Foundation PHP for Dreamweaver 8

David Powers

Foundation PHP for Dreamweaver 8

ISBN (pbk): 1-59059-569-6

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit www.apress.com.

The source code for this book is freely available to readers at www.friendsofed.com in the Downloads section.

Credits

Lead Editor
Chris Mills

Technical Reviewer
Jason Nadon

Editorial Board
Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser

Project Manager
Richard Dal Porto

Copy Edit Manager
Nicole LeClerc

Copy Editor
Andy Carroll

Assistant Production Director
Kari Brooks-Copony

Production Editor
Kelly Winquist

Compositor
Katy Freer

Proofreader
Nancy Sixsmith

Indexer
Diane Brenner

Cover Artist
Corné van Dooren

Interior and Cover Designer
Kurt Krames

Manufacturing Director
Tom Debolski

Excerpt from Chapter 11

DISPLAYING A BLOG AND PHOTO GALLERY

What this chapter covers:

- Storing and formatting dates in MySQL
- Handling image details in a database
- Displaying a message when no records are found
- Alternating the colors of table rows
- Displaying an extract of a long text item
- Using Live Data view with a URL parameter
- Dynamically generating the code for the dimensions of an image
- Using a commercial extension to loop horizontally

Now that you know how to store, update, and delete records, it's time to turn to the display and formatting of material stored in a database. I've chosen a blog because it helps illustrate such things as working with dates. It also gives me an opportunity to show you how to display a short extract from a longer item and link to the full version on a different page, a technique that can also be used in online catalogs and a wide variety of other applications.

The other main focus of this chapter is how you work with images in a database-driven site. A lot of beginners try to store images in their database, and are then distressed to discover that, instead of a beautiful image, they get a stream of incomprehensible text. You *can* store images in a database, but I'll explain why it's not a good idea, and what to do instead.

Creating the blog back-end

As always, the first step in building a database-driven site is deciding what information needs to be stored in the database. You then need to build four pages for each table:

- An insert form for new records
- A page to list all records, with links to update and delete forms
- An update form
- A delete form

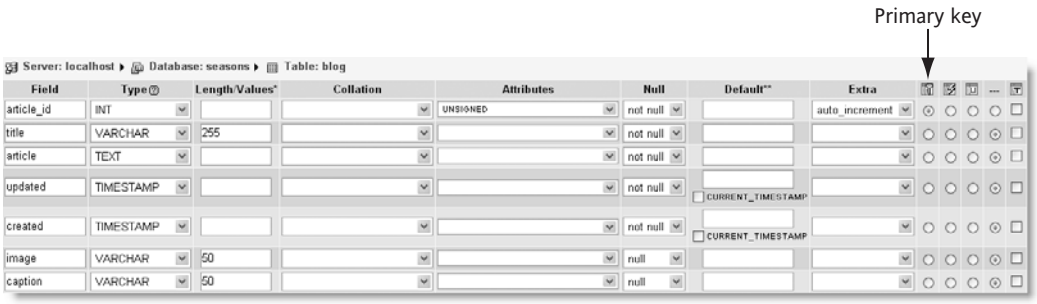
Planning and building the required components

In the interest of keeping things simple, the blog won't have such features as being sortable by categories, a calendar, or a facility for others to post comments. Each record will simply have a title, an article, the date and time it was originally created, the date and time it was most recently updated, and the option to include an image. The images will be kept outside the database in an ordinary folder, so all that needs to be stored in the database is the filename. Each image will need some alternative text, which can also double up as a caption. This means that a single database table with seven columns, plus one set of administration pages, will be sufficient.

Creating the blog table in the database

1. Open phpMyAdmin and select the `seasons` database. Create a new table called `blog` with seven columns (fields).
2. Define the columns as follows (see also screenshot at the top of the next page):
   - `article_id` is the table's primary key.
   - `title` has been given the maximum length (255) permitted for `VARCHAR` column types prior to MySQL 5.0. This should be more than enough for a headline.
   - `article` is where you will write your blogging masterpieces, so it has been given a column type of `TEXT`.

- updated and created will store the date and time of changes to each article, along with the date and time of its original creation. Although it may seem more logical to put created first, the columns *must* be in this order. I'll explain why shortly. When you select the TIMESTAMP column type, phpMyAdmin adds a check box labeled CURRENT_TIMESTAMP. Leave this unchecked for both columns.
- image will store the name of any image associated with the article. I've used a VARCHAR column type with a length of 50, which should be more than adequate. Because not every article will be associated with an image, the Null drop-down has been set to null.
- caption will store the alt text for the image. Captions and alt text are best kept short on the Web, so I've used a VARCHAR column with a length of 50, and set the Null drop-down to null.



3. Click Save when you have finished. The table structure should look like this:

	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	article_id	int(10)		UNSIGNED	No		auto_increment	
☐	title	varchar(255)	latin1_swedish_ci		No			
☐	article	text	latin1_swedish_ci		No			
☐	updated	timestamp		ON UPDATE CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP		
☐	created	timestamp			No	0000-00-00 00:00:00		
☐	image	varchar(50)	latin1_swedish_ci		Yes	*NULL*		
☐	caption	varchar(50)	latin1_swedish_ci		Yes	*NULL*		

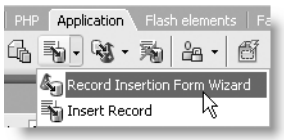
Note the Attributes and Default values for the updated field. (If you are using an old version of phpMyAdmin or MySQL prior to the 4.1 series, the values will be empty. However, the behavior of the updated column will be the same.) These indicate a useful feature of MySQL TIMESTAMP columns. The first TIMESTAMP column in a table is automatically updated to the current date and time whenever changes are made to a record. However, subsequent TIMESTAMP columns have to be set specifically.

What this means is that you can set the value of created when you first insert a record into the table, and it will never change. The updated column, on the other hand, will always change whenever the record is updated. What's more, because they're TIMESTAMP columns, MySQL automatically inserts the date and time in the correct format without any need for user input.

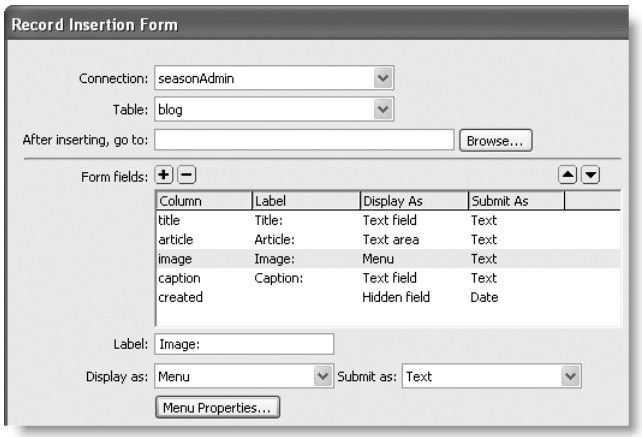
Creating an insert form for new blog entries

Because of the emphasis on simplicity, the insert and update forms won't include any validation. The quickest way to build the forms is with the Dreamweaver wizards.

1. Create a new PHP page in the `admin` folder. Save it as `blog_insert.php`, and attach the `admin.css` stylesheet. Give the page a suitable title and heading.
2. Make sure that your cursor is outside the closing `</h1>` tag, and click the Record Insertion Form Wizard icon on the Application Insert bar. Set the Connection field to seasonAdmin, and the Table field to blog.

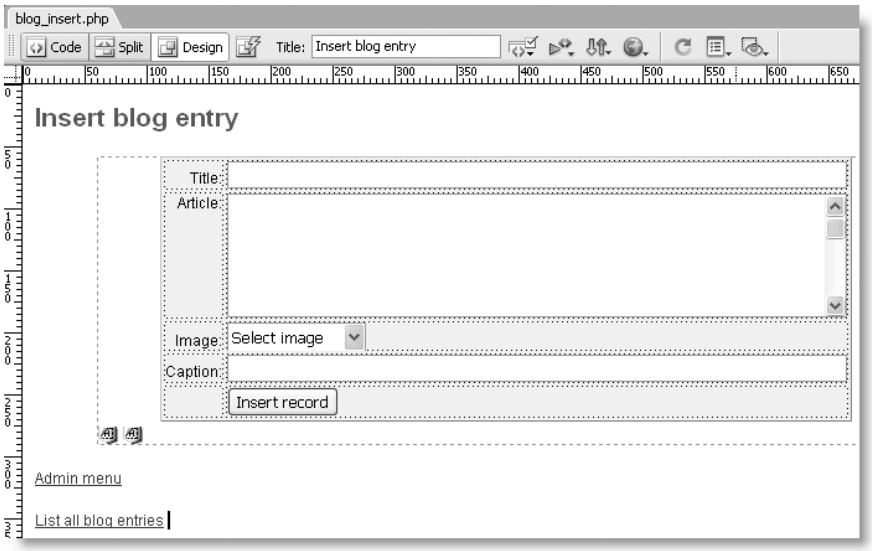


3. Use the minus button to remove article_id and updated from the Form fields area.
4. Set the Display as setting for article to Text area.
5. Set the Display as setting for created to Hidden field, and type NOW() in the Default value field. Use the down arrow button to move created to the bottom of the list. `NOW()` is a MySQL function that generates a current date and time. Dreamweaver doesn't have a method of passing MySQL functions like this, so you need to tweak the server behavior code later.
6. Set the Display as setting for image to Menu. This reveals a button labeled Menu Properties. Click the button, and enter Select image in the Label field. Leave the other fields blank, and click OK.
7. Leave the settings for caption unchanged, check that the Record Insertion Form dialog box settings are like those in the screenshot, and click OK.



8. After Dreamweaver has built the insert form, select the title text field, and choose widebox from the Class drop-down menu in the Property inspector. This is a style in `admin.css` that makes the text field the same width as the text area beneath it. Set Max Chars to 255 to avoid entering text that exceeds the maximum length for the `title` column.
9. Select the caption text field and apply the `widebox` class. Set Max Chars to 50.

10. Highlight the form in the Tag selector, and press your right arrow key once to move the cursor outside the form. Add two text links: one to the admin menu (`menu.php`), and the other to `list_blog.php`. This second page will display a list of all blog entries. If necessary, create `list_blog.php` as a blank page. Your page should now look like this.



11. The page still needs the code for the `image` drop-down menu, which will be added in the next section. Check your code so far against `blog_insert_01.php` in `site_check/ch11`.

Building a list of images in a folder

One of the main reasons for using a server-side language like PHP is to reduce the amount of effort it takes to create a website. In Chapter 9, you learned how to populate a drop-down menu from a database. It's not only a time-saver; a dynamically populated menu is always up to date. However, you don't always need to store information in a database for PHP to be able to interact with it. PHP has an extensive suite of built-in functions that allow you to work directly with your computer's file system (`www.php.net/manual/en/ref.filesystem.php`).

I have written a custom function that makes use of several of these built-in functions to open an images folder, build an array of all image files, close the folder, and then use the array to populate a drop-down menu. Although I don't intend to explain all the details of this rather complex function, the basic way most PHP file system functions work is that you create a variable known as a file handler to open the file or folder, and then pass the file handler as the argument to the functions that interact directly with the file or folder. To traverse the contents of a folder called `images_blog`, for instance, this is the sequence of events:

```
// create a file handler
$fileHandler = opendir('path_to_folder/images_blog/');
// use a while loop to go through the folder
while (readdir($fileHandler)) {
  // do something with the files in the folder
  }
// close the folder
closedir($fileHandler);
```

As you can see, the function names are intuitive: `opendir()`, `readdir()`, and `closedir()`, with `dir` being short for "directory," which most Mac and Windows users now refer to as a "folder." (Such changes in terminology are aimed at making computers seem more "friendly," but they often end up simply confusing people.) The equivalent functions that work with individual files are called `fopen()`, `fread()`, and `fclose()`. You will find links to descriptions of all of them, together with examples of how they're used in the PHP manual at the URL given earlier in this section.

The custom function is in one of the PHP snippets that you installed in Chapter 6, so most of the work in this next section is done for you.

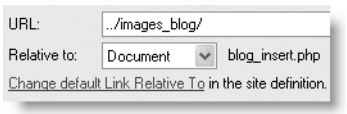
Dynamically populating the image menu

Continue working with the same file as in the previous section, or use `blog_insert_01.php` from `site_check/ch11`.

1. Select the menu object in Design view to locate the underlying code, and switch to Code view. Insert a call to the custom function like this (new code is shown in bold):

   ```
   <td><select name="image">
     <option value="">Select picture</option>
     <?php buildImageList('../images_blog/"); ?>
   </select></td>
   ```

   The `buildImageList()` function can take up to two arguments, but only the first one is required—the path to the folder from which you want to build the list. I have given you the correct relative path to `images_blog`, but if you're ever in doubt about the path to a folder, right-click/*Ctrl*-click in Code view and select Code Hint Tools ➤ URL Browser from the context menu. Click Browse and navigate to the target folder. The correct relative path appears in the URL field of the Select File dialog box. Highlight the contents of the field, and copy (*Ctrl+C*/⌘+*C*) it to your clipboard. Click Cancel to close the dialog box, and then paste (*Ctrl+V*/⌘+*V*) the URL into your script.



2. Unless you put it in an external file, it doesn't matter where you define a PHP function, so scroll down to the bottom of the page, create a pair of PHP tags after the closing `</html>` tag, and place your cursor between them. Open the Snippets panel, and double-click Build image list in the PHP-DW8 folder. (Alternatively, you can right-click/*Ctrl*-click the snippet name, and select Insert from the context menu.) It will insert the following code:

```
function buildImageList($imageFolder, $recordset=NULL) {
// Check whether image folder has trailing slash, add if needed
$imageFolder = strrpos($imageFolder,'/') == strlen($imageFolder-1)
➥ ? $imageFolder : "$imageFolder/";
// Execute code if images folder can be opened, or fail silently
if ($theFolder = @opendir($imageFolder)) {
  // Create an array of image types
  $imageTypes = array('jpg','jpeg','gif','png');
  // Traverse images folder, and add filename to $img array if an image
  while (($imageFile = readdir($theFolder)) !== false) {
    $fileInfo = pathinfo($imageFile);
    if (in_array($fileInfo['extension'],$imageTypes)) {
      $img[] = $imageFile;
      }
    }
  // Close the stream from the images folder
  closedir($theFolder);
  // Check the $img array is not empty
  if ($img) {
    // Sort in natural, case-insensitive order, and populate menu
    natcasesort($img);
    foreach ($img as $image) {
      echo "<option value='$image'";
      // Set selected image if recordset details supplied
      if ($recordset != NULL && $recordset == $image) {
        echo  ' selected="selected"';
        }
      echo ">$image</option>\n";
      }
    }
  }
}
```

The inline comments explain what each stage of the function does, but I don't expect most readers to delve into the details. If you *are* interested, visit the PHP online documentation at www.php.net/manual/en and type the names of the functions I've used into the search field at the top of the page.

The second (optional) argument is needed only when you want one of the images to be selected when the menu is displayed. I'll explain how to set the second argument when building the update page.

If you decide to put the definition in an external file, the include command must come before you call the function.

3. Finally, you need to tweak the server behavior code to pass the MySQL `NOW()` function to the SQL query. Scroll up until you find this section of code:

```
33 if ((isset($_POST["MM_insert"])) && ($_POST["MM_insert"] == "form1")) {
34   $insertSQL = sprintf("INSERT INTO blog (title, article, image, caption, created) VALUES (%s,
   %s, %s, %s, %s)",
35                        GetSQLValueString($_POST['title'], "text"),
36                        GetSQLValueString($_POST['article'], "text"),
37                        GetSQLValueString($_POST['image'], "text"),
38                        GetSQLValueString($_POST['caption'], "text"),
39                        GetSQLValueString($_POST['created'], "date"));
```

4. `GetSQLValueString()` is a Dreamweaver custom function that prepares user input for insertion into a SQL query with all quotes properly escaped. It's not capable of handling the `NOW()` function. Amend the final line (shown on line 39 of the preceding screenshot) so that just the `POST` variable, the closing parenthesis, and the semicolon are left remaining. It should look like this:

   ```
   $_POST['created']);
   ```

   This change may cause Dreamweaver to display a warning message about a JavaScript error. Just click OK. Both the page and Dreamweaver remain fully functional. Check your code against `blog_insert_02.php` in `site_check/ch11`.

Dreamweaver frequently runs JavaScript in the background to populate the various panels and the Property inspector, and a missing piece of code usually triggers an error. If you close and reopen `blog_insert.php`*, you will see a detailed message telling you where the error occurred. This page is now complete, so you can close it to avoid being disturbed by error messages.*

Displaying a message when no records are found

Before you test the insert record page, let's build `list_blog.php`, as this presents an opportunity to use the Show Region server behavior. As the name suggests, this server behavior selectively shows a section of your page, depending on the results of a recordset. If the recordset produces no results, you can hide the section where the results would normally be displayed and show an alternative message instead. The process involves two stages:

1. Apply a Show Region server behavior to the section of the page where the results are normally displayed.
2. Create alternative content that you want to display if the recordset is empty, and apply the opposite Show Region server behavior to the alternative content.

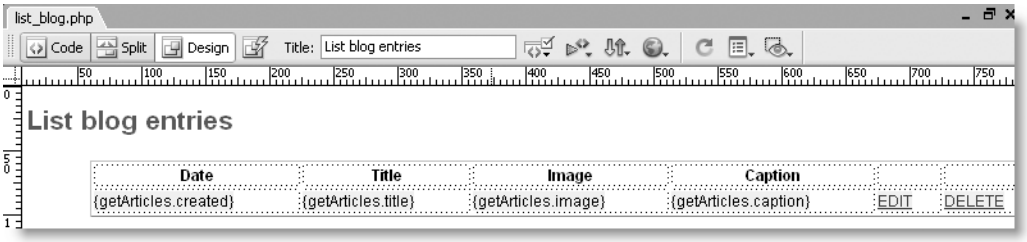
Before you can do that, you need to build the table to display the recordset results in the normal way.

Displaying a list of blog entries

1. If you haven't already created a blank page for `list_blog.php`, do so now. Also create two blank pages for `blog_update.php` and `blog_delete.php`. Save all three pages in the `admin` folder.
2. Give `list_blog.php` a suitable title and heading, and attach the `admin.css` stylesheet.
3. Make sure the cursor is outside the closing `</h1>` tag of the heading, and insert a table. In the Table dialog box, set Rows to 2, Columns to 6, and Table width to 750 pixels. Leave Border thickness, Cell padding, and Cell spacing blank, and select Top for the Header setting. Click OK.
4. Select the entire table, and set Table Id in the Property inspector to striped. This will enable it to pick up the style rules in `admin.css` that set padding on the table cells. Using CSS instead of the `cellpadding` attribute enables you to set different amounts of padding on each side of a table cell, rather than a single value all round.
5. Create a recordset. You should know how to do this by now, so I'll just give the basic details. Use the simple Recordset dialog box with the following settings:
   - Name: getArticles
   - Connection: seasonAdmin
   - Table: blog
   - Columns: Selected (highlight article_id, title, created, image, and caption)
   - Filter: None
   - Sort: created Descending

> *If you installed MX Kollection in the previous chapter, you will notice a new button labeled* QuB3 *in the Recordset dialog box. This is for the MX Kollection advanced SQL query builder. Since it's not a standard part of Dreamweaver, it's not covered in this book. Read the MX Kollection help files for an explanation of how to use it.*

6. Expand Recordset (getArticles) in the Bindings panel, and drag dynamic text placeholders for created, title, image, and caption into the first four cells of the table's second row. Create EDIT and DELETE links in the final two cells in exactly the same way as you have done since Chapter 8 (refer to the "Creating a list of all records" section in that chapter if you need to refresh your memory). The EDIT link should point to `blog_update.php`, and the DELETE link to `blog_delete.php`. Both links should use a parameter named `article_id`, which draws its value from `article_id` in the getArticles recordset. Insert appropriate column headings in the first four cells of the top row. Your page should now look like this:



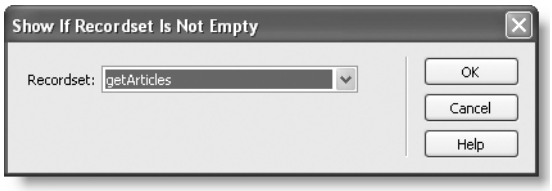
7. Highlight the second table row, and apply a Repeat Region server behavior (this was covered in the same section of Chapter 8). Select the option to show all records.
8. Save `list_blog.php`, and press *F12*/*Opt+F12* to view the page in a browser. As long as you have no records in the `blog` table, you should see something like this:

If you hover your mouse pointer over either of the links, you will see that the query string contains no value for the `article_id` parameter. For an administration page that the public will never see, this is perhaps acceptable, but it would give a very bad impression in the front-end of a dynamic website. The answer lies in using the Show Region server behavior, which I'll show you how to apply in the next section. If you want to check your code so far, compare it with `list_blog_01.php` in `site_check/ch11`.

Applying the Show Region server behavior

Continue working with the same file as in the previous section, or use `list_blog_01.php` from `site_check/ch11`.

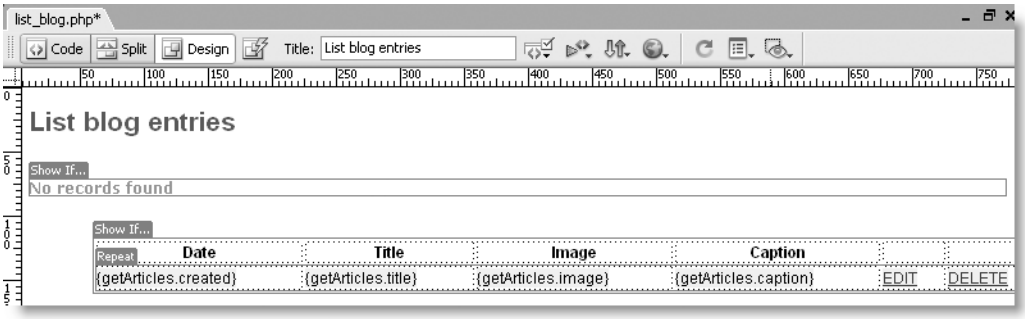
1. Position your cursor anywhere inside the table, and click the `<table#striped>` tag in the Tag selector to select the whole table.
2. Click the plus button in the Server Behaviors panel, and select Show Region ➤ Show If Recordset Is Not Empty. Alternatively, you can select the same options by clicking the Show Region icon on the Application Insert bar, or from the Application Objects submenu of the Insert menu.
3. The dialog box that opens couldn't be simpler. All you need to do is select the correct recordset. Since `getArticles` is the only recordset on the page, just click OK.



4. Dreamweaver puts a gray border around the table with a Show If tab at the top left, as shown alongside.

5. Save the page, and view it in a browser. All you should see is the page heading.

6. Since you want to display a message indicating that no records have been found, it doesn't matter whether you place it above or below the table. Either the table will be shown and the message hidden, or the other way round. Click in a blank area of the Document window to make sure nothing is selected, and then insert a new paragraph containing the words "No records found." To make the message stand out, apply the `warning` class to the paragraph.
7. Highlight the paragraph by clicking the `<p.warning>` tag in the Tag selector. Then apply a Show Region server behavior. This time choose Show If Recordset Is Empty. Again, the dialog box simply requires you to select the recordset. Since there is only one on the page, just click OK. Your page should now look like this:



8. Save `list_blog.php`, and view it in a browser. You should now see the No records found message. You can check your code against `list_blog_02.php` in `site_check/ch11`.

As soon as you insert the first record in the `blog` table, `list_blog.php` will hide the No records found message and start displaying the date and time of creation, the title, and the name of any image linked with a blog entry.

A quick way to load some blog entries To save you the effort of typing out genuine masterpieces of blog literature, open a temporary page in Dreamweaver and use the Lorem and More extension to create about 15 paragraphs of corporate mumbo-jumbo, which you can cut and paste into the Article field of `blog_insert.php` to insert three or four records. Display the list of blog entries by clicking the link at the bottom of `blog_insert.php`. You should see something like Figure 11-1.

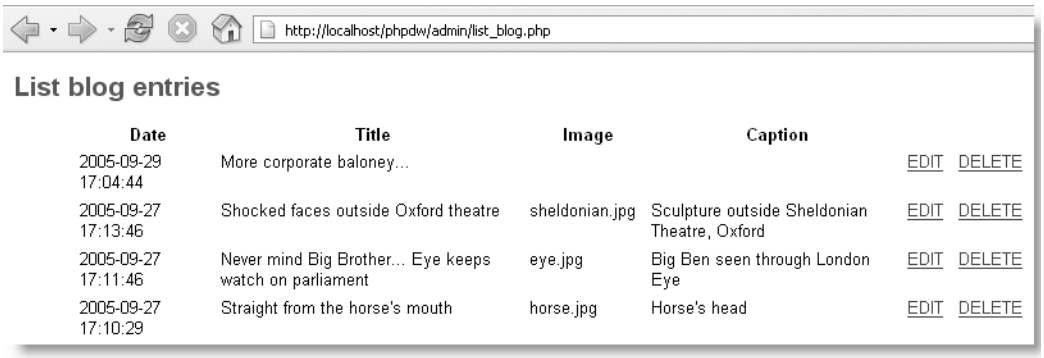


Figure 11-1. The value of the MySQL `TIMESTAMP` column isn't in a very friendly format.

Although the page is reasonably usable, the format of the date and time is rather inelegant. In fact, if your hosting company is still using MySQL 3.23 or 4.0, the date and time in the first line would look like this: 20050929170444. The numbers are the same, but with no punctuation the date and time becomes extremely difficult to read. Definitely a case for treatment . . .

Formatting dates and time in MySQL

Figure 11-1 shows the TIMESTAMP format used by MySQL since the release of the 4.1 series. The date portion at the beginning follows the only pattern that MySQL accepts for dates—year, month, and day of month *in that order*. Because the values were automatically generated, MySQL used the correct format. I'll come back at the end of the chapter, in the section "Storing dates in MySQL," to discuss the best way of handling dates in user input, but, for the moment, your main concern is how to display a date that has already been stored in this format. Fortunately, converting dates from MySQL format to a more user-friendly one is relatively easy, and it's best done as part of the SQL query when building a recordset.

MySQL has a wide range of date and time functions, all of which are listed together with examples at http://dev.mysql.com/doc/refman/5.0/en/date-and-time-functions.html. They allow you to perform many useful calculations, such as working out people's ages from their birthdates, calculating the difference between two dates, and adding to or subtracting from dates. The function that concerns us here is DATE_FORMAT(), which does exactly what its name suggests. The syntax for DATE_FORMAT() is as follows:

```
DATE_FORMAT(date, format)
```

Normally, *date* is the name of the table column that you want to format, and *format* is either a string or another function that tells MySQL which format to use. The DATE_FORMAT() function works in a very similar way to the PHP date() function, in that you build the format string from specifiers. Table 11-1 lists those most commonly used.

Table 11-1. Frequently used MySQL date format specifiers

Period	Specifier	Description	Example
Year	%Y	Four-digit format	2005
	%y	Two-digit format	05
Month	%M	Full name	January, September
	%b	Abbreviated name, 3 letters	Jan, Sep
	%m	Number, with leading zero	01, 09
	%c	Number, no leading zero	1, 9
Day of month	%d	With leading zero	01, 25
	%e	No leading zero	1, 25

Period	Specifier	Description	Example
	%D	With English text suffix	1st, 25th
Weekday name	%W	Full text	Monday, Thursday
	%a	Abbreviated name, 3 letters	Mon, Thu
Hour	%H	24-hour clock, with leading zero	01, 23
	%k	24-hour clock, no leading zero	1, 23
	%h	12-hour clock, with leading zero	01, 11
	%l (lowercase *L*)	12-hour clock, no leading zero	1, 11
Minute	%i	With leading zero	05, 25
Second	%S	With leading zero	08, 45
AM/PM	%p		

The specifiers can be combined with ordinary text or punctuation in the format string. As always, when using a function in a SQL query, assign the result to an alias using the AS keyword, in the same way as you did with CONCAT() in Chapter 9. Referring to Table 11-1, you can now format the date in the created column in a variety of ways. To present the date in a common U.S. style and retain the name of the original column, use the following:

```
DATE_FORMAT(created, '%c/%e/%Y') AS created
```

To format the same date in European style, reverse the first two specifiers like this:

```
DATE_FORMAT(created, '%e/%c/%Y') AS created
```

Keeping all these specifiers in your head is a problem, so MySQL 4.1 introduced a new function that creates some standard formats using easy-to-remember strings. It's called GET_FORMAT(), and it takes two arguments: the type of date output that you want and the format. The first argument can be one of three values, DATE, TIME, or DATETIME, which produce the date or time on its own, or both together. The two most useful formats are 'USA' and 'EUR'. You use GET_FORMAT() like this:

```
DATE_FORMAT(date, GET_FORMAT(type, format))
```

Table 11-2 shows how different combinations of GET_FORMAT() render the TIMESTAMP value in the first blog entry in Figure 11-1 (2005-09-29 17:04:44).

Table 11-2. The same date and time value rendered in different ways by `GET_FORMAT()`

Arguments	Output
`GET_FORMAT(DATE, 'USA')`	09.29.2005
`GET_FORMAT(DATE, 'EUR')`	29.09.2005
`GET_FORMAT(TIME, 'USA')`	05:04:44 PM
`GET_FORMAT(TIME, 'EUR')`	17:04:44
`GET_FORMAT(DATETIME, 'USA')`	2005-09-29 17.04.44
`GET_FORMAT(DATETIME, 'EUR')`	2005-09-29 17.04.44

These formats are fixed, so if you don't like periods instead of slashes, you need to build your own formats with the specifiers listed in Table 11-1, as you'll also need to do if your server uses a version of MySQL earlier than 4.1.

You can now format the `TIMESTAMP` value in `list_blog.php` in a way that's easier to read.

Formatting the date in the blog entry list

Continue working with the same file as in the "Applying the Show Region server behavior" section, or use `list_blog_02.php` from `site_check/ch11`.

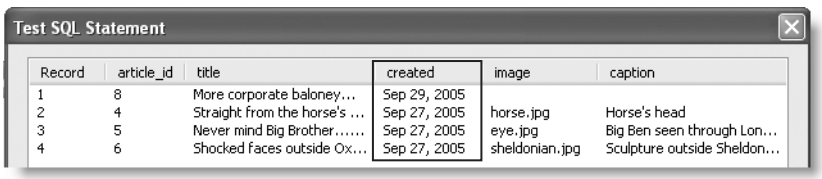
1. In the Server Behaviors panel, double-click Recordset (getArticles). When the Recordset dialog box opens, click the Advanced button.
2. The first line of the SQL query looks like this:

   ```
   SELECT article_id, title, created, image, caption
   ```

3. For a U.S.-style date, amend it like this:

   ```
   SELECT article_id, title, DATE_FORMAT(created, '%b %e, %Y') AS created,
   ➥ image, caption
   ```

   Reverse the order of the first two specifiers for a European-style date.
4. Click the Test button to make sure that everything is working correctly. You should still have a column called created, but the dates will now be formatted as shown here:



If Dreamweaver displays a MySQL error message instead, check that you have not left any space between DATE_FORMAT and the opening parenthesis of the function. Although some computer languages allow you to leave a space, MySQL doesn't. Also make sure that the format string is enclosed in matching quotes. Although I have used single quotes, double quotes are equally acceptable.

5. Assuming that the test is successful, click OK to close the test panel and then to close the Recordset dialog box.
6. Save list_blog.php, and view it in a browser. The dates should now be formatted in a more user-friendly way. You can check your code against list_blog_03.php in site_check/ch11.

Creating striped table rows

Viewing a long list of similar items on a computer screen can be tiring on the eyes, so it's often useful to give alternate rows a subtle background color. This is very easy to do with a CSS style rule, but it can be a nightmare to maintain on a static site. You need to apply the CSS class to every alternate row, and if you add another row in the middle of the table at a later date, everything has to be restyled. With a dynamically generated table, that's no problem.

The way that you do it relies on a little bit of simple math and the way that PHP treats 1 and 0. In Chapter 5, I introduced you to the modulo operator (%), which produces the remainder of a division. If you divide any number by 2, the remainder will always be either 1 or 0. I also explained in Chapter 5 that PHP treats 0 as false, but 1 is treated as true. So, if you create a variable as a counter, and increment it by 1 each time a new table row is added, you can use modulo to divide the variable by 2. This will produce a result of 1 (true) or 0 (false) every alternate row, which can be used as the test of an if statement that inserts the CSS class for the different background. Let's try it out on list_blog.php.

Using modulo to create stripes in alternate rows

Continue working with the same file, or use list_blog_03.php from site_check/ch11.

1. Switch to Code view, and locate the following section of code:

```
30      </tr>
31      <?php do { ?>
32        <tr>
33          <td><?php echo $row_getArticles['created']; ?></td>
```

The code shown on line 31 is the start of the do . . . while loop that iterates through the getArticles recordset to display the list of blog entries. (The do . . . while loop was explained in the "Using loops to handle repetitive tasks" section of Chapter 6.)

2. Amend the code like this (new code is shown in bold type):

```
</tr>
<?php $counter = 0; // initialize counter outside loop ?>
<?php do { ?>
  <tr <?php if ($counter++ % 2) {echo 'class="hilite"';} ?>>
    <td><?php echo $row_getArticles['created']; ?></td>
```

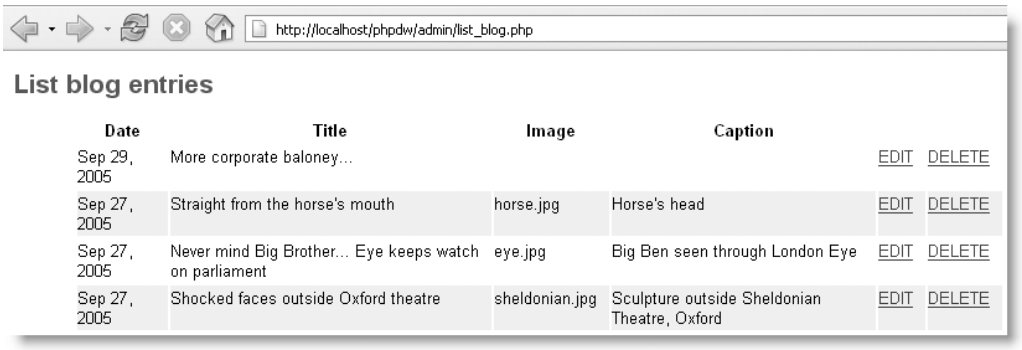
It involves just two short blocks of PHP code to perform the following tasks:

- Initialize the counter outside the loop
- Increment the counter by 1 inside the loop, and use modulo to create a Boolean (`true`/`false`) test to insert the `hilite` class in every alternate row

As you may remember from Chapter 5, the increment operator (++) performs the current calculation and then adds 1 to the variable. So, the first time through the loop `$counter` will be 0. When divided by 2, the remainder will be 0. Since this equates to `false`, the `hilite` class won't be inserted into the `<tr>` tag. However, `$counter` is incremented by 1, so the next time through the loop, the calculation will attempt to divide 1 by 2, producing a remainder of 1, which equates to `true`. The third time through the loop, `$counter` will be 2, producing a remainder of 0, and so on until the loop comes to an end.

When hand-coding, I would normally combine the initialization of the counter with the do PHP block. However, I have used separate blocks here to avoid breaking Dreamweaver's Repeat Region server behavior code.

3. Save `list_blog.php`, and view it in a browser. Voilà, stripes. You can check your code against `list_blog_04.php` in `site_check/ch11`.



The code inserted in step 2 creates a class that affects only even-numbered rows. Some developers use slightly more complex code to insert a different class in odd-numbered rows, too. This isn't necessary. By utilizing the cascade in your CSS, you can set a default background color for the table, and override it with the `hilite` class like this:

```
#striped tr {background-color: #EEE;}
#striped tr.hilite {background-color: #E8F2F8;}
```

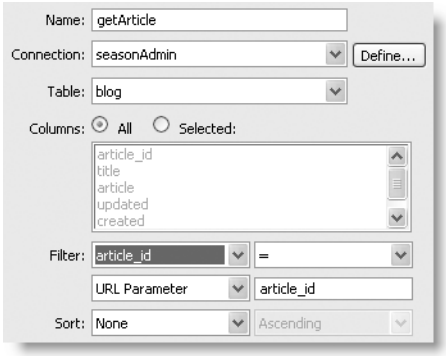
These rules will produce alternate pale gray and pale blue stripes in a table with an ID called `striped`. If you want to use the same effect in more than one table, change `striped` from an ID to a class.

Finishing the back-end

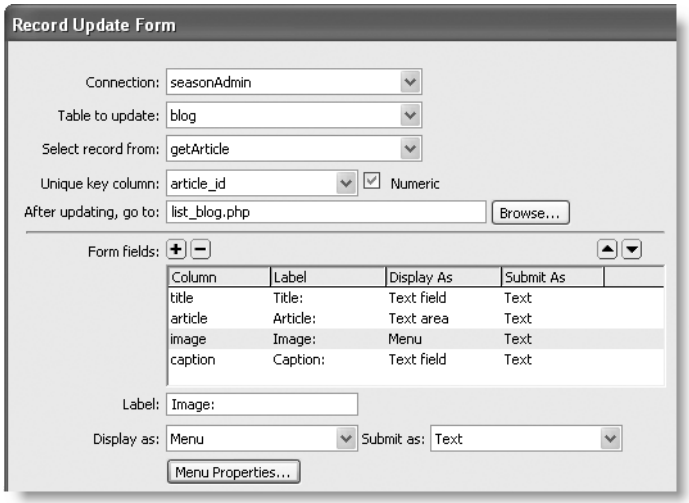
All that remains for the back-end of the blog is to build the update and delete forms. These will be very simple to build, again using the Dreamweaver wizards. The only point that requires special attention is in the update form. The drop-down menu that lists the images will need to display the correct filename. That's easily handled with the `buildImageList()` function.

Creating the update page

1. Open `blog_update.php`, the blank page that you created earlier in the chapter. Attach the `admin.css` stylesheet, and give the page a suitable title and heading.
2. You need a recordset called `getArticle` to retrieve the details of the blog entry that you are going to update. Use the simple Recordset dialog box with the settings shown in the screenshot alongside. Click OK to create the recordset.

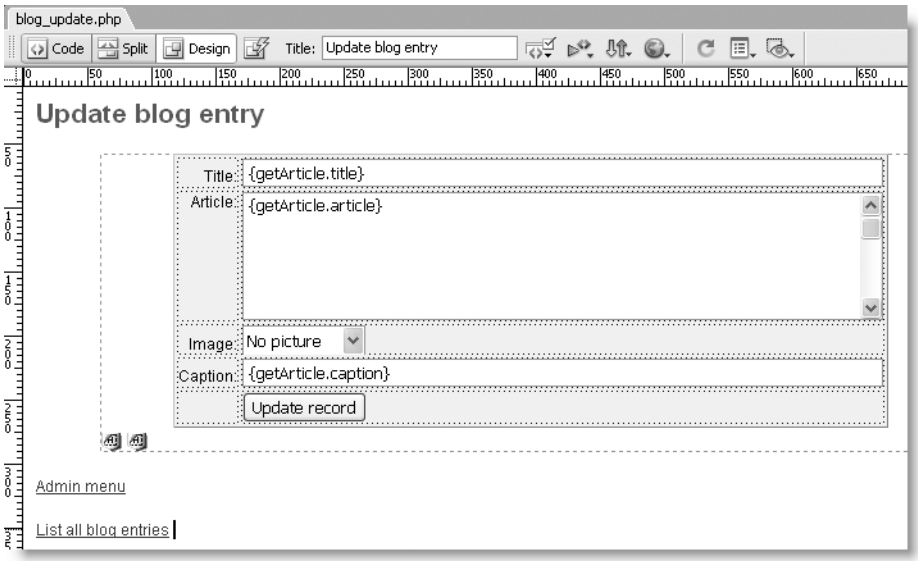


3. Make sure that your cursor is to the right of the closing `</h1>` tag of the heading, and select Record Update Form Wizard from the Application Insert bar.
4. In the Record Update Form dialog box, set Connection to seasonAdmin and Table to update to blog. Dreamweaver should automatically set the correct values for Select record from and Unique key column.
5. Set the value of the After updating go to field to list_blog.php.
6. In the Form fields area, use the minus button to remove article_id, updated, and created. MySQL will automatically insert a new date and time value in the `updated` column, and you don't want to change the value of the `created` column.
7. Change the value of Display as for article to Text area.
8. Change the value of Display as for image to Menu. Click the Menu Properties button, and set the value of Label to No picture. Leave the other fields as they are, and click OK.
9. Check that the settings in the Record Update Form dialog box look like this, and click OK to create the form.



10. Highlight the title text field, and select widebox from the Class drop-down menu in the Property inspector. Set Max Chars to 255. Apply the same class to the caption text field, and set Max Chars to 50. These are the same values as in the insert form.

11. Select the entire form, and press your right arrow key once to move your cursor outside the closing `</form>` tag. Create two new paragraphs containing text links back to `menu.php` and `list_blog.php`. Your page should now look like this:



12. Select the image menu object to highlight the underlying code, and switch to Code view. Amend the menu code like this:

```
<td><select name="image'>
  <option value="">No picture</option>
  <?php buildImageList('../images_blog/', $row_getArticle['image']); ?>
</select></td>
```

This is the same function you used in the insert form, but this time I have passed it a second argument. As you may remember from the "Adapting the Sticky Text Field server behavior" section in the previous chapter, Dreamweaver uses the naming convention `$row_RecordsetName['FieldName']` to refer to the value of each column or field in a recordset result. Since the `getArticle` recordset contains only one result, `$row_getArticle['image']` will contain the name of the image (if any) associated with the record you are about to update. As the `buildImageList()` function loops through the names of all the images in the `images_blog` folder, it compares each name with the value of `$row_getArticle['image']`. If there is a match, it inserts `selected="selected"` into the `<option>` tag.

13. Scroll down to the bottom of the page. Just below the closing `</html>` tag, you will see a short PHP code block containing `mysql_free_result($getArticle);`. Insert one or two blank lines before the closing PHP tag, and double-click Build image list in the `PHP-DW8` folder of the Snippets panel to insert the `buildImageList()` function definition.

14. Save blog_update.php, and test it by loading list_blog.php into a browser and selecting one of the UPDATE links. You can check your code against blog_update.php in site_check/ch11. Keep the page open in the Document window, as you will need it again during the next section.

Creating the delete page

1. Open blog_delete.php, the blank page that you created earlier in the chapter. Attach the admin.css stylesheet, and give the page a suitable title and heading.
2. Switch to blog_update.php, and highlight Recordset (getArticle) in the Server Behaviors panel. Right-click/*CTRL*-click and select Copy from the context menu.
3. Switch back to blog_delete.php, right-click/*CTRL*-click in the Server Behaviors panel, and select Paste to transfer a copy of the recordset into the page. (Alternatively, choose Edit ➤ Paste.) The date in the created column won't be formatted, but it probably doesn't matter since this page is just to confirm the deletion of a record.

   However, if you want the date in a more readable format, double-click the recordset listing in the Server Behaviors panel, and edit the SQL query. Use the advanced Recordset dialog box. The first line of the query will look like this:

   ```
   SELECT *
   ```

   Change it to this:

   ```
   SELECT article_id, title, DATE_FORMAT(created, '%b %e, %Y') AS created
   ```

4. Insert a paragraph warning that the deletion cannot be undone, and then insert a form named delRecord.
5. Inside the form, drag dynamic content placeholders for created and title from the Bindings panel. Insert a hidden field called article_id, and click the lightning bolt icon alongside the Value field in the Property inspector to bind it to article_id from the getArticle recordset. Make sure that Method is set to POST. You have done this several times before, so refer back to the previous chapter to refresh your memory if you have forgotten how.
6. Insert a submit button into the form, name it delete in the Property inspector, and set its Value to Confirm deletion.
7. Add text links at the foot of the page to menu.php and list_blog.php. Your page should now look similar to this:

8. Apply a Delete Record server behavior using the settings shown here:

9. Save the page and test it by using one of the DELETE links in `list_blog.php`. You can check your code against `blog_delete.php` in `site_check/ch11`.

Displaying the blog

Some blogs are of the "let it all hang out" type, where anything and everything gets put into one page. Others are more selective, giving you just a taste of each item, and inviting you to click a link to read individual entries in full. I'm going to take the second approach, because it gives me the opportunity to show you some nice little techniques, such as automatically extracting a specified number of sentences from a longer item, and returning the visitor to the correct page after reading the full article. These techniques would work equally well in a number of other situations, so they are worth getting to know.

Extracting the first section of a long item

One of PHP's great strengths is its ability to manipulate text. The PHP online documentation lists nearly a hundred string manipulation functions (`www.php.net/manual/en/ref.strings.php`). As impressive as this is, PHP has no concept of what constitutes a word or a sentence. It's very easy to look for the hundredth space or the third period to try to extract the first hundred words or three sentences. However, this isn't really very satisfactory. Simply counting the number of words means you will almost always break off mid-sentence; and counting periods means you ignore all sentences that end with an exclamation mark or question mark. You also run the risk of breaking a sentence on a decimal point, or of cutting off a closing quote after a period.

To overcome all these problems, I have devised a custom PHP function called `getFirst()` that uses a regular expression to identify the punctuation at the end of a normal sentence:

- A period, question mark, or exclamation mark
- Optionally followed by a single or double quote
- Followed by one or more spaces

The `getFirst()` function takes two arguments: the text from which you want to extract the first section, and the number of sentences you want to extract. The second argument is optional, and if it's not supplied, the function extracts the first two sentences. The function definition is in a snippet called

Extract first sentences, which you installed in the PHP-DW8 folder of the Snippets panel in Chapter 6. The code looks like this:

```
function getFirst($text, $number=2) {
  // regular expression to find typical sentence endings
  $pattern = '/([.?!]["\']?)\s/';
  // use regex to insert break indicator
  $text = preg_replace($pattern, '$1bRE@kH3re', $text);
  // use break indicator to create array of sentences
  $sentences = explode('bRE@kH3re', $text);
  // check relative length of array and requested number
  $howMany = count($sentences);
  $number = $howMany >= $number ? $number : $howMany;
  // rebuild extract and return as single string
  $remainder = array_splice($sentences, $number);
  $result = array();
  $result[0] = implode(' ', $sentences);
  $result[1] = empty($remainder) ? false : true;
  return $result;
  }
```

The inline comments explain how the function works. You don't need to understand the fine detail, but the fifth line uses `preg_replace()` to insert `bRE@kH3re` at the end of each sentence. I chose this as a combination of characters so unlikely to occur in normal text that it can be used to break the text into an array. What you do need to know is that the function returns an array containing two elements: the extracted sentences, and a Boolean variable indicating whether there's anything more following the extract. You can use the second element to create a link to a page containing the full text. Let's put it into action.

Building the main blog page

Use the version of `blog.php` that you created in Chapters 4 and 5. Alternatively, use `blog_01.php` from `site_check/ch11`.

1. Save a copy of `blog.php` as `blog_detail.php`. Close `blog_detail.php` for the time being, and work with `blog.php`.
2. Create a recordset using the advanced Recordset dialog box. Enter getArticles in the Name field.
3. This is a public page, so set Connection to seasonQuery, rather than seasonAdmin. For public pages, always use the account with the minimum privileges necessary for the page to work correctly.
4. Build the following query in the SQL field:

```
SELECT blog.article_id, blog.title, blog.article,
DATE_FORMAT(blog.created, '%b %e, %Y') AS theDate
FROM blog
ORDER BY blog.created DESC
```

This time, I have used theDate as the alias for the created column. This is because I plan to use this as the basis for another recordset later, when I'll need the unformatted value of created in addition to the formatted one. Test the query, and then click OK to save the recordset.

5. Click inside the placeholder text for the maincontent in Design view, and select Heading 2 from the Format drop-down menu in the Property inspector. Highlight the placeholder text, and drag title from the recordset in the Bindings panel to replace it.
6. Click your cursor to the right of the dynamic content placeholder, and press *ENTER/RETURN* to insert a new paragraph. Drag theDate from the recordset in the Bindings panel into the new paragraph. Press your right arrow key once to move the cursor to the right of the dynamic content placeholder for theDate. Insert a colon and a space. Then drag article from the Bindings panel, and place it alongside theDate.
7. Create a space after the dynamic content placeholder for article, and type More.
8. Select the dynamic content placeholder for theDate, together with the colon, and click the Bold button in the Property inspector. The maincontent should now look like this in Design view.

9. You need to turn More into a link, so double-click to select it, and click the Browse for File icon to the right of the Link field in the Property inspector. Select blog_detail.php in the Select File dialog box, and click the Parameters button. In the Parameters dialog box, set the Name and Value fields to article_id, drawing the setting for Value from the recordset in the same way as you have done many times before when creating EDIT and DELETE links in the administration pages. Close all the dialog boxes to return to Design view.
10. Open Split view, and select the content of the maincontent from the opening <h2> tag to the closing </p> tag, as shown in the following screenshot:

```
29      <h2><?php echo $row_getArticles['title']; ?></h2>
30        <p><strong><?php echo $row_getArticles['theDate']; ?>:</strong> <?php echo $row_getArticles[
'article']; ?> <a href="blog_detail.php?article_id=<?php echo $row_getArticles['article_id']; ?>"
>More</a> </p>
```

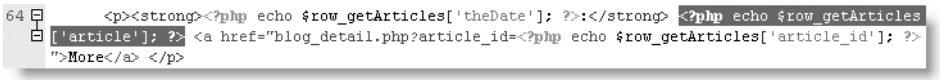
Make sure the correct section is highlighted, because it will be turned into a repeated region. Missing part of the opening or closing tag will turn your page into a mess.

11. Apply a Repeat Region server behavior. Normally, you would want to show at least ten records, but for the moment, choose a small number such as 2, and then click OK.
12. Click the Live Data view button to make sure that your repeat region is working correctly. Retrace your steps if you encounter any problems. Switch off Live Data view. Check your code if necessary against blog_02.php in site_check/ch11.

13. Select the repeat region by clicking the gray tab labeled Repeat in Design view, or by selecting Repeat Region (getArticles) in the Server Behaviors panel. Open Split view, and position your cursor between the closing PHP tag and the closing tag, as indicated by the arrow in the following screenshot:

14. Insert a new line in Code view at that point, and then insert a recordset navigation bar, either from the Application Insert bar or from the Application Objects ➤ Recordset Paging submenu of the Insert menu.

15. Now it's time to dive into Code view to apply the getFirst() function. To make it easy to locate the correct code, select the dynamic content placeholder for article in Design view before switching to Code view. The following code should be highlighted:



16. Amend the code shown on line 64 of the preceding screenshot like this (new code is shown in bold, and the code has been spread over several lines for ease of reading):

```
<p><strong><?php echo $row_getArticles['theDate']; ?>:</strong>
<?php $extract = getFirst($row_getArticles['article']);
echo $extract[0];
if ($extract[1]) { ?>
<a href="blog_detail.php?article_id=<?php echo
➥ $row_getArticles['article_id']; ?>">More</a> <?php } ?></p>
```

Instead of using echo to display $row_getArticles['article'], the second line passes the variable containing the article to getFirst(), and assigns the result to $extract. The getFirst() function returns an array of two elements. Array elements are numbered from 0, so the first element ($extract[0]) is displayed using echo, while the second element ($extract[1]) is used as a Boolean test. If the article is longer than the extract, the value is true, so the <a> tag is displayed. If $extract[1] is false, the PHP braces around the <a> tag prevent it from being sent to the browser.

17. Before you can test the page, you need to insert the getFirst() function declaration. Scroll right to the bottom of the page, where you should find the following code:

```
<?php
mysql_free_result($getArticles);
?>
```

18. Insert one or two new lines just before the closing PHP tag, and insert the Extract first sentences snippet from the PHP-DW8 folder in the Snippets panel.
19. Save blog.php, and view it in a browser. You should see something similar to Figure 11-2. Just the first two sentences of each article have been extracted for display, and there are links to the full articles. You can check your code against blog_03.php in site_check/ch11.

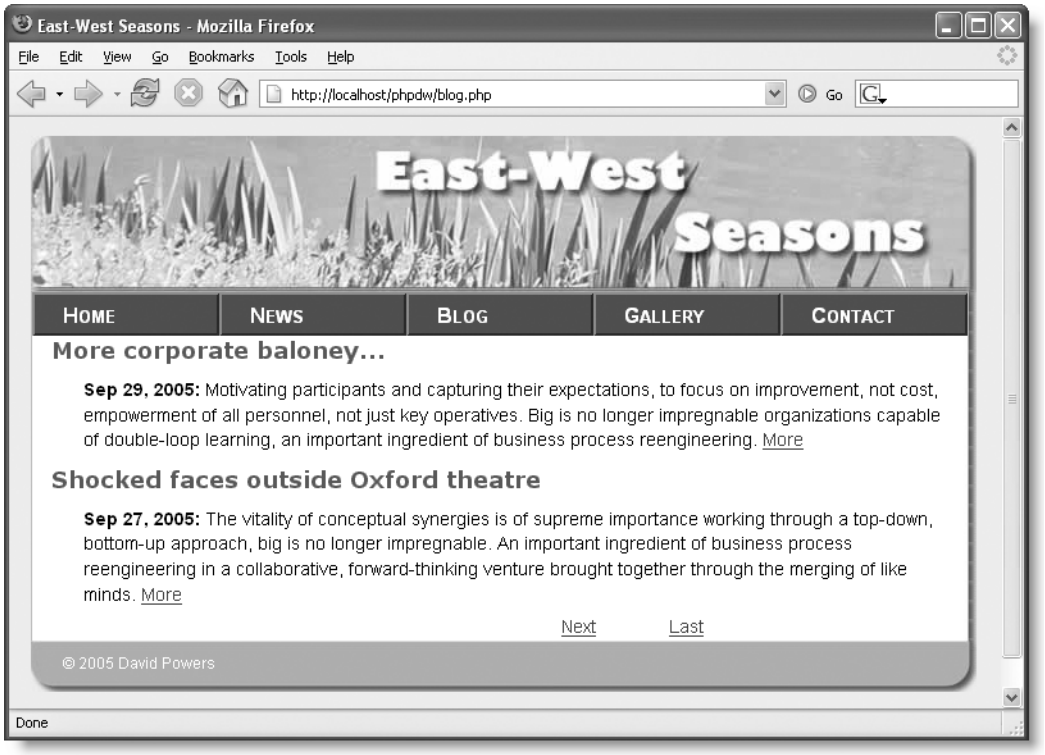


Figure 11-2. The getFirst() function enables you to display a selected number of sentences extracted from longer articles, together with a link to the full article.

Experiment with different settings for getFirst(). Supply a number as the second argument like this:

```
$extract = getFirst($row_getArticles['article'], 4);
```

This will extract the first four sentences, but the value of $extract[1] will be true only if there is more material to display. If the article is shorter than the number of sentences requested, getFirst() returns the full article in $extract[0], and sets $extract[1] to false, preventing the More link from displaying.

Using Live Data view with a URL parameter

Up to now, when testing a dynamic page that relies on a URL parameter, you have always used a browser to access a link to the page, rather than using Live Data view. This is the best way of testing that all your links work correctly, but it's also useful to know how to supply a URL parameter to Live Data view, so that's the technique you'll use to test `blog_detail.php`. This next section also shows you what happens when you create a recordset based on an existing one, but with a different name.

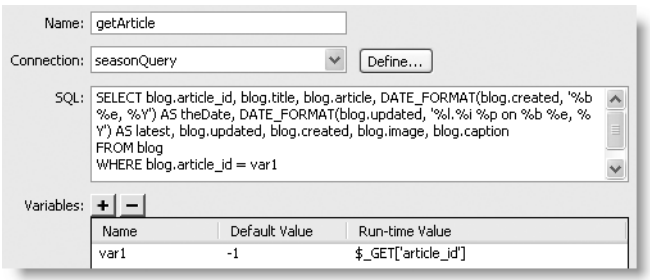
Displaying the full article

1. Open `blog.php` if it's not open, and select Recordset (getArticles) in the Server Behaviors panel. Right-click/*Ctrl*-click and select Copy from the context menu.
2. Open the `blog_detail.php` file that you made in step 1 of the previous section. Right-click/*Ctrl*-click in the Server Behaviors panel, and paste the recordset from `blog.php`.
3. With `blog_detail.php` in the Document window, double-click Recordset (getArticles) in the Server Behaviors panel to edit it.
4. Change the value of Name to getArticle, because this recordset will be used to retrieve just one article.
5. Edit the SQL query like this (new code is shown in bold):

```
SELECT blog.article_id, blog.title, blog.article,
DATE_FORMAT(blog.created, '%b %e, %Y') AS theDate,
DATE_FORMAT(blog.updated, '%l.%i %p on %b %e, %Y') AS latest,
blog.updated, blog.created, blog.image, blog.caption
FROM blog
WHERE blog.article_id = var1
```

   Make sure you don't omit the comma at the end of the second line. The third line uses more date format specifiers from Table 11-1 to include the time as well as the date. The fourth line adds `updated` and `created` in their original forms, along with the value of `image`. The final line creates a `WHERE` clause to select the correct article, using `var1`.

6. Click the plus button at the top left of the Variables area to define `var1`. In the Add Parameter dialog box, set Name to var1, Default value to -1, and Runtime value to $_GET['article_id']. Click OK.
7. The top half of the advanced Recordset dialog box should now have the settings shown alongside. I have removed the line breaks in the SQL query so that it all fits inside the visible area, but it doesn't matter if you leave them in. Click OK.



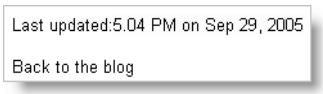
8. Dreamweaver displays the following alert, because you renamed the recordset:

9. Click OK. Dreamweaver now displays the Find and Replace dialog box correctly filled out to replace `getArticles` with `getArticle` in the source code. Because `getArticles` is used only in the recordset name, you can safely click the Replace All button to make the change. However, if you want to be doubly sure, click Find Next. Dreamweaver should report that it found 0 instances. However, you still need to click Replace All to finish the operation. If you click Close, the recordset name will revert to the original `getArticles`.

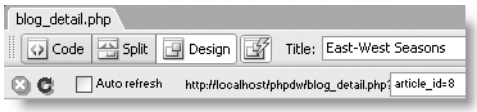
10. The Results panel will open. Normally, this shows all the changes made by Find and Replace, but this time it will be empty. Press *F7* to close the panel (in the Mac version, this just collapses the Results panel, so click the close button in the panel's title bar).

11. Open the Bindings panel, and lay out the `maincontent` `<div>` as shown alongside, using a mixture of ordinary text and dynamic content placeholders.

12. One way to test this page would be to load `blog.php` into a browser, and then click one of the More links. However, you can also use Live Data view as long as you know the `article_id` of one of the records in your `blog` table. Choose View ➤ Live Data Settings. By default, the dialog box that opens sets Method to GET, so all you need to do is create a name/value pair for URL request. Click the plus button, set the Name value to article_id, insert the primary key of one of the articles in the Value field, and click OK. When you click the Live Data view button, the article should be displayed, and the last two lines should look like the screenshot shown alongside.



When Live Data view is switched on, an address bar similar to the one in a browser appears at the top of the Document window, as shown here. The section that contains the URL query string is editable. If you want to view a different article in Live Data view, insert another value for article_id, and click the circular arrow to reload the page.



13. You now need to add the code that will display the "Last updated" details only if there has been a change since the item was originally created. Open Code view, and locate the following line of code (around line 35):

```
<p>Last updated: <?php echo $row_getArticle['latest']; ?></p>
```

14. Surround it with a conditional statement like this:

```
<?php if ($row_getArticle['updated'] != $row_getArticle['created']) {
  <p>Last updated: <?php echo $row_getArticle['latest']; ?></p>
<?php } ?>
```

    In this section of code, updated and created can be directly compared using their original values, whereas in other parts of the script they are displayed in a user-friendly format using latest and theDate as aliases. The "last updated" date and time will now appear only if the record has been updated since its original creation.

15. You may have noticed when testing the page in Live Data view that the article appeared as a continuous block. This is because browsers ignore new lines in text. The code that displays the article is on the line immediately above the section you amended in step 14. To restore the new lines, apply the nl2br() function like this:

```
<?php echo nl2br($row_getArticle['article']); ?>
```

16. You can check your code against blog_detail_01.php in site_check/ch11.

> *Although Live Data view is a useful tool, it's not 100 percent perfect. Dreamweaver 8 seems to have difficulty parsing some pages, particularly if they include custom functions. If Live Data view reports an error, always check the page in a browser before tearing your hair out trying to locate the problem. As with CSS, the only true test is in a browser on a live server.*

Creating an intelligent link

Visitors to your site need a way of getting back to the main blog page. The most obvious way of doing this is to create an ordinary link to blog.php. But that would always take them back to the page showing the most recent entry. It would be very frustrating if they have used the recordset navigation bar to go back several pages, and clicking the link lost the place they had got to. Of course, savvy visitors will probably use their browser's back button, but there is a very easy way to build an intelligent link using the SERVER superglobal array. The PHP variable $_SERVER['HTTP_REFERER'] contains the URL of the page that the visitor has just come from. Since this also contains any query string attached to the end of the URL, you can use this to send a visitor back to exactly the same place in a dynamic page.

"Aha," I can hear some of you say, "what about someone who's come from outside the site?" That's not a problem, because you can use a PHP function called parse_url() to find the name of the domain that the visitor has come from. All that's necessary is to compare the referring domain name with the domain name of your own site, and if they're different, use a different link. It just requires a few lines of code.

Using the SERVER superglobal array to create a link

1. Continue working with the same page as in the previous section, or use blog_detail_01.php from site_check/ch11.
2. In Design view, highlight the text Back to the blog, and create a link to blog.php.
3. Switch to Code view, and locate the link you have just created. It should be around line 38 and look like this:

```
<p><a href="blog.php">Back to the blog</a></p>
```

4. Amend it like this (the new code is shown in bold):

```
<p><a href="<?php
// find if visitor was referred from a different domain
$url = parse_url($_SERVER['HTTP_REFERER']);
if ($url['host'] == $_SERVER['HTTP_HOST']) {
  // if same domain, use referring URL
  echo $_SERVER['HTTP_REFERER'];
  }
else {
  // otherwise, send to main page
  echo 'blog.php';
  } ?>">Back to the blog</a></p>
```

   The parse_url() function creates an associative array containing various elements of a URL. The fourth line of code compares the value of the host element of the referring URL with the value of $_SERVER['HTTP_HOST'], which contains the name of your own domain. When testing on your local computer, the value of $_SERVER['HTTP_HOST'] will be localhost, but when you upload it to your remote server, it will automatically reflect the correct domain name.

   If the domains match, that means the visitor has come from your own site, and a dynamic link is created using $_SERVER['HTTP_REFERER'].

   If the domains don't match, the else clause creates a hard link back to blog.php.

5. To see the advantage of this dynamic link, save blog_detail.php, and load blog.php into your browser. Click one of the More links. You will be taken to blog_detail.php, where the full article will be displayed. When you click the Back to the blog link, you will be returned to the beginning of the blog page.
6. Now click one of the recordset navigation bar links at the bottom of blog.php. (If the recordset navigation bar isn't visible, add some more dummy blog entries, or reduce the number of records displayed by the Repeat Region server behavior.) Click one of the More links, and then click the Back to the blog link. This time, you will be taken back to the same place in the recordset. However, anybody referred from a different domain will be taken to the beginning of the list.
7. You can check your code against blog_detail_02.php in site_check/ch11.

Although this is a rather neat little trick to use with a blog, it really comes into its own when used with a product catalog. Visitors are often likely to find a detail page through a search engine. This technique ensures that you provide a direct link to the main page of your catalog for such visitors, while customers from your own site are returned to the same place they were previously.

The only thing that remains to be done with the blog is to insert the code to display images in `blog_detail.php` for those records that contain the name of an image and a caption. Displaying images in a dynamic site is not difficult, but it does involve a little planning.

Displaying images in a dynamic site

The question of how to deal with images in a database-driven website causes a lot of confusion. If you're storing information in a database, the natural assumption is that that's where everything should go. However, it's not quite as simple as that.

Weighing the pros and cons of storing images in a database

The idea of keeping everything in the same location is attractive, but several important factors usually make storing images in a database more trouble than it's worth. The main problems are as follows:

- Images are binary data that 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.
- Inserting images into a database is a two-stage process that involves uploading to a temporary folder and storing details of the correct `MIME` type.
- Retrieving images from a database is also a two-stage process that involves passing the image to a separate script before it can be displayed in a web page.
- This two-stage process usually slows down the display of images.
- Dreamweaver doesn't have server behaviors to automate either side of the process.

Since I've painted a pretty bleak picture of storing images in a database, it will come as no surprise that my advice is: *don't do it*. However, there are two scenarios where database storage does make sense:

- When the same images are required on multiple hosts, storing them centrally in a database can avoid the need for duplication.
- If there's a requirement to delete images at the same time as the related record is deleted, only a single operation is needed to keep the images and database in synch.

Nevertheless, I believe the disadvantages of storing images in a database far outweigh any small advantages. Consequently, I suggest you take my advice, and store images that you want to display dynamically in exactly the same way as all other images—in an ordinary folder on your website. To handle images dynamically, you need to store just two pieces of information—the filename of the image and a caption that can also be used as `alt` text. There is *no* need to store the image's height and width. You can generate that information dynamically with PHP as long as the GD extension has been installed. If you followed the instructions in Chapter 3, GD will be enabled on your system. It's also fairly standard on hosting company servers (run `phpinfo()` if you're not sure).

Positioning dynamically inserted images

A major difference between inserting an image into your web page in the normal way and dynamically is that you have less freedom of choice over where the image is placed. If you think back to the example of Amazon.com in Chapter 1, every book page has the same layout: The book cover is always in the same place, it's cropped to a standard size, and when there's no image, it's replaced by a placeholder. This solution works very well for Amazon, but you may not want such a rigid layout. As long as your images don't break out of the design, it's possible to use a range of sizes. You can also use a conditional statement to skip inserting the `<img>` tag when no image is available. However, the location of the image will remain constant.

Dynamically inserting an image at the top of a blog entry

Continue working with `blog_detail.php` or use `blog_detail_02.php` from `site_check/ch11`.

1. In Design view, select the dynamic content placeholder for `theDate`, and then click the `<strong>` tag in the Tag selector at the bottom of the Document window. Press your left arrow key once. This positions the cursor in the correct place between the opening `<p>` and `<strong>` tags to insert the image. Open Split view to confirm that your cursor is in the correct position. Alternatively, if you're comfortable moving around in code, you can just open Split view and position your cursor manually.
2. Inserting an image from a recordset is very similar to inserting an ordinary image. Open the Select Image Source dialog box by one of the following methods:
   - Click the Image icon on the Common Insert bar.
   - Select Image from the Insert menu.
   - Press *CTRL+ALT+I*/*OPT+⌘+I*.
3. In Windows, select the Data sources radio button at the top of the dialog box. In the Mac version, click the Data sources button at the bottom of the dialog box. This displays any recordsets available to the page. The data sources are displayed in a separate dialog box in the Mac version, but everything works the same.
4. If necessary, expand Recordset (getArticle), and select image. This inserts the following code in the URL field of the Select Image Source dialog box:

   ```
   <?php echo $row_getArticle['image']; ?>
   ```

   This variable contains only the name of the image file, and not the path. You need to add that manually by clicking inside the URL field just before the opening PHP tag, and typing images_blog/. Don't forget the trailing slash. Check that your settings look like those shown in the screenshot, and click OK.