
# Democracy, The PHP Way

## By Vikram Vaswani

# Table of Contents

# Proof And Pudding

In your travels across the Web, you've probably seen (maybe even participated in) online polls, quick measurements of what visitors to a site think of the hot−button issues of the day. And back when portals where a Good Thing, online polls appeared on each and every one of them; they were − and still are − a simple and effective method of promoting a sense of community amongst the visitors to a Web site, and of generating demographic data on a site's visitors.

If you have a Web site of your own, an online poll offers a quick way to find out what your visitors are thinking, to add an element of dynamism to your Web site, and to have a few laughs (you'd be surprised how many Internet users, when polled, think that Elvis is still alive!)

Building an online poll isn't very hard; as a matter of fact, with a middling knowledge of PHP, you can slap one together in a couple of hours. And you won't even need a degree in rocket science to accomplish this feat − PHP makes it so easy to build and maintain a Web application like this that you'll wonder why you never did it before.

Over the next few pages, I'm going to demonstrate this by showing you how to build a simple polling system, one which you can quickly integrate into your own Web site. And if you're new to PHP, the process should also offer some insight into how to apply PHP's capabilities to a real−world problem, and create a simple and elegant solution.

Not a believer? Don't take my word for it − taste the pudding for yourself...

# The Plan

The first order of business is to decide the features to be supported by this application. Obviously, there needs to be a mechanism by which the user can view a question, and then select from a list of possible answers. Once the "vote" has been captured, it's traditional to offer the voter an opportunity to look at the results generated thus far.

For purposes of this tutorial, I'll assume that each poll consists of a single question, with three possible responses.

So far as the results are concerned, it's quite easy to provide a tabular report of the votes for each possible option. However, I plan to make the application look more professional by providing a graphical report, in the form of a bar graph. This might seem difficult to do, since the graph would need to be dynamically generated depending on the votes, but PHP's image generation functions make it a snap.

A quick review of the various Web portals suggests that most of them also allow visitors to view the results of previous polls (this is particularly useful if the poll question changes on a daily basis). This is not too difficult to do − so let's add that to the feature list as well.

With this skeleton in mind, the next step is to design a database that supports these requirements.

# Design View

This is a good time for you to download the source code, so that you can refer to it throughout this tutorial (you will need a Web server capable of running PHP and a mySQL database in order to run the application).

poll.zip

After spending an hour looking out the window and another hour at lunch (hey, these are billable hours!), this is the database structure I came up with.

---

```
#
# Table structure for table 'poll'
# poll.sql in the source archive

CREATE TABLE poll (
id int(10) unsigned NOT NULL auto_increment,
question varchar(255) NOT NULL,
response1 varchar(255) NOT NULL,
response2 varchar(255) NOT NULL,
response3 varchar(255) NOT NULL,
votes1 int(10) unsigned DEFAULT '0' NOT NULL,
votes2 int(10) unsigned DEFAULT '0' NOT NULL,
votes3 int(10) unsigned DEFAULT '0' NOT NULL,
date date DEFAULT '0000-00-00' NOT NULL,
PRIMARY KEY (id)
);

#
# Column descriptions:
#
# id - a unique identifier for each poll/question
# question - the poll question
# response1 - possible response #1
# response2 - possible response #2
# response3 - possible response #3
# votes1 - number of votes for response #1
# votes2 - number of votes for response #2
# votes3 - number of votes for response #3
# date - date on which poll was posted
#

Just to get things started, I also INSERTed the first question
into the
database, together with three possible responses.
```

```
#
# Dumping data for table 'poll'
#

INSERT INTO poll (id, question, response1, response2,
response3, votes1,
votes2, votes3, date) VALUES ( '1', 'The Oscar for Best
Picture should go
to...', 'Gladiator', 'Erin Brockovich', 'Traffic', '0', '0',
'0',
'2001-03-07');
```

# Start Me Up

With the database taken care of, it's time to put together the Web pages that the user sees. The first of these is "start.php", which connects to the database to get the latest poll, and displays it with a list of possible responses.

---

```
<html>
<head>
<basefont face="Arial">
</head>

<body bgcolor="white">

<?
// start.php - displays poll and responses

// includes
include("config.php");
include("common.php");

// connect to database and query
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

$query = "SELECT id, question, response1, response2, response3
from $table
ORDER BY id DESC LIMIT 0,1";

$result = mysql_db_query($database, $query, $connection) or
die ("Could not
execute query: $query. " . mysql_error());

// if questions are available, display vote form
if (mysql_num_rows($result) > 0)
{
list ($id, $question, $response1, $response2, $response3) =
mysql_fetch_row($result);

?>
<form method="post" action="vote.php">
<b><? echo $question; ?></b>
<p>
<input type="Radio" name="response" value="1"><? echo
$response1; ?>
```

```
<p>
<input type="Radio" name="response" value="2"><? echo
$response2; ?>
<p>
<input type="Radio" name="response" value="3"><? echo
$response3; ?>
<input type="hidden" name="id" value="<? echo $id; ?>">
<p>

<!-- explanation coming up - keep reading -->
<font size=-2><a href="archive.php?id=<? echo $id; ?>">view
results</a></font>
<font size=-2><a href="archive.php">view past polls</a></font>
<p>
<input type=submit name=submit value="Vote">
</form>
<?
}
// or display a status message
else
{
?>
<i>No polls available!</i>
<?
}

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


</body>
</html>
```

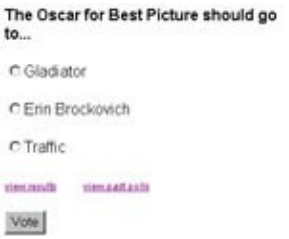
Pay special attention to the SQL query I'm running − I'm using the ORDER BY, DESC and LIMIT keywords to ensure that I get the latest record (read: question) from the database. Once the query returns a result, the list() function is used to walk through the result set and assign each field to a variable; these are then displayed in a form. The identifier for the poll question is also included in the form, as a hidden field; when the form is submitted, this identifier will be used to ensure that the correct record is updated.

If the database is empty, an error message is displayed. In this case, I've already inserted one question into the database, so you won't see it at all; however, it's good programming practice to ensure that all eventualities are accounted for, even the ones that don't occur that often.

In case you're wondering about the files include()d at the top of the script − they simply contain variables and functions common to the application. Here's what "config.php" looks like:

```
<?
// config.php - global variables for all database operations
$hostname="somehost";
$user="us54738";
$pass="7834535";
$database="db54738";
$table="poll";
?>
```

Here's what it looks like:



# Vote Now, Or Forever Hold Your Peace

Once the form is submitted, "vote.php" takes over to process the vote. This script first checks to ensure that the form has been correctly submitted, by verifying the presence of the $submit and $response variables

```
<?
// vote.php - record votes

// check to ensure that the form has been submitted
if (!$submit || !$response)
{
?>
<html>
<head>
<basefont face="Arial">
</head>

<body bgcolor="white">
<i>Error! Please <a href=start.php>try again</a></i>
<?
}
else
{
// vote processing code
}
?>
```

and, assuming that all is well, updates the database to reflect the new vote and displays an appropriate message.

```
<?
// vote.php - record vote

// check to ensure that the form has been submitted
if (!$submit || !$response)
{
// error message
}
else
{
```

```
// all is well − process the vote
?>
<html>
<head>
<basefont face="Arial">
</head>

<body bgcolor="white">

<?
// includes
include("config.php");
include("common.php");

// connect and update table
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");
$fieldname = "votes" . $response;
$query = "UPDATE $table SET $fieldname = $fieldname+1 WHERE id
= $id";
$result = mysql_db_query($database, $query, $connection) or
die ("Could not
execute query: $query. " . mysql_error());

// query successful,display status message...
if ($result)
{
echo "Thank you for voting. Here are the results so far:<p>";
// code to display results − keep reading!
}
// or error in query − display error message
else
{
echo "<i>Error! Please <a href=start.php>try again</a></i>";
}

// close connection
mysql_close($connection);

}
?>

</body>
</html>
```

As you can see, the value of the $response variable is used to determine which option field is updated with the

user's vote.

# The Number Game

Once the database has been updated with the vote, it's a good idea to display the current results of the poll. This involves connecting to the database, using the $id variable to extract the correct record, calculate the total number of votes, and the percentage each option has of the total, and displaying this information in a table.

Here's what all that looks like in PHP:

---

```
<?
// query successful,display status message...
if ($result)
{
echo "Thank you for voting. Here are the results so far:<p>";
// ...and tabulated results

// get a complete count of votes in each category
$query = "SELECT question, response1, response2, response3,
votes1,
votes2, votes3, date from $table WHERE id = $id";

$result = mysql_db_query($database, $query, $connection) or
die ("Could
not execute query: $query. " . mysql_error());

// assign the returned values to variables
list($question, $response1, $response2, $response3, $votes1,
$votes2,
$votes3, $date) = mysql_fetch_row($result);

// count the total votes
$total = $votes1 + $votes2 + $votes3;

// calculate each as a percentage of the total, round to two
decimals
$perc_votes1 = round(($votes1/$total)*100,2);
$perc_votes2 = round(($votes2/$total)*100,2);
$perc_votes3 = round(($votes3/$total)*100,2);

// print it all in a neat table
echo "<table border=0 cellspacing=0 cellpadding=5>";
echo "<tr><td colspan=3><b>$question</b></td></tr>";

// also display an image graph − more on this later!
echo "<tr><td>$response1</td><td> $votes1
($perc_votes1%)</td><td
```

```
rowspan=4 valign=top><img
src=graph.php?votes1=$votes1otes3=$votes3
border=0></td></tr>";
echo "<tr><td>$response2</td><td> $votes2
($perc_votes2%)</td></tr>";
echo "<tr><td>$response3</td><td> $votes3
($perc_votes3%)</td></tr>";
echo "<tr><td><font size=-2>Posted on " . fixDate($date) .
"</font></td><td><font size=-2>$total total
votes</font></td></tr>";
echo "</table><p>";
}
// or error in query - display error message
else
{
echo "<i>Error! Please <a href=start.php>try again</a></i>";
}
?>
```

You need to be careful when converting the absolute numbers into percentages - if there aren't any votes yet, you can get some pretty strange "division by zero" errors. This error is not likely at this stage - after all, you've just added a vote - but it can crop up at a later stage. As we progress, you'll see the correction I've used to account for this situation.

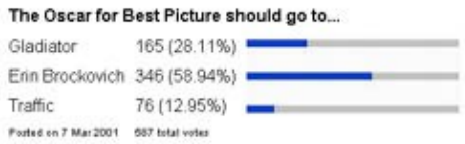
The code snippet above references an image named "graph.php". If you're familiar with PHP's image generation function, you'll immediately divine that this is the PHP script used to dynamically generate the bar graph. I'll be discussing this a little later, so ignore it for the moment.

Finally, you'll see a reference to a fixDate() function in the last line of the table. This is a very simple function I wrote to convert the default MySQL date format into something a little more readable. Here's the function:

```
<?
// format the date so that it looks prettier
function fixDate($val)
{
$dateArray = explode("-", $val);
$val = date("j M Y", mktime(0,0,0, $dateArray[1],
$dateArray[2],
$dateArray[0]));
return $val;
}
?>
```

Feed fixDate() a date in the default MySQL format (say, "2001−03−07") and it will return something much friendlier ("7 Mar 2001").

And here's what the final result page looks like:



# Down Memory Lane

So that takes care of the mechanics of displaying a question, registering votes, and displaying totals. The next item to address is the stated requirement to be able to view previous polls. In this application, the script to accomplish this is called "archive.php".

"archive.php" is extremely simple. It connects to the database, queries for a list of all available polls and votes, and then uses a "while" loop to iterate through the result set and display the final tally for each poll. Take a look:

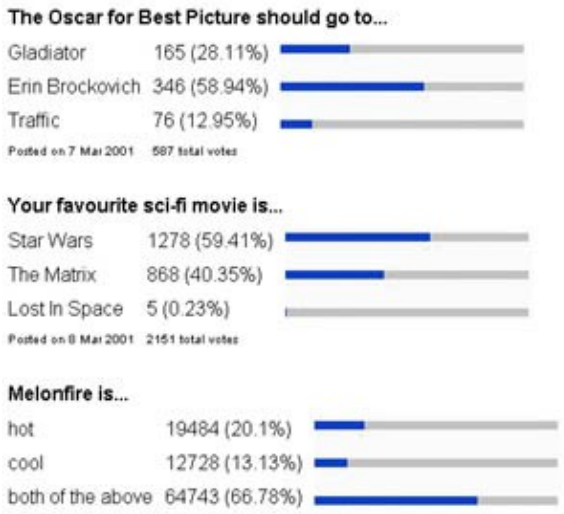
---

```
<html>
<head>
<basefont face="Arial">
</head>

<body bgcolor="white">
<?

// includes
include("config.php");
include("common.php");

// connect and get a list of previous questions
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

$query = "SELECT question, response1, response2, response3,
votes1, votes2,
votes3, date from $table";


$result = mysql_db_query($database, $query, $connection) or
die ("Could not
execute query: $query. " . mysql_error());

// iterate through resultset
while(list($question, $response1, $response2, $response3,
$votes1,
$votes2, $votes3, $date) = mysql_fetch_row($result))
{
// calculate the total votes
$total = $votes1 + $votes2 + $votes3;

if ($total > 0)
```

```
{
// convert sub-totals into percentages
$perc_votes1 = round(($votes1/$total)*100,2);
$perc_votes2 = round(($votes2/$total)*100,2);
$perc_votes3 = round(($votes3/$total)*100,2);
}
// no votes yet - can cause "division by zero" errors
else
{
$perc_votes1 = $votes1;
$perc_votes2 = $votes2;
$perc_votes3 = $votes3;
}

// print a table with results
echo "<table border=0 cellspacing=0 cellpadding=5>";
echo "<tr><td colspan=3><b>$question</b></td></tr>";

// and a graph
echo "<tr><td>$response1</td><td> $votes1
($perc_votes1%)</td><td
rowspan=4 valign=top><img
src=graph.php?votes1=$votes1otes3=$votes3
border=0></td></tr>";
echo "<tr><td>$response2</td><td> $votes2
($perc_votes2%)</td></tr>";
echo "<tr><td>$response3</td><td> $votes3
($perc_votes3%)</td></tr>";
echo "<tr><td><font size=-2>Posted on " . fixDate($date) .
"</font></td><td><font size=-2>$total total
votes</font></td></tr>";
echo "</table><p>";
}

// close connection
mysql_close($connection);
?>
<p>
<font size="-2"><a href="start.php">back to main
page</a></font>
</body>
</html>
```

In this case, I've checked that the total number of votes is greater than zero before calculating percentages, to avoid "division by zero" errors - this is the correction I mentioned a few pages back.

Here's what it looks like:




There's one more thing I'd like to do before leaving "archive.php" − modify it so that, in addition to displaying a complete list of previous polls and results, it also has the capability to display the results of any single, specified poll. This comes in handy on my first page, since it allows me to offer my users a couple of additional options before they cast their vote.

```
<!-- from start.php -->
<font size=-2><a href="archive.php?id=<? echo $id; ?>">view
results</a></font>
<font size=-2><a href="archive.php">view past polls</a></font>
```

If "archive.php" is called without any parameters, it should display the complete archive; if called with a specific $id identifier, it should only display results for that specific poll question.

This is pretty easy to accomplish − I just need to inject the following code snippet into the script:

```
<?
// archive.php

$query = "SELECT question, response1, response2, response3,
votes1, votes2,
votes3, date from $table";

// if a specific id is requested, modify query to only return
that question
```

```
if ($id)
{
$query .= " WHERE id = $id";
}

$result = mysql_db_query($database, $query, $connection) or
die ("Could not
execute query: $query. " . mysql_error());

?>
```

---

# One Picture, A Thousand Words

Both "archive.php" and "vote.php" rely on a dynamically−generated bar graph to spiff up the page design. And this dynamically−generated graph is created through the magic of PHP's image generation functions, as embodied in the file "graph.php".

If you take a close look at the scripts above, you'll see that "graph.php" is always passed a few variables via the URL GET method; these variables represent the number of votes for each of the three available options. The function of "graph.php" thus becomes to calculate appropriate percentages for each of these options, and represent these percentages graphically. Let's take a look:

---

```
<?
// graph.php − generates a bar graph of votes using an
existing, blank image

// calculate the total
$total = $votes1 + $votes2 + $votes3;

if ($total > 0)
{
// convert sub−totals into percentages
$perc_votes1 = round(($votes1/$total)*100,2);
$perc_votes2 = round(($votes2/$total)*100,2);
$perc_votes3 = round(($votes3/$total)*100,2);
}
// "division by zero" correction
else
{
$perc_votes1 = $votes1;
$perc_votes2 = $votes2;
$perc_votes3 = $votes3;
}

// header
Header("Content−Type: image/jpeg");

// set up image and colours
$im = ImageCreateFromJPEG("graph.jpg");
$blue = ImageColorAllocate($im, 8, 63, 206);

// fill with colour up to specific length
// length of colour bar depends on percentage of votes
ImageFilledRectangle($im, 0, 3, ($perc_votes1*2), 9, $blue);
ImageFilledRectangle($im, 0, 34, ($perc_votes2*2), 40, $blue);
ImageFilledRectangle($im, 0, 65, ($perc_votes3*2), 71, $blue);
```

```
// output to browser
ImageJPEG($im);
?>
```

The first few lines are familiar − the conversion of absolute numbers into percentages.

```
<?
// calculate the total
$total = $votes1 + $votes2 + $votes3;

if ($total > 0)
{
// convert sub-totals into percentages
$perc_votes1 = round(($votes1/$total)*100,2);
$perc_votes2 = round(($votes2/$total)*100,2);
$perc_votes3 = round(($votes3/$total)*100,2);
}
// "division by zero" correction
else
{
$perc_votes1 = $votes1;
$perc_votes2 = $votes2;
$perc_votes3 = $votes3;
}
?>
```

Once that's done with, a header is sent and a base image, "graph.jpg", read into memory. This is the base image for all graphs to be generated, and it looks like this:

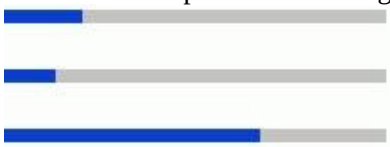
The length of each empty bar in the image above is 200 pixels (100%). Now, based on the percentage values already generated, "graph.php" will fill each bar with a specified colour; the length of the fill is determined by the quantum of votes. For example, if a specific option receives 25% of the votes, the fill will be 50 pixels long (25% of 200).

```
<?
// fill with colour up to specific length
// length of colour bar depends on percentage of votes
ImageFilledRectangle($im, 0, 3, ($perc_votes1*2), 9, $blue);
ImageFilledRectangle($im, 0, 34, ($perc_votes2*2), 40, $blue);
ImageFilledRectangle($im, 0, 65, ($perc_votes3*2), 71, $blue);
?>
```

Once the bars are filled, the image is sent to the browser for display.

```
<?
// output to browser
ImageJPEG($im);
?>
```

Here's an example of what it might look like.



In case this didn't make any sense, you should take a look at the article entitled "Image Generation With PHP" at http://www.devshed.com/Server_Side/PHP/ImageGeneration/. And after you're done with that, come back here and experiment with displaying poll results in the form of a pie chart, rather than a bar graph. Go on − it isn't nearly as hard as it looks!

# Cookie−Cutter Code

The way things are currently set up, a single user can vote for a specific option than once, thereby contravening one of the basic principles of democracy: one citizen, one vote. Not many users would have the patience or inclination to do this; however, it *is* a hole, and should be plugged.

I've decided to make it slightly more difficult for users to vote more than once by setting a cookie on their system, once their vote has been successfully cast. With the addition of a few lines of script, I can now check for the presence or absence of the cookie, and thereby decide whether or not to accept the vote.

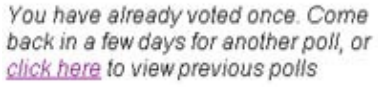
Here's the updated "vote.php" script:

```
<?
// vote.php − record vote

// check to ensure that the form has been submitted
if (!$submit || !$response)
{
// rest of error message code
}
// check the cookie to ensure that user has not voted already
else if ($lastpoll &&$lastpoll == $id)
{
?>
<html>
<head>
<basefont face="Arial">
</head>

<body bgcolor="white">
<i>You have already voted once. Come back in a few days for
another poll,
or <a href=archive.php>click here</a> to view previous
polls</i>
<?
}
// all is well − refresh the cookie (or set a new one) and
process the vote
else
{
setCookie("lastpoll", $id, time()+2592000);
// rest of vote processing code
}
?>
```

Once the user votes, a cookie is set on the client browser; this cookie contains the name−value pair

```
lastpoll=$id
```

Now, on each subsequent vote attempt, the script will first check for the presence of the cookie and, if it exists, the value of the cookie variable $lastpoll. Only if the cookie is absent (indicating that this is a first−time voter) or the value of $lastpoll is different from the identifier for the current poll question (indicating that the user has voted previously, but in response to a different question) will the vote be accepted.




This is by no means foolproof − any reasonably adept user can delete the cookie from the client's cache and vote more than once − but it does perhaps offer an additional layer of security to the process. The ideal method, of course, is to track voters on the server itself, and deny votes to those who have already voted − and indeed, this is a feasible alternative if a site requires users to register with unique usernames before accessing its online polls.

# Adding More...

The final item on today's menu is perhaps the simplest − a form which allows administrators to easily add new questions to the system.

This script, named "add.php", is divided into two sections. The initial section is the form itself, with fields for the question and possible responses.

---

```
<html>
<head>
<basefont face="Arial">
</head>

<body bgcolor="white">

<?
// this script is meant only for the administrator − add new
polls to the
database

// $submit does not exist −> forms has not been submitted −>
display
initial page
if (!$submit)
{
?>
<table border="0" cellspacing="5" cellpadding="5">
<form action="add.php" method="post">

<tr>
<td>Poll question<br><input type="Text" name="question"
size="25"></td>
</tr>

<tr>
<td>Response 1<br><input type="Text" name="response1"
size="25"></td>
</tr>

<tr>
<td>Response 2<br><input type="Text" name="response2"
size="25"></td>
</tr>

<tr>
```

```
<td>Response 3<br><input type="Text" name="response3"
size="25"></td>
</tr>

<tr>
<td align=center><input type=submit name=submit value="Add
Question"></td>
</tr>

</form>
</table>

<?
}
// form has been submitted − process data
else
{
// form processing code goes here
}
?>

</body>
</html>
```

---

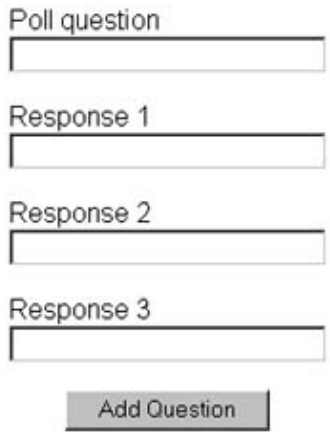
Once the form has been submitted, the data from the form fields is assimilated into an SQL query and INSERTed into the database.

---

```
<html>
<head>
<basefont face="Arial">
</head>

<body bgcolor="white">

<?
// this script is meant only for the administrator − add new
polls to the
database

// $submit does not exist −> forms has not been submitted −>
display
initial page
if (!$submit)
{
```

```
// initial form goes here
}
// form has been submitted − process data
else
{

// includes
include("config.php");
include("common.php");

// connect and insert form data into database
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

$query = "INSERT INTO $table (question, response1, response2,
response3,
date) VALUES ('$question', '$response1', '$response2',
'$response3',
NOW())";

$result = mysql_db_query($database, $query, $connection) or
die ("Could not
execute query: $query. " . mysql_error());

// check for result code
if ($result)
{
echo "<i>Entry successfully added. Click here to <a
href=start.php>view</a></i>";
}
else
{
echo "<i>Error! Please <a href=add.php>try again</a></i>";
}
// close connection
mysql_close($connection);
}

?>

</body>
</html>
```

And here's what it looks like:




And that's about it. Hopefully, this exercise gave you some insight into how PHP can be used to build a simple Web application, and illustrated its power and flexibility as a rapid development tool for the Web medium. You can use the example scripts above to build your own simple poll, or even modify them a little bit and create an online quiz (it *is* the same basic principle − one question, three answers). Either way, have fun...and stay healthy!