LAMP, MySQL/PHP Database Driven Websites - Part III

Navigate: PHP Tutorials > MySQL with PHP

Author: Gizmola
Date: 04/26/2005
Version 1.0
Experience Level: Intermediate

This is Part 3 of a 3 Part Series.
Part 1 of the series is here.
Part 2 of the series is here.

Joining Tables

Remember that one of our goals is to handle the many to many data we need to establish the relationship between Games and Platforms. This is resolved in the GamePlatform table. What this table allows, is for the GameStatus system to indicate when a game is available on multiple platforms. (Refer again to the Entity Relationship Diagram). In order to makes things work as planned, we need two arrays: the array of *PlatformSeq*'s from GamePlatform for the specific Game we're adding or editting, and the list of *PlatformSeq*'s from the Platform table, that *aren't already in GamePlatform*. In other words, any Platforms (PS/2, XBox) that the game wasn't released on.

An easy way to think about this is to think of the entire list of Platforms as a hat. Inside the hat there is one piece of paper for each platform with the name of that platform written on it. If you reach into the hat and remove the piece of paper that says "XBox" and put it on your desk, the set of "Platforms" in the hat, is now "All platforms *except for XBox*

.

There are many different ways at the database level we can determine with SQL what Platforms are included in the GamePlatform list, and which are not. I'm going to use a single query that will return me a result set I can use to fill both arrays. You might be thinking "isn't this a *chicken and egg* problem? We haven't even added a Game to the

GameStatus database, so how could we have any GamePlatform rows? What we'll attempt to do, is create a single query that works for our purposes no matter whether we're adding a new row, or editing an existing one. The important thing is that we will get a list of *all available Platforms* and this list will also indicate whether or not a particular Platform is already in a GamePlatform row for a particular Game. At that point all we have to do is go through the result set, and add to the appropriate (selected or unselected) array.

When you do a standard inner join in mysql (or any relational database for that matter) you will get a row whenever the two joined columns are able to match values. If there isn't a match on that column between at least one row in each of the joined tables, you will not get any rows in your result set.

Mysql can also do an **outer join** aka **left (outer) join** which unlike the inner join, will return a row for every row that exists in the the FROM table, whether or not a corresponding row exists in the table that it is LEFT OUTER JOINed to.

Refer back to the data model, and consider the purpose of the GamePlatform table. For a Game, if there's a version available for a Platform, there will be a row in GamePlatform with the corresponding GameSeq and corresponding PlatformSeq of that Platform. As discussed, the total universe of Platforms is what you get from *SELECT * FROM Platform*.

We know going into this operation, which Game (and Game.GameSeq) we are interested in. So the exercise is to JOIN Platform to GamePlatform so that we get a list of ALL the platforms, but also with an indication of which Platforms exist for our Game. How will we know which Platforms have a release for our Game? When the tables are outer joined, GamePlatform.GameSeq will be equal to our GameSeq. *For Platforms where the Game was not released*, **GamePlatform.GameSeq will be NULL!**

Let's build our query step by step. Our first decision is which table to start with. We know we need to start with the table that has all the Platforms -- namely Platform. In our result set we need Platform.PlatformSeq and Platform.Name so we can fill our chooser with all the available values. We also need GamePlatform.GameSeq, so we can check whether or not the value is NULL. So let's start with what we know:

PHP Example: (!)

```
SELECT Platform.PlatformSeq, Platform.Name, GamePlatform.GameSeq
FROM Platform
```

You should be familiar with this type of query by now, although it should be obvious that we have an issue with the FROM section. We've listed Platform but not GamePlatform. This is where our LEFT OUTER JOIN comes in. We're going to OUTER JOIN to the table on the LEFT hand side of our query, namely platform. Again, in doing this, we're guaranteed to get a row for every row in Platform. So now we can add our LEFT OUTER JOIN.

PHP Example: (!)

```
SELECT Platform.PlatformSeq, Platform.Name, GamePlatform.GameSeq
FROM Platform
LEFT OUTER JOIN GamePlatform
```

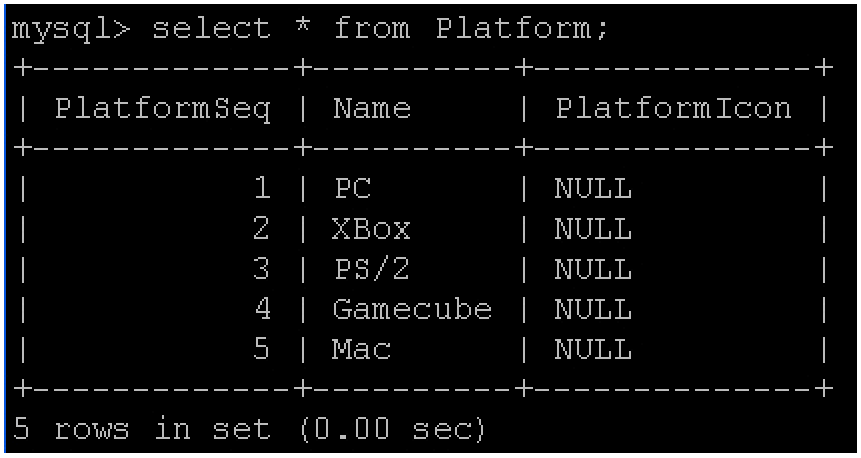
Now we've provided enough information to resolve both the tables we're dealing with. We still need to specify which columns to join ON. Again referencing the model, it should be clear that we need to join on PlatformSeq. So we add this to the query.

PHP Example: (!)

```
SELECT Platform.PlatformSeq, Platform.Name, GamePlatform.GameSeq
FROM Platform
LEFT OUTER JOIN GamePlatform
ON Platform.PlatformSeq = GamePlatform.PlatformSeq;
```

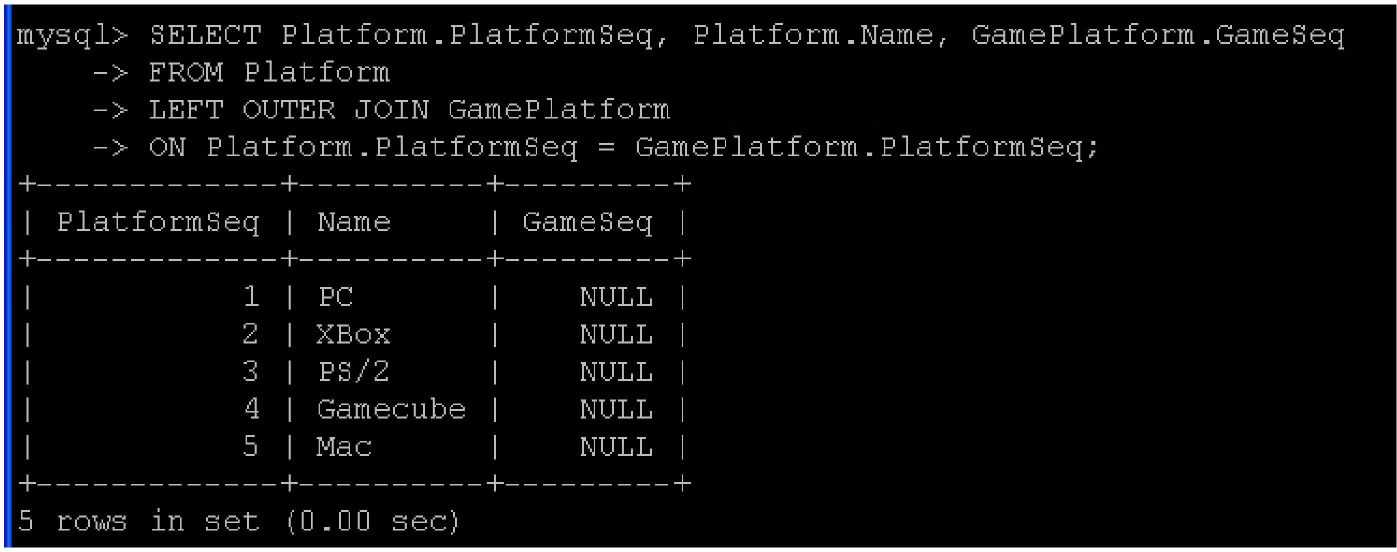
Exploring the Outer Join

If our Platform table contains this:



```
mysql> select * from Platform;
+-------------+----------+--------------+
| PlatformSeq | Name     | PlatformIcon |
+-------------+----------+--------------+
|           1 | PC       | NULL         |
|           2 | XBox     | NULL         |
|           3 | PS/2     | NULL         |
|           4 | Gamecube | NULL         |
|           5 | Mac      | NULL         |
+-------------+----------+--------------+
5 rows in set (0.00 sec)
```

Our query will return exactly what we wanted -- a row for every platform, even though the GamePlatform table is empty.


```
mysql> SELECT Platform.PlatformSeq, Platform.Name, GamePlatform.GameSeq
    -> FROM Platform
    -> LEFT OUTER JOIN GamePlatform
    -> ON Platform.PlatformSeq = GamePlatform.PlatformSeq;
+-------------+----------+---------+
| PlatformSeq | Name     | GameSeq |
+-------------+----------+---------+
|           1 | PC       |    NULL |
|           2 | XBox     |    NULL |
|           3 | PS/2     |    NULL |
|           4 | Gamecube |    NULL |
|           5 | Mac      |    NULL |
+-------------+----------+---------+
5 rows in set (0.00 sec)
```

We still have an issue however. The GamePlatform table is not going to be empty for long. Once we start entering Games, and indicating Platforms we're going to have rows in it. What will happen to our LEFT OUTER JOIN? Let's put some test rows into GamePlatform and try it out.

Let's pretend we have 2 Games (GameSeq = 1 and GameSeq = 2). For GameSeq = 1, let's pretend that it was released on Platform 4. To do our test, we'll insert a row into GamePlatform. Although we have no Game in the Game table with GameSeq 1, what's important is that we already have populated our Platform Table with rows that represent the available Game Platforms. A sample set of data is provided here, although ideally you would have made a simple set of admin scripts modelled upon the ones we created in Part I, for Developer. With Platforms to work with we can test out our assumptions about outer joins.

Here's an INSERT (can be done from command line mysql or phpMyAdmin) so we can test.

PHP Example: (!)
```
INSERT INTO GamePlatform (GameSeq, PlatformSeq) VALUES (1, 4);
```

Now when we execute our LEFT OUTER JOIN query, we notice something interesting about GamePlatform.GameSeq. It is no longer NULL for all the rows. Note that for the GameCube row (PlatformSeq #4) we now see a GameSeq of 1. Let's continue with our test and add 2 more rows, this time for hypothetical Game #2. This time we'll assume releases on platform 3 AND 4.

PHP Example: (!)
```
INSERT INTO GamePlatform (GameSeq, PlatformSeq) VALUES (2, 3),(2,4);
```

Now let's execute our LEFT OUTER JOIN Query again.

Notice that not only do we have now have non NULL GameSeq values, but we also have 2 Rows for the GameCube, indicating that for both Game 1 and 2, they were released on the GameCube. This is not what what want for our chooser. What we're missing is an additional element that will refine the query so that we only get rows for the Game we're interested in. Initially many people would be tempted to add a WHERE clause to the query. In this type of query, a WHERE clause operates on the existing result set. So if for example we tried this:

Adding a WHERE isn't quite right

PHP Example: (!)

```
SELECT Platform.PlatformSeq, Platform.Name, GamePlatform.GameSeq
FROM Platform
LEFT OUTER JOIN GamePlatform
ON Platform.PlatformSeq = GamePlatform.PlatformSeq
WHERE GamePlatform.GameSeq = 1;
```

Now we only get one row back -- not at all what we need. Again this is because, the result set was already determined by the JOIN, and the WHERE clause could only further constrain that. Once we add WHERE GamePlatform.GameSeq =1 ALL we get rows that have GameSeq =1, and we lose the NULL GameSeq's which are the point of the entire exercise. What we have to accomplish is to add to the query after the LEFT OUTER JOIN condition using AND so that it is taken into consideration by MySQL at the same time that it is determining how to handle the LEFT OUTER JOIN.

This is what we actually need

PHP Example: (!)

```
SELECT Platform.PlatformSeq, Platform.Name, GamePlatform.GameSeq
FROM Platform
LEFT OUTER JOIN GamePlatform
ON Platform.PlatformSeq = GamePlatform.PlatformSeq
AND GamePlatform.GameSeq = 1;
```

Adding Aliases & Integrating the LEFT OUTER JOIN

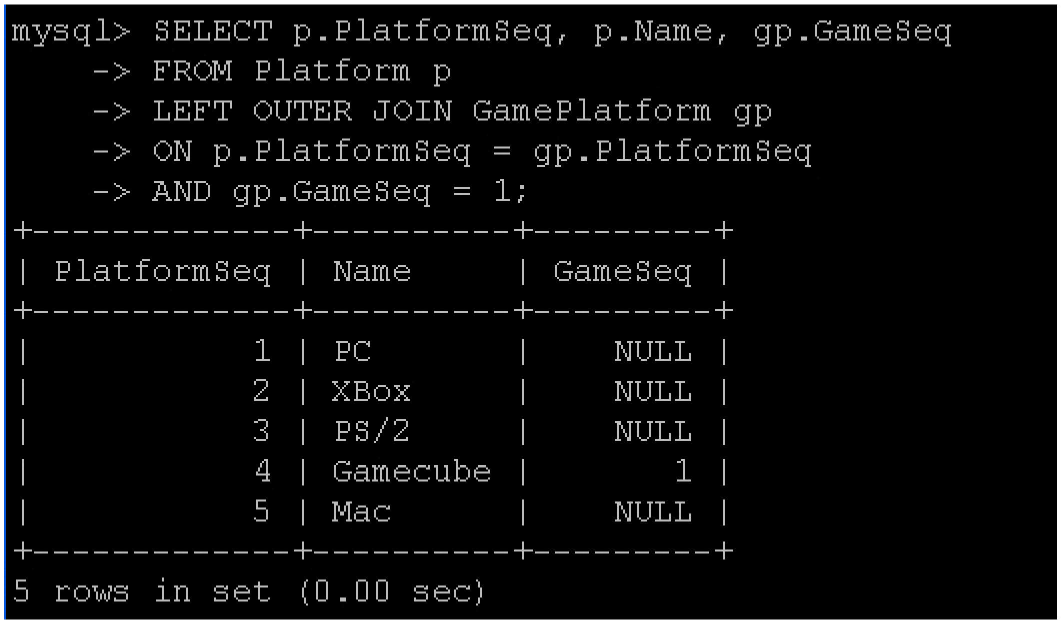
We have arrived finally at our ultimate query. Even though we may have hundreds of games and gameplatform rows, for any one game, we can determine all the platforms it was released on as well as those platforms it was not released on, in one query, even if the game has not yet been released on ANY platforms. You can try this for yourself by introducing an arbitrary GameSeq of your liking -- for example, try 132. You'll notice that you correctly get back a result set where all the GameSeq's are NULL.

Before we move on, I'm going to refactor this to use SQL aliases, because that's what I prefer. A table alias is basically an abbreviation or substitute for using the entire table name. You certainly don't have to use them, but I like to save myself some typing with complicated queries. I tend to use one or two character aliases that reflect the table names. In this case, I'm going to use p for Platform and gp for GamePlatform.

PHP Example: (!)

```
SELECT p.PlatformSeq, p.Name, gp.GameSeq
FROM Platform p
LEFT OUTER JOIN GamePlatform gp
ON p.PlatformSeq = gp.PlatformSeq
AND gp.GameSeq = 1;
```

As you can see, I can add an alias immediately following the first use of the table name, be that in the FROM or after the LEFT OUTER JOIN. Wherever I would have to qualify a column name using a table name, I'm now free to substitute my alias instead. This query is exactly the same as far as MySQL is concerned to the query noted above, and our output with all the previous assumptions about our test data should match what you see here:



```
mysql> SELECT p.PlatformSeq, p.Name, gp.GameSeq
    -> FROM Platform p
    -> LEFT OUTER JOIN GamePlatform gp
    -> ON p.PlatformSeq = gp.PlatformSeq
    -> AND gp.GameSeq = 1;
+-------------+----------+---------+
| PlatformSeq | Name     | GameSeq |
+-------------+----------+---------+
|           1 | PC       |    NULL |
|           2 | XBox     |    NULL |
|           3 | PS/2     |    NULL |
|           4 | Gamecube |       1 |
|           5 | Mac      |    NULL |
+-------------+----------+---------+
5 rows in set (0.00 sec)
```

Now that the heavy lifting of handling our many to many relationship has been taken care of for us by mysql and the LEFT OUTER JOIN, we are left with the task of taking the result set we now understand how to create, and figuring out how to apply it to the chooser. Just as a bit of housekeeping for those following along, make sure you TRUNCATE TABLE GamePlatform; so that our test rows are removed. We're going to add in the chooser to gamedetail.php and then add the ability to update the database once we add our gamedetailpost script.

Let's refactor once again, by introducing our latest changes to the gamedetail.php script.

gamedetail3.php

PHP Example: (!)

```
<?php

$m = $_GET["mode"];
$sort = $_GET["sort"];
$seq = $_GET["seq"];
$s = $_GET["status"];

include("adminheader.php");
require("config.php");
require("common.function.php");
require("chooser.class.php");

$dbh1 = mysql_connect($dbhost, $dbuser, $dbpasswd) or die("Could not
connect");
mysql_select_db($dbname,$dbh1) or die("Could not select database");
if ($m == 'edit') {
    $q1 = "SELECT * FROM Game where GameSeq=$seq";
  $rslt1 = mysql_query($q1, $dbh1) or die("Query failed");
    $row1 = mysql_fetch_assoc($rslt1);
} else {
    $seq = 0;
}
// Get Our Chooser Data
$q = "SELECT p.PlatformSeq, p.Name, gp.GameSeq
                FROM Platform p
                LEFT OUTER JOIN GamePlatform gp
                ON p.PlatformSeq = gp.PlatformSeq
                AND gp.GameSeq = $seq";

$r_selected = array();
$r_unselected = array();
$rslt1 = mysql_query($q, $dbh1) or die("Query failed");
//
// Load up selected and unselected for chooser
//
while ($row = mysql_fetch_assoc($rslt1)) {
  if ($row['GameSeq'] > 0) {
        $r_selected[$row['Name']] = $row['PlatformSeq'];
    } else {
        $r_unselected[$row['Name']] = $row['PlatformSeq'];
    }
}
/* Free resultset */
```

```
mysql_free_result($rslt1);
/* Closing connection */

// Make our chooser
$Chooser = new selectChooser($r_unselected, $r_selected);
$Chooser->outputjavascript();


$backurl = 'gamelist.php';
$posturl = 'gamepost.php';
$addurl = 'gamedetail.php?mode=add';

if (isset($sort)) {
    $backurl .= "?sort=$sort";
    $posturl .= "?sort=$sort";
    $addurl .= "&sort=$sort";
}

echo "<a href=$backurl>Back</a>  <a href=$addurl>Add New
Game</a><br><br>";
echo "<form name="form1" method="POST" action="$posturl">";
echo "<input type="hidden" name="seq" value="$seq">";
echo '<table>';
echo '<tr><th colspan="2">Game Detail</th></tr>';
echo "<tr><td>Seq:</td><td>$seq</td></tr>";
echo '<tr><td>Title:</td><td><input type="text" name="title" size="80"
maxlength="80"';
if ($m == 'edit') {
    echo " value='".htmlentities($row1['Title'],ENT_QUOTES)."'";
}
echo '></td></tr>';
echo '<tr><td>Status:</td><td>';
if ($m == 'edit') {
    echo makelookup('Status', 'StatusCode', 'Description',
$row1['StatusCode']);
} else {
    echo makelookup('Status', 'StatusCode', 'Description');
}
echo '</td></tr>';

echo '<tr><td>Genre:</td><td>';
if ($m == 'edit') {
    echo makelookup('Genre', 'GenreSeq', 'Name', $row1['GenreSeq']);
} else {
    echo makelookup('Genre', 'GenreSeq', 'Name');
}
echo '</td></tr>';

echo '<tr><td>Developer:</td><td>';
if ($m == 'edit') {
    echo makelookup('Developer', 'DeveloperSeq', 'Name',
$row1['DeveloperSeq']);
} else {
    echo makelookup('Developer', 'DeveloperSeq', 'Name');
}
echo '</td></tr>';
echo '<tr><td>Publisher:</td><td>';
if ($m == 'edit') {
    echo makelookup('Publisher', 'PublisherSeq', 'Name',
$row1['PublisherSeq']);
} else {
    echo makelookup('Publisher', 'PublisherSeq', 'Name');
}
echo '</td></tr>';

echo '<tr><td>Website:</td><td><input type="text" name="website"
size="80" maxlength="255"';
if ($m == 'edit') {
    echo " value='".htmlentities($row1['Website'],ENT_QUOTES)."'";
}
echo "></td></tr>";

echo '<tr><td>Description:</td><td><textarea name="description"
rows="4" cols="80">';
if ($m == 'edit') {
    echo htmlentities($row1['Description'],ENT_QUOTES);
}
echo "</textarea></td></tr>";

echo '<tr><td>Review:</td><td><textarea name="review" rows="4"
cols="80">';
if ($m == 'edit') {
    echo htmlentities($row1['Review'],ENT_QUOTES);
}
echo "</textarea></td></tr>";

echo '<tr><td>Platforms:</td><td>' . $Chooser->outputChooser() .
'</td></tr>';

echo "<tr><td colspan=2 align=center><input type='submit'
name='SubmitFrm' value='Save'></td></tr>";
echo "</table>";
echo "</form>";
if (isset($s)) {
    echo "Edit Status: $s <br>";
}
if ($m == 'edit') {
    $addUrl = "gamedetail.php?mode=add&seq=$seq";
    if (isset($s))
        $addUrl .= "&sort=$s";
}
?>
```

If you have a diff tool available you might want to fire it up and compare the differences between the prior version without the chooser (gamedetail2.php) and new one (gamedetail3.php) with it. You'll find that the changes are relatively minimal.

The first thing we have to add is our chooser class. The script won't work without it, so we'll require it:

PHP Example: (!)

```
require("chooser.class.php");
```

The next issue we face is that instead of doing a single query to get only the Game Row, we also need to fill our Chooser arrays indicating the selected and unselected items. We're already doing a number of queries on the page, simply because we put the loading of our lookup tables into a function that manages those details for us. We could do the same with our LEFT OUTER JOIN query, but in the interest of keeping things simple, we'll just reuse the database connection that we were only creating in edit mode to read in an existing Game. We'll use that same connection now to load the chooser data and load the Game if we need to. In order to do this we need to move a couple of lines out of the edit mode condition, so that we make the database connection no matter whether we're editing or not.

The old code

PHP Example: (!)

```
if ($m == 'edit') {
    $dbh1 = mysql_connect($dbhost, $dbuser, $dbpasswd) or die("Could
not connect");
    mysql_select_db($dbname,$dbh1) or die("Could not select database");
    $q1 = "SELECT FROM Game where GameSeq=$seq";
  $rslt1 = mysql_query($q1, $dbh1) or die("Query failed");
    $row1 = mysql_fetch_assoc($rslt1);
  /* Free resultset */
  mysql_free_result($rslt1);
} else {
    $seq = 0;
}
/* Closing connection */
```

The new code

PHP Example: (!)

```
$dbh1 = mysql_connect($dbhost, $dbuser, $dbpasswd) or die("Could not
connect");
mysql_select_db($dbname,$dbh1) or die("Could not select database");
if ($m == 'edit') {
    $q1 = "SELECT * FROM Game where GameSeq=$seq";
  $rslt1 = mysql_query($q1, $dbh1) or die("Query failed");
    $row1 = mysql_fetch_assoc($rslt1);
} else {
    $seq = 0;
}
// Get Our Chooser Data
$q = "SELECT p.PlatformSeq, p.Name, gp.GameSeq
                FROM Platform p
                LEFT OUTER JOIN GamePlatform gp
                ON p.PlatformSeq = gp.PlatformSeq
                AND gp.GameSeq = $seq";

$r_selected = array();
$r_unselected = array();
$rslt1 = mysql_query($q, $dbh1) or die("Query failed");
```

We've also added our LEFT OUTER JOIN query which now takes into account the $seq of the Game. The seq gets passed via a url parameter on edit, or is set to 0 if this is a new row. We also declare the two arrays which we will pass to our chooser. Once we execute the LEFT OUTER JOIN query, all we have left to do is load the arrays. We do that by adding this small block of code which fetches each row from the result set returned from the LEFT OUTER JOIN of Platform to GamePlatform.

PHP Example: (!)

```
/
// Load up selected and unselected for chooser
//
while ($row = mysql_fetch_assoc($rslt1)) {
  if ($row['GameSeq'] > 0) {
        $r_selected[$row['Name']] = $row['PlatformSeq'];
    } else {
        $r_unselected[$row['Name']] = $row['PlatformSeq'];
    }
}
```

As you can see, if GameSeq is > 0, which means that the row was NOT NULL, we know that this is a Platform that the game was released on. In that case we set the name and value pair in our **$r_selected** array for the chooser. If not we know we set $r_unselected.

Instantiating a Chooser

Now we are ready to create the chooser object using the two arrays. Once we've created a chooser object we then us it to output the javascript code the chooser requires.

PHP Example: (!)

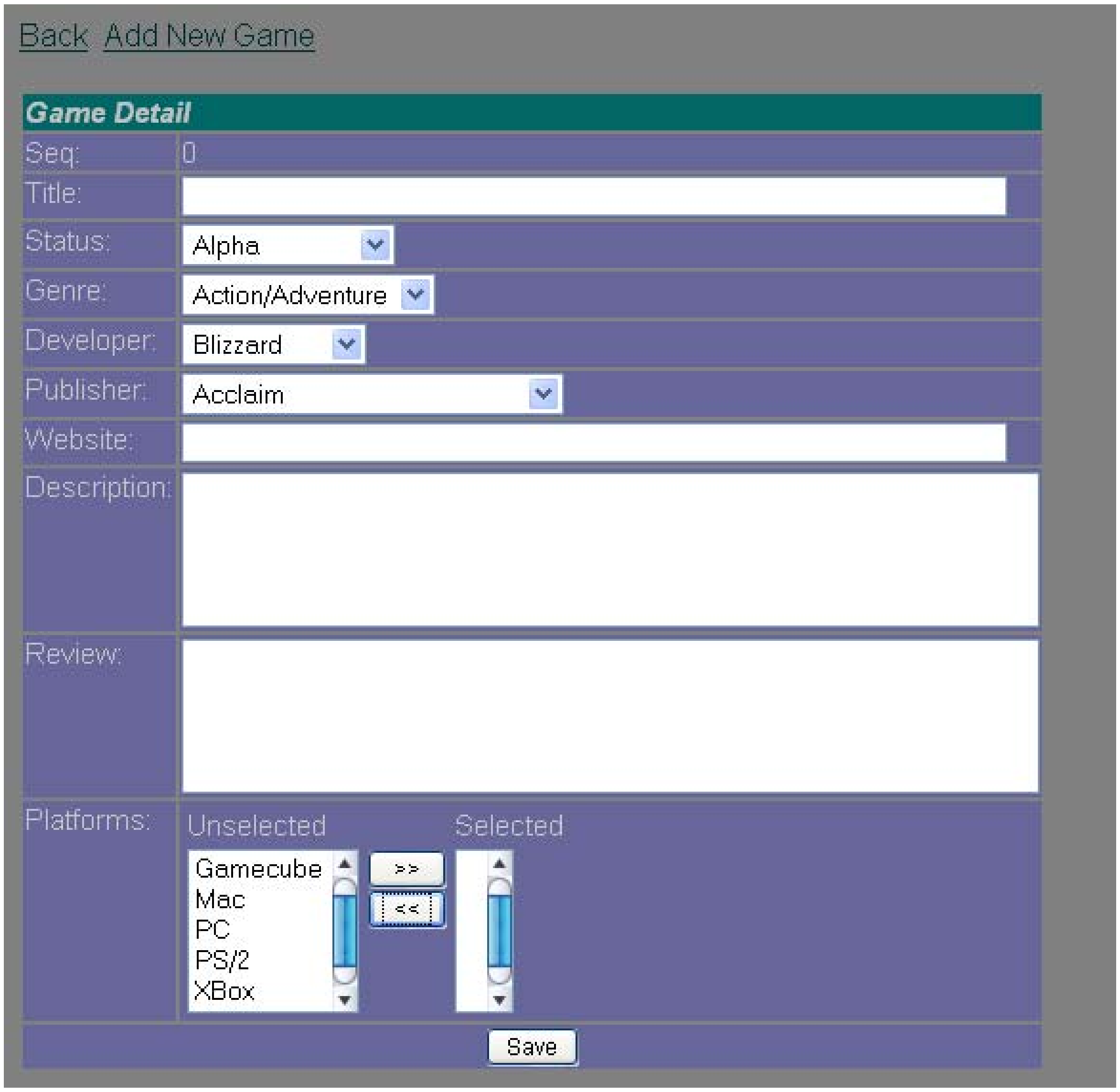
```
// Make our chooser
$Chooser = new selectChooser($r_unselected, $r_selected);
$Chooser->outputjavascript();
```

Last but not least, we need to add the Chooser html into the form. I chose to do this at the bottom of the form, just before the submit button. Again the Chooser object does all the work for us, so that our only real concern is where in the form to place it.Adding a table row with a prompt and a call to the outputChooser() method is all we need.

PHP Example: (!)

```
echo '<tr><td>Platforms:</td><td>' . $Chooser->outputChooser() .
'</td></tr>';
```

If you run the script at this point you should now have a fully functional chooser filled with the values from the Platform table that you can play with.


Back Add New Game

Game Detail

Seq:	0
Title:	
Status:	Alpha
Genre:	Action/Adventure
Developer:	Blizzard
Publisher:	Acclaim
Website:	
Description:	
Review:	
Platforms:	Unselected: Gamecube, Mac, PC, PS/2, XBox; Selected:

Handling the POST

None of this is particularly useful if we can't actually add new rows or save edits. So our next test is to write our post script. If you look at the form code, you'll note that this script should be called gamepost.php. The basic structure of gamepost.php is the same as that of developerpost.php from part 1. The framework of the file and our strategy is not going to change, so the best thing to do is copy all the code from developerpost.php into gamepost.php. If you refer back to that script, you'll find that there's 3 basic situations gamepost needs to handle.

1. We're deleting a game (mode == 'delete', $delseq = GameSeq to delete)
2. We're editting an existing game (gameseq > 0)
3. It's a new game (gameseq == 0)

The heart of the post is an *if* -- *elseif* -- *else* that handles these 3 cases. So essentially we can use the same logic as was used in the simpler developerpost.php script by keeping the basic structure and replacing the developer specific code with code that supports the Game table.

But before we go too far, it's good practice to test the gamedetail3.php form out, by putting in some test data and seeing what we get in our $_POST. There are various ways to do this including using the print_r function or using a foreach, but we're going to provide a function that will give us the same sort of output as print_r, but in an easier to understand format.

Some of the elements in the $_POST are going to be arrays, so it's not enough to simply foreach through $_POST if we want to know what we're getting. Although it's not something we'll encounter in our $_POST, it is conceptually possible for a php array to contain many nested levels of arrays. To handle this possibility we'll use recursion in our function. It's not important to the overall goal of this tutorial that you understand recursion, but for the sake of completeness I wanted to point out that I used it in this function

function dbgpost

PHP Example: (!)

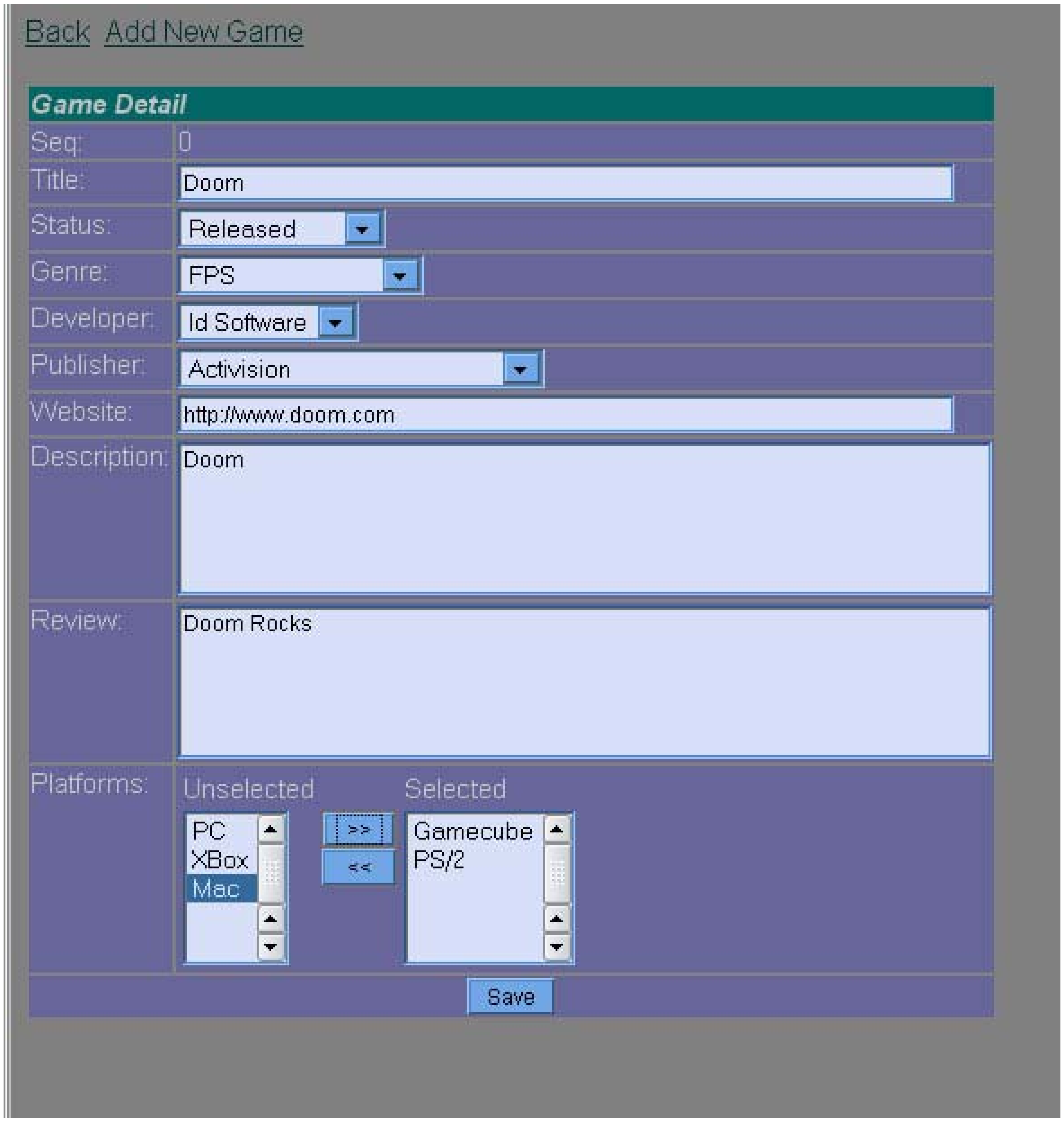
```
function dbgpost($post) {
    foreach($post as $key => $value) {
      if (is_array($value)) {
          echo "--$key (array)---start--<br />n";
            dbgpost($value);
            echo "--$key (array)---end----<br />n";
        } else {
            echo "$key: $value <br>n";
        }
    }
}
```

Note that dbgpost calls itself in the case that an element in the array it is echoing also turns out to be an array. In using this technique we can be sure that no matter how many nested arrays are part of the array being output, this function should be able to handle it. This would be a good function to add to our common functions include file, but for now we'll simply add it to the top of the gamepost.php file. Now we can add a quick call to it, and die()

PHP Example: (!)

```
// For debugging
dbgpost($_POST);
die();
```

Now we can run the gamedetail3.php script and actually submit the form, and see what $_POST data we have to work with, paying special attention to what the chooser does for us.



Game Detail	
Seq:	0
Title:	Doom
Status:	Released
Genre:	FPS
Developer:	Id Software
Publisher:	Activision
Website:	http://www.doom.com
Description:	Doom
Review:	Doom Rocks
Platforms:	Unselected: PC, XBox, Mac; Selected: Gamecube, PS/2

Notice that we used the chooser to indicate that Doom was released on the GameCube and PS/2 platforms. When we submit the form our dbgpost() output shows us this:

PHP Example: (!)

```
seq: 0
title: Doom
Status: R
Genre: 3
Developer: 21
Publisher: 5
website: http://www.doom.com
description: Doom
review: Doom Rocks
```

```
--list_u (array)---start--
0: 5
--list_u (array)---end----
SubmitFrm: Save
```

Immediately you should notice a few things. First off, our drop down lists are working as intended, namely providing the keys we'll need to save the appropriate foreign keys we need, be they integers or codes (as in the case of Status).

Select Multiple Issues

Something is not quite right with the chooser. We got an array called list_u which contained one item with a value of 5. list_u is our unselected items list, which is if no interest to us, and even that array doesn't reflect the full set of unselected platforms. What gives?

To understand this, you have to understand the nature of the underlying html form element. The select returns only items that are SELECTED. Look again at the screen shot of the form as it was posted, and you'll note that while our Chooser did move items back and forth, the only item that was selected was the "Mac" platform in the unselected box. Mac happens to have a PlatformSeq of 5, and this is why it comes over in the list_u array in the $_POST.

Unfortunately we need to trick our form into checking list_s, and setting any Platforms it contains to selected, The chooser provides a javascript function to do this, but we have to manually modify our form so it calls this function when the form is submitted, otherwise we'll never get the list of selected items we need.

The reason I started with the strategy of using my dbgpost() function was to illustrate the type of thinking you need to employ when you're doing anything tricky or complicated, or in general are trying to debug problems in your scripts. Have some tools in your tool kit so you can find out what you're working with and track down issues. By testing our form, we were able to identify problems before we'd written a lot of code based on our assumptions of what should work. It's not a great idea to write a lot of code based on our assumption of how things work, only to find that our gamepost.php script wouldn't work as expected because unbeknownst to us, our assumptions were wrong.

As it happens, the Chooser provides a javascript function we need, named SelectListAll(). It requires the name of the selected items object in the form. We will need to call this function when the form is submitted, using the onSubmit event hook, so that it will be executed just before the script is posted. The tricky part is knowing exactly how to specify this using the browser Document Object Model. By default the chooser class names the selected list **list_s[]**. So we can get javascript to pass it to our function by using the onSubmit event:

PHP Example: (!)

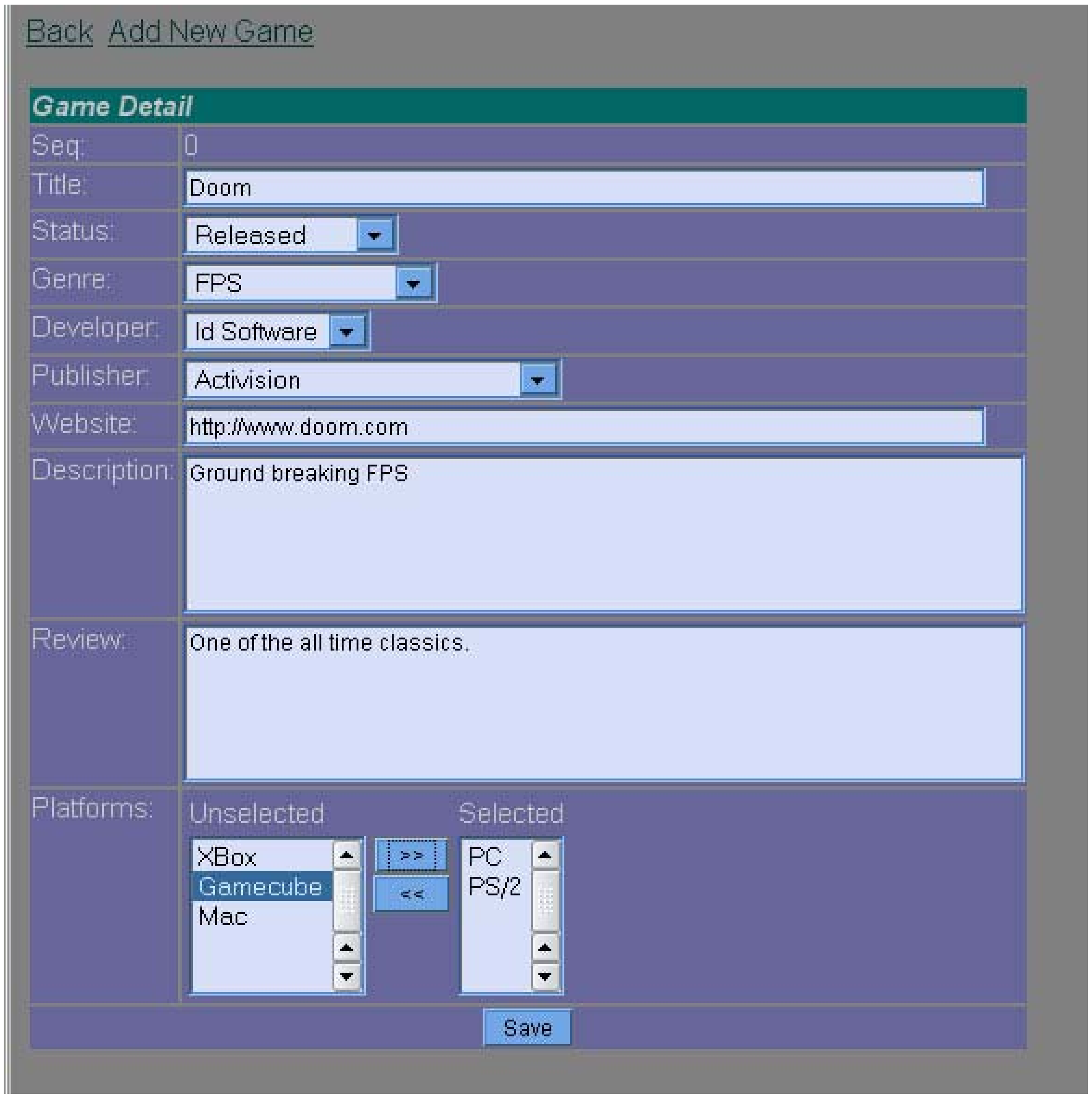
```
onSubmit="SelectListAll(document.forms[0].elements['list_s[]'])"
```

Although it's not the easiest thing to read, we can embed this in an interpolated php string by changing the line that echos our form so that it now reads like this:

PHP Example: (!)

```
/code>]
echo "<form name="form1"
onSubmit="SelectListAll(document.forms[0].elements['list_s[]'])"
method="POST" action="$posturl">";
```

If you're following along, although it's only a one line change, I've saved this as gamedetail4.php. To break this change down, the javascript DOM starts with document (your browser instance) and from there includes an array of forms. Because we only have one form, the one we're interested in is forms[0]. From there javascript allows us to access form objects by name using the elements[] array, so we can access the select we want using **document.forms[0].elements["list_s[]"]**. This is all that SelectListAll needs to work. SelectListAll, like its name implies, simply goes through and makes sure that every item in the list is **selected** just before the form is submitted. This insures that we will get the PlatformSeq numbers of those items in the list_s[] array.



Back Add New Game

Game Detail

Seq:	0
Title:	Doom
Status:	Released
Genre:	FPS
Developer:	Id Software
Publisher:	Activision
Website:	http://www.doom.com
Description:	Ground breaking FPS
Review:	One of the all time classics.
Platforms:	Unselected: XBox, Gamecube, Mac — Selected: PC, PS/2

Retesting with SelectListAll installed

Let's run this script and see what our debug function in the gamepost.php shows us.

PHP Example: (!)

```
seq: 0
title: Doom
Status: R
Genre: 3
Developer: 21
Publisher: 5
```

```
website: http://www.doom.com
description: Ground breaking FPS
review: One of the all time classics.
--list_u (array)---start--
0: 4
--list_u (array)---end----
--list_s (array)---start--
0: 1
1: 3
--list_s (array)---end----
SubmitFrm: Save
```

Now that the SelectListAll function is executing, our $_POST includes the list_s array, and as we can see, it properly contains 2 items corresponding to the PlatformSeq's of the Platforms which were indicated as released through the chooser. All that's left is making use of this. Let's start by making local variables from the $_POST variables in the same way we made local variables in developerpost.php. Our test output provides us a nice roadmap for what we need.

We'll also want to comment out the debug function so that we fall through to our posting logic. We want to do two things now. Since we're approaching the finish line, we want to make sure that our work in progress script (gamedetail4.php) is also saved as gamedetail.php. In other words, we want gamedetail.php to be the same as gamedetail4.php. If you're using the scripts provided, you will find that gamedetail.php and gamedetail4.php are in fact the same. This is important because gamepost.php needs to return to a script, and we want that script to be named gamedetail.php. At this point our scripts are complete, so I'd suggest you refer to gamedetail.php, and use that for any further testing you do.

gamepost.php

PHP Example: (!)

```php
<?php
    function dbgpost($post) {
        foreach($post as $key => $value) {
          if (is_array($value)) {
              echo "--$key (array)---start--<br />n";
                dbgpost($value);
                echo "--$key (array)---end----<br />n";
            } else {
                echo "$key: $value <br>n";
            }
        }
    }

    //For debugging
    //dbgpost($_POST);
    //die();

    require("config.php");
    function update_gameplatform($gameseq, $r_selected) {
        global
            $dbhost,
            $dbname,
            $dbuser,
            $dbpasswd;


        $dbh = mysql_connect($dbhost, $dbuser, $dbpasswd) or die("Could
not connect");
        mysql_select_db($dbname,$dbh) or die("Could not select
database");
        //
        // Start by getting rid of the old selections
        //
        $sql = "DELETE FROM GamePlatform WHERE GameSeq = $gameseq";
        $rslt = mysql_query($sql, $dbh) or die("Query failed:
GamePlatform Delete.");
        //
        // Now Add new ones, if needed
        //
        if (count($r_selected) > 0) {
            $sql = 'INSERT INTO GamePlatform (GameSeq, PlatformSeq)
VALUES ';
            foreach ($r_selected as $value) {
                $sql .= "($gameseq, $value),";
            }
            // Get rid of last unneeded ','
            $sql = substr($sql, 0, -1);
            $rslt = mysql_query($sql, $dbh) or die("Query failed:
GamePlatform Adds");
        }
    }

    $sort = $HTTP_GET_VARS["sort"];
    $mode = $HTTP_GET_VARS["mode"];
    $delseq = $HTTP_GET_VARS["delseq"];
    $seq = $_POST['seq'];

    $title = $_POST['title'];
    $statuscode = $_POST['Status'];
    $genre = $_POST['Genre'];
    $developer = $_POST['Developer'];
    $publisher = $_POST['Publisher'];
    $website = $_POST['website'];
    $description = $_POST['description'];
    $review = $_POST['review'];
    $r_selected = $_POST['list_s'];

    $dbh1 = mysql_connect($dbhost, $dbuser, $dbpasswd) or die("Could
not connect");
    mysql_select_db($dbname,$dbh1) or die("Could not select database");

    if (isset($mode) && ($mode == 'delete') && ($delseq > 0)) {
        $status = "Deleted";
        $q1 = "DELETE FROM GamePlatform WHERE GameSeq = $delseq";
        $rslt1 = mysql_query($q1, $dbh1) or die("Query failed");
        $q1 = "DELETE FROM Game WHERE GameSeq = $delseq";
        $rslt1 = mysql_query($q1, $dbh1) or die("Query failed");
    } elseif ($seq > 0) {
        $status = "Saved";
        $sql = "UPDATE Game
                        SET Title='$title',
                        GenreSeq = $genre,
                        PublisherSeq = $publisher,
                        DeveloperSeq = $developer,
                        StatusCode = '$statuscode',
                        Website = '$website',
                        Description = '$description',
                        Review = '$review'
                        WHERE GameSeq = $seq";

        $rslt1 = mysql_query($sql, $dbh1) or die("Query failed, Game
Update");
        // Now add the GamePlatforms
        update_gameplatform($seq, $r_selected);

    } else {
        if ($seq == 0) {
            $status = "Added";
            $q1 = "INSERT INTO Game
                  (Title, GenreSeq, PublisherSeq,
                         DeveloperSeq, StatusCode, Website,
                         Description, Review)
                         VALUES
                         ('$title', $genre, $publisher,
                          $developer, '$statuscode', '$website',
                            '$description', '$review')";
            $rslt1 = mysql_query($q1, $dbh1) or die("Query failed:
Doing Game Insert");
             $seq = mysql_insert_id($dbh1);
            update_gameplatform($seq, $r_selected);
        }
    }


    if ($mode == 'delete') {
        $returnurl = "gamelist.php";
        if (isset($sort))
            $returnurl .= "?sort=$sort";
    } else {
           $returnurl =
"gamedetail.php?mode=edit&seq=".$seq."&status=".$status;
        if (isset($sort))
            $returnurl .= "&sort=$sort";
    }

  mysql_close($dbh1);
    header("Location: $returnurl");
?>
```

Here's our completed gamepost.php. It follows the same basic logic used in developerpost.php, although we have had to add some code to handle storing the many to many data for gameplatform.

We've chosen to put that code into a function. One of the best reasons to do this, is that the same code needs to be called in two different circumstances -- in the case that we're editting an existing Game, or we're Adding a new game. We call this function update_gameplatform().

function update_gameplatform

PHP Example: (!)

```
function update_gameplatform($gameseq, $r_selected) {
        global
            $dbhost,
            $dbname,
            $dbuser,
            $dbpasswd;


        $dbh = mysql_connect($dbhost, $dbuser, $dbpasswd) or die("Could
not connect");
        mysql_select_db($dbname,$dbh) or die("Could not select
database");
        //
        // Start by getting rid of the old selections
        //
        $sql = "DELETE FROM GamePlatform WHERE GameSeq = $gameseq";
        $rslt = mysql_query($sql, $dbh) or die("Query failed:
GamePlatform Delete.");
        //
        // Now Add new ones, if needed
        //
        if (count($r_selected) > 0) {
            $sql = 'INSERT INTO GamePlatform (GameSeq, PlatformSeq)
VALUES ';
            foreach ($r_selected as $value) {
                $sql .= "($gameseq, $value),";
            }
            // Get rid of last unneeded ','
            $sql = substr($sql, 0, -1);
            $rslt = mysql_query($sql, $dbh) or die("Query failed:
GamePlatform Adds");
        }
    }
```

Notice that the function takes two parameters, the gameseq of the game in question, and the array of selected items, with the assumption that this array contains platformseqs. Because there is no data that relates to GamePlatform, we employ a strategy for simplicity sake, and assume that each time we call this function, we will first delete any existing GamePlatform rows for the Game we're updating. In many cases I find this to be an acceptable approach, especially when the universe of possible values is relatively small as it is in the case of Platform.

Hopefully at this juncture, it's clear to you that the deletion is accomplished by these lines:

PHP Example: (!)

```
    //
        // Start by getting rid of the old selections
        //
        $sql = "DELETE FROM GamePlatform WHERE GameSeq = $gameseq";
        $rslt = mysql_query($sql, $dbh) or die("Query failed:
GamePlatform Delete.");
```

It's important to keep in mind that a DELETE query that doesn't actually delete any rows, is still valid as far as the database is concerned. This is no different than doing a SELECT with a WHERE clause that doesn't actually match any rows. The result set would be empty, but the query would still be legal as far as MySQL is concerned.

Our next step is to go through the $r_selected array and add back any selected platforms. However, the first thing we need to do is make sure that there were some. It's entirely possible that the user may have decided to clear the list of gameplatforms for some reason, even if that's an unlikely proposition. So the first thing we do is use **count()** to make sure there are actually some elements in the array, and it's not just empty. Once we know we have platforms to add, we'll build up our insert statement. MySQL allows you to do multiple inserts in one INSERT statement by adding a series of values after the VALUES statement. We'll use this capability to minimize the load on the database.

We start with our basic INSERT query in a string:

PHP Example: (!)

```
$sql = 'INSERT INTO GamePlatform (GameSeq, PlatformSeq) VALUES ';
```

Now we foreach through $r_selected and in each case add **(gameseq, platformseq) ,**. Note that because we don't know how many values we may have, we'll have one piece of housekeeping to do when we exit the foreach statement, and that's to strip off the final extraneous comma.

PHP Example: (!)

```
    $sql = 'INSERT INTO GamePlatform (GameSeq, PlatformSeq) VALUES ';
            foreach ($r_selected as $value) {
                $sql .= "($gameseq, $value),";
            }
            // Get rid of last unneeded ','
            $sql = substr($sql, 0, -1);
```

PHP's **substr()** is an incredibly flexible sub-string function and makes the task of stripping the last character off a string very easy. Using substr($str, 0, -1), you have passed -1 as the length parameter, which tells substr we want the entire length of the string, minus one last character.

In a lot of other languages, you'd have to do the length calculation yourself. In this case, the last line of the function, simply strips off that extraneous comma I mentioned earlier.

What's left to handle after POST??

Now that we have a function to handle our many to many resolver table GamePlatform, we can return to looking at how the Game table is different from the Developer table. For the most part, it's really just has more columns, even if some of those columns are foreign keys. Needless to say, the first issue is making sure that we get each of them from the $_POST[].

PHP Example: (!)

```
    $title = $_POST['title'];
    $statuscode = $_POST['Status'];
    $genre = $_POST['Genre'];
    $developer = $_POST['Developer'];
    $publisher = $_POST['Publisher'];
    $website = $_POST['website'];
    $description = $_POST['description'];
    $review = $_POST['review'];
    $r_selected = $_POST['list_s'];
```

With these variables we're in a position to handle Adding or Updating a Game. There is no difference in the framework being employed for Game: if we're deleting a game, we have to also delete any GamePlatform rows for that Game, so this is handled by doing a delete query on GamePlatform followed by one on Game.

When editting a row, we employ an UPDATE query, plugging the values we got from the $_POST[]. The only additional concern is a call to the update_gameplatform() function.

PHP Example: (!)

```
    $sql = "UPDATE Game
        SET Title='$title',
        GenreSeq = $genre,
        PublisherSeq = $publisher,
        DeveloperSeq = $developer,
        StatusCode = '$status',
        Website = '$website',
        Description = '$description',
        Review = '$review'
        WHERE GameSeq = $seq";
    $rslt1 = mysql_query($sql, $dbh1) or die("Query failed, Game
Update");
    // Now add the GamePlatforms
    update_gameplatform($seq, $r_selected);
```

When adding a new row, we have only one small complication. Because Game.GameSeq is a mysql AUTO_INCREMENT column, we won't know the value of GameSeq until after the row is inserted. Note that we do not even specify the GameSeq column in the INSERT query. Immediately following the insert, we can call mysql_insert_id() and receive the new GameSeq, so that we can then call update_gameplatform(), and have it insert the appropriate GamePlatform rows.

PHP Example: (!)

```
$q1 = "INSERT INTO Game
     (Title, GenreSeq, PublisherSeq,
     DeveloperSeq, StatusCode, Website,
     Description, Review)
     VALUES
     ('$title', $genre, $publisher,
      $developer, '$status', '$website',
    '$description', '$review')";
$rslt1 = mysql_query($q1, $dbh1) or die("Query failed: Doing Game
Insert");
$seq = mysql_insert_id($dbh1);
update_gameplatform($seq, $r_selected);
```

Because the List-Detail-Post paradigm requires some URL variables to interconnect scripts, we do have to check that these variables take us back to the right scripts. It should be apparent that the following block of code was altered accordingly to reflect that this post was designed to handle the Game table.

PHP Example: (!)

```
    if ($mode == 'delete') {
        $returnurl = "gamelist.php";
        if (isset($sort))
            $returnurl .= "?sort=$sort";
    } else {
           $returnurl =
"gamedetail.php?mode=edit&seq=".$seq."&status=".$status;
        if (isset($sort))
            $returnurl .= "&sort=$sort";
    }
```

If you're playing around with the gamedetail.php script you should find that you can now successfully add and even edit Games. The problem of course, is that we have a detail form, but no list form. We copy developerlist.php to gamelist.php, but how do we attack the question of what to change?

Writing the GameList

The first thing to consider is the question of what you might want to see, and sort by in a columnar list of Games. This is a judgement call, but I would lean towards this list: **GameSeq, Game Title, Status, Genre, Developer and Publisher.** I leave text descriptions, notes and relationships that don't fit the idea of a "one row/ one line" interface.

We might not want to actually display GameSeq, but we'll want to query it nevertheless, as it will be passed as a url parameter to the gamedetail.php script. With this in mind, the first thing we need to do is figure out the query we need to accomplish this. Unlike Developer, a Game row contains several foreign keys. It won't do much good to display DeveloperSeq 14 in a column, so we will need to use standard inner joins to all the related tables so we can display meaningful information in the list like the actual Developer's name.

The query isn't really complicated if you go through it methodically starting with the columns you need, then the tables required, and finally adding the joins in the WHERE clause. When all is said and done your query will probably look something like this:

PHP Example: (!)

```
$q1 = "SELECT GameSeq, Title, s.Description as Status,
    ge.Name as Genre, d.Name as Developer,
    p.Name as Publisher
    FROM Game g, Status s, Genre ge, Developer d, Publisher p
    WHERE s.StatusCode = g.StatusCode AND
    ge.GenreSeq = g.GenreSeq AND
    d.DeveloperSeq = g.DeveloperSeq AND
    p.PublisherSeq = g.PublisherSeq
    ORDER BY $sortby";
```

You should recall that a List includes the option to resort the list by it's various columns, so we have a $sortby variable that will control the ORDER BY.

Once we have a working List query, the next step is to do a search and replace on developer. It's a safe bet that where we were using **developer** we're going to be using game. Simply doing a search and replace, and looking at what your editor wants to change should remind you of the main strategies of the list, and how it interacts with the detail.

The primary exercise that remains is to expand the section that handles the setting of the $sortby variable, and correcting the html table so that it includes the right number of table columns.

gamelist.php

PHP Example: (!)

```
<?php
    $s = $HTTP_GET_VARS["sort"];
    include("adminheader.php");
?>
<script language="javascript">
function confirmdelete(delurl) {
    var msg = "Are you sure you want to Delete this Row?";
    if (confirm(msg))
        location.replace(delurl);
}
</script>
</head>
<?php
    require("config.php");

    function loadlist($s) {
        Global $dbhost, $dbname, $dbuser, $dbpasswd;

        switch ($s) {
            case "name" :
                $sortby = "Title";
                break;
            case "status" :
                $sortby = "Status";
                break;
            case "genre" :
                $sortby = "Genre";
                break;
            case "developer" :
                $sortby = "Developer";
                break;
            case "publisher" :
                $sortby = "Publisher";
                break;
            default:
                $sortby = "GameSeq";
        }

        $dbh1 = mysql_connect($dbhost, $dbuser, $dbpasswd) or
die("Could not connect");
           mysql_select_db($dbname,$dbh1) or die("Could not select
database");
```

```
        $q1 = "SELECT GameSeq, Title, s.Description as Status,
                    ge.Name as Genre, d.Name as Developer,
                    p.Name as Publisher
                    FROM Game g, Status s, Genre ge, Developer d,
Publisher p
                    WHERE s.StatusCode = g.StatusCode AND
                    ge.GenreSeq = g.GenreSeq AND
                    d.DeveloperSeq = g.DeveloperSeq AND
                    p.PublisherSeq = g.PublisherSeq

                    ORDER BY $sortby";
           $rslt1 = mysql_query($q1, $dbh1) or die("Query failed");

        echo "<table><tr><th colspan=7>Game List</th></tr>";
        echo "<tr><th><a
href=".$_SERVER['SCRIPT_NAME'].">Seq</a></th><th><a
href=".$_SERVER['SCRIPT_NAME']."?sort=name>Name</a></th>
                    <th><a
href=".$_SERVER['SCRIPT_NAME']."?sort=status>Status</a></th>
                    <th><a
href=".$_SERVER['SCRIPT_NAME']."?sort=genre>Genre</a></th>
                    <th><a
href=".$_SERVER['SCRIPT_NAME']."?sort=developer>Developer</a></th>
                    <th><a
href=".$_SERVER['SCRIPT_NAME']."?sort=publisher>Publisher</a></th>
                    <th></th></tr>n";
           while ($row1 = mysql_fetch_array($rslt1, MYSQL_ASSOC)) {
               echo "t<tr>n";
            $dtlUrl = "gamedetail.php?mode=edit&seq=".$row1['GameSeq'];
            $delUrl =
"gamepost.php?mode=delete&delseq=".$row1['GameSeq'];
            if (isset($s)) {
                $dtlUrl .= "&sort=$s";
                $delUrl .= "&sort=$s";
            }
            echo "tt<td><a href=$dtlUrl>".$row1['GameSeq']."</a></td>";
            echo "tt<td><a href=$dtlUrl>".$row1['Title']."</a></td>";
            echo "tt<td><a href=$dtlUrl>".$row1['Status']."</a></td>";
            echo "tt<td><a href=$dtlUrl>".$row1['Genre']."</a></td>";
            echo "tt<td><a
href=$dtlUrl>".$row1['Developer']."</a></td>";
            echo "tt<td><a
href=$dtlUrl>".$row1['Publisher']."</a></td>";
            echo "<td><input type="button" value="Delete"
onClick="confirmdelete('".$delUrl."');"></td>t</tr>n";
        }
        echo "</table>n";
        /* Free resultset */
        mysql_free_result($rslt1);
        /* Closing connection */
        mysql_close($dbh1);
    }

// Main Script
    $addUrl = "gamedetail.php?mode=add";
    if (isset($s))
        $addUrl .= "&sort=$s";
    echo "<a href=adminmenu.php>Back</a>  <a href=$addUrl>Add New
Game</a><br><br>";
    loadlist($s);
?>
```

In particular the switch statement for the $sortby url parameter needs to be expanded.

PHP Example: (!)

```
switch ($s) {
            case "name" :
                $sortby = "Title";
                break;
            case "status" :
                $sortby = "Status";
                break;
            case "genre" :
                $sortby = "Genre";
                break;
            case "developer" :
                $sortby = "Developer";
                break;
            case "publisher" :
                $sortby = "Publisher";
                break;
            default:
                $sortby = "GameSeq";
        }
```

As you can see, all we really needed to do, was add cases for our new columns, and our $sortby variable takes care of the rest.

We also need to add new table rows for the additional columns, and increase the **colspan=** number accordingly, but these should be obvious changes to you at this point.

Developer and Game

Earlier in the series, I talked about the relationship between developer and Game as being 1-M. Now would be the time to revisit the Developer Detail script. In part I of the series, the developer list, detail and post were the scripts used to illustrate how the list-detail-post paradigm works, but the original script didn't account for a developers Games. Now that we've done a Game list, it's a good time to go back and reconsider what we'd need to add to the developer detail script to add a list of the developer's games.

In doing the Game list, we've done the majority of the work we need to add a list of a Developer's Games to the Developer Detail script. The only difference is that a standalone game list will show all games, while a developer game list should show *only the games for that developer*.

I would hope that those who have followed along through this tutorial system, will have a good idea of what we need to do now, but for the sake of completeness, I'll present the revisions necessary to get basic support for an embedded detail list installed.

To begin with take a look back at the list query on the previous page, and compare it to this one we'll use inside of developerdetail.php

PHP Example: (!)

```
$q1 = "SELECT GameSeq, Title, s.Description as Status,
                    ge.Name as Genre, d.Name as Developer,
                    p.Name as Publisher
                    FROM Game g, Status s, Genre ge, Developer d,
Publisher p
                    WHERE g.DeveloperSeq = $seq AND
                    s.StatusCode = g.StatusCode AND
                    ge.GenreSeq = g.GenreSeq AND
                    d.DeveloperSeq = g.DeveloperSeq AND
                    p.PublisherSeq = g.PublisherSeq";
```

Very little needs to change. The main thing we needed to do was add an additional condition to the WHERE clause: **g.DeveloperSeq = $seq.** As this is really the most important condition (only return the Games tied to *this developer* (the one from the url parameter $seq), I will usually make sure that's the first condition in the WHERE clause. The rest of the exercise is simply bumping down the original criteria. You will also note, that I removed the ORDER BY. In our embedded list, we need to simplify things a bit, because it would be difficult to support all the functionality of a standalone list, without having to come up with a bunch of new url parameters. We really don't need sortation as much with a smaller list of games, so we'll simply strip that code out once we copy it.

Where do we stick an embedded detail list?

The obvious place is to add the embedded list after your existing form. This way it doesn't interfere with the form functionality, or require any modification. The embedded list simply becomes complementary, and we can decide how much or little functionality we want that list to have without much concern for how it will effect the original detail code.

The other thing to consider is under what conditions should we display the list. For a new developer, it doesn't make much sense to have a list, since a new developer won't have any games to display. So we'll only do the work to display an embedded list when **(mode == 'edit').**

Here's our modified script.

developerdetail.php (with embedded game list)

PHP Example: (!)

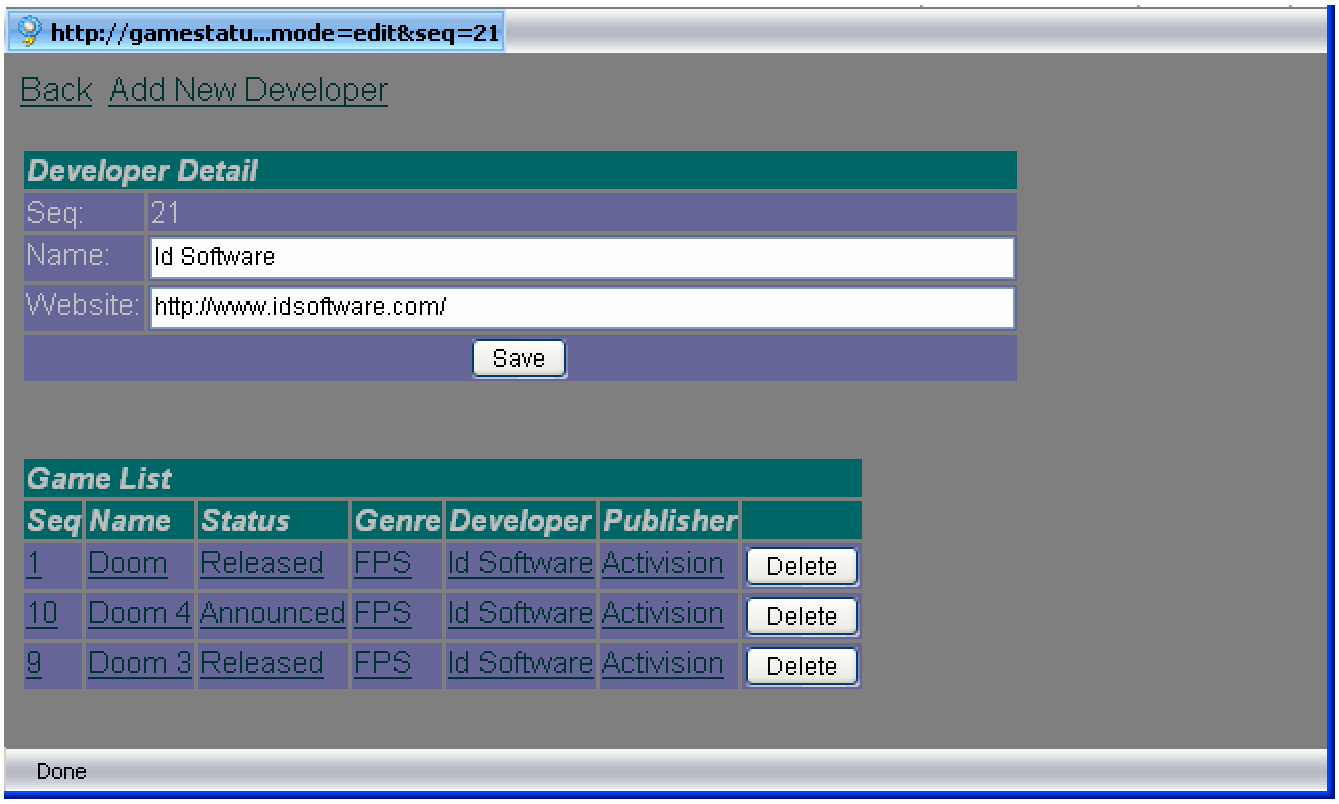
```
<?php

$m = $HTTP_GET_VARS["mode"];
$sort = $HTTP_GET_VARS["sort"];
$seq = $HTTP_GET_VARS["seq"];
$s = $HTTP_GET_VARS["status"];


include("adminheader.php");
require("config.php");
//require("common.class.php");

if ($m == 'edit') {
    $dbh1 = mysql_connect($dbhost, $dbuser, $dbpasswd) or die("Could
not connect");
    mysql_select_db($dbname,$dbh1) or die("Could not select database");
    $q1 = "SELECT * FROM Developer where DeveloperSeq=$seq";
  $rslt1 = mysql_query($q1, $dbh1) or die("Query failed");
    $row1 = mysql_fetch_array($rslt1, MYSQL_ASSOC);
    /* Free resultset */
  //mysql_free_result($rslt1);
} else {
    $seq = 0;
}
$backurl = "developerlist.php";
$posturl = "developerpost.php";
$addurl = "developerdetail.php?mode=add";

if (isset($sort)) {
    $backurl .= "?sort=$sort";
    $posturl .= "?sort=$sort";
    $addurl .= "&sort=$sort";
}

echo "<a href=$backurl>Back</a>  <a href=$addurl>Add New
Developer</a><br><br>";
echo '<form name="form1" method="POST" action="'.$posturl.'">';
echo "<input type='hidden' name='seq' value='$seq'>";
echo "<table>";
echo "<tr><th colspan=2>Developer Detail</th></tr>";
echo "<tr><td>Seq:</td><td>$seq</td></tr>";
echo '<tr><td>Name:</td><td><input type="text" name="name" size="80"
maxlength="80"';
if ($m == 'edit') {
    echo " value='".htmlentities($row1['Name'],ENT_QUOTES)."'";
}
echo "></td></tr>";
echo '<tr><td>Website:</td><td><input type="text" name="website"
size="80" maxlength="255"';
```

```
if ($m == 'edit') {
    echo " value='".htmlentities($row1['Website'],ENT_QUOTES)."'";
}
echo "></td></tr>";

echo "<tr><td colspan=2 align=center><input type='submit'
name='SubmitFrm' value='Save'></td></tr>";
echo "</table>";
echo "</form>";
if (isset($s)) {
    echo "Edit Status: $s <br>";
}
if ($m == 'edit') {
    $addUrl = "developerdetail.php?mode=add&seq=$seq";
    if (isset($s))
        $addUrl .= "&sort=$s";
}
if ($m == 'edit') {
    echo "<br />n";
    $q1 = "SELECT GameSeq, Title, s.Description as Status,
                    ge.Name as Genre, d.Name as Developer,
                    p.Name as Publisher
                    FROM Game g, Status s, Genre ge, Developer d,
Publisher p
                    WHERE g.DeveloperSeq = $seq AND
                    s.StatusCode = g.StatusCode AND
                    ge.GenreSeq = g.GenreSeq AND
                    d.DeveloperSeq = g.DeveloperSeq AND
                    p.PublisherSeq = g.PublisherSeq";

    $rslt1 = mysql_query($q1, $dbh1) or die("Query failed on SELECT");
    echo "<table><tr><th colspan=7>Game List</th></tr>";
    echo "<tr><th>Seq</th><th>Name</th>
                <th>Status</th>
                <th>Genre</th>
                <th>Developer</th>
                <th>Publisher</th>
                <th></th></tr>n";

    while ($row1 = mysql_fetch_array($rslt1, MYSQL_ASSOC)) {
        echo "t<tr>n";
        $dtlUrl = "gamedetail.php?mode=edit&seq=".$row1['GameSeq'];
        $delUrl = "gamepost.php?mode=delete&delseq=".$row1['GameSeq'];

        echo "tt<td><a href=$dtlUrl>".$row1['GameSeq']."</a></td>";
        echo "tt<td><a href=$dtlUrl>".$row1['Title']."</a></td>";
        echo "tt<td><a href=$dtlUrl>".$row1['Status']."</a></td>";
        echo "tt<td><a href=$dtlUrl>".$row1['Genre']."</a></td>";
        echo "tt<td><a href=$dtlUrl>".$row1['Developer']."</a></td>";
        echo "tt<td><a href=$dtlUrl>".$row1['Publisher']."</a></td>";
        echo "<td><input type="button" value="Delete"
onClick="confirmdelete('".$delUrl."');"></td>t</tr>n";
    }
    echo "</table>n";
}
?>
```

As you can see, we've really just copied the working list code from gamelist.php, modified the query as discussed, and removed anything related to the $sortorder variable.

One detail we need to attend to is the reuse of the database connection. In the original developerdetail.php script we closed the connection when we were done with it using **mysql_free_result($rslt1).** Since we want to reuse the database connecion for our embedded game list query, we either need to delete or comment out this line, or else we'd have to create a new database connection. In many databases, creating a database connection can take a good deal of time, but in MySQL it's a fairly lightweight operation. It wouldn't be a major performance hit to reopen the conenction. However, there's not really any benefit to doing so here. Furthermore, there's really no reason in PHP to close a mysql database connection since PHP's garbage collection will do that when the page has finished executing. In this case I just commented out the call.

Now when we click on a Developer which has associates Games, we should see something like this:



Developer Detail

Seq:	21
Name:	Id Software
Website:	http://www.idsoftware.com/

Game List

Seq	Name	Status	Genre	Developer	Publisher	
1	Doom	Released	FPS	Id Software	Activision	Delete
10	Doom 4	Announced	FPS	Id Software	Activision	Delete
9	Doom 3	Released	FPS	Id Software	Activision	Delete

This is certainly a great start. We have a lot of functionality in the embedded Game List, although there are some issues that now arise. For example, it would be nice to be able to Add a new Game from this page, that would already have the DeveloperSeq set. We could add a link pretty easily, but we'd need to go into the game post script and add some intelligence in the add condition, that currently isn't there. There's also the problem of what happens with the back link when you drill into a Game, particularly if you were to change it. There are solutions for these problems, and I leave it to you to ponder them, and explore how to make this work, if you're interested enough in doing so.

Summary

When I began this series of articles I wanted to peal back a lot of the layers of interactive website development with MySQL and PHP, and in the process, reveal something about the necessary thought process and approach a typical developer needs to employ. Considering all the different topics covered in this series, I also managed to achieve another one of my goals, namely demonstrating exactly why it is such a challenging and underestimated pursuit.

The GameStatus system certainly involves some sophistication and some neat tricks, but it intentionally avoids issues of security, user authentication, session, and many of the other important concepts that have been covered in other phpFreaks tutorials and on other PHP community sites. I hope I made my point in that web development needs to begin in most cases with consideration of the database design. One also needs to understand not only html (and .css to be current) but also relational database design and theory, SQL, structured programming, the design of HTTP and how it works, and even in some cases, the javascript language and DOM.

It is a large interconnected area of knowledge, and a challenge for any one person to master in a short amount of time. While the breadth of material may seem daunting, it is one that you will only get better at through exploration, reading and most of all, designing and developing your own applications. There's a lot to learn, and I hope that this series showed how you can start with something simple, and build upon it to achieve something moderately complicated, learning as you go. It also demonstrates how a conceptual framework can help with the front end and the structure of the code you use. List-Detail-Post is a simple but capable paradigm and well suited to this series. It is certainly not the only one, and I recommend you research other frameworks and paradigms (Mojavi [an MVC library], PHP Fusebox, and Cake [based on Ruby on Rails], all of which come with classes and code libraries to allow for rapid frontend development. Even if you roll your own, having a paradigm will provide your system the consistency it needs, and will lead to a natural inclination to develope classes and functions that will allow you to reuse rather than reinvent the same thing repeatedly.

Dedication

On a personal note, it took me well over a year and a half to complete this series, even though I had much of it done when Part I was published. Between the time Part I was published and now, I've received numerous emails from people who read Part I, asking me when Part II would be released. That provided me with a lot of encouragement, and I appreciated those emails. Although I made some progress over the months, I found many other demands on my time, and kept putting it off.

During this period, my father passed away. Aside from being a wonderful Dad who taught me a lot of things, and allowed me to pursue my own interests in a supportive way, he also passed on many of the values I've come to believe are important to a man who wants to be both a good parent, and live a principled life. He managed this, most of all by the example he set in the way he lived his daily life. One of those values was "always finish what you start." In the back of my mind it was always his voice that spoke to me of the commitment I undertook when I unleashed Part I of this series on an unsuspecting world, even if that commitment was primarily to myself, as a way of giving back to the PHP community. Now that the series is finished, I take pride in dedicating it to the memory of my father, **Frank Rolston III**, *Veteran, Engineer, Husband, Father and Grandfather.* You are greatly missed.
