A complete example

We’ll write a simple example program that allows users to get a quick geography quiz about capital cities.

The quiz will work as follows:

- Users will select (from a drop-down menu) a world region.
- A country in that region will be randomly selected.
- All the cities in that country will be displayed to the user.
- The user will select which city is the capital.
- Their selection will be marked wrong or right.

The scripts

We’ll implement this with three scripts region.php for the initial region selection, question.php to generate the question, and answer.php to report on the answer.

Note that it is possible—and in fact quite usual—to do all of these within a single script, but it makes that script more complicated.
Region Selector

The region.php script needs to get a list of regions from the database, and then construct a HTML menu item with that list of regions.

```php
require_once "db3240.php";
$conn = getConnection();
$sql = "SELECT DISTINCT region FROM Country";
$result =& $conn->query($sql);
if (MDB2::isError($result)) {
    die($result->getUserInfo());
}

regions =& $result->fetchCol(0);
if (MDB2::isError($regions)) {
    die($regions->getUserInfo());
}
```

Menu making

Menu making is frequently needed, so an ideal candidate for a function to be added to db3240.php.

```php
function makeMenu($name, $items) {
    print "<select name="$name">";
    $len = count($items);
    print "<option selected>$items[0]</option>";
    for ($i=1; $i<len; $i++) {
        print "<option>$items[$i]</option>";
    }
    print "</select>";
}
```

The only new thing to notice here is that in order to create a double-quote inside a double-quoted string, we have to use an escape sequence ".

What the user sees

The HTML

Now the PHP and HTML must be appropriately mixed: first the PHP to extract the region list:

```html
<?php
// the code that creates $regions
?>
<form action="question.php" method="post">
<p>
<?php makeMenu("region", $regions);?>
<input type="submit" value="Go!">
</p>
</form>
```

Notice that the second portion of PHP code has access to the variable $regions that was defined in the first portion.
Getting the countries

The destination script will now use the control's value to find all the countries from that region — making sure that we only list countries that have capital cities!

```php
<?php
require_once "db3240.php";

$conn = getConnection();
$sql = "SELECT C.name, C.code FROM Country C WHERE " . "C.region = "$region" AND C.capital IS NOT NULL";

$result =& $conn->query($sql);
if (MDB2::isError($result)) {
    die($result->getUserInfo());
}

Note the use of the string concatenation .= to build the query.
```

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III 9 / 26

Choosing a random country

Now we need to find a random row from the result set.

```php
$nr = $result->numRows();
$cn = rand(0,$nr-1);
$row = $result->fetchRow(MDB2_FETCHMODE_ORDERED,$cn);
$result->free();
```

This uses the PHP function rand() to generate a random row number $cn, and then fetchRow() to fetch that particular row.

The first argument of fetchRow() is an MDB2 constant that means that the array should be returned as an ordered array, so that the elements are $row[0] and $row[1].

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III 10 / 26

Getting that country's cities

Now $row[0] contains the random country's name, and $row[1] contains its code, so the next step is to find all of the cities listed for that country.

```php
$sql = "SELECT T.name FROM City T WHERE " . "$row[1]" ORDER BY T.name"
$result =& $conn->query($sql);
if (MDB2::isError($result)) {
    die($result->getUserInfo());
}
$cities = $result->fetchCol(0);
```

The main new item here is the fetchCol() method that returns an entire column of a result set as an array.

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III 11 / 26

The HTML

The HTML is again very simple, with most of the work being done by our makeMenu() function.

```php
<p>
What is the capital of <?= $row[0] ?>.
</p>
<form action="answer.php" method="post">
<p>
<?php makeMenu("capital", $cities);?>
<input type="hidden" name="cName" value="<?=$row[0]?>">
<input type="submit" value="Go!">
</p>
</form>
```

However this uses two new concepts - a hidden control and a new PHP tag.

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III 12 / 26
Hidden fields

The question.php script needs to pass the country name to answer.php. One way of doing this is to ensure that the HTML form contains a control with this value. However we cannot let the user change the value of this control.

It would be possible to create a menu with one item (so the user could not select any other), but it looks silly, so it is better to just create a control which is not displayed to the user.

Note: This is not a security measure, merely a way of controlling the display.
Compare the answers

Now the script knows the user's "guess" and the real answer so it can compare them and prepare the correct sort of response.

```php
if ($answer == $guess) {
    $msg = "Well done, you got that one!";
} else {
    $msg = "Whoops... not quite."
}
```

Notice that at the moment we are not "counting up" the scores of the user on the quiz, just treating each question as a separate thing.

The HTML

The HTML simply prints out the already-prepared variable containing the message, along with the correct capital.

```html
<?php
print $msg;
?>
<p>
The capital of $cName is $answer.
</p>
```

Start over

Finally, we add some mechanism to allow the user to return to the first script and have another go.

```html
<form action="region.php">
<input type="submit" value="Try Again!">
</form>
```

This is a form containing no controls other than a button that links back to the initial script. (This could of course be a static hyperlink as the form contains no controls that can be changed by the user.)

What the user sees

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III 17 / 26

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III 18 / 26

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III 19 / 26

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III 20 / 26
Scores

Of course, in an online quiz, the user will probably want some summary information about their performance in terms of numbers of questions answered correctly and incorrectly.

In order to do this we need to be able to maintain state — in other words, have the scripts remember the user’s prior performance.

In this simple example, it could be managed purely with hidden controls in the forms (because the only navigation is through the button). However using PHP’s sessions is more general.

Session variables

We will use the session variables $_SESSION['numCorrect'] and $_SESSION['numQuestions'] to store the number of correct answers and total number of attempts.

The first script region.php must therefore be able to distinguish between two situations:
- A user starting the quiz
- A user returning for their next question

Setting up

The script region.php can tell whether the user is new or not by checking to see if the session variables are already set or not.

```php
session_start();
if (!isset($_SESSION['numCorrect'])) {
    $_SESSION['numCorrect'] = 0;
    $_SESSION['numQuestions'] = 0;
}
```

If they are NOT already set, then they should be created and given initial values of 0.

Update

When the user has attempted a question, then the values should be updated.

This in the answer.php script some additional work needs to be done after the answer is checked — this code must occur sometime after session_start() has again been called.

```php
if ($answer == $guess) {
    $msg = "Well done, you got that one!";
    $_SESSION['numCorrect']++;
} else {
    $msg = "Whoops... not quite.";
}
$_SESSION['numQuestions']++;```
Some user feedback

Inside the HTML part of the script we add another paragraph

<p>You have answered <strong><?=$_SESSION['numCorrect']?> </strong> correctly out of <strong><?=$_SESSION['numQuestions']?> </strong>.</p>

(GF Royle 2006-8, N Spadaccini 2008) HTML/PHP III