
# Building a Quick-and-Dirty PHP/MySQL Publishing System

**By icarus**

# Table of Contents

# RAD, Dude!

In the highly competitive world of Web development, there's a crying need for a toolkit that allows developers to rapidly and efficiently construct dynamic, robust and scalable Web sites. This toolkit needs to be feature−rich yet easy to use, cost−effective yet labour−efficient, simple yet scalable.

Now, I may be biased or a fool (or maybe even both), but I'm going to go out on a limb here and say that PHP is that toolkit.

Think about it. The language is fast, robust and scalable. It's easy to use, comes with a great manual, and is backed by an enthusiastic user community. It comes with a rich feature set, and includes support for almost every new technology you can think of. It is, in short, the best open−source scripting language available today.

This is not just advertising. I've been using the language for over three years now, and it still amazes me with its capabilities. Constructing a dynamic, database−driven Web site in Perl would take me a week; with PHP, I'm done in two days, and my code is cleaner, more readable and less convoluted than my best efforts in Perl.

Now, if you're a cynical software programmer, all the rhetoric in the world isn't going to convince you. You're not going to believe a word I say until you see the truth with your own eyes. And so, over the new few pages, I'm going to demonstrate PHP's RAD capabilities by using it to build − very rapidly − a simple content publishing system for a Web site.

Before we begin, a little context. The application you're about to see was an actual development effort I undertook a few weeks ago for an existing customer of my company. This customer already had a PHP−based Web site, which we'd developed a year ago; however, it now required an addition, in the form of a dynamic, database−driven page for corporate news and press releases. I was tagged as the person to implement this addition.

I was told that the requirement was an urgent one, and that the customer needed it "yesterday". Since time travel is a feat I have yet to master, I was forced to decline this deadline, and instead promised to have something up and running in two days. This, then, is the story of how I spent those two days.

# A Little Slug−gish

The first task in any development effort is to figure out the customer's requirements. I accomplished this via a phone call to the person most likely to be using the application, the customer's Media/PR department head. She told me exactly what she wanted...and it boiled down to this:

1. An additional couple of pages added to the existing site to list the latest published news and press releases.

2. An administration interface so that individual editors within the PR department could easily publish new press releases to the Web site (her exact words were, "give me something simple, please! I don't have time for anything complicated!").

Since I was already thinking of using a database to store the press releases, I thought this would be a good time to get a little information to help me design this database. So I also asked her to define the attributes of a press release, so that I could build an appropriate administration module, and the functions available through this module.

After much thought and consultation, we concluded that a press release could be broken down into:

− a title (which she called a "slug", much to my amusement);

− a main body containing the text of the press release or news item;

− an information section containing the publication date and the name of the person to be contacted for further information on that particular release.

As to the functions available in the administration module, she said that all she was interested in at the moment was:

− the ability to add new press releases to the site (obviously!);

− the ability to edit existing releases, in order to make corrections or update them with new information;

− the ability to delete older, out−of−date releases and news items.

On the basis of this conversation, I was able to define the amount of work involved quite clearly. So I sent out some email to her and to my boss to get their approval and acceptance of the requirements document, and decided to start work immediately.

# A Maniac Is Born

Now, I can implement this application using a variety of different scripting languages and technologies. I can use Perl, PHP or JSP. I can store the data in a database, in an ASCII text file, or in an XML−encoded document. My personal preference here is PHP for the scripting, and a MySQL database for the data storage − first, because I find development much faster with PHP (and I also happen to like the language a great deal), and second, because the customer's existing Web site is also built on PHP and already has an active database server.

Before reading further, you should download the source code for this case study, so that you can refer to it throughout this article (you will need a Web server capable of running PHP and a mySQL database).

news.zip

Got it? Good. Now, the first thing I'm going to do is design the database tables that will hold the news information, using the information provided to me on the attributes of a press release.

Here's what I came up with:

```
CREATE TABLE news (
id smallint(5) unsigned NOT NULL auto_increment,
slug text NOT NULL,
content text NOT NULL,
contact varchar(255),
timestamp datetime DEFAULT '0000−00−00 00:00:00' NOT NULL,
PRIMARY KEY (id)
);

#
# id − unique item identifier
# slug − item title
# content − item body
# contact − contact person
# timestamp − item publication date
#
```

As you can see, this maps right into the information on the previous page − I have one field for every element of a press release.

For my initial development, I'm going to populate this table with a couple of dummy records. Here goes:

```
INSERT INTO news (id, slug, content, contact, timestamp)
VALUES ( '1',
'Megalomaniacs Inc. Is Born', 'EARTH −− A new star was born
today on the
```

```
planet third closest to the sun. Megalomaniacs Inc., a venture
of
WeWantItAll Corp., today threw open its doors for business in
the ritzy
Jefferson Square business district.

Created with the sole goal of colonizing every single planet
in the known
Universe (and beyond), Megalomaniacs Inc. hopes to quickly
acquire a
monopoly over the vast tracts of uncharted real estate in
space. Speaking
at a press conference, Megalomaniacs Inc. CEO warned reporters
that
Megalomaniacs Inc. would \&quot;take everything it could, and
then
some\&quot;. ', 'Peter Paul (peter@megalo.mania)', '2001−02−01
17:29:25');

INSERT INTO news (id, slug, content, contact, timestamp)
VALUES ( '2',
'Megalomaniacs Inc. Expands To Mars', 'MARS −− As part of its
business
strategy of \&quot;expand and swallow\&quot;, Megalomaniacs
Inc. today
announced that it had successfully sent a team of corporate
raiders to
Mars, in an effort to persuade the inhabitants of that planet
to surrender
their planet for colonization.

Megalomaniacs Inc. COO today said that the move was a
\&quot;friendly
overture\&quot;, but that a failure to comply with the
company\'s
colonization plans would result in a \&quot;swift and sure
eviction of
those little green guys\&quot;. ', 'Tim Jr.
(tim@megalo.mania)',
'2001−07−11 12:13:48');
```

Right. Let's get on with some code.

# The Simple Stuff

You'll remember, from the requirements discussion a couple pages back, that this development effort can broadly be split into two parts. One part consists of the scripts that retrieve the list of newest items from the database and display this list to the user; the other consists of administration scripts that allow editors to manage this list, enter new information, and edit or delete existing information.

Since the first part is simpler, I'm going to get that out of the way first. The scripts involved in this are:

"list.php" − the script which retrieves a list of the five newest entries in the database;

"story.php" − the script which displays the full text for the selected story.

These scripts are stored within the "user" directory in the source code archive.

Here's my first script, which displays the list of five most recent press releases from the database.

```
<?
// list.php - display list of five most recent press releases
?>

<!-- page header - snip -->

<ul>
<?
// includes
include("../conf.php");
include("../functions.php");

// open database connection
$connection = mysql_connect($host, $user, $pass) or die
("Unable to
connect!");

// select database
mysql_select_db($db) or die ("Unable to select database!");

// generate and execute query
$query = "SELECT id, slug, timestamp FROM news ORDER BY
timestamp DESC
LIMIT 0, 5";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// if records present
if (mysql_num_rows($result) > 0)
```

```
{
// iterate through resultset
// print article titles
while($row = mysql_fetch_object($result))
{
?>
<li><font size="−1"><b><a href="story.php?id=<? echo $row−>id;
?>"><?
echo $row−>slug; ?></a></b></font>
<br>
<font size="−2"><? echo formatDate($row−>timestamp); ?></font>
<p>
<?
}
}
// if no records present
// display message
else
{
?>
<font size="−1">No press releases currently available</font>
<?
}

// close database connection
mysql_close($connection);
?>
</ul>

<!−− page footer − snip −−>
```

There's no magic here at all. This script simply connects to the database, retrieves a set of records, and formats them for display in a Web browser. Let's look at it in detail.

# An Anatomical Exploration

The first step is, obviously, to open a connection to the database through which queries can be transmitted − this is accomplished via PHP's mysql_connect() function:

```
<?
// open database connection
$connection = mysql_connect($host, $user, $pass) or die
("Unable to
connect!");
?>
```

In case you're wondering, the host name, user name and password are all variables sourced from the configuration file "conf.php". This file has been include() at the top of the script, and it looks like this:

```
<?
// conf.php − configuration parameters

// database configuration
$host = "localhost";
$user = "mm_273";
$pass = "secret";
$db = "mm_db6388";

// default contact person
$def_contact = "Johnny Doe (jd@megalo.mania)";
?>
```

Extracting this configuration information into a separate file makes it easier to update the application in case the database username or password changes. It's far easier to update a single file than it is to update multiple scripts, each with the values hard−wired into it.

Next, a database needs to be selected, via the mysql_select_db() function:

```
<?
// select database
mysql_select_db($db) or die ("Unable to select database!");
?>
```

Again, the database name comes from the configuration file "conf.php".

With the connection open, the next step is to execute an SQL query, via the mysql_query() function:

```
<?
// generate and execute query
$query = "SELECT id, slug, timestamp FROM news ORDER BY
timestamp DESC
LIMIT 0, 5";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());
?>
```

Note the addition of the DESC clause in the SELECT statement above − this orders the items in the order of most recent first − and the additional LIMIT clause, which restricts the resultset to five items only.

Once a resultset has been returned, I've used the mysql_fetch_object() function, in combination with a "while" loop, to iterate through it and format the fields in each row for display.

```
<?
// if records present
if (mysql_num_rows($result) > 0)
{
// iterate through resultset
// print article titles
while($row = mysql_fetch_object($result))
{
?>
<li><font size="−1"><b><a href="story.php?id=<? echo $row−>id;
?>"><?
echo $row−>slug; ?></a></b></font>
<br>
<font size="−2"><? echo formatDate($row−>timestamp); ?></font>
<p>
<?
}
}
// if no records present
// display message
else
{
?>
<font size="−1">No press releases currently available</font>
<?
}
?>
```

In case you're wondering, the mysql_fetch_object() function is a nifty little thing I only discovered recently. It converts each row into a PHP object, and represents each column within that row as a property of that object; field values can then be accessed in standard object notation. Therefore, in order to access the column named "slug" of each row, I would simply use the code

```
$row->slug
```

The formatDate() function you see in the code above is a function I wrote to turn a MySQL timestamp into a human−friendly date string. Here's the function definition:

```
<?
// format MySQL DATETIME value into a more readable string
function formatDate($val)
{
$arr = explode("-", $val);
return date("d M Y", mktime(0,0,0, $arr[1], $arr[2],
$arr[0]));
}
?>
```

Note the addition of several lines of code that tell the script what to do if no records are returned by the query. If I didn't have this, and the database turned out to be empty (which might happen the very first time the application was installed for use), the generated page would be completely empty − not a very nice thing to show to your users, especially on a potentially high−traffic page. So I've worked around this with some code that displays a neat little message if no records are found in the database.

```
<?
// if records present
if (mysql_num_rows($result) > 0)
{
// iterate through resultset
// print article titles
}
// if no records present
// display message
else
{
?>
<font size="-1">No press releases currently available</font>
<?
}
?>
```

It's important, as a developer, to think through all possible situations and write code that handles each one intelligently. The possibility of an empty database doesn't even occur to many novice developers − and this can lead to embarrassing situations if you're demonstrating the application to your boss...or worse, the customer.

Finally, with all the records processed and displayed, the mysql_close() function terminates the database connection.

```
<?
// close database connection
mysql_close($connection);
?>
```

Here's what it all looks like:

# Bedtime Stories

You'll see, from the code on the previous page, that every press release title is linked to a script named "story.php" via its unique ID. This "story.php" script displays the complete text for the selected press release − and it looks like this:

```
<?
// story.php − display contents of selected press release
?>

<!−− page header − snip −−>

<?
// includes
include("../conf.php");
include("../functions.php");

// open database connection
$connection = mysql_connect($host, $user, $pass) or die
("Unable to
connect!");

// select database
mysql_select_db($db) or die ("Unable to select database!");

// generate and execute query
$query = "SELECT slug, content, contact, timestamp FROM news
WHERE id =
'$id'";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// get resultset as object
$row = mysql_fetch_object($result);

// print details
if ($row)
{
?>
<p>
<b><? echo $row−>slug; ?></b>
<p>
<font size="−1"><? echo nl2br($row−>content); ?></font>
<p>
<font size="−2">This press release was published on <? echo
formatDate($row−>timestamp); ?>. For more information, please
```

```
contact <?
echo $row->contact; ?></font>
<?
}
else
{
?>
<p>
<font size="-1">That press release could not be located in our
database.</font>
<?
}

// close database connection
mysql_close($connection);
?>

<!-- page footer - snip -->
```

Again, extremely simple − connect, use the ID to get the full text for the corresponding item, and display it. Here's what it looks like:


With that, I've successfully completed the first part of this development effort. I now have a primitive publishing system that can be used to provide users of a Web site with news, press releases and other information.

# Admin Ahoy!

At this point in time, I do not really have a simple way to update the database with new information. In order to insert or edit information into the database, I need to know SQL and have access to a MySQL client. This works fine for me, the developer − but remember what the customer said about wanting something simple and easily usable?

Obviously, I cannot ask the customer to learn SQL just to update the database. So I need to develop a simple, friendly interface that she can use to update the database. Which brings me to the second part of the development effort − the administration module.

Based on the functions described to me by the customer, it seems clear (to me, at least) that I will need the following four scripts:

"list.php" − the starting point for the administration module, which lists all press releases currently in the database and allows the administrator to select an individual record for an edit or delete operation;

"edit.php" − the script which allows the administrator to update a selected record;

"delete.php" − the script which allows the administrator to delete the selected record;

"add.php" − the script which allows the administrator to add a new record.

These scripts are stored within the "admin" directory in the source code archive. When the application is finally uploaded to the customer's Web site, this directory will need to be protected against unauthorized usage via Apache's HTTP authentication mechanism (for more information on how this works, take a look at the links at the end of this article).

Let's look at each of these in turn.

# A List In Time

First up, "list.php". As described above, it simply displays a list of all press releases currently stored in the database, with links to the scripts to actually edit or delete them. Here goes:

---

```
<?
// list.php - display list of all press releases
?>

<!-- page header - snip -->

<?
// includes
include("../conf.php");
include("../functions.php");

// open database connection
$connection = mysql_connect($host, $user, $pass) or die
("Unable to
connect!");

// select database
mysql_select_db($db) or die ("Unable to select database!");

// generate and execute query
$query = "SELECT id, slug, timestamp FROM news ORDER BY
timestamp DESC";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// if records present
if (mysql_num_rows($result) > 0)
{
// iterate through resultset
// print title with links to edit and delete scripts
while($row = mysql_fetch_object($result))
{
?>
<font size="-1"><b><? echo $row->slug; ?></b> [<? echo
formatDate($row->timestamp); ?>]</font>
<br>
<font size="-2"><a href="edit.php?id=<? echo $row->id;
?>">edit</a> | <a
href="delete.php?id=<? echo $row->id; ?>">delete</a></font>
<p>
<?
```

```
}
}
// if no records present
// display message
else
{
?>
<font size="−1">No press releases currently
available</font><p>
<?
}

// close connection
mysql_close($connection);
?>
<font size="−2"><a href="add.php">add new</a></font>

<!−− page footer − snip −−>
```

As you can see, this is almost identical to the code used in the other "list.php" − and well it should be, since it performs a nearly−identical function. Here's what it looks like:


Pay special attention to the links to "edit.php" and "delete.php" in the script above; you'll see that each of these scripts is passed an additional $id variable, which contains the unique record identifier for that particular item.

```
<font size="−2"><a href="edit.php?id=<? echo $row−>id;
?>">edit</a> | <a
href="delete.php?id=<? echo $row−>id; ?>">delete</a></font>
```

Don't worry too much about why I'm doing this − all will be explained shortly.

# Splitting Up

Next, "add.php". If you think about it, you'll realize that this script actually has two components to it: a form, which displays fields for the administrator to enter information, and a form processor, which validates the input and inserts it into the database.

Now, I could write this as two separate scripts; however, it's a habit of mine to put both components into the same script and wrap a conditional test around them to decide which one gets used when. So, the broad outline of my "add.php" script would look like this:

```
<?
// form not yet submitted
// display initial form
if (!$submit)
{
// code to display form goes here
}
// form submitted
// process it
else
{
// code to process form data goes here
}
?>
```

Based on the presence or absence of the $submit variable, the script can take a decision as to whether to display the initial form, or initiate the form processor.

Now, when an administrator first accesses the page through a browser, the $submit variable will not exist, and so control will shift to the first section of the script, which displays an HTML form. Let's look at that first:

```
<?
// add.php - add a new press release
?>

<!-- page header - snip -->

<?
// form not yet submitted
// display initial form
if (!$submit)
{
?>
<table cellspacing="5" cellpadding="5">
<form action="<? echo $PHP_SELF; ?>" method="POST">
```

```
<tr>
<td valign="top"><b><font size="−1">Slug</font></b></td>
<td><input size="50" maxlength="250" type="text"
name="slug"></td>
</tr>
<tr>
<td valign="top"><b><font size="−1">Content</font></b></td>
<td><textarea name="content" cols="40"
rows="10"></textarea></td>
</tr>
<tr>
<td valign="top"><font size="−1">Contact person</font></td>
<td><input size="50" maxlength="250" type="text"
name="contact"></td>
</tr>
<tr>
<td colspan=2><input type="Submit" name="submit"
value="Add"></td>
</tr>
</form>
</table>
<?
}
else
{
// form processor code here
}
?>

<!−− page footer − snip −−>
```

Here's what it looks like:


Now, once the administrator enters data into this form and submits it, the same script is called again to process

the data (note the presence of the special $PHP_SELF variable in the form's ACTION attribute). Since the $submit variable will now exist, control will transfer to the latter half of the script, which looks like this:

```
<?
// add.php − add a new press release
?>

<!−− page header − snip −−>

<?
if (!$submit)
{
// form display code goes here
}
else
{
// includes
include("../conf.php");
include("../functions.php");

// set up error list array
$errorList = array();
$count = 0;

// validate text input fields
if (!$slug) { $errorList[$count] = "Invalid entry: Slug";
$count++; }

if (!$content) { $errorList[$count] = "Invalid entry:
Content"; $count++; }

// set default value for contact person
if (!$contact) { $contact = $def_contact; }

// check for errors
// if none found...
if (sizeof($errorList) == 0)
{
// open database connection
$connection = mysql_connect($host, $user, $pass) or die
("Unable to
connect!");

// select database
mysql_select_db($db) or die ("Unable to select database!");

// generate and execute query
$query = "INSERT INTO news(slug, content, contact, timestamp)
```

```
VALUES('$slug', '$content', '$contact', NOW())";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// print result
echo "<font size=−1>Update successful. <a href=list.php>Go
back to the
main menu</a>.</font>";

// close database connection
mysql_close($connection);
}
else
{
// errors found
// print as list
echo "<font size=−1>The following errors were encountered:
<br>";
echo "<ul>";
for ($x=0; $x<sizeof($errorList); $x++)
{
echo "<li>$errorList[$x]";
}
echo "</ul></font>";
}
}
?>

<!−− page footer − snip −−>
```

Fairly simple, this. The first thing to do is ensure that all required values are present, and to generate errors if not. These errors are stored in the array $errorList.

```
<?
// set up error list array
$errorList = array();
$count = 0;

// validate text input fields
if (!$slug) { $errorList[$count] = "Invalid entry: Slug";
$count++; }

if (!$content) { $errorList[$count] = "Invalid entry:
Content"; $count++; }
?>
```

In the event that the contact person field is left empty, a default value is used; this value is pulled in from the configuration file "conf.php".

```
<?
// set default value for contact person
if (!$contact) { $contact = $def_contact; }
?>
```

Once all the data validation is complete, the $errorList array is checked for entries. If entries are present in this array, a message is displayed listing the errors; if not, an INSERT query is generated to add the data to the database, and a success message is printed to the browser.

```
<?
// check for errors
// if none found...
if (sizeof($errorList) == 0)
{
// snip

// generate and execute query
$query = "INSERT INTO news(slug, content, contact, timestamp)
VALUES('$slug', '$content', '$contact', NOW())";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// print result
echo "<font size=−1>Update successful. <a href=list.php>Go
back to the
main menu</a>.</font>";

// snip
}
else
{
// errors found
// print as list
echo "<font size=−1>The following errors were encountered:
<br>";
echo "<ul>";
for ($x=0; $x<sizeof($errorList); $x++)
{
echo "<li>$errorList[$x]";
}
echo "</ul></font>";
}
```

```
?>
```

# Erasing The Past

So that takes care of adding new data to the database. Now, how about deleting it?

You'll remember, from the discussion of "list.php" a few pages back, that the script "delete.php" is passed a $id variable, which holds the unique database identifier for the selected news item. The script "delete.php" needs this identifier in order to delete the correct record from the database.

Here's the code that makes up "delete.php":

```
<?
// delete.php - delete a press release
?>

<!-- page header - snip -->

<?
// includes
include("../conf.php");
include("../functions.php");

// open database connection
$connection = mysql_connect($host, $user, $pass) or die
("Unable to
connect!");

// select database
mysql_select_db($db) or die ("Unable to select database!");

// generate and execute query
$query = "DELETE FROM news WHERE id = '$id'";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// close database connection
mysql_close($connection);

// print result
echo "<font size=-1>Deletion successful. <a href=list.php>Go
back to the
main menu</a>.</font>";
?>

<!-- page footer - snip -->
```

This is so simple it hardly requires any explanation. The ID passed to the script via the $id variable is used to construct and execute a DELETE query, which removes the corresponding record from the database. Short, sweet and quite efficient.

# Changing Things Around

The last item on the agenda involves updating, or editing, a news item. The script that does this is called "edit.php", and it's a combination of both "add.php" and "delete.php".

Like "delete.php", "edit.php" also gets the record's unique identifier via the $id variable. It now needs to display a form similar to that used by "add.php", except that this form needs to be pre−filled with the data for that news item.

Let's see how this is accomplished:

```
<?
// edit.php − edit a press release
?>

<!−− page header − snip −−>

<?
// includes
include("../conf.php");
include("../functions.php");

// form not yet submitted
// display initial form with values pre−filled
if (!$submit)
{
// open database connection
$connection = mysql_connect($host, $user, $pass) or die
("Unable to
connect!");

// select database
mysql_select_db($db) or die ("Unable to select database!");

// generate and execute query
$query = "SELECT slug, content, contact FROM news WHERE id =
'$id'";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// if a result is returned
if (mysql_num_rows($result) > 0)
{
// turn it into an object
$row = mysql_fetch_object($result);
```

```
// print form with values pre−filled
?>
<table cellspacing="5" cellpadding="5">
<form action="<? echo $PHP_SELF; ?>" method="POST">
<input type="hidden" name="id" value="<? echo $id; ?>">
<tr>
<td valign="top"><b><font size="−1">Slug</font></b></td>
<td><input size="50" maxlength="250" type="text" name="slug"
value="<?
echo $row−>slug; ?>"></td>
</tr>
<tr>
<td valign="top"><b><font size="−1">Content</font></b></td>
<td><textarea name="content" cols="40" rows="10"><? echo
$row−>content;
?></textarea></td>
</tr>
<tr>
<td valign="top"><font size="−1">Contact person</font></td>
<td><input size="50" maxlength="250" type="text"
name="contact" value="<?
echo $row−>contact; ?>"></td>
</tr>
<tr>
<td colspan=2><input type="Submit" name="submit"
value="Update"></td>
</tr>
</form>
</table>
<?
}
// no result returned
// print graceful error message
else
{
echo "<font size=−1>That press release could not be located in
our
database.</font>";
}
}
else
{
// form submitted
// start processing it
}
?>

<!−− page footer − snip −−>
```

Using the identifier from "list.php", "edit.php" queries the database for the fields relevant to that particular record, and uses that information to pre−fill an HTML form. Note that the $id variable is also attached to this form as a hidden variable; this ID will be used by the form processor when constructing the UPDATE query.

Here's what it looks like:


You might be wondering why I've bothered to check the number of rows returned by the query, and written code to display an error if no rows were returned. This is necessary because, if the identifier provided to "edit.php" is invalid or non−existent, the query will return zero rows, and the administrator will be faced with a form with no data in it.

Most of the time, this additional check is redundant, since the identifier will be generated from "list.php" and will therefore usually be valid. However, in the event that someone − a malicious hacker or, more likely, a company employee with too much time on his hands − decides to experiment with the URL string, changing the ID that gets appended to it to an invalid value, it could result in a series of ugly error messages or − even worse − cause the application to break. Therefore, by adding this check, I'm increasing the overall security of the application and simultaneously reducing the possibility of error.

Now, once the form gets submitted, the data entered into it needs to be validated and used to update the database. Let's see how that works:

```
<?
// edit.php − edit a press release
?>

<!−− page header − snip −−>

<?
// form not yet submitted
// display initial form with values pre−filled
if (!$submit)
{
// form display code
}
```

```
else
{
// form submitted
// start processing it

// set up error list array
$errorList = array();
$count = 0;

// validate text input fields
if (!$slug) { $errorList[$count] = "Invalid entry: Slug";
$count++; }

if (!$content) { $errorList[$count] = "Invalid entry:
Content"; $count++; }

// set default value for contact person
if (!$contact) { $contact = $def_contact; }

// check for errors
// if none found...
if (sizeof($errorList) == 0)
{
// open database connection
$connection = mysql_connect($host, $user, $pass) or die
("Unable to
connect!");

// select database
mysql_select_db($db) or die ("Unable to select database!");

// generate and execute query
$query = "UPDATE news SET slug = '$slug', content =
'$content', contact =
'$contact', timestamp = NOW() WHERE id = '$id'";
$result = mysql_query($query) or die ("Error in query: $query.
" .
mysql_error());

// print result
echo "<font size=−1>Update successful. <a href=list.php>Go
back to the
main menu</a>.</font>";

// close database connection
mysql_close($connection);
}
else
{
```

```
// errors occurred
// print as list
echo "<font size=−1>The following errors were encountered:
<br>";
echo "<ul>";
for ($x=0; $x<sizeof($errorList); $x++)
{
echo "<li>$errorList[$x]";
}
echo "</ul></font>";
}
}
?>

<!−− page footer − snip −−>
```

This is almost identical to "add.php", with the obvious difference that this query string uses an UPDATE command, while that one used an INSERT command.

# Game Over

And that just about concludes this case study. I now have an application that meets all the requirements defined by the customer. All that's left is to upload it to their Web site, run a few tests on it, and give them a call to let them know where to send the cheque.

As you can see, it's extremely easy to build a simple publishing system with PHP and MySQL. The two technologies, combined together, are so powerful that putting together dynamic, robust Web applications, like the one just described, is a snap. It's also fast − I estimate that I spent a total development time just under four hours on this project − which can come in very handy when working against aggressive deadlines.

If you'd like to learn more about some of the issues, techniques and functions described throughout the course of this article, here are a few links:

The Fundamentals of Relational Database Design:
http://www.microsoft.com/TechNet/Access/technote/ac101.asp?a=printable

Date functions available in PHP: http://www.php.net/manual/en/ref.datetime.php

mySQL functions available in PHP: http://www.php.net/manual/en/ref.mysql.php

Protecting Web pages with HTTP authentication: http://www.apacheweek.com/issues/96−10−18#userauth

If you'd like to read other case studies like this, do consider visiting the following links:

Miles To Go Before I Sleep: http://www.devshed.com/Server_Side/PHP/MilesToGo/

Cracking The Vault: http://www.devshed.com/Server_Side/PHP/Cracking/

The Perfect Job: http://www.devshed.com/Server_Side/MySQL/PerfectJob/

Till next time...stay healthy!

Note: All examples in this article have been tested on Linux/i686 with Apache 1.3.12 and PHP 4.1.0. Examples are illustrative only, and are not meant for a production environment. A fictitious company name has been used in this case study to protect the identity and reputation of the original subject. Melonfire provides no warranties or support for the source code described in this article. YMMV!