height of an image ready for insertion into an <img> tag. So $imageSize[3] inserts the correct dimensions for the main image.

9. Test the page again. It should still look the same as Figure 12-4, but the images and caption are being drawn dynamically from the database. You can check your code against gallery_mysql03.php, gallery_mysqli03.php, or gallery_pdo03.php.

Building the dynamic elements

The first thing that you need to do after converting the static page is to display all the thumbnails and build dynamic links that will enable you to display the large version of any thumbnail that has been clicked. Displaying all the thumbnails is easy—just loop through them (we'll work out how to display them in rows of two later). Activating the link for each thumbnail requires a little more thought. You need a way of telling the page which large image to display.

Passing information through a query string

In the last section, you used `$mainImage` to identify the large image, so you need a way of changing its value whenever a thumbnail is clicked. The answer is to add the image's filename to a query string at the end of the URL in the link like this:

```
<a href="gallery.php?image=filename">
```

You can then check whether the `$_GET` array contains an element called `image`. If it does, change the value of `$mainImage`. If it doesn't, leave `$mainImage` as the filename from the first record in the result set.

Time to dive back into the code . . .

PHP Solution 12-2: Activating the thumbnails

Continue working with the same file as in the previous section. Alternatively, use `gallery_mysql03.php`, `gallery_mysqli03.php`, or `gallery_pdo03.php` from the download files.

1. Locate the `<a>` tag surrounding the thumbnail. It looks like this:

```
<a href="gallery.php">
```

Change it like this:

```
<a href="<?php echo $_SERVER['PHP_SELF']; ?>?image=<?php echo ➥
$row['filename']; ?>">
```

Be careful when typing the code. It's easy to mix up the question marks in the PHP tags with the question mark at the beginning of the query string. It's also important there are no spaces surrounding `?image=`.

So, what's all this about? `$_SERVER['PHP_SELF']` is a handy predefined variable that refers to the name of the current page. You could just leave `gallery.php` hard-coded in the URL, but I suspect that many of you will use the download files, which have a variety of names. Using `$_SERVER['PHP_SELF']` ensures that the URL is pointing to the correct page. The rest of the code builds the query string with the current filename.

2. Save the page, and load it into a browser. Hover your mouse pointer over the thumbnail, and check the URL displayed in the status bar. It should look like this:

```
http://localhost/phpsolutions/gallery.php?image=basin.jpg
```

If nothing is shown in the status bar, click the thumbnail. The page shouldn't change, but the URL in the address bar should now include the query string. Check that there are no gaps in the URL or query string.

3. To show all the thumbnails, you need to wrap the table cell in a loop. Insert a new line after the XHTML comment about repeating the row, and create the first half of a do... while loop like this (see Chapter 3 for details of the different types of loops):

```
<!-- This row needs to be repeated -->
<?php do { ?>
```

4. You already have the details of the first record in the result set, so the code to get subsequent records needs to go after the closing </td> tag. Create some space between the closing </td> and </tr> tags, and insert the following code. It's slightly different for each method of database connection.

For the MySQL original extensions, use this:

```
  </td>
<?php
$row = mysql_fetch_assoc($result);
} while ($row);
?>
</tr>
```

For the MySQL Improved object-oriented interface, use this:

```
  </td>
<?php
$row = $result->fetch_assoc();
} while ($row);
?>
</tr>
```

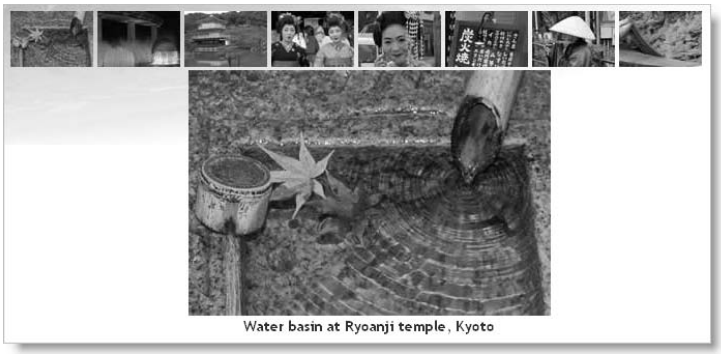
For PDO, use this:

```
  </td>
<?php
$row = $result->fetch();
} while ($row);
?>
</tr>
```

This fetches the next row in the result set and sends the loop back to the top. Because $row['filename'] and $row['caption'] have different values, the next thumbnail and its associated alt text are inserted into a new table cell. The query string is also updated with the new filename.

5. Save the page, and test it in a browser. You should now see all eight thumbnails in a single row across the top of the gallery, as shown at the top of the next page.

Hover your mouse pointer over each thumbnail, and you should see the query string display the name of the file. You can check your code against gallery_mysql04.php, gallery_mysqli04.php, or gallery_pdo04.php.



6. Clicking the thumbnails still doesn't do anything, so you need to create the logic that changes the main image and its associated caption. Locate this section of code in the block above the DOCTYPE declaration:

```
// get the name and caption for the main image
$mainImage = $row['filename'];
$caption = $row['caption'];
```

Highlight the line that defines $caption, and cut it to your clipboard. Wrap the other line in a conditional statement like this:

```
// get the name for the main image
if (isset($_GET['image'])) {
  $mainImage = $_GET['image'];
  }
else {
  $mainImage = $row['filename'];
  }
```

The $_GET array contains values passed through a query string, so if $_GET['image'] has been set (defined), it takes the filename from the query string and stores it as $mainImage. If $_GET['image'] doesn't exist, the value is taken from the first record in the result set as before.

7. You finally need to get the caption for the main image. It's no longer going to be the same every time, so you need to move it to the loop that displays the thumbnails. It goes right after the opening curly brace of the loop. Position your cursor after the brace and insert a couple of lines, and then paste the caption definition that you cut in the previous step. You want the caption to match the main image, so if the current record's filename is the same as $mainImage, that's the one you're after. Wrap the code that you have just pasted in a conditional statement like this:

```
<?php
do {
  // set caption if thumbnail is same as main image
  if ($row['filename'] == $mainImage) {
    $caption = $row['caption']; // this is the line you pasted
    }
?>
```

8. Save the page and reload it in your browser. This time, when you click a thumbnail, the main image and caption will change. Check your code, if necessary, against `gallery_mysql05.php`, `gallery_mysqli05.php`, or `gallery_pdo05.php`.

Passing information through a query string like this is an important aspect of working with PHP and database results. Although form information is normally passed through the `$_POST` array, the `$_GET` array is frequently used to pass details of a record that you want to display, update, or delete. Like the `$_POST` array, the `$_GET` array automatically inserts backslashes if magic quotes are turned on in `php.ini`. Since only the filename is being passed through the query string, there's no need to use the `nukeMagicQuotes()` function from Chapter 3 because quotes are illegal in filenames. That's one reason I didn't pass the caption through the query string. Getting it directly from the database avoids the problem of handling backslashes.

Creating a multicolumn table

With only eight images, the single row of thumbnails across the top of the gallery doesn't look too bad. However, it's useful to be able to build a table dynamically with a loop that inserts a specific number of table cells in a row before moving to the next row. You do this by keeping count of how many cells have been inserted. When you get to the limit for the row, check whether any more rows are needed. If so, insert a closing tag for the current row and an opening tag for the next one. What makes it easy to implement is the modulo operator, `%`, which returns the remainder of a division.

This is how it works. Let's say you want two cells in each row. When the first cell is inserted, the counter is set to 1. If you divide 1 by 2 with the modulo operator (1%2), the result is 1. When the next cell is inserted, the counter is increased to 2. The result of 2%2 is 0. The next cell produces this calculation: 3%2, which results in 1; but the fourth cell produces 4%2, which is again 0. So, every time that the calculation results in 0, you know—or to be more exact, PHP knows—you're at the end of a row.

So how do you know if there are any more rows left? Each time you iterate through the loop, you extract the next record into an array called `$row`. By using `is_array()`, you can check whether `$row` contains the next result. If it does, you add the tags for the next row. If `is_array($row)` is `false`, you've run out of records in the result set. Phew . . . let's try it.

PHP Solution 12-3: Looping horizontally and vertically

Continue working with the files from the preceding section. Alternatively, use `gallery_mysql05.php`, `gallery_mysqli05.php`, or `gallery_pdo05.php`.

1. You may decide at a later stage that you want to change the number of columns in the table, so it's a good idea to create a constant at the top of the script, where it's easy to find, rather than burying the figures deep in your code. Insert the following code just before the database connection:

```
// define number of columns in table
define('COLS', 2);
```

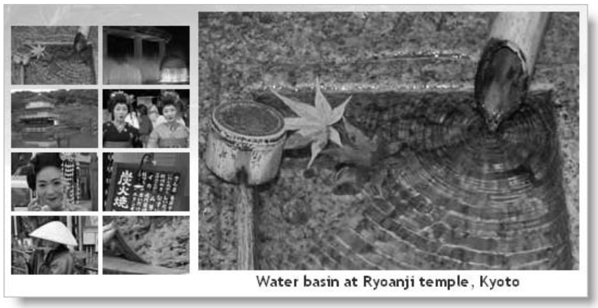
2. You need to initialize the cell counter outside the loop, so amend the beginning of the loop like this:

```
<?php
// initialize cell counter outside loop
$pos = 0;
do {
  // set caption if thumbnail is same as main image
  if ($row['filename'] == $mainImage) {
```

3. The remainder of the code goes after the table cell. It should be easy to follow with the inline comments and the description at the beginning of this section. Amend the code as follows (the first line of code inside the block is part of the existing code, and will look slightly different if you're using MySQLI or PDO):

```
    </td>
  <?php
  $row = mysql_fetch_assoc($result);
  // increment counter after next row extracted
  $pos++;
  // if at end of row and records remain, insert tags
  if ($pos%COLS === 0 && is_array($row)) {
    echo '</tr><tr>';
    }
  } while($row);  // end of existing loop
  // new loop to fill in final row
  while ($pos%COLS) {
    echo '<td>&nbsp;</td>';
    $pos++;
    }
  ?>
  </tr>
</table>
```

A new loop is added at the end of the existing loop. If there are no more records, and $pos%COLS doesn't equal 0, it means you have an incomplete row at the bottom of the table, so the second loop continues incrementing $pos while $pos%COLS produces a remainder (which is interpreted as true) and inserting an empty cell. Note that this second loop is *not* nested inside the first. It runs only after the first loop has ended.



4. Save the page and reload it in a browser. The single row of thumbnails across the top of the gallery should now be neatly lined up two by two, as shown to the right.

Try changing the value of COLS and reloading the page. See how easy it is to control the number of cells in each row by changing just one number. You can check your code against gallery_mysql06.php, gallery_mysqli06.php, or gallery_pdo06.php.

Paging through a long set of records

The grid of eight thumbnails fits quite comfortably in the gallery, but what if you have 28 or 48? The answer is to limit the number of results displayed on each page, and build a navigation system that lets you page back and forth through the results. You've seen this technique countless times when using a search engine; now you're going to learn how to build it yourself.

The task can be broken down into the following two stages:

1. Selecting a subset of records to display
2. Creating the navigation links to page through the subsets

Both stages are relatively easy to implement, although it involves applying a little conditional logic. Keep a cool head, and you'll breeze through it.

Selecting a subset of records

Limiting the number of results on a page is simple. Add the LIMIT keyword to the SQL query like this:

```
SELECT * FROM images LIMIT startPosition, maximum
```

The LIMIT keyword can be followed by one or two numbers. If you use just one number, it sets the maximum number of records to be retrieved. That's useful, but it's not what we need to build a paging system. For that, you need to use two numbers: the first indicates which record to start from, and the second stipulates the maximum number of records to be retrieved. MySQL counts records from 0, so to display the first six images, you need the following SQL:

```
SELECT * FROM images LIMIT 0, 6
```

To show the next set, the SQL needs to change to this:

```
SELECT * FROM images LIMIT 6, 6
```

There are only eight records in the images table, but the second number is only a maximum, so this retrieves records 7 and 8.

To build the navigation system, you need a way of generating these numbers. The second number never changes, so let's define a constant called SHOWMAX. Generating the first number (call it $startRecord) is pretty easy, too. Start numbering the pages from 0, and multiply the second number by the current page number. So, if you call the current page $curPage, the formula looks like this:

```
$startRecord = $curPage * SHOWMAX;
```

And for the SQL, it becomes this:

```
SELECT * FROM images LIMIT $startRecord, SHOWMAX
```

If $curPage is 0, $startRecord is also 0 (0 × 6), but when $curPage increases to 1, $startRecord changes to 6 (1 × 6), and so on.

Since there are only eight records in the images table, you need a way of finding out the total number of records to prevent the navigation system from retrieving empty result sets. In the last chapter, you used $numRows to get this information. However, the technique that was used for the original MySQL extension and the MySQL Improved object-oriented interface won't work, because mysql_num_rows() and the num_rows property report the number of records in the *current* result set. Since you're limiting the number of records retrieved at any one time to a maximum of six, you need a different way to get the total. If you're using PDO, you already know the answer is this:

```
SELECT COUNT(*) FROM images
```

COUNT() is a SQL function that calculates the total number of results in a query. When used like this in combination with an asterisk, it gets the total number of records in the table. So, to build a navigation system, you need to run both SQL queries: one to find the total number of records, and the other to retrieve the required subset. MySQL is fast, so the result is almost instantaneous.

I'll deal with the navigation links later. Let's begin by limiting the number of thumbnails on the first page.

PHP Solution 12-4: Displaying a subset of records

Continue working with the same file. Alternatively, use gallery_mysql06.php, gallery_mysqli06.php, or gallery_pdo06.php.

1. Define SHOWMAX and the SQL query to find the total number of records in the table. Amend the code toward the top of the page like this (new code is shown in bold):

```
// define number of columns in table
define('COLS', 2);
// set maximum number of records per page
define('SHOWMAX', 6);
// create a connection to MySQL
$conn = dbConnect('query');
// prepare SQL to get total records
$getTotal = 'SELECT COUNT(*) FROM images';
```

Although COLS and SHOWMAX are defined as constants, it doesn't prevent you from offering visitors a choice of how many columns and items to display on a page. You could use variables as the second arguments to define(), and draw their values from user input.

2. You now need to run the new SQL query. The code goes immediately after the code in the preceding step, but differs according to the type of MySQL connection.

If you're using the original MySQL extension, add this:

```
// submit query and store result as $totalPix
$total = mysql_query($getTotal);
$row = mysql_fetch_row($total);
$totalPix = $row[0];
```

This introduces a new function, `mysql_fetch_row()`, which gets a single record from a result set as an indexed array (one that refers to elements by numbers). The result of `SELECT COUNT(*)` contains just one field, so you access it as `$row[0]`.

For MySQL Improved, use this:

```
// submit query and store result as $totalPix
$total = $conn->query($getTotal);
$row = $total->fetch_row();
$totalPix = $row[0];
```

This uses the MySQLI equivalent of `mysql_fetch_row()` just described. The result set for the query has been saved as `$total`, so `$total->fetch_row()` gets the record as an indexed array.

For PDO, use this:

```
// submit query and store result as $totalPix
$total = $conn->query($getTotal);
$row = $total->fetchColumn();
$totalPix = $row[0];
$total->closeCursor();
```

This is the same as in the previous chapter, using `fetchColumn()` to get a single result, and `closeCursor()` to free the database connection for the next query.

3. Next, set the value of `$curPage`. The navigation links that you will create later pass the value of the required page through a query string, so you need to check whether `curPage` has been set in the `$_GET` array. If it has, use that value. Otherwise, set the current page to 0. Insert the following code immediately after the code in the previous step:

```
// set the current page
$curPage = isset($_GET['curPage']) ? $_GET['curPage'] : 0;
```

This uses the conditional operator (see Chapter 3). If you find the conditional operator hard to understand, use the following code instead. It has exactly the same meaning.

```
if (isset($_GET['curPage'])) {
  $curPage = $_GET['curPage'];
  }
else {
  $curPage = 0;
  }
```

4. You now have all the information that you need to calculate the start row, and to build the SQL query to retrieve a subset of records. Add the following code immediately after the code in the preceding step:

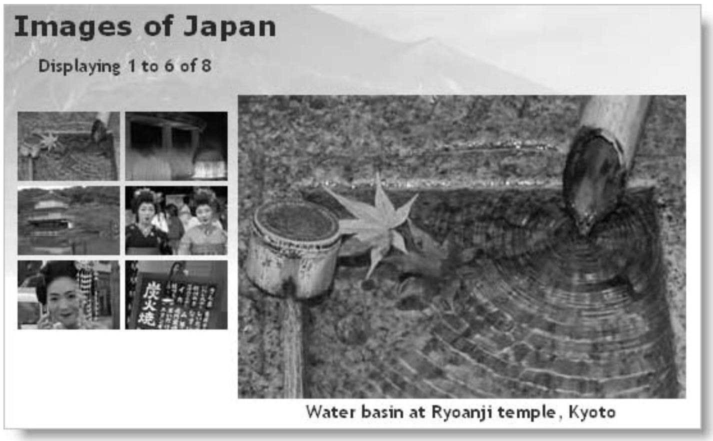
```
// calculate the start row of the subset
$startRow = $curPage * SHOWMAX;
```

The original SQL query should now be on the next line. Amend it like this:

```
// prepare SQL to retrieve subset of image details
$sql = "SELECT * FROM images LIMIT $startRow,".SHOWMAX;
```

Notice that I've used double quotes this time, because I want PHP to process `$startRow`. *Unlike variables, constants aren't processed inside double-quoted strings. So* `SHOWMAX` *is added to the end of the SQL query with the concatenation operator (a period). The comma inside the closing quotes is part of the SQL, separating the two arguments of the* `LIMIT` *clause.*

5. Save the page and reload it into a browser. Instead of eight thumbnails, you should see just six, as shown here:



Change the value of SHOWMAX to see a different number of thumbnails. The text above the thumbnail grid doesn't update because it's still hard-coded, so let's fix that.

6. Locate the following line of code in the main body of the page:

```
<p id="picCount">Displaying 1 to 6 of 8</p>
```

Replace it with this:

```
<p id="picCount">Displaying <?php echo $startRow+1;
if ($startRow+1 < $totalPix) {
  echo ' to ';
  if ($startRow+SHOWMAX < $totalPix) {
    echo $startRow+SHOWMAX;
    }
  else {
    echo $totalPix;
    }
  }
echo " of $totalPix";
?></p>
```

Let's take this line by line. The value of `$startRow` is zero-based, so you need to add 1 to get a more user-friendly number. So, `$startRow+1` displays 1 on the first page and 7 on the second page.

In the second line, `$startRow+1` is compared with the total number of records. If it's less, that means the current page is displaying a range of records, so the third line displays the text "to" with a space on either side.

You then need to work out the top number of the range, so a nested `if... else` conditional statement adds the value of the start row to the maximum number of records to be shown on a page. If the result is less than the total number of records, `$startRow+SHOWMAX` gives you the number of the last record on the page. However, if it's equal to or greater than the total, you display `$totalPix` instead.

Finally, you come out of both conditional statements and display "of" followed by the total number of records.

7. Save the page and reload it in a browser. You still get only the first subset of thumbnails, but you should see the second number change dynamically whenever you alter the value of `SHOWMAX`. Check your code, if necessary, against `gallery_mysql07.php`, `gallery_mysqli07.php`, or `gallery_pdo07.php`.

Navigating through subsets of records

As I mentioned in step 3 of the preceding section, the value of the required page is passed to the PHP script through a query string. When the page first loads, there is no query string, so the value of `$curPage` is set to 0. Although a query string is generated when you click a thumbnail to display a different image, it includes only the filename of the main image, so the original subset of thumbnails remains unchanged. To display the next subset, you need to create a link that increases the value of `$curPage` by 1. It follows, therefore, that to return to the previous subset, you need another link that reduces the value of `$curPage` by 1.

That's simple enough, but you also need to make sure that these links are displayed only when there is a valid subset to navigate to. For instance, there's no point in displaying a back link on the first page, because there isn't a previous subset. Similarly, you shouldn't display a forward link on the page that displays the last subset, because there's nothing to navigate to.

Both issues are easily solved by using conditional statements. There's one final thing that you need to take care of. You must also include the value of the current page in the query string generated when you click a thumbnail. If you fail to do so, $curPage is automatically set back to 0, and the first set of thumbnails is displayed instead of the current subset.

PHP Solution 12-5: Creating the navigation links

Continue working with the same file. Alternatively, use gallery_mysql07.php, gallery_mysqli07.php, or gallery_pdo07.php.

1. I have placed the navigation links in an extra row at the bottom of the thumbnail table. Insert this code between the placeholder comment and the closing </table> tag:

```
<!-- Navigation link needs to go here -->
<tr><td>
<?php
// create a back link if current page greater than 0
if ($curPage > 0) {
  echo '<a href="'.$_SERVER['PHP_SELF'].'?curPage='.($curPage-1).'"> ➥
&lt; Prev</a>';
  }
// otherwise leave the cell empty
else {
  echo '&nbsp;';
  }
?>
</td>
<?php
// pad the final row with empty cells if more than 2 columns
if (COLS-2 > 0) {
  for ($i = 0; $i < COLS-2; $i++) {
    echo '<td>&nbsp;</td>';
    }
  }
?>
<td>
<?php
// create a forward link if more records exist
if ($startRow+SHOWMAX < $totalPix) {
  echo '<a href="'.$_SERVER['PHP_SELF'].'?curPage='.($curPage+1).'"> ➥
Next &gt;</a>';
  }
```

```
// otherwise leave the cell empty
else {
  echo '&nbsp;';
  }
?>
</td></tr>
</table>
```

It looks like a lot, but the code breaks down into three sections: the first creates a back link if $curPage is greater than 0; the second pads the final table row with empty cells if there are more than two columns; and the third uses the same formula as before ($startRow+SHOWMAX < $totalPix) to determine whether to display a forward link.

When typing this code, make sure that you get the combination of quotes right in the links. The other point to note is that the $curPage-1 and $curPage+1 calculations are enclosed in parentheses to avoid the period after the number being misinterpreted as a decimal point. It's used here as the concatenation operator to join the various parts of the query string.

2. You now need to add the value of the current page to the query string in the link surrounding the thumbnail. Locate this section of code:

```
<a href="<?php echo $_SERVER['PHP_SELF']; ?>?image=<?php echo ➥
$row['filename']; ?>">
```

Change it like this:

```
<a href="<?php echo $_SERVER['PHP_SELF']; ?>?image=<?php echo ➥
$row['filename']; ?>&amp;curPage=<?php echo $curPage; ?>">
```

You want the same subset to be displayed when clicking a thumbnail, so you just pass the current value of $curPage through the query string.

Notice that I have used the HTML entity &amp; *to add a second name/value pair to the query string. This is displayed in the browser status bar or address bar simply as an ampersand. Although using an ampersand on its own also works,* &amp; *is required for valid XHTML.*

3. Save the page and test it. Click the Next link, and you should see the remaining subset of thumbnails, as shown in Figure 12-5. There are no more images to be displayed, so the Next link disappears, but there's a Prev link at the bottom left of the thumbnail grid. The record counter at the top of the gallery now reflects the range of thumbnails being displayed, and if you click the right thumbnail, the same subset remains onscreen while displaying the appropriate large image. You're done!

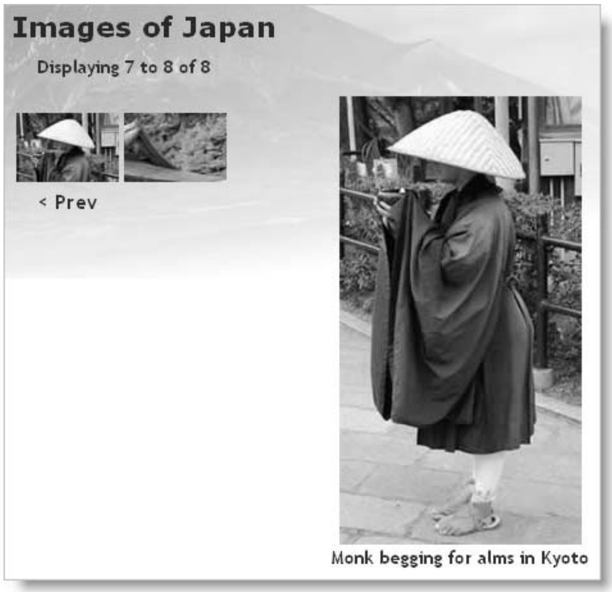


Figure 12-5. The page navigation system is now complete.

Check your code, if necessary, against `gallery_mysql08.php`, `gallery_mysqli08.php`, or `gallery_pdo08.php`.

Summary

Wow! In a few pages, you have turned a boring list of filenames into a dynamic online gallery, complete with a page navigation system. All that's necessary is to create a thumbnail for each major image, upload both images to the appropriate folder, and add the filename and a caption to the `images` table in the database. As long as the database is kept up to date with the contents of the `images` and `thumbs` folders, you have a dynamic gallery. Not only that, you've learned how to select subsets of records, link to related information through a query string, and build a page navigation system.

The more you use PHP, the more you realize that the skill doesn't lie so much in remembering how to use lots of obscure functions, but in working out the logic needed to get PHP to do what you want. It's a question of if this, do that; if something else, do something different. Once you can anticipate the likely eventualities of a situation, you can normally build the code to handle it.

So far, you've concentrated on extracting records from a simple database table. In the next chapter, I'll show you how to insert, update, and delete material.