By The Disenchanted Developer

Table of Contents

Back To Work

In the first part of this article, I explained the need and rationale for a timesheet application to log and analyze employee work hours, and put together a set of functional requirements for this application. I then constructed a database schema to address my requirements, designed a simple user interface, and wrote a few of the basic scripts required by such a system.

The job's nowhere near complete, though − I have yet to write the code which accepts and validates user input, displays timesheet entries, and analyzes the raw data to create a useful report. I plan to address all these items, and a few more, over the next few pages. So keep reading.

Getting Creative

If you look at the main menu, you'll see that the third menu item allows the user to view entries for a specified date. When you analyze this a little, though, you'll see that this menu item has broader implications − it must allow the user to:

view a list of timesheet entries for the specified date;

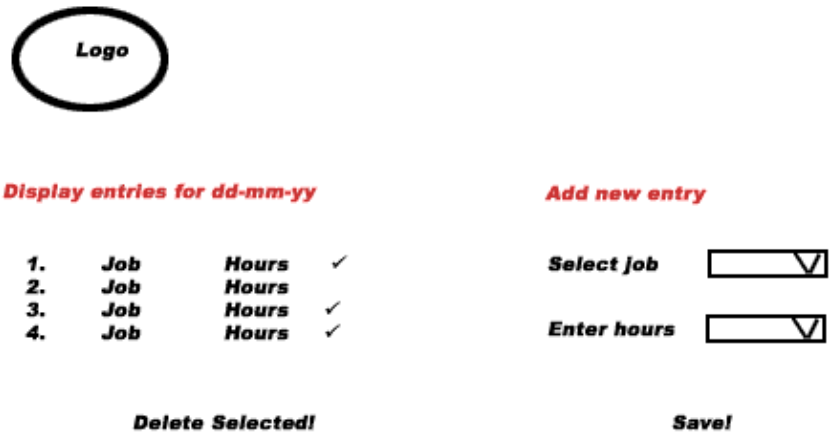
add new entries for the specified date;

delete existing entries from the list.

In order to perform these functions, the script − I've called it "view.php" − must necessarily receive a valid datestamp (created from the variables $d, $m and $y). These three variables are generated from the drop−down boxes in "menu.php" − you may remember this from last time's article (if not, I'd strongly suggest you review the source code for "menu.php" and then come back here). Once "view.php" receives these values, it generates a datestamp, checks to ensure that the date is a valid one, and then queries the "log" table for all relevant data.

So that takes care of displaying existing entries. But what about adding new ones, or deleting existing ones?

Well, after much thought and a couple of conversations with our resident interface designer, I've come up with a clever way to combine both these functions into "view.php". Here's a rough sketch of what I have in mind:



Essentially, I plan to split the page into two main sections. The left side of the page will contain a list of entries retrieved from the database, with a checkbox next to each; the user can select an entry for deletion by checking the corresponding box. The right side of the page will hold a form, which can be used to add new entries to the timesheet. Entries added from the form on the right side will immediately appear in the list on the left side.

This sounds complicated, but it's actually not all that difficult to implement. The important thing here is to take the pieces one at a time and deal with them separately. And so, the first thing I need to do is set up a page template.

```
<?
// view.php - view/add/delete entries for a specific date

// includes

// check for valid user session
session_start();
if(!session_is_registered("SESSION_UID"))
{
header("Location: error.php?ec=1");
exit;
}

// check for valid date
if (!checkdate($m, $d, $y))
{
header("Location: error.php?ec=2");
exit;
}

<html>
<head>
</head>

<body bgcolor="white">

<?
// display page header
?>

<!-- table (1r, 3c) -->
<table width="100%" border="0" cellspacing="0"
cellpadding="0">
<tr>
<td valign="top" align="left" width="60%">
<!-- table for existing timesheet records goes here - snip -->
</td>

<!-- spacer -->
<td width="10%">
 
</td>

<td valign="top" align="left" width="30%">
<!-- table for new timesheet records goes here - snip -->
</td>
</tr>
</table>
```

```
<?
// display page footer
?>

</body>
</html>
```

The script begins with the usual checks and includes, and generates an HTML page containing a (1r, 3c) table. The first and last of these cells will be used for the existing and new timesheet data respectively.

Note the checkdate() function at the top of the script − I'm using this to validate the date value received by "view.php". This check is necessary to catch incorrect date values − for example, 30 February or 31 April. If the date selected by the user is invalid, the script will redirect to the generic error handler, which should display a message indicating the error.

Split Personality

Let's now look at the code which connects to the database and retrieves a list of entries, given the user and date.

```
<!-- table (1r, 3c) -->
<table width="100%" border="0" cellspacing="0"
cellpadding="0">
<tr>
<td valign="top" align="left" width="60%">
<!-- table for existing timesheet records goes here -->

<?
// create datestamp
$datestamp = $y . "-" . $m . "-" . $d;

// initialize variable to hold total hours worked
$totalHours = 0;

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

// get timesheet records for this user and this date
$query = "SELECT lid, pname, tname, hours FROM projects,
tasks, log WHERE
projects.pid = log.pid AND tasks.tid = log.tid AND date =
'$datestamp' AND
uid = '$SESSION_UID'";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());
?>

<table border="0" cellspacing="2" cellpadding="2">
<tr>
<td colspan=4><b><font color="#3098C3">Timesheet for <? echo
fixDate($datestamp); ?></font></b></td>
</tr>

<?
// if records exist
if (mysql_num_rows($result) > 0)
{
?>
<form action="delete.php" method="post">
```

```
<tr>
<td><b>Project</b></td>
<td><b>Task</b></td>
<td><b>Hours</b></td>
<td> </td>
</tr>


<?
// display
while (list($lid, $pname, $tname, $hours) =
mysql_fetch_row($result))
{
$totalHours= $totalHours + $hours;
?>
<tr>
<td><? echo $pname; ?></td>
<td><? echo $tname; ?></td>
<td align="center"><? echo sprintf("%1.01f", $hours); ?></td>
<td><input type="Checkbox" name="lid[]" value="<? echo $lid;
?>"></td>
</tr>
<?
}
?>

<tr>
<td colspan=2> </td>
<td align="center"><hr><b><? echo sprintf("%1.01f",
$totalHours);
?></b></td>
<td> </td>
</tr>

<tr>
<td colspan="4" align="right"><input type="Submit"
name="submit"
value="Delete Selected Records"></td>
</tr>

<!-- send date as hidden values - used to redirect back to
this page -->
<input type="hidden" name="d" value="<? echo $d; ?>">
<input type="hidden" name="m" value="<? echo $m; ?>">
<input type="hidden" name="y" value="<? echo $y; ?>">

</form>
<?
}
```

```
// if no existing records
// display message
else
{
?>
<tr>
<td colspan=4>No records found</td>
</tr>
<?
}
?>
</table>
</td>

<!-- spacer -->
<td width="10%">
 
</td>

<td valign="top" align="left" width="30%">
<!-- table for new timesheet records - snip -->
</td>
</tr>
</table>
```

Here are a few quick notes on how this code works:

1. The first order of business is to open a connection to the database and retrieve a list of entries for this user and this date; I've used a join so that I can display descriptive project and task names rather than meaningless IDs.

```
<?
// get timesheet records for this user and this date
$query = "SELECT lid, pname, tname, hours FROM projects,
tasks, log WHERE
projects.pid = log.pid AND tasks.tid = log.tid AND date =
'$datestamp' AND
uid = '$SESSION_UID'";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());
?>
```

2. Before proceeding to use the generated resultset, I'm also initializing a variable named $totalHours - this variable will be used to calculate the total for the hours column in the list displayed.

```
<?
// initialize variable to hold total hours worked
$totalHours = 0;
?>
```

3. The fixDate() function call in the first row of the table

```
<tr>
<td colspan=4><b><font color="#3098C3">Timesheet for <? echo
fixDate($datestamp); ?></font></b></td>
</tr>
```

is used to create a human−readable version of the $datestamp variable. Here's what the function looks like:

```
<?
// function to format DATE values
function fixDate($val)
{
// split it up into components
$datearr = explode("−", $val);
// create a timestamp with mktime(), format it with date()
return date("d M Y", mktime(0, 0, 0, $datearr[1], $datearr[2],
$datearr[0]));
}
?>
```

4. With all that out of the way, I'm iterating through the resultset to display each entry, complete with project name, task name and hours worked. Since I want to allow the user to delete selected entries, a form checkbox is added next to each entry as well.

Each iteration also adds the current hours value to the $totalHours value, which is displayed in a separate row once the resultset has been completely processed.

```
<?
// snip

// display
while (list($lid, $pname, $tname, $hours) =
mysql_fetch_row($result))
{
$totalHours= $totalHours + $hours;
?>
```

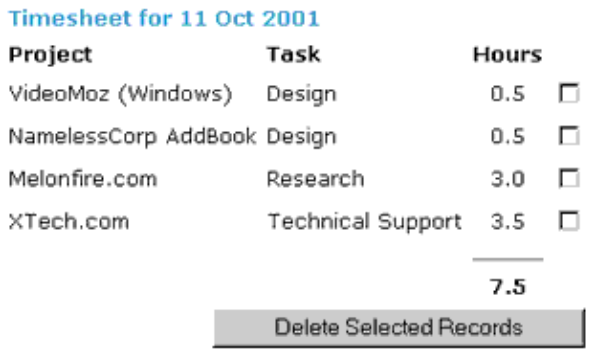
```
<tr>
<td><? echo $pname; ?></td>
<td><? echo $tname; ?></td>
<td align="center"><? echo sprintf("%1.01f", $hours); ?></td>
<td><input type="Checkbox" name="lid[]" value="<? echo $lid;
?>"></td>
</tr>
<?
}
?>
```

When the user hits the delete button and submits the form, the selected entries will be passed to the form processor "delete.php" as an array − this array is named $lid (you can see it attached to each checkbox above) and contains the unique record identifier for each selected entry. The "delete.php" processor will use this information to identify which records are to be deleted from the "log" table.

5. Finally, I'm including the three date variables $d, $m and $y in the form as hidden values. My intent here is to pass these values to the "delete.php" script when the form is submitted; "delete.php", in turn, will use them to redirect the browser back to the correct "view.php" instance.

```
<!-- send date as hidden values - used to redirect back to
this page -->
<input type="hidden" name="d" value="<? echo $d; ?>">
<input type="hidden" name="m" value="<? echo $m; ?>">
<input type="hidden" name="y" value="<? echo $y; ?>">
```

Here's what it looks like:



The right side of the page contains a form, which allows the user to add a new entry to the "log" table by specifying the hours worked on a specific task for a specific project.

```
<!-- table (1r, 3c) -->
<table width="100%" border="0" cellspacing="0"
```

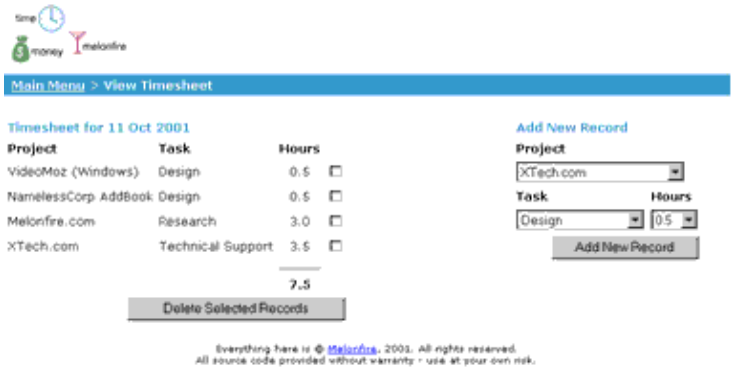
```
cellpadding="0">
<tr>
<td valign="top" align="left" width="60%">
<!-- table for existing timesheet records goes here - snip -->
</td>

<!-- spacer -->
<td width="10%">
 
</td>

<td valign="top" align="left" width="30%">
<!-- table for new timesheet records goes here -->
<table border="0" cellspacing="2" cellpadding="2">
<form action="add.php" method="post">

<tr>
<td colspan="2"><b><font color="#3098C3">Add New
Record</font></font></td>
</tr>

<tr>
<td colspan="2"><b>Project</b></td>
</tr>

<tr>
<td colspan="2">
<select name="pid">
<?
// get project list
$query = "SELECT pid, pname from projects";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());
while (list($pid, $pname) = mysql_fetch_row($result))
{
echo "<option value=$pid>$pname</option>";
}
mysql_free_result($result);
?>
</select>
</td>
</tr>


<tr>
<td><b>Task</b></td>
<td><b>Hours</b></td>
</tr>
```

```
<tr>
<td>
<select name="tid">
<?
// get task list
$query = "SELECT tid, tname from tasks";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());
while (list($tid, $tname) = mysql_fetch_row($result))
{
echo "<option value=$tid>$tname</option>";
}
mysql_free_result($result);
mysql_close($connection);
?>
</select>
</td>
<td>
<select name="h">
<?
for ($x=0.5; $x<=16; $x+=0.5)
{
echo "<option value=$x>" . sprintf("%1.01f", $x) .
"</option>";
}
?>
</select>
</tr>

<tr>
<td colspan=2 align=right><input type="submit" name="submit"
value="Add
New Record"></td>
</tr>

<!-- send date as hidden values - used to redirect back to
this page -->
<input type="hidden" name="d" value="<? echo $d; ?>">
<input type="hidden" name="m" value="<? echo $m; ?>">
<input type="hidden" name="y" value="<? echo $y; ?>">

</form>
</table>
</td>
</tr>
</table>
```

This is much simpler to read and understand than the previous listing. Essentially, I'm setting up drop−down boxes for the various project, task and hour values (in increments of 0.5), and inviting the user to add an entry to the timesheet by simply selecting appropriate values from each. I'm also including the three hidden date variables in the form, for the same reasons previously stated.

Here's what this half of the page looks like:

And here's what the complete product looks like:



In case you're still confused, don't give up just yet − the next couple of scripts should make things clearer.

In...

The two halves of "view.php" correspond to two different form processors, "delete.php" and "add.php" respectively. Let's look at "add.php" first.

```
<?
// add.php − add timesheet entry

// includes

// checks

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

// create datestamp
$datestamp = $y . "−" . $m . "−" . $d;

// insert data
$query = "INSERT INTO log (pid, tid, uid, hours, date) VALUES
('$pid',
'$tid', '$SESSION_UID', '$h', '$datestamp')";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// close connection
mysql_close($connection);

// redirect back
header("Location:view.php?d=$d&m=$m&y=$y");
?>
```

The data received from the form in "view.php" − project, task and hour values − is used by this script to formulate an INSERT query and insert the entry into the "log" table. The $SESSION_UID variable is used to identify which user this entry belongs to.

Once the data has been inserted, the browser is redirected back to the page it came from, using the datestamp values passed to it from "view.php".

...And Out

The opposite of "add.php" is "delete.php", which is designed to accept a list of record identifiers and delete the corresponding records from the "log" table. An array of these identifiers is passed to the script via the $lid array − take a look:

```
<?
// delete.php − delete timesheet entries

// includes and checks

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

// iterate through checkbox array, delete records
for ($x=0; $x<sizeof($lid); $x++)
{
$query = "DELETE FROM log WHERE lid = '$lid[$x]'";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());
}

// close connection
mysql_close($connection);

// redirect back
header("Location:view.php?d=$d&m=$m&y=$y");
?>
```

Here too, once the deletion has been accomplished, the user is taken back to the originating instance of "view.php"

The Number Game

Yes, I know that was a little complicated, but you have to admit it works like a charm. One of the major battles is now behind us, with one still ahead − report generation.

Before I start writing code, I'd like to spend a little time analyzing the problem ahead. First, there are innumerable types of reports possible with the data available to us. Numbers can be massaged and interpreted in different ways to produce different conclusions (as Disraeli famously said, "there are lies, damned lies and statistics") and so it's essential (especially since I'm running out of time) to identify which types of reports are most useful, and concentrate my efforts on those.

In order to get a better feel for the problem, I decide to turn to the Boss for help. After all, he runs a company too − surely he'll have some idea of the kind of data that would be most helpful in making business decisions on resource allocation and usage. And he does...

"Well, lemme see," he drawls, spinning in his swivel chair to look out of the window. "The most important thing, in my opinion, would be to see the amount of time spent on the different tasks within a project. I'd find it very useful to know, for example, how much time you spend on code design and development, versus the amount of time the System Test people take to test your code. If I can break up a project into tasks and attach a dollar value to the hours spent on each task, I can then calculate the total time and money spent on the project, compare it with the revenue earned, and find out if we're actually making any money!" (this last accompanied with a fist slamming down on the table.)

"Next, " he continues, calming down a little, "I want to see a breakdown of hours spent on a project by user, so that I know who's pulling the weight and who's slacking off. I notice that you, for example, spend an inordinate amount of time at lunch − I'd be very curious to see how those hours are logged."

So he's been keeping tabs on me, huh? Gotta wonder how this guy has time to manage a company, given the amount of time he spends keeping track of employee lunch hours...

But the Boss is just warming up.

"And, since I've also happened to notice that this company's output drops dramatically every time I have to go out of town, I'd very much like the ability to see a big−picture overview for a specific period of time − this should show me the time spent on *all* our active customers, broken down by user and task. And you know something − I bet that if I compare that data for the weeks I'm in the office and the weeks I'm out of town, I'd be able to draw some interesting conclusions," smirking away like he's just thought of something funny.

Right. I've just about had enough of this guy. So I do the only thing that occurs to me − I stand up, walk around the desk, tip his swivel chair backwards and, as he flaps around helplessly on the floor, turn on my heel, and stride through the door with my long black coat flowing behind me like a cape...

Exercising Restraint

No, I didn't really do that − I still need a paycheck. But every dog has his day, and mine will come soon...

Anyway, disturbing though that conversation was, I still managed to get some useful data from it. I can now state with certainty that the system should generate the following three types of reports:

a big−picture overview of the time spent by users on all active projects for a specific period of time,

a big−picture overview of the time spent on different tasks over a specific period of time;

a focused report of time spent by different users on the different components of a project for a specific period.

With this in mind, let's quickly review the code for the report generation menu item in "menu.php"

```
<!−− menu.php −−>
<li>
<!−− generate report option −−>
<form name="report" action="report.php" method="post">
<a href="javascript:submitForm(1)">Generate activity
reports</a> between
<? generateDateSelector("s"); ?> and <?
generateDateSelector("e"); ?> for
<select name="pid">
<option value="0">&lt;all projects&gt;</option>
<?
// open connection to database
$connection = mysql_connect($hostname, $user, $pass) or die
("Unable to
connect!");

// get project list
$query = "SELECT pid, pname from projects";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());
while (list($pid, $pname) = mysql_fetch_row($result))
{
echo "<option value=$pid>$pname</option>";
}
mysql_free_result($result);
?>
</select>
</form>
<p>
<?
}
```

```
?>
```

This has all the data I need to build any of the three report types − a starting date (created from the date variables $sd, $sm and $sy), an ending date (created from the date variables $ed, $em and $ey) and a project identifier (which may be 0 for "all projects"). When this form is submitted, it sends these variables to the script "report.php", which performs the actual report generation.

If you look at "report.php", you'll see that it's basically one gigantic "if" loop − the first part generates big−picture type reports, while the second part generates more specific reports.

```
<?
// report.php − generate reports

// includes

// check for valid session and valid administrator

// check for valid dates
if (!checkdate($sm, $sd, $sy) || !checkdate($em, $ed, $ey))
{
header("Location: error.php?ec=2");
exit;
}

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

// create start and end datestamp
$sdate = $sy . "−" . $sm . "−" . $sd;
$edate = $ey . "−" . $em . "−" . $ed;

?>
<?
// main "if" loop begins

// if report required for ALL projects
if ($pid == 0)
{
// code goes here
}
// report required for a specific project
else
{
// code goes here snip!
}
```

```
// main "if" loop done - now clean up and print page footer
mysql_close($connection);
?>

<? include("footer.inc.php"); ?>

</body>
</html>
```

Let's write some code for the general report first - once that's done, writing code to build a report for a specific project should be a piece of cake.

The Big Picture...

My general report, such as it is, is going to look something like this:

	User 1	User 2	...	User x	
Project 1	(x1,y1)	(x2,y1)	...		rTotal
Project 2	(x1,y2)	(x2,y2)	...		rTotal
...					
Project y					
	cTotal	cTotal			Total

Since I'm going to be using a table to display the report, it helps to think of this in terms of rows and columns. The first row contains a list of users; every subsequent row represents a new project, with the numbers in the columns representing the total hours worked by that user on that project. The last column display the row totals (total time spent by all users on a specific project), while the last row contains column totals (total time spent on all projects by a specific user).

Putting together the first row should be simple enough − simply query the database for a list of users and print them.

```
<!-- projects vs. users table -->
<table width=100% border="0" cellspacing="2" cellpadding="5">
<tr>
<td> </td>
<?
// get user list
// this resultset is useful during report generation, so make
sure that it
is retained!
$query = "SELECT uid, uname FROM users";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// print in top row
while (list($uid, $uname) = mysql_fetch_row($result))
{
echo "<td valign=top align=center><font
color=#D03468>$uname</font></td>";
}

?>
```

```
<td> </td>
</tr>
```

Next, I need a project list, which will be used to generate each row:

```
<?
// get project list
$query2 = "SELECT pid, pname FROM projects";
$result2 = mysql_db_query($database, $query2, $connection) or
die ("Error
in query: $query2. " . mysql_error());
?>
```

I need to create a couple of variables, one to hold the row totals and one to hold the column totals.

```
<?
// create variables to hold totals
$columnTotals = array();
$rowTotal = 0;
?>
```

For each project (row) in this list, I need to look up the total hours worked by each user (column) and print these numbers as cells. As I obtain each cell total, I need to add that number to the appropriate row and column total variable, for use at the end of every row and column respectively.

```
<?
// do this for each project in project list...
while (list($pid, $pname) = mysql_fetch_row($result2))
{
echo "<tr>";
echo "<td valign=top><font color=#D03468>$pname</font></td>";
$rowTotal = 0;
$count = 0;

// go back to top of user list
// select a uid...
mysql_data_seek($result, 0);

while (list($uid, $uname) = mysql_fetch_row($result))
{
// calculate the sum of the intersection of user and project
$query3 = "SELECT SUM(hours) from log WHERE pid = '$pid' AND
uid =
```

```
'$uid' AND date >= '$sdate' AND date <= '$edate'";
$result3 = mysql_db_query($database, $query3, $connection) or
die
("Error in query: $query3 . " . mysql_error());
list($sum) = mysql_fetch_row($result3);

// correction if sum is zero − explicitly assign it 0
if (!$sum) { $sum = 0; }

// keep track of the row total
$rowTotal = $rowTotal + $sum;

// and the column total
$columnTotals[$count] = $columnTotals[$count] + $sum;
$count++;

// print the number
echo "<td align=center>" . sprintf("%1.01f", $sum) . "</td>";
}
// end of the row, print the row total
echo "<td align=center><b>" . sprintf("%1.01f", $rowTotal) .
"</b></td>";
}
echo "</tr>";
?>
```

Once all the rows are done, I need to add one extra row for the column totals (the row totals have already been printed).

```
<tr>
<?
// last row of table contains column totals
// ie. total hours by this user on all projects
echo "<td> </td>";
// the sum of the column totals gives a grand total
$sumOfColumnTotals = 0;

// print column totals
for ($x=0; $x<sizeof($columnTotals); $x++)
{
$sumOfColumnTotals = $sumOfColumnTotals + $columnTotals[$x];
echo "<td align=center><b>" . sprintf("%1.01f",
$columnTotals[$x]) .
"</b></td>";
}

// print grand total
```

```
echo "<td align=center><b>" . sprintf("%1.01f",
$sumOfColumnTotals) .
"</b></td>";

// clean up resultsets
mysql_free_result($result);
mysql_free_result($result2);
mysql_free_result($result3);
?>
</tr>
</table>
```

Here's what it looks like:

Projects/Users:

	john	joe	vanessa	sherry	
XTech.com	1.0	5.0	0.0	4.0	**10.0**
Melonfire.com	0.0	0.0	0.0	6.0	**6.0**
VideoMoz (Linux)	0.0	0.0	0.0	0.0	**0.0**
VideoMoz (Windows)	0.0	0.0	0.0	0.5	**0.5**
NamelessCorp AddBook	8.5	0.5	0.0	0.5	**9.5**
NamelessCorp invDB	0.0	0.5	0.0	0.0	**0.5**
	9.5	**6.0**	**0.0**	**11.0**	**26.5**

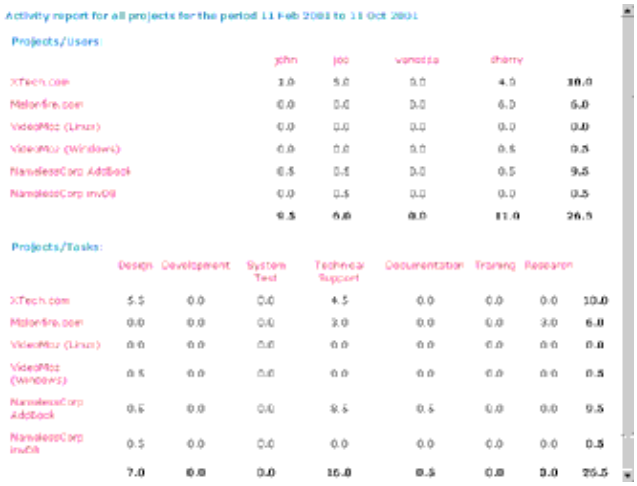
Now, on this same page, I need to include the second type of report, this one offering a breakdown of projects versus tasks. The code is identical to what you've just seen − simply replace all references to users with corresponding references to tasks.

```
<!-- projects versus tasks table -->
<table width=100% border="0" cellspacing="2" cellpadding="5">
<tr>
<td> </td>
<?
// get task list
// this resultset is useful further down
$query = "SELECT tid, tname FROM tasks";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// print tasks in top row
while (list($tid, $tname) = mysql_fetch_row($result))
{
echo "<td valign=top align=center><font
color=#D03468>$tname</font></td>";
```

```
}

?>
<td> </td>
</tr>

<?
// create variable to hold column totals
$columnTotals = array();

// get project list
$query2 = "SELECT pid, pname FROM projects";
$result2 = mysql_db_query($database, $query2, $connection) or
die ("Error
in query: $query2. " . mysql_error());

// for each project in list...
while (list($pid, $pname) = mysql_fetch_row($result2))
{
// go back to top of task list
mysql_data_seek($result, 0);

// print project name as first column
echo "<tr>";
echo "<td valign=top><font color=#D03468>$pname</font></td>";
$rowTotal = 0;
$count = 0;

// get intersection of this task and this project
while (list($tid, $tname) = mysql_fetch_row($result))
{
$query3 = "SELECT SUM(hours) from log WHERE pid = '$pid' AND
tid =
'$tid' AND date >= '$sdate' AND date <= '$edate'";
$result3 = mysql_db_query($database, $query3, $connection) or
die
("Error in query: $query3 . " . mysql_error());
list($sum) = mysql_fetch_row($result3);

// correction for zero values
if (!$sum) { $sum = 0; }

// keep track of totals for later use
$rowTotal = $rowTotal + $sum;
$columnTotals[$count] = $columnTotals[$count] + $sum;
$count++;

// print sum
echo "<td align=center>" . sprintf("%1.01f", $sum) . "</td>";
```

```
}

// print row total
echo "<td align=center><b>" . sprintf("%1.01f", $rowTotal) .
"</b></td>";
}
echo "</tr>";
?>

<tr>
<?
echo "<td> </td>";
$sumOfColumnTotals = 0;

// print last row - column totals
for ($x=0; $x<sizeof($columnTotals); $x++)
{
$sumOfColumnTotals = $sumOfColumnTotals + $columnTotals[$x];
echo "<td align=center><b>" . sprintf("%1.01f",
$columnTotals[$x]) .
"</b></td>";
}

// print last cell - grand total aka sum of column totals
echo "<td align=center><b>" . sprintf("%1.01f",
$sumOfColumnTotals) .
"</b></td>";

// clean up resultsets
mysql_free_result($result);
mysql_free_result($result2);
mysql_free_result($result3);
?>
</tr>
</table>
```

And here's what the finished report looks like:



Activity report for all projects for the period 11 Feb 2001 to 11 Oct 2001

Projects/Users:

	john	joe	vanessa	sherry	
XTech.com	1.0	5.0	0.0	4.0	**10.0**
Melon5ie.com	0.0	0.0	0.0	6.0	**6.0**
VideoMoz (Linux)	0.0	0.0	0.0	0.0	**0.0**
VideoMoz (Windows)	0.0	0.0	0.0	0.5	**0.5**
NanolessCorp AddSock	0.5	0.5	0.0	0.5	**9.5**
NanolessCorp invDB	0.0	0.5	0.0	0.0	**0.5**
	9.5	**6.0**	**0.0**	**11.0**	**26.5**

Projects/Tasks:

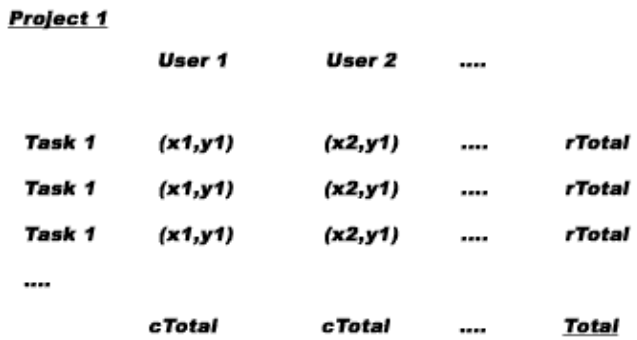
	Design	Development	System Test	Technical Support	Documentation	Training	Research	
XTech.com	5.5	0.0	0.0	4.5	0.0	0.0	0.0	**10.0**
Melon5ie.com	0.0	0.0	0.0	3.0	0.0	0.0	3.0	**6.0**
VideoMoz (Linux)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**0.0**
VideoMoz (Windows)	0.5	0.0	0.0	0.0	0.0	0.0	0.0	**0.5**
NanolessCorp AddSock	0.5	0.0	0.0	9.5	0.5	0.0	0.0	**9.5**
NanolessCorp invDB	0.5	0.0	0.0	0.0	0.0	0.0	0.0	**0.5**
	7.0	**0.0**	**0.0**	**16.0**	**0.5**	**0.0**	**3.0**	**26.5**

Suitable for printing or framing, eh?

...And The Little Brush Strokes

That takes care of the first half of the "if" loop − the "gimme−a−picture−of−all−projects" report. But while the Customer's HRD minions will be deliriously happy with this information, the Billing guys have yet to be satisfied. My next (and final) task, therefore, is to wrap things up by filling in the second half of that "if" loop and generating a focused report for a specific project across a specific time period.

Given what I've just accomplished, this should be a snap, especially if I treat it as a subset of the general report above. Here's what I anticipate it will look like:



And here's the code to accomplish it:

```
<!−− tasks vs. users table for a specific project −−>
<table width=100% border="0" cellspacing="2" cellpadding="5">
<tr>
<td> </td>
<?
// get user list
// this resultset is useful during report generation, so make
sure that it
is retained!
$query = "SELECT uid, uname FROM users";
$result = mysql_db_query($database, $query, $connection) or
die ("Error in
query: $query. " . mysql_error());

// print users in top row
while (list($uid, $uname) = mysql_fetch_row($result))
{
echo "<td valign=top align=center><font
color=#D03468>$uname</font></td>";
}

?>
```

```
<td> </td>
</tr>

<?
// create variables to hold row and column totals (useful
later)
$rowTotals = array();
$columnTotals = array();

// get list of tasks
$query2 = "SELECT tid, tname FROM tasks";
$result2 = mysql_db_query($database, $query2, $connection) or
die ("Error
in query: $query2 . " . mysql_error());

// iterate through resultset
while (list($tid, $tname) = mysql_fetch_row($result2))
{
echo "<tr>";
echo "<td valign=top align=left><font
color=#D03468>$tname</font></td>";

mysql_data_seek($result, 0);

// for each task-user combination
while (list($uid, $uname) = mysql_fetch_row($result))
{
// get intersection
$query3 = "SELECT SUM(hours) from log WHERE pid = '$pid' AND
tid = '$tid'
AND uid = '$uid' AND date >= '$sdate' AND date <= '$edate'";
$result3 = mysql_db_query($database, $query3, $connection) or
die ("Error
in query: $query3 . " . mysql_error());
list($sum) = mysql_fetch_row($result3);

// correction
if (!$sum) { $sum = 0; }

// keep track of totals
$rowTotals[$tid] = $rowTotals[$tid] + $sum;
$columnTotals[$uid] = $columnTotals[$uid] + $sum;

// print value
echo "<td valign=top align=center>" . sprintf("%1.01f", $sum)
. "</td>";
}

// print row total
```

```
echo "<td valign=top align=center><b>" . sprintf("%1.01f",
$rowTotals[$tid]) . "</b></td>";

echo "</tr>";
}
?>

<tr>
<td> </td>
<?
// back to top of user list
mysql_data_seek($result, 0);

// print column totals
while (list($uid, $uname) = mysql_fetch_row($result))
{
$sumOfColumnTotals = $sumOfColumnTotals + $columnTotals[$uid];
echo "<td valign=top align=center><b>" . sprintf("%1.01f",
$columnTotals[$uid]) . "</b></td>";
}

// print grand total
echo "<td valign=top align=center><b>" . sprintf("%1.01f",
$sumOfColumnTotals) . "</b></td>";
?>
</tr>
</table>
```

Since this is a report for a specific project, I can assume that the variable $pid will have a value other than 0. That said, the procedure is almost identical to that used in the general report, except that in this case, the users are represented by columns and the tasks by rows. As always, I'll generate the top row first, querying the table for a list of users and displaying them. Next, I get a list of tasks and, for each task−user combination for the given project, calculate the total hours worked.

Two arrays, $columnTotals and $rowTotals, indexed by user ID and task ID respectively, hold the total hours worked on both axes, and are used to build the last column and row of the table.

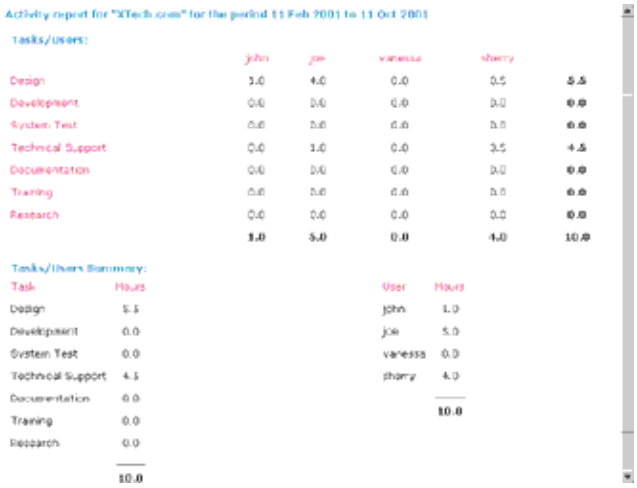
Once this report has been generated, I'd also like to print two summary reports, one listing the total hours worked by each user on the project, and the other listing the total time spent on each task within the project. These summary reports are essentially the row and column totals, which I'm displaying again to make the data easier to analyze − both the Boss and I concur that these summaries will probably be the most valuable bits of the report, as they provide a bird's−eye view of resource allocation across tasks and users in a project.

Here's the code to generate these two summary tables (remember, these are just the column and row totals which have already been calculated above):

```
<!-- tasks summary - these are the row totals -->
<table border="0" cellspacing="2" cellpadding="5">
<tr>
<td valign=top><font color=#D03468>Task</font></td>
<td valign=top align=center><font color=#D03468>Hours</td>
</tr>

<?
// iterate through task list
mysql_data_seek($result2, 0);

// for each task, get corresponding row total and print
while (list($tid, $tname) = mysql_fetch_row($result2))
{
$sumOfRowTotals = $sumOfRowTotals + $rowTotals[$tid];
echo "<tr>";
echo "<td valign=top align=left>$tname</td>";
echo "<td valign=top align=center>" . sprintf("%1.01f",
$rowTotals[$tid])
. "</td>";
echo "</tr>";
}
?>

<tr>
<td> </td>
<td valign="top" align="center"><hr><b><? echo
sprintf("%1.01f",
$sumOfRowTotals); ?></b></td>
</tr>
</table>

<!-- user hours summary - these are the column totals -->
<table border="0" cellspacing="2" cellpadding="5">
<tr>
<td valign=top><font color=#D03468>User</font></td>
<td valign=top align=center><font color=#D03468>Hours</td>
</tr>

<?
// iterate through user list
mysql_data_seek($result, 0);

// print column totals
while (list($uid, $uname) = mysql_fetch_row($result))
{
echo "<tr>";
echo "<td valign=top align=left>$uname</td>";
echo "<td valign=top align=center>" . sprintf("%1.01f",
```

```
$columnTotals[$uid]) . "</td>";
echo "</tr>";
}
?>

<tr>
<td> </td>
<td valign="top" align="center"><hr><b><? echo
sprintf("%1.01f",
$sumOfColumnTotals); ?></b></td>
</tr>

</table>
```

And here's the finished product looks like:



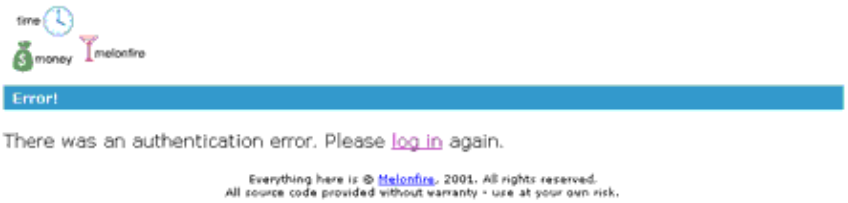
When Things Go Wrong

The last script − another extremely simple one − is the error handler, "error.php". If you look at the source code, you'll notice many links to this script, each one passing it a cryptic error code via the $ec variable. Very simply, "error.php" intercepts the variable and converts it to a human−readable error message, which is then displayed to the user.

```
<?
// error.php − displays error messages based on error code $ec

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

switch ($ec)
{
// login failure
case 0:
$message = "There was an error logging you in. <a
href=index.html>Please
try again.</a>";
break;

// session problem
case 1:
$message = "There was an authentication error. Please <a
href=index.html>log in</a> again.";
break;

// bad datestamp
case 2:
$message = "You selected an invalid date range. Please <a
href=menu.php>try
again</a>.";
break;

// default action
default:
$message = "There was an error performing the requested
action. Please <a
href=index.html>log in</a> again.";
break;
}

?>
<html>
<head>
```

```
<basefont face="Verdana">
<style type="text/css">
TD {font-family: Verdana; font-size: smaller}
</style>
</head>

<body bgcolor="White">

<? $title="Error!"; ?>
<? include("header.inc.php"); ?>

<? echo $message; ?>

<? include("footer.inc.php"); ?>

</body>
</html>
```

Here's what it looks like.



Simple and elegant − not to mention flexible. Found a new error? No problem − assign it an error code and let "error.php" know.

Happy Endings

And that just about concludes this case study. Throughout this development effort, I have made widespread use of PHP's session management capabilities, date and string functions, HTTP header functions, and database access capabilities. If you are new to PHP, I hope that the effort has been instructive, and that it has helped you gain a greater understanding of these powerful open−source tools.

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

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

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

String functions http://www.php.net/manual/en/ref.strings.phpHeader functions

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

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

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

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

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

Despite my protestations to the contrary (and my fear that I'm going to have to cut short my overly−long lunch hour once the Boss gets his hands on it), I believe that a tool such as the one described over the preceding pages offers tremendous benefits to any organization in its efforts to streamline business processes and allocate resources more efficiently. By obtaining and storing information in electronic format, it reduces paperwork and simplifies resource accounting; by imposing a structure on user information, it makes it easier and quicker to locate, present and analyze raw data; and by using a database, it ensures that data does not get corrupted.

It should be noted also that this is an initial release of the application, and I expect it to evolve further, with new features being added and old features being upgraded. It's always a good idea to review both design and code as the application evolves − I plan to do this a little further down the road, and to make changes to both the database schema and the scripts themselves. This process should take place in conjunction with the development plan for new features, so that the addition of new features does not add to overhead and cause performance degradation.

That said, let me also add that when I delivered the finished product to the Customer at his uptown office (a couple hours ahead of deadline), he was thrilled to bits, and the severe HRD minions looked delirious with delight. In fact, the Customer was so impressed that, in the five minutes I spent alone in his office, he:

offered me one of his foul−smelling cigars (which I politely declined);

offered me a job (which I also politely declined);

and offered us a long−term software development contract (which I was happy to accept on behalf of the Boss);

Needless to say, the Boss was equally thrilled with the outcome and indicated that I would shortly be receiving a bonus and some vacation time for my efforts...naturally, after executing the Customer's next project. In fact, the Boss said, he was thinking of making me point man for the Customer's entire development contract, a statement which made my blood freeze and my stomach rumble in fear of the impending ulcers...

What, you were expecting a happy ending? Get real!

Note: All examples in this article have been tested on Linux/i586 with Apache 1.3.12, mySQL 3.23 and PHP 4.06. Examples are illustrative only, and are not meant for a production environment. YMMV!