CHAPTER 6

Adding Support for Multiple Pages

So far you've created an extremely basic blog. But what good is a blog if a user can't find out more about its author?

In this chapter, you'll learn how to modify your application to support multiple pages, so you can add an "About the Author" page. To do this requires that you learn how to accomplish each of the following steps:

- Add a `page` column to the `entries` table
- Modify functions to use a page as part of the `WHERE` clause in your MySQL query
- Add a hidden input to the form on `admin.php` to store the page
- Modify `update.inc.php` to save page associations in the database
- Use an `.htaccess` file to create friendly URLs
- Add a menu
- Modify display options for the "About the Author" and "Blog" pages

By the end of this chapter, your blog will have two pages: one will support multiple entries, while the other will support only a single entry.

Add a page Column to the entries Table

Your first task is learning to identify what entries belong on what page. Essentially, you need to add a page identifier. This could be a number or a string. Your application is pretty simple, so you can just use the name of the page as your identifier.

To add this to your entries, you need to get back into your database controls, located at `http://localhost/phpmyadmin`. Open the `simple_blog` database, then the `entries` table. You need to add a column called `page` to the `entries` table, which will hold the name of the page to which each entry belongs.

This column cannot be blank, or the entries will get lost. To avoid this, you can set the column to `NOT NULL` and provide a default value. Most entries will end up on the blog page, so set the default to "blog." Finally, for organizational purposes, you want to put the column right after the `id` column; you can accomplish this in your query by using `AFTER id`.

Additionally, you can speed up your queries by adding an index to the `page` column. This is as simple as appending `ADD INDEX (page)` to the end of the query, separated by a comma. The full query looks like this:

```
ALTER TABLE entries
ADD page VARCHAR(75) NOT NULL DEFAULT 'blog'
AFTER id,
ADD INDEX (page)
```

Now execute the preceding query in the SQL tab of `http://localhost/phpmyadmin`. When the query finishes, click the Browse tab to verify that the page column has been created and that all the pages have been identified as blogs.

Modify Your Functions to Accept Page Parameters

Now that your entries have a page associated with them, you can start using the page as a *filter* to retrieve only the data that matches your current page. This is really similar to the way you used the id column to filter your query to only return one entry. By using the page, you filter the query to only return entries for one page.

Accepting Page Information in the URL

First—and this is very important—you need to somehow pass a page variable to your script. You do this in the same way that you previously passed an entry ID to the script, using the URL and the `$_GET` superglobal.

For example, you navigate to the following address to look at the blog page:

```
http://localhost/simple_blog/?page=blog
```

Navigating to an entry within the blog requires that you use a URL similar to the following:

```
http://localhost/simple_blog/?page=blog&id=2
```

To use the preceding URL format, you need to modify `index.php` to use the page variable passed in the URL, then modify `functions.inc.php` to accept the page variable and use it in your database query.

Begin by opening `index.php` (full path: `/xampp/htdocs/simple_blog/index.php`) and adding the code in bold to the top of the script:

```
<?php
    /*
     * Include the necessary files
     */
    include_once 'inc/functions.inc.php';
    include_once 'inc/db.inc.php';

    // Open a database connection
    $db = new PDO(DB_INFO, DB_USER, DB_PASS);
    /*
     * Figure out what page is being requested (default is blog)
     * Perform basic sanitization on the variable as well
     */
    if(isset($_GET['page']))
    {
        $page = htmlentities(strip_tags($_GET['page']));
    }
    else
    {
        $page = 'blog';
    }

    // Determine if an entry ID was passed in the URL
    $id = (isset($_GET['id'])) ? (int) $_GET['id'] : NULL;

    // Load the entries
    $e = retrieveEntries($db, $page, $id);

    // Get the fulldisp flag and remove it from the array
    $fulldisp = array_pop($e);

    // Sanitize the entry data
    $e = sanitizeData($e);
?>
```

Here you add a line that collects the page variable from the `$_GET` superglobal array, then assigns its value (or a default value, which you've set to "blog") to a variable called `$page`.

Next, you add the `$page` variable as an argument in your call to `retrieveEntries($db, **$page**, $id);` so that you can use the information in retrieving entry data.

For now, you're finished in `index.php`. Next, you need to modify your `retrieveEntries()` function.

Using the Page Information to Filter Entries

The first thing you need to do is to alter `retrieveEntries()` to accept the `$page` parameter you've just added. Open `functions.inc.php` and alter the function definition to read as follows:

```
function retrieveEntries($db, $page, $url=NULL)
{
```

The page is being sent to your entry retrieval function, so you can use the information to filter your query and return only results relevant to the page being viewed. You accomplish this using a `WHERE` clause.

Originally, your query for retrieving entries when no entry ID was supplied looked like this:

```
SELECT id, title, entry
FROM entries
ORDER BY created DESC
```

Adding the `WHERE` clause means you can no longer simply execute the query because you're now relying on user-supplied data, which is potentially dangerous. To keep your script secure, you need to use a prepared statement. Your query uses a placeholder for the page variable and looks something like this:

```
SELECT id, page, title, entry
FROM entries
WHERE page=?
ORDER BY created DESC
```

Now you can retrieve only the entries that correspond to the page being viewed. The next step is to update your query in `functions.inc.php` (full path: `/xampp/htdocs/simple_blog/inc/functions.inc.php`). This snippet starts at `line 25` in the file; add the changes highlighted in bold:

```
/*
 * If no entry ID was supplied, load all entry titles for the page
 */
else
{
    $sql = "SELECT id, page, title, entry
            FROM entries
            WHERE page=?
            ORDER BY created DESC";
    $stmt = $db->prepare($sql);
    $stmt->execute(array($page));

    $e = NULL; // Declare the variable to avoid errors
```

In this snippet, you create a prepared statement out of the query you wrote previously, then execute the statement using the `$page` variable you passed to `retrieveEntries()` from `index.php`.

This code also adds a line declaring the `$e` variable as `NULL`. This part serves as a precautionary measure against empty result sets, which would otherwise result in an error notice if no entries exist for the specified page.

■ **Tip** It's a good habit to get into to always declare a variable as `NULL` if there's the potential for a query or loop to come back empty. This means any variable defined in a conditional statement or used to store the result of a database query should contain a `NULL` value before the query or loop is executed.

You changed the method you use to execute the query, so now you need to modify the way you store the result set. Add the following code in bold where indicated in `functions.inc.php`, immediately beneath the script you just altered, starting at `line 39`:

```
// Loop through returned results and store as an array
while($row = $stmt->fetch()) {
    $e[] = $row;
}
```

Once this code is in place, each result array is stored as an array element in `$e`; this means that your script will now work. Save `functions.inc.php` and navigate to `http://localhost/simple_blog/?page=blog` in a browser. At this point, you should see the previews of the blog entry (see Figure 6-1).

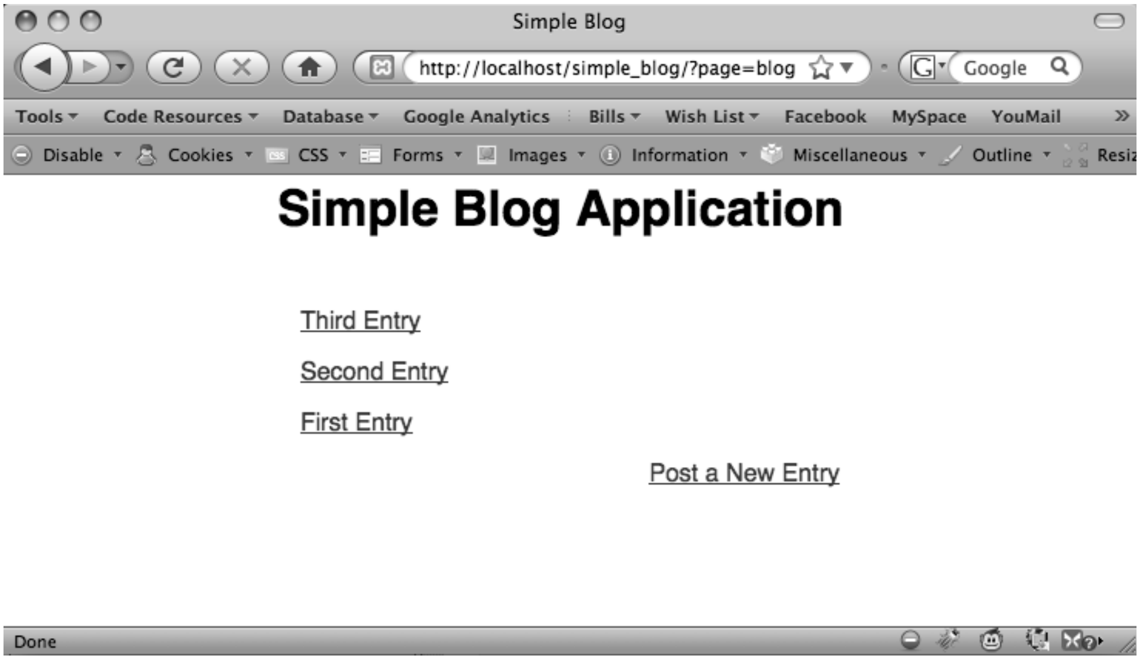
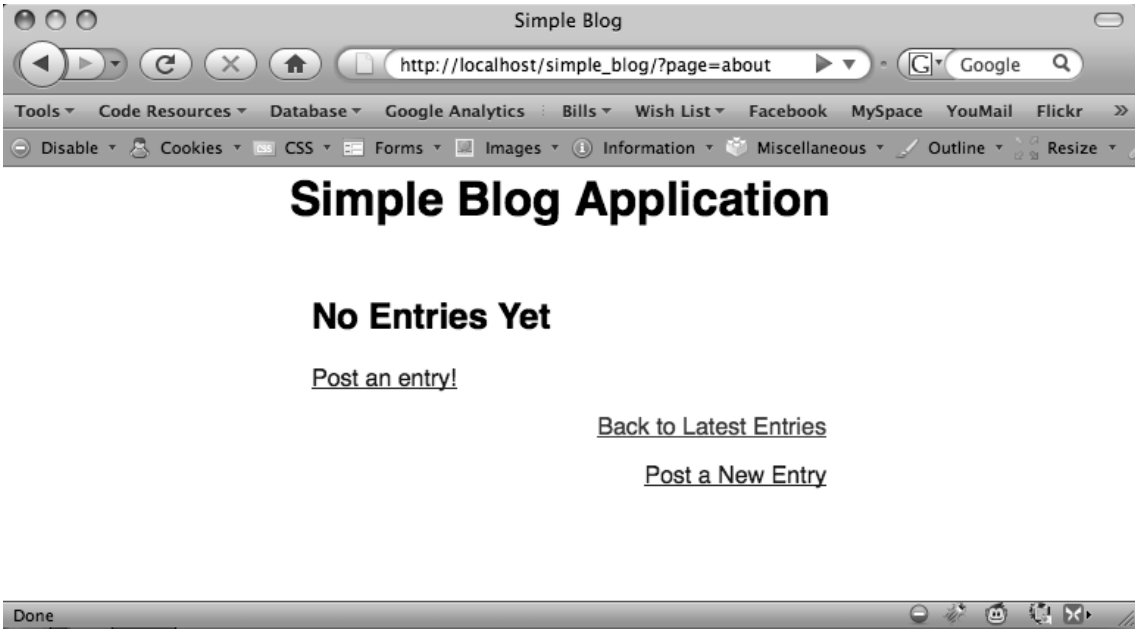


Figure 6-1. *The blog previews page loaded with URL variables*

The blog is the default page, so previews will also load without the page variable. To see the power of what you've just built, navigate to a page that doesn't exist yet: your "About the Author" page. Navigate to `http://localhost/simple_blog/?page=about` in a browser, and you should see your default "No Entries" message (see Figure 6-2).


Simple Blog
http://localhost/simple_blog/?page=about
Tools Code Resources Database Google Analytics Bills Wish List Facebook MySpace YouMail Flickr
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize
Done

Simple Blog Application

No Entries Yet

Post an entry!

Back to Latest Entries

Post a New Entry

Figure 6-2. *The "About the Author" page with no entries supplied*

Here you face with a slight problem: you have a "Back to Latest Entries" link on your "About the Author" page. This could prove misleading because it might give your users the impression that there are more entries about the author.

Additionally, the "Post a New Entry" link appears on this page. You want only one entry to appear on the "About the Author" page, so you don't want this link to appear here.

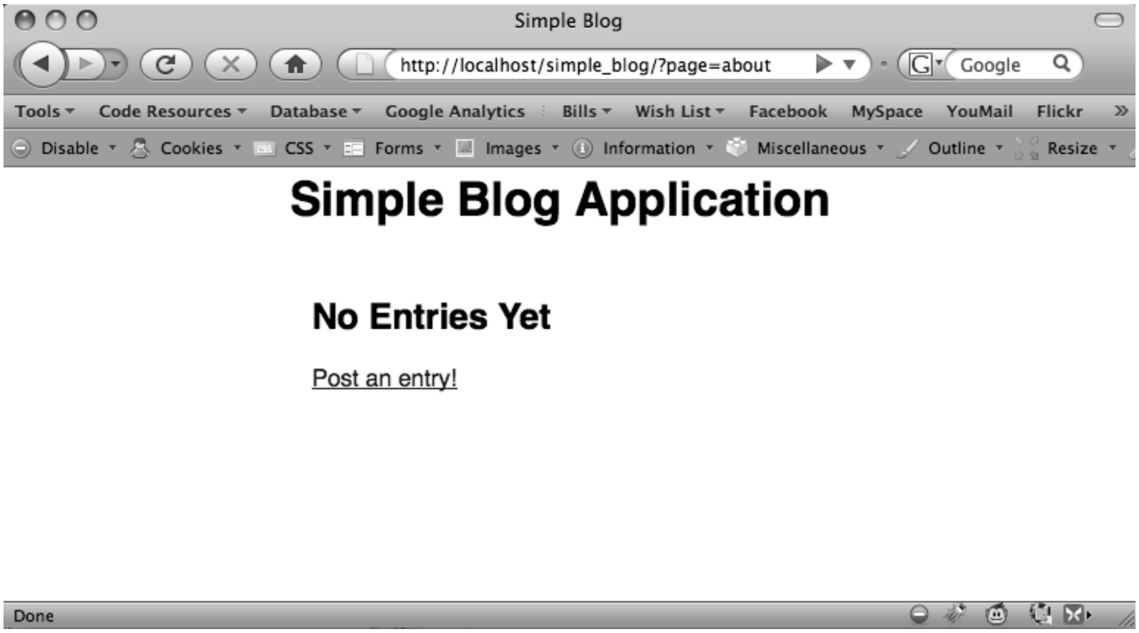
To correct this, you must modify `index.php` with a conditional statement that displays the "Back to Latest Entries" and "Post a New Entry" links only on the "Blog" page. Accomplish this by opening index.php and adding the code in bold to the body of the document:

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <link rel="stylesheet" href="css/default.css" type="text/css" />
    <title> Simple Blog </title>
</head>
```

```
<body>

    <h1> Simple Blog Application </h1>

    <div id="entries">

<?php

// If the full display flag is set, show the entry
if($fulldisp==1)
{

?>

        <h2> <?php echo $e['title'] ?> </h2>
        <p> <?php echo $e['entry'] ?> </p>
        <?php if($page=='blog'): ?>
        <p class="backlink">
            <a href="./">Back to Latest Entries</a>
        </p>
        <?php endif; ?>

<?php

} // End the if statement

// If the full display flag is 0, format linked entry titles
else
{
    // Loop through each entry
    foreach($e as $entry) {

?>

        <p>
            <a href="?id=<?php echo $entry['id'] ?>">
                <?php echo $entry['title'] ?>

            </a>
        </p>

<?php

    } // End the foreach loop
} // End the else

?>
```

```
        <p class="backlink">
        <?php if($page=='blog'): ?>
            <a href="/simple_blog/admin/<?php echo $page ?>">
                Post a New Entry
            </a>
        <?php endif; ?>
        </p>

    </div>

</body>

</html>
```

Now you don't see the potentially misleading links when you load `http://localhost/simple_blog/?page=about` (see Figure 6-3).



Simple Blog Application

No Entries Yet

Post an entry!

Figure 6-3. *The "About the Author" page without potentially misleading links*

The next step is to create an entry for the "About the Author" page. However, you need to update your `admin.php` script before you can create this entry.

Modifying admin.php to Save Page Associations

Saving the page an entry is associated with is as easy as adding another input to your form. However, there are a couple reasons you don't want to require the user to fill out the page an entry belongs on. First, it's inconvenient for the user; second, it increases the risk of typos or confusion.

Fortunately, HTML forms allow you to insert *hidden inputs*, which contain a value that is passed in the `$_POST` superglobal, but isn't displayed to the user. In your `admin.php` script (full path: `/xampp/htdocs/simple_blog/admin.php`), add a hidden input to your form by inserting the lines in bold:

```
<?php
if(isset($_GET['page']))
{
    $page = htmlentities(strip_tags($_GET['page']));
}
else
{
    $page = 'blog';
}
?>
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
    <meta http-equiv="Content-Type"
        content="text/html;charset=utf-8" />
    <link rel="stylesheet"
        href="/simple_blog/css/default.css" type="text/css" />
    <title> Simple Blog </title>
</head>

<body>
    <h1> Simple Blog Application </h1>

    <form method="post" action="/simple_blog/inc/update.inc.php">
        <fieldset>
            <legend>New Entry Submission</legend>
            <label>Title
                <input type="text" name="title" maxlength="150" />
            </label>
            <label>Entry
                <textarea name="entry" cols="45" rows="10"></textarea>
            </label>
            <input type="hidden" name="page"
                value="<?php echo $page ?>" />
            <input type="submit" name="submit" value="Save Entry" />
            <input type="submit" name="submit" value="Cancel" />
        </fieldset>
    </form>
</body>

</html>
```

In the first line of this script, you retrieve the page variable, which will be passed in the URL. To make sure a variable was passed, you use the *ternary operator* (a shortcut syntax for the `if else` statement) to check whether `$_GET['page']` is set. If so, you perform basic sanitization by removing any HTML tags from the string, then encoding any special characters that could cause problems in your script. If not, you provide a default page, `blog`, to avoid any unexpected behavior.

Then, in the form itself, you insert a hidden input with the name of "page" and a value that contains the sanitized value from the URL.

This means that creating an entry with an associated page requires that you access `admin.php` using a path that includes a page variable:

```
http://localhost/simple_blog/admin.php?page=about
```

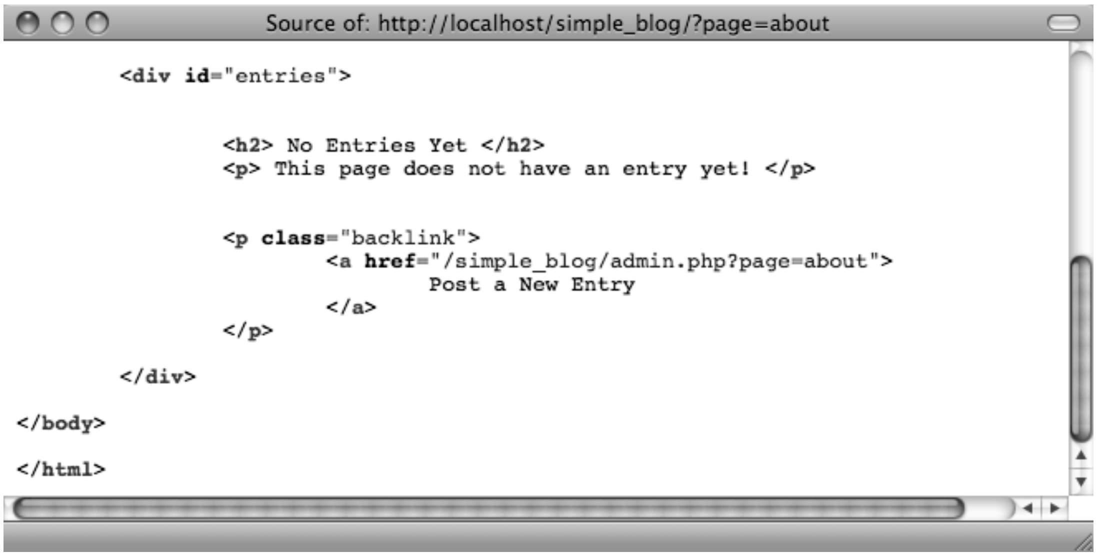
This means that you need to make some adjustments to `index.php` to ensure that a page variable is passed when a user clicks the link to create a new entry.

In `index.php`, starting at `line 100`, modify the link to create a new entry as follows:

```
        <p class="backlink">
            <a href="/simple_blog/admin.php?page=<?php echo $page ?>">
                Post a New Entry
            </a>
        </p>
```

This entry takes the `$page` variable you stored at the beginning of the script and uses it to make a link for posting a new entry pass to the page. You can test this by navigating to `http://localhost/simple_blog/?page=about`; this URL lets you use your browser to look at the page value stored in the "Post a New Entry" link (see Figure 6-4).

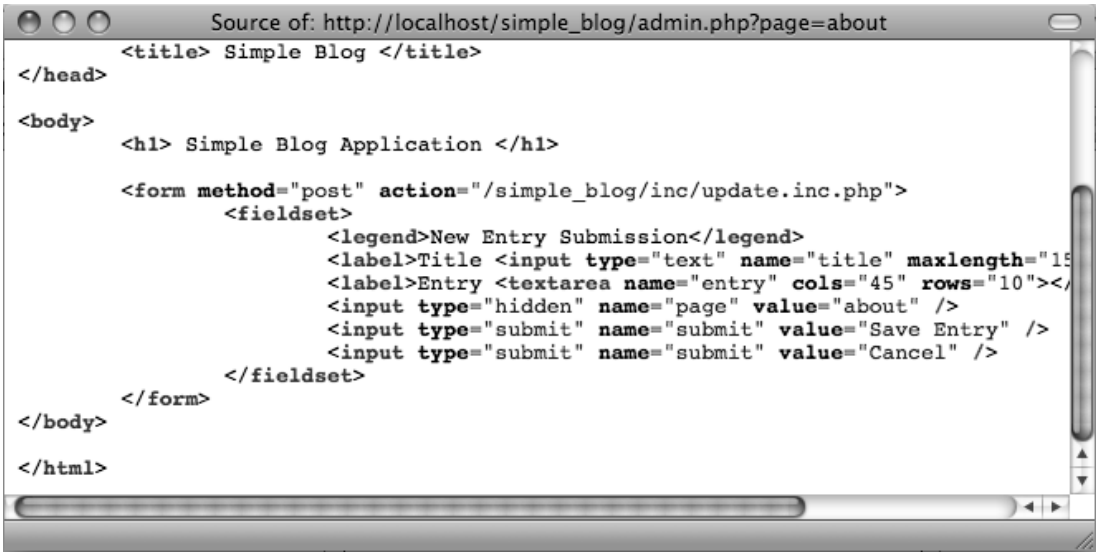
■ **Tip** You can view the source code in a PHP project by select View from the browser menu, then (depending on the browser being used) Source, Page Source, or View Source.



```
        <div id="entries">


                <h2> No Entries Yet </h2>
                <p> This page does not have an entry yet! </p>


                <p class="backlink">
                        <a href="/simple_blog/admin.php?page=about">
                                Post a New Entry
                        </a>
                </p>

        </div>

</body>

</html>
```

Figure 6-4. *The source code of* *`http://localhost/simple_blog/?page=about`*

Next, you need to make sure that you're storing the page in the hidden input properly. Click the "Post a New Entry" link on `http://localhost/simple_blog/?page=about`, which should direct you to `http://localhost/simple_blog/admin.php?page=about`. There, you can see your form as usual, but looking at the source code should reveal that the hidden input now contains the "about" value that was passed in the URL (see Figure 6-5).


Source of: http://localhost/simple_blog/admin.php?page=about

```
        <title> Simple Blog </title>
</head>

<body>
        <h1> Simple Blog Application </h1>

        <form method="post" action="/simple_blog/inc/update.inc.php">
                <fieldset>
                        <legend>New Entry Submission</legend>
                        <label>Title <input type="text" name="title" maxlength="15
                        <label>Entry <textarea name="entry" cols="45" rows="10"></
                        <input type="hidden" name="page" value="about" />
                        <input type="submit" name="submit" value="Save Entry" />
                        <input type="submit" name="submit" value="Cancel" />
                </fieldset>
        </form>
</body>

</html>
```

Figure 6-5. *The source of* *`http://localhost/simple_blog/admin.php?page=about`*

Now you know that the page will be passed to the form. This means that you have access, via the `$_POST` superglobal, to whatever page the entry is associated with after the new entry is submitted.

However, bear in mind that the page association won't be saved until you make some adjustments to `update.inc.php` to handle this new information.

Saving Page Associations

Saving the page association in your database when new entries are created requires that you modify your query in `update.inc.php`, as well as a couple more checks to ensure that errors don't occur.

To save the entry information, you need to:

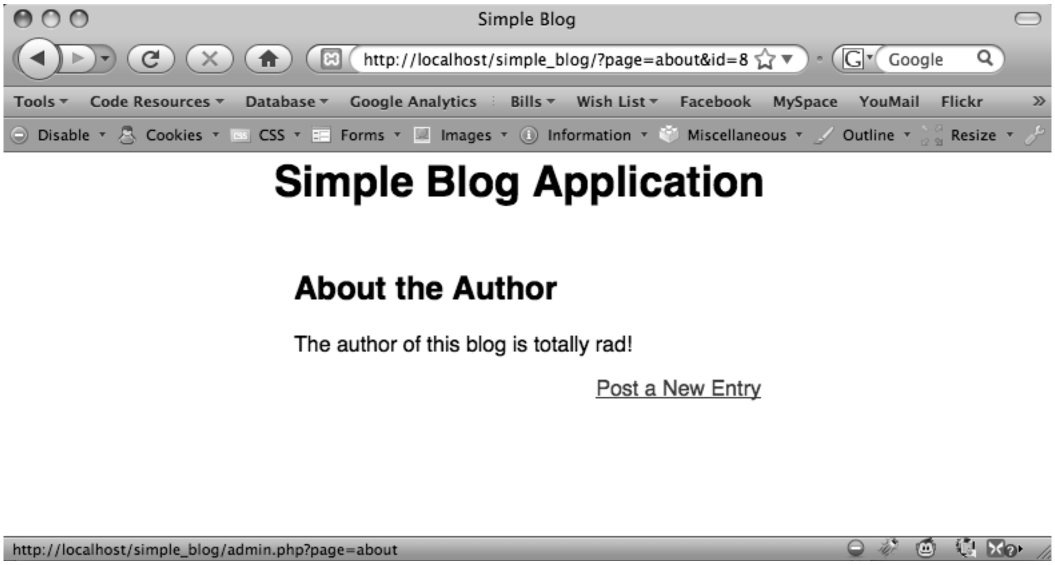
1. Make sure the page was specified before processing
2. Add the page to the query to be saved
3. Sanitize the data
4. Use the sanitized page information to send the user back to the created entry

In `update.inc.php`, modify the script to include the lines highlighted in bold:

```
<?php

if($_SERVER['REQUEST_METHOD']=='POST'
    && $_POST['submit']=='Save Entry'
    && !empty($_POST['page'])
    && !empty($_POST['title'])
    && !empty($_POST['entry']))
{

    // Include database credentials and connect to the database
    include_once 'db.inc.php';
    $db = new PDO(DB_INFO, DB_USER, DB_PASS);

    // Save the entry into the database
    $sql = "INSERT INTO entries (page, title, entry)
            VALUES (?, ?, ?)";
    $stmt = $db->prepare($sql);
    $stmt->execute(
        array($_POST['page'],$_POST['title'],$_POST['entry'])
    );
    $stmt->closeCursor();

    // Sanitize the page information for use in the success URL
    $page = htmlentities(strip_tags($_POST['page']));
    // Get the ID of the entry you just saved
    $id_obj = $db->query("SELECT LAST_INSERT_ID()");
    $id = $id_obj->fetch();
    $id_obj->closeCursor();

    // Send the user to the new entry
    header('Location: /simple_blog/?page='.$page.'&id='.$id[0]);
    exit;
}

else
{
    header('Location: ../');
    exit;
}

?>
```

Making these changes, effectively ensures that a page association is passed to the update script; you can then insert the association using your prepared statement. Afterward, you sanitize the page information and store it in the `$page` variable. Finally, you send the user to the new entry by passing the page in the URL, along with the ID of the new entry.

Save `update.inc.php` and navigate to `http://localhost/simple_blog/?page=about`, then click the "Post a New Entry" link. Now create an "About the Author" entry and click "Save Entry"; this should take you to the entry saved with the "about" page association (see Figure 6-6).



Simple Blog Application

About the Author

The author of this blog is totally rad!

Post a New Entry

http://localhost/simple_blog/admin.php?page=about

Figure 6-6. *The "About the Author" page with an entry created*

Using .htaccess to Create Friendly URLs

When building applications for the web, it's important to look at some of the marketing tricks involved in getting a site noticed. There are full courses dedicated to web site marketing, but every developer should know a little bit about how sites get noticed on the web.

One of the most widely discussed areas of web site marketing is *search engine optimization*, or SEO. This is the practice of maximizing the value of a web site in the eyes of search engines like Google and Yahoo! by placing key words in important areas of the site.

One of the most important areas of any site is the URL itself. For instance, a web site selling t-shirts with rubber ducks on them would want locate its products page at an URL something like this:

```
http://rubberducktees.com/products/t-shirts
```

This is far more desirable than a URL with a bunch of confusing IDs, such as:

```
http://rubberducktees.com?page=4&category=5&product=67
```

Neither approach is wrong, but the former is far easier to read and to remember. It is also far more likely to make sense to the average user. Also, you should be aware that search engines are much more likely to *index*, or store as a search result, sites with key words in the URL.

■ **Note** Some search engines won't index URLs beyond the first question mark they encounter, which means non-optimized URLs won't make it into search results at all.

Your URLs right now aren't exactly optimal because you're using variables such as `?page=blog&entry=2` to identify entries to the script. So a good next step is to figure out a way to allow your users to get to a blog using a much easier URL, such as:

```
http://localhost/simple_blog/blog/first-entry
```

Doing this requires that you use a pair of advanced coding techniques: *.htaccess* and *regular expressions.*

What .htaccess Does

One of the best parts about using Apache servers is the ability to use `.htaccess` files. These allow developers to control a number of things, including file-access permissions, how certain file types are handled. The files also serve as an especially useful tool for rewriting URLs.

You won't need to know too much about how `.htaccess` works for the examples described in this book. However, I teach you everything you need to know about rewriting a URL so it is much more user-friendly.

Using Regular Expressions

Regular expressions are, in essence, patterns that enable complex matching in strings. They are tricky, and they employ a syntax that can be very hard to understand; however, they are also one of the most powerful tools available to developers, so it behooves you to learn about them.

Once you get the basics of regular expressions (or *regex*), you can use them in your `.htaccess` files to match URL patterns and rewrite them so they are compatible with your scripts.

Creating Your .htaccess File

Begin by firing up Eclipse and creating a new file called `.htaccess` in the `simple_blog` project. You should place this file in the top level of the project (full path: `/xampp/htdocs/simple_blog/.htaccess`).

■**Note** `.htaccess` files start with a period, so your new file might not show up in your file list under the project in Eclipse. If you close `.htaccess` and need to open it, use `File > Open...` to access it again. Alternatively, you can set Eclipse to show resources starting with a period by selecting the upside-down triangle (third button from the right) in the "Project Explorer" panel and clicking "Customize View..." In the Filters tab, uncheck `.* resources`, and the `.htaccess` file should appear in your file list.

■**Caution** If you choose to view `.* resources` in project folders, be sure that you do *not* edit or delete the `.project` or `.buildpath` files. Doing so can cause Eclipse to lose its association with your projects, which means you'll have to recreate them before you can access any of your files for editing.

The file should bring up a blank file in the editor. In this file, you need to accomplish the following:

- Turn on URL rewriting
- Declare the base-level folder for rewriting
- Set up a rule to stop the rewrite if certain file types are accessed directly
- Set up a special rule for admin.php
- Set up a rule for page-only URLs
- Set up a rule for page and entry URLs

Step 1: Turn on URL Rewriting

The first line of your `.htaccess` file lets the server know that URL rewriting is allowed in this directory. To do this, add the following line to your currently blank `.htaccess` file:

```
RewriteEngine on
```

Step 2: Declare the Base-Level Folder for Rewriting

Next, you need to let the server know which folder to use as a base when rewriting. If you want the root folder to be the base folder, you could use a single forward slash (`/`) to set the root folder (the `htdocs` folder when using XAMPP) as the base folder. Your project is in the `simple_blog` folder in root, so you need to specify it as the base. To do so, add the following to `line 2` in `.htaccess`:

```
RewriteBase /simple_blog/
```

Step 3: Set Up a Rule to Stop Rewriting for Certain File Types

You must be able to access some files by their real path, not a rewritten one, so you need to set up a rewrite rule that says certain file types, such as images and included scripts, will stop the rewrite engine from doing anything and exit the `.htaccess` file.

■ **Tip** For a complete breakdown of URL rewriting, visit the `mod_rewrite` documentation on the Apache website at `http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html`.

This is a slightly more complicated area of the file, and it's here you start using regex. Begin by creating the rule; I'll break down what it does and how it works afterward:

```
RewriteRule \.(gif|jpg|png|css|js|inc\.php)$ - [L]
```

In `.htaccess` files, rewrite rules follow this pattern:

```
RewriteRule pattern replacement [flags]
```

Adding a `RewriteRule` lets you signify to the server that the rest of the line contains commands for URL rewriting.

Patterns

The *pattern*, which is what you want to match, goes next. In your rule just described, the pattern section of the rule is `\.(gif|jpg|png|css|js|inc\.php)$`. In plain English, this means that any URL that calls a file ending with the extension `.gif`, `.jpg`, `.png`, `.css`, `.js`, or `.inc.php` will match your pattern.

The first part, a backslash followed by a period, signifies the "dot" that precedes the file extension (as in, "dot jpg"). Next, you have your file extensions wrapped in parentheses and separated with a vertical bar (`|`). The parentheses enclose a *group* of characters to match, and the vertical bar acts as an "or" command. Finally, the dollar sign means that the match must fall at the end of the URL, so a URL ending in `image.jpg` would trigger the command, but a URL ending in `image.jpg?mischief` would not.

■ **Note** To learn more about regular expressions, check out `http://regular-expressions.info`.

Replacements

The *replacement* is a new format for the data matched in the rule's pattern. Each group can be accessed to create a different URL structure that scripts can use instead of the one navigated to by the user.

In the case of your rule, the replacement is a hyphen (`-`). This signifies that nothing is to be done. You'll near more about replacements momentarily, when I cover the next rule.

Flags

The *flags* for rewrite rules are a set of controls that allow users to determine what to do *after* a rule matches. In the preceding example, the flag for your rule is `[L]`, which means that this is the last rule checked if this rule is matched. This means that no more rewriting will occur. Some of the other available rules include:

- `nocase/NC` *(no case)*: Makes the rule case-insensitive
- `forbidden/F` *(forbidden)*; Force a resource to return a 403 (FORBIDDEN) response
- `skip/S=num` *(skip the next rule)*: If current rule is matched, the next *num* rules are skipped

---

■ **Note** There are many other flags that you can in `.htaccess` rewrite rules. For a full list, visit the RewriteRule Directive section of the `mod_rewrite` documentation on the Apache web site at `http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html#rewriterule`.

Step 4: Set Up a Rule for Admin Page Access

Your admin page is a little different from the rest of your site because it relies on a different file (`admin.php`). This means your URL rewrites need to take this into account and behave accordingly.

You want your administrative page to be accessed with the following URL structure:

```
http://localhost/simple_blog/admin/blog
```

However, your scripts need the URL to be structured like this:

```
http://localhost/simple_blog/admin.php?page=blog
```

To do this, you need to catch any URL that begins with "admin/" and take the rest of the string and pass it as the page variable. In your `.htaccess` file, add the following rule:

```
RewriteRule ^admin/(\w+) admin.php?page=$1 [NC,L]
```

In this rewrite rule, you require that the URL path start with `admin/` using the carat (`^`), then use the shorthand `\w+` to store one or more word characters (`a-z`, `0-9`, and `_` [underscore]) as a *backreference*, which is functionally equivalent to a variable in PHP. When working with rewrite rules, you access backreferences using a dollar sign and the number corresponding to the group (*i.e.*, the first backreference is `$1`, the second is `$2`, and so on). You can create only nine backreferences for each rule.

The replacement URL is a link to `admin.php`, with the page variable set to the value of your backreference. So, if the URL path supplied to the rule is `admin/blog`, it is rewritten to `admin.php?page=blog`.

Finally, you set the flags to `NC`, which means the rule isn't case-sensitive (`ADMIN/` is equivalent to `admin/` for purposes of matching), and `L`, which means more rewriting will occur if this rule is matched.

You don't want the site to throw an error if the user accesses `http://localhost/simple_blog/admin`, so you need to add an additional rule. This rule will go immediately above the previous rewrite rule, and handle administrative URLs that don't specify the page.

Add the following rule to `.htaccess`:

```
RewriteRule ^admin/?$ admin.php [NC,L]
```

This rule makes sure that the user has used either `http://localhost/simple_blog/admin` or `http://localhost/simple_blog/admin/` to access the site. If this rule is matched, the user is directed to `admin.php`.

Step 5: Set Up a Rule for Page-Only URLs

You can access your publicly displayed pages in one of two ways: either page-only (*i.e.*, `http://localhost/simple_blog/blog/`) or page-and-entry (*i.e.*, `http://localhost/simple_blog/blog/first-entry`). Your next rewrite rule catches the page-only URLs and directs the user to the proper place.

In `.htaccess`, add the following rule:

```
RewriteRule ^(\w+)/?$ index.php?page=$1
```

This rule captures the beginning of the URL path, stopping at the first forward slash. The dollar sign after the forward slash means that you can't use any additional characters after the first forward slash, or the rule won't match. You can then use the captured characters as the page variable in the rewritten URL.

Also, note the use of a question mark following the slash. This makes the expression lazy, which means it doesn't need to match the last slash. This covers you if the user enters a URL without a trailing slash, like this one: `http://localhost/simple_blog/blog`. You don't need any flags for this rule, so you simply leave them out altogether.

Step 6: Set Up a Rule for Page-and-Entry URLs

Finally, you need a rule that passes the page and entry information to your script in a format it can understand. This is similar to the previous rule, except you're going to be using two backreferences, as well as a new concept known as *character classes*.

In `.htaccess`, add the following rule:

```
RewriteRule ^(\w+)/([\w-]+) index.php?page=$1&url=$2
```

The first part of the rule, `^(\w+)/`, is the same as the last rule. It matches any word character that starts the URL path until the first forward slash.

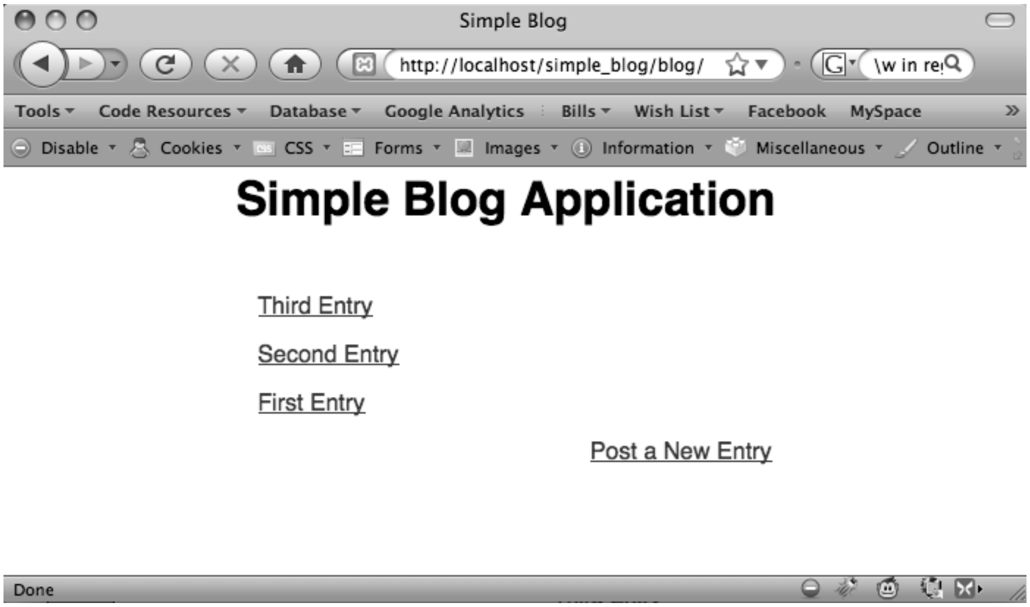
The second part of the URL, `([\w-]+)`, creates a second backreference to a character class, which is a group of characters enclosed in square brackets that you can use for a valid match. You can match any word character in this character class, as well as the hyphen (`-`). The plus sign (`+`) means that one or more characters will be matched.

You add a hyphen in this case because you're going to use hyphens to replace spaces in your URLs (you can learn more about this in the next section, "Creating Friendly URLs Automatically").

The replacement URL generated passes the first backreference as the page variable, and it passes the second backreference as a variable called `url`, which you use in place of an entry ID from here on out.

Trying It Out

At this point, your blog should accept friendly URLs. You can test whether this is true by navigating to `http://localhost/simple_blog/blog/` in a browser to see the results (see Figure 6-7).



Simple Blog Application

Third Entry

Second Entry

First Entry

Post a New Entry

Figure 6-7. *Previewing your blog entries with a friendly URL*

In the next section, you'll modify your application to use friendly entry URLs to access individual entries.

Creating Friendly URLs Automatically

Now that your site can rewrite friendly URLs for your site to process correctly, you need to modify your application to use the new format. You must implement the following steps to make your application run properly:

1. Add a `url` column to the `entries` table
2. Modify `functions.inc.php` to search by and return the `url` value
3. Modify `index.php` to use the `url` value
4. Write a function to create friendly URLs automatically
5. Modify `update.inc.php` to save the new URL

Step 1: Add a url Column to the entries Table

There are a few ways you can make your application use custom URLs. Perhaps the easiest method is to create a custom friendly URL and store it in the database, which is the approach you'll take in this project.

Begin by creating a new column in the entries table, which you call `url`. To do this, navigate to `http://localhost/phpmyadmin`, then select the `simple_blog` database, and, finally, the `entries` table. Click the SQL tab and enter the following command:

```
ALTER TABLE entries
ADD url VARCHAR(250)
AFTER entry
```

This creates a new column in the `entries` table that accepts 250 characters. This field stores your entry URLs.

You can populate this column for pre-existing entries quickly by hand. Click the Browse tab, then scroll to the bottom of the entry listings and click "Check All." To the right of the "Check All" option, click the pencil to edit all the entries at once.

The format you'll be using for entry URLs is to take the title of the entry, remove all special characters (such as apostrophes, punctuation, and so on), and replace spaces with hyphens. For example, "First Entry" would become "first-entry" after processing.

Enter a URL for each entry in the `url` field of each entry, then click the Go button to save.

Step 2: Modify functions.inc.php to Handle URLs

Next, open `functions.inc.php` in Eclipse. You need to change your queries to use the `url` column rather than the `id` column to look up entries, as well as making sure the URL is returned by your page-only query.

First, you modify the parameters of `retrieveEntries()` to accept the URL in place of the ID. Next, you check whether the URL was provided. If so, you retrieve the entry that matches the provided URL. If not, you retrieve entry information for all the entries that correspond to the provided page.

To accomplish this, you need to make the following modifications (shown in bold) to `retrieveEntries()`:

```
function retrieveEntries($db, $page, $url=NULL)
{
    /*
     * If an entry URL was supplied, load the associated entry
     */
    if(isset($url))
    {
        $sql = "SELECT id, page, title, entry
                FROM entries
                WHERE url=?
                LIMIT 1";
        $stmt = $db->prepare($sql);
        $stmt->execute(array($url));
        // Save the returned entry array
        $e = $stmt->fetch();

        // Set the fulldisp flag for a single entry
        $fulldisp = 1;
    }

    /*
     * If no entry URL provided, load all entry info for the page
     */
    else
    {
        $sql = "SELECT id, page, title, entry, url
                FROM entries
                WHERE page=?
                ORDER BY created DESC";
        $stmt = $db->prepare($sql);
        $stmt->execute(array($page));

        $e = NULL; // Declare the variable to avoid errors

        // Loop through returned results and store as an array
        while($row = $stmt->fetch()) {
            $e[] = $row;
            $fulldisp = 0;
        }

        /*
         * If no entries were returned, display a default
         * message and set the fulldisp flag to display a
         * single entry
         */
        if(!is_array($e))
        {
            $fulldisp = 1;
            $e = array(
                'title' => 'No Entries Yet',
                'entry' => 'This page does not have an entry yet!'
            );
        }
    }
```

```
    // Add the $fulldisp flag to the end of the array
    array_push($e, $fulldisp);

    return $e;
}
```

Now your function can search by URL, as well as return the URL information for use in link creation.

Step 3: Modify index.php to Handle URLs

Next, you need to open `index.php` in Eclipse and make some changes. First, you need to swap out your check for an ID in the `$_GET` superglobal and check instead for a `url` variable. Then you need to pass the stored URL information to `retrieveEntries()` for processing.

In `index.php`, make the following modifications to the script at the top (shown in bold highlight):

```
<?php

    /*
     * Include the necessary files
     */
    include_once 'inc/functions.inc.php';
    include_once 'inc/db.inc.php';

    // Open a database connection
    $db = new PDO(DB_INFO, DB_USER, DB_PASS);

    // Figure out what page is being requested (default is blog)
    if(isset($_GET['page']))
    {
        $page = htmlentities(strip_tags($_GET['page']));
    }
    else
    {
        $page = 'blog';
    }

    // Determine if an entry URL was passed
    $url = (isset($_GET['url'])) ? $_GET['url'] : NULL;

    // Load the entries
    $e = retrieveEntries($db, $page, $url);
```

```
  // Get the fulldisp flag and remove it from the array
    $fulldisp = array_pop($e);

    // Sanitize the entry data
    $e = sanitizeData($e);

?>
```

Your “About the Author” page is displayed as a full entry, so you need to an extra line to your full entry display to ensure that administrative links are built properly, even though the entry isn’t accessed with a URL beyond this page. Add the code in bold to `index.php` in the code block starting on line 59:

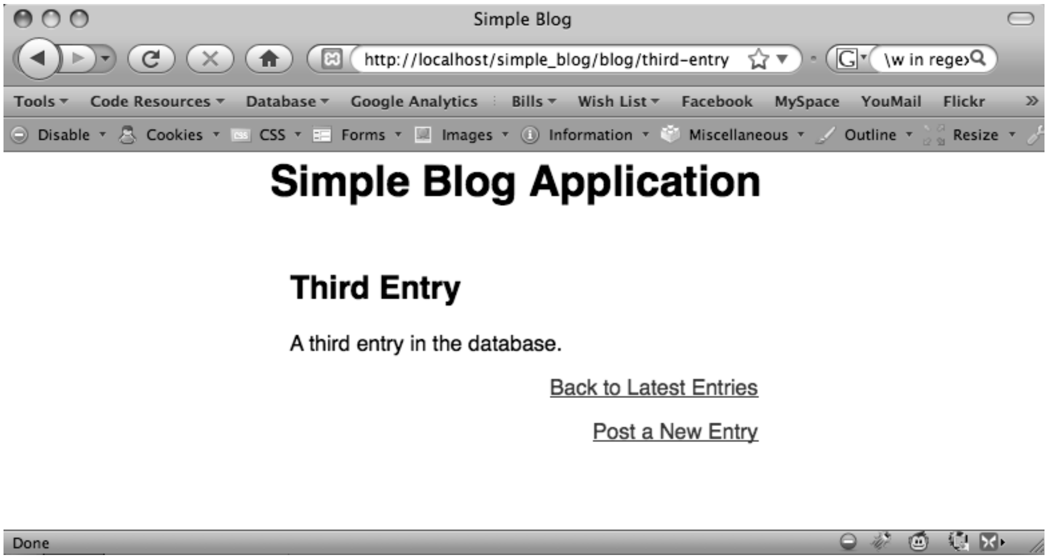
```
<?php

// If the full display flag is set, show the entry
if($fulldisp==1)
{

    // Get the URL if one wasn't passed
    $url = (isset($url)) ? $url : $e['url'];

?>
```

You also need to change how entry previews are formatted, changing the links to reflect your new URL format. Finally, you need to switch your “Post a New Entry” link so it uses the new URL format.

Make the following code modifications in the bottom half of `index.php`, starting at `line 78`, as marked in bold:

```
// If the full display flag is 0, format linked entry titles
else
{
    // Loop through each entry
    foreach($e as $entry) {

?>

        <p>
            <a href="/simple_blog/<?php echo $entry['page'] ?>/<?php echo $entry['url'] ?>">
                <?php echo $entry['title'] ?>

            </a>
        </p>
<?php

    } // End the foreach loop
} // End the else

?>

        <p class="backlink">
            <a href="/simple_blog/admin/<?php echo $page ?>">
                Post a New Entry
            </a>
        </p>

    </div>
```

Now navigate to `http://localhost/simple_blog/blog/` in a browser to see your code in action. Click one of the entries for a full view and note that the URL is now easy to read (see Figure 6-8).



Simple Blog Application

Third Entry

A third entry in the database.

Back to Latest Entries

Post a New Entry

Figure 6-8. *A full entry loaded with a custom URL*

Step 4: Write a Function to Create Friendly URLs Automatically

You don't want to create a URL manually for every entry created, so you need to write a function that generates a URL following your format automatically. To do this, you need to use regular expressions again, but with the twist that this time you combine it with a PHP function called `preg_replace()`:

```
mixed preg_replace  ( mixed $pattern  , mixed $replacement  ,
        mixed $subject  [, int $limit= -1  [, int &$count  ]] )
```

Essentially, `preg_replace()` works similarly to the rewrite rules you're already created. It accepts *patterns* to match (`$pattern`), *replacements* for those matches (`$replacement`), a string in which to search (`$subject`), and two optional parameters: the maximum number of replacements to be performed (`$limit`), which defaults to `-1` (no limit), and the number of replacements made (`$count`).

One of the most convenient features of `preg_replace()` is its ability to accept arrays as both the `$pattern` and `$replacement` parameters, which allows you to perform multiple pattern matches and replacements with a single function call.

To create your URLs, you can accept the title of an entry as a string, which you run through `preg_replace()`. You want to match two patterns: first, you want to replace all spaces with a hyphen; second, you want to remove all non-word characters (excluding hyphens) by replacing them with an empty string.

You place your function, called `makeUrl()`, in `functions.inc.php`. Open the file in Eclipse and insert the following code at the bottom of the file:

```
function makeUrl($title)
{
    $patterns = array(
        '/\s+/',
        '/(?!-)\W+/'
    );
    $replacements = array('-', '');
    return preg_replace($patterns, $replacements, strtolower($title));
}
```

In this function, you begin by defining your array of patterns. The first pattern, `/\s+/`, matches any one or more whitespace characters, such as spaces and tabs. The special character shorthand, `\s`, denotes any whitespace, and the plus sign means "one or more," as explained previously.

The second pattern, `/(?!-)\W+/`, matches one or more non-word characters, excluding the hyphen. The first bit after the *delimiter* (see the note on using regex, which I'll introduce momentarily), `(?!-)`, is a complicated bit of syntax called a *negative lookahead*. Essentially, it means, "if the following match can be made *without* using the character noted, consider it valid." The syntax for this is `(?!`*character*`)`, where *character* is the character or characters that will render a match invalid.

The `\W` is shorthand for the *opposite* of word characters. This means punctuation, whitespace, and other special characters that aren't alphanumeric or the underscore character.

Next, you declare your replacement array. This follows the order of the patterns, which means that you replace the first element of the pattern array with the first element in the replacement array. Your first replacement is for whitespace characters, which you want to replace with a hyphen. You use the second replacement to get rid of other special characters, which you accomplish simply by supplying an empty string.

With your patterns and replacements stored and ready, you return the result of `preg_replace()` when passed your entry title (which you convert to lowercase using the function `strtolower()`). This return value is your custom URL.

■ **Note** When using regex in PHP, or any Perl-compatible language, you need to enclose patterns in delimiters, which are forward slashes. This means that if your pattern is `\w+`, you must enclose it in forward slashes (`/\w+/`) to delimit it as a pattern.

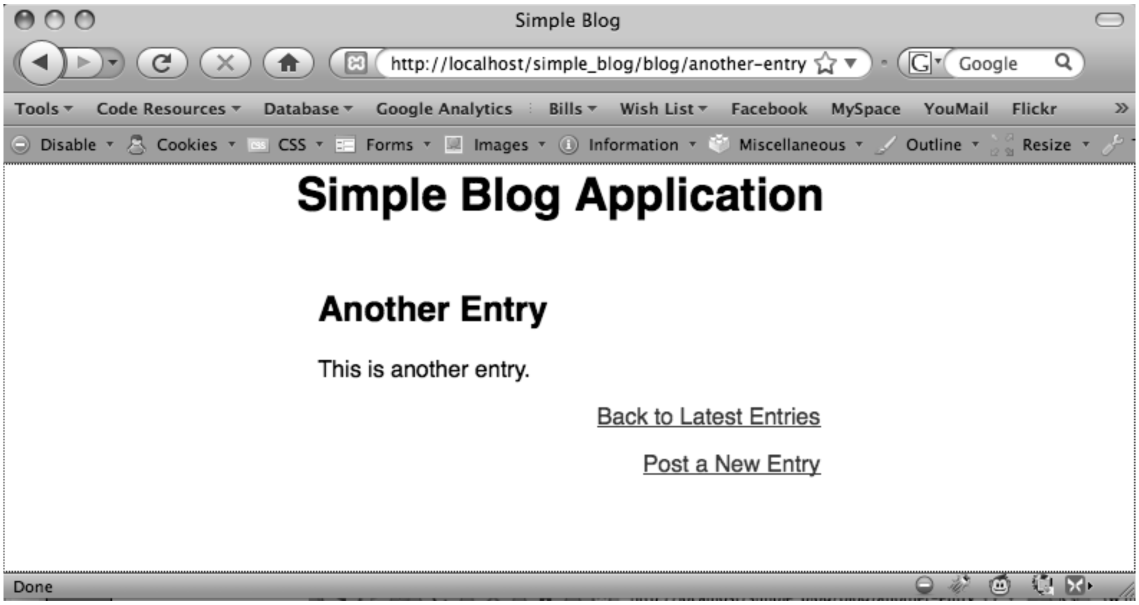
Step 5. Modify update.inc.php to Save URLs in the Database

You now need a way to store the results of `makeUrl()` in the database. To do this, you need to modify your query in `update.inc.php`. Open this file in Eclipse and make the modifications shown in bold:

```
<?php

// Include the functions so you can create a URL
include_once 'functions.inc.php';

if($_SERVER['REQUEST_METHOD']=='POST'
    && $_POST['submit']=='Save Entry'
    && !empty($_POST['page'])
    && !empty($_POST['title'])
    && !empty($_POST['entry']))
{
    // Create a URL to save in the database
    $url = makeUrl($_POST['title']);

    // Include database credentials and connect to the database
    include_once 'db.inc.php';
    $db = new PDO(DB_INFO, DB_USER, DB_PASS);

    // Save the entry into the database
    $sql = "INSERT INTO entries (page, title, entry, url)
            VALUES (?, ?, ?, ?)";
    $stmt = $db->prepare($sql);
    $stmt->execute(
        array($_POST['page'], $_POST['title'], $_POST['entry'], $url)
    );
    $stmt->closeCursor();

    // Sanitize the page information for use in the success URL
    $page = htmlentities(strip_tags($_POST['page']));
// Send the user to the new entry
    header('Location: /simple_blog/'.$page.'/'.$url);
    exit;
}

else
{
    header('Location: ../');
    exit;
}

?>
```

When a new entry is created now, `functions.inc.php` is loaded and `makeUrl()` is called on the new title. The result is stored in the `url` column of the `entries` table, which leaves you with a dynamically generated, custom URL for your new entry. You can then use this custom URL to direct the user to the new entry.

Navigate to `http://localhost/simple_blog/admin/blog` and create a new entry with the title ("Another Entry") and the text ("This is another entry."). Clicking the Save Entry button takes you to your new entry at the URL, `http://localhost/simple_blog/blog/another-entry` (see Figure 6-9).



Simple Blog Application

Another Entry

This is another entry.

Back to Latest Entries

Post a New Entry

Figure 6-9. *An entry with a dynamically created custom URL*

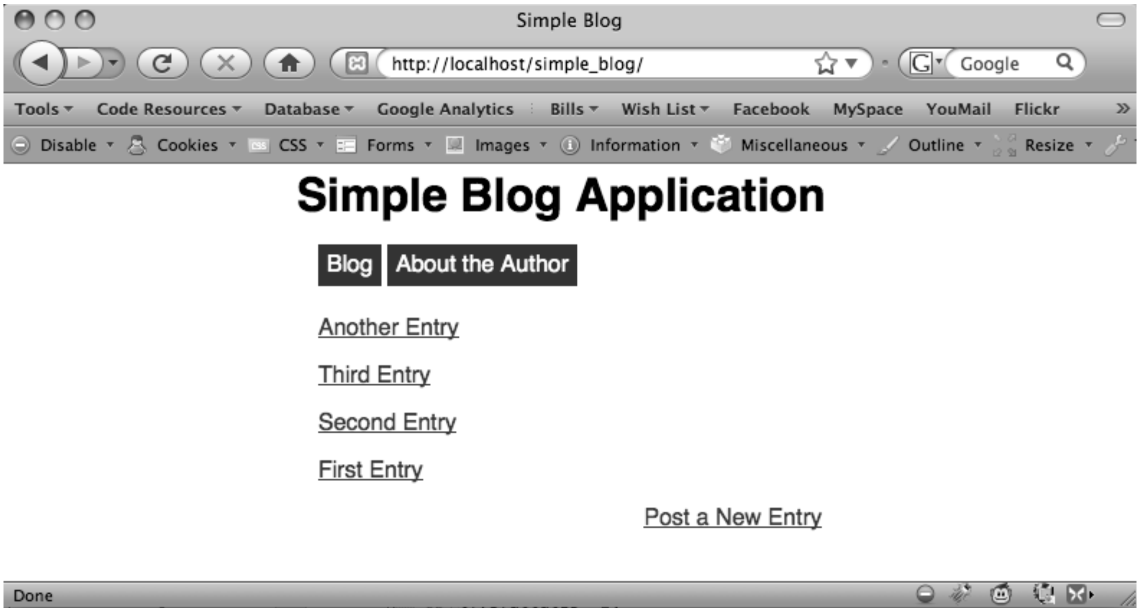
Adding a Menu

Your blog should include a menu item that enables your users to get between the "Blog" and "About the Author" pages. Accomplishing this is a simple matter of adding a snippet of HTML to `index.php` below the main heading.

Open `index.php` in Eclipse and insert the HTML code in bold into the body of the page as indicated, just below the `<h1>` tag:

```
<body>

    <h1> Simple Blog Application </h1>
    <ul id="menu">
        <li><a href="/simple_blog/blog/">Blog</a></li>
        <li><a href="/simple_blog/about/">About the Author</a></li>
    </ul>

    <div id="entries">
```

Next, navigate to `http://localhost/simple_blog/` to see the menu (see Figure 6-10).



Simple Blog Application

Blog About the Author

Another Entry

Third Entry

Second Entry

First Entry

Post a New Entry

Figure 6-10. *Your blog application with a menu in place*

Creating Different Viewing Styles for the Pages

You're nearly finished with this stage of your blog's development. However, if you navigate to the "About the Author" page, you'll see that it shows an entry preview. This is undesirable because there should be only one entry on this page.

To fix this, you need to modify your `retrieveEntries()` function to force the "About the Author" entry to be a full-display entry, even without a URL being supplied.

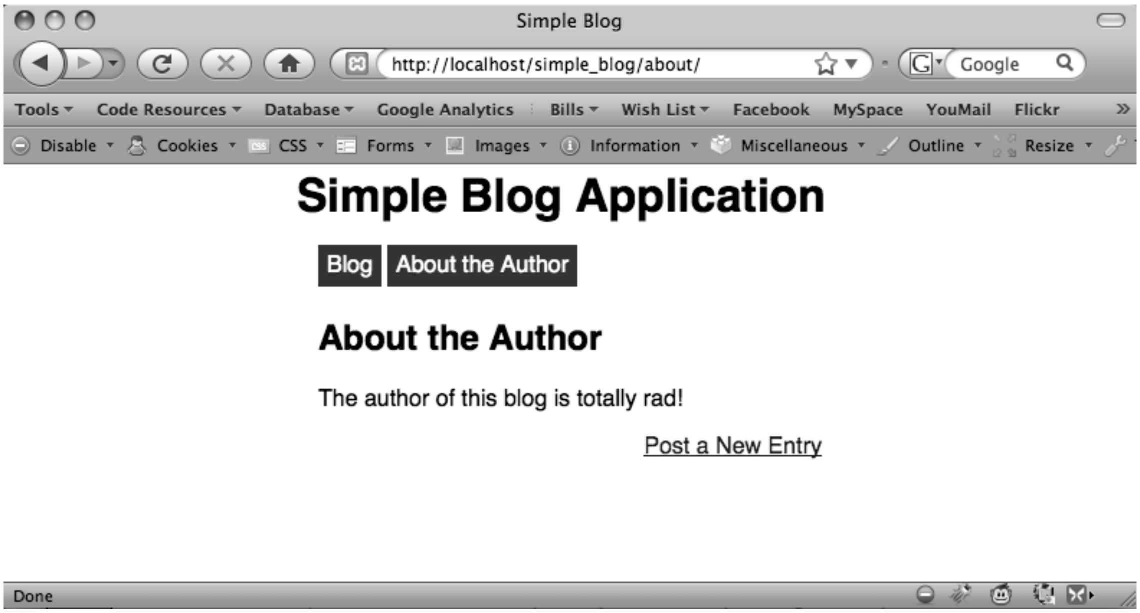
Open `functions.inc.php` and modify `retrieveEntries()` by inserting the following code in bold into the `while` loop starting at `line 38`:

```
// Loop through returned results and store as an array
while($row = $stmt->fetch()) {
    if($page=='blog')
    {
        $e[] = $row;
        $fulldisp = 0;
    }
    else
    {
        $e = $row;
        $fulldisp = 1;
    }
}
```

You also need to remove the original line that defaults `$fulldisp` to `0`. Remove this line from `functions.inc.php`:

```
// Set the fulldisp flag for multiple entries
$fulldisp = 0;
```

Now you can view the full "About the Author" entry by navigating to `http://localhost/simple_blog/about/` in your browser (see Figure 6-11).


Simple Blog
http://localhost/simple_blog/about/
Google
Tools Code Resources Database Google Analytics Bills Wish List Facebook MySpace YouMail Flickr
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize
Done

Simple Blog Application

Blog | About the Author

About the Author

The author of this blog is totally rad!

Post a New Entry

Figure 6-11. *The full "About the Author" entry display*

Summary

In this chapter, you've learned a ton of information. Some of it was pretty advanced, so congratulate yourself! You can now:

- Support multiple pages in your application
- Create custom URLs using regular expressions and `.htaccess`
- Differentiate between multi-entry and single-entry pages

In the next chapter, you'll learn how to update entries that have already been created, bringing you one step closer to having a fully customizable blogging application.