

# MySQL

## Displaying Multiple Records Per Row in a MySQL Query Result Set

Contributed by [Peter Cole](#)

2004-05-19

[\[ Send Me Similar Content When Posted \]](#)

[\[ Add Developer Shed Headlines To Your Site \]](#)



[DISCUSS](#)



[NEWS](#)



[SEND](#)

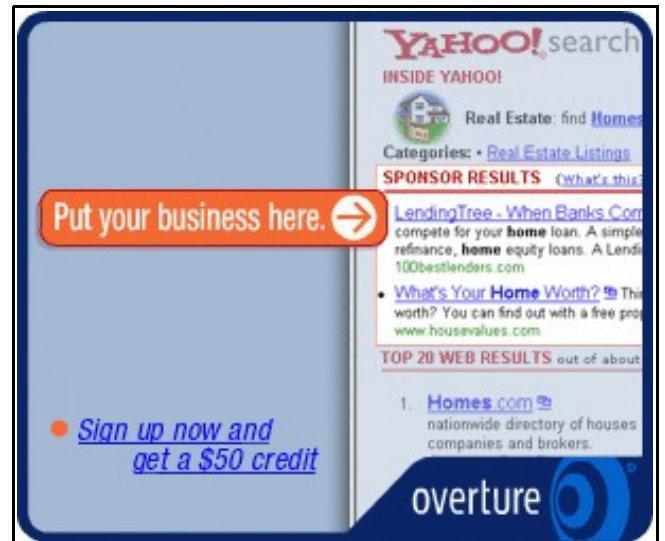


[PRINT](#)



[PDF](#)

advertisement



**Article Index:** Ever wonder how you can query a database and display the result set in something other than a one record per row layout? Keeping in mind that the simple answer is always the best one, I found a solution that keeps to that premise and solves an issue that seemed impossible not long ago. In a word, the answer, lies in the loop. An additional one, that is. But first, lets define and explain the general look and functionality of our project.

The project I was working on for a client dealt with a photo database that involved four related tables, the schema for which follows in a dump from the popular open source mysql manager phpMyAdmin: I've added a bit of extra comment to each table to give you an idea what each table is designed to do.

### *Design and Construct Database Tables*

We will assume here you understand how to create a database in MySQL. You can do so from the command line after logging into the mysql server (not a bad idea if you are logged in remotely to a 'NIX' server) by using the CREATE DATABASE command. It is also possible to use the mysqladmin utility to create the database remotely. While its a good thing to get comfortable with the command line, there are numerous mysql management applications now available in both open source and commercial varieties that can create and manage mysql databases with a lot less pain and suffering. My personal favorite is [phpmyadmin](#).

Below is a 'dump' or export of the table structure of the photogalleryDB we will be exploring in this article. Take a look at the table relationships and how the master table 'images' has hooks or match fields to the other tables by way of a key field. With the relationships in place, we can create query statements that display data based on more than one relationship.

In our example, we will be looking for all the images from the 'mountaineering' category that come from the 'huntington ravine' album and are 'located' in new Hampshire. Sound complicated? No ... not really. Read on.

```
# phpMyAdmin SQL Dump
```

```

# version 2.5.6
# http://www.phpmyadmin.net
#
# Host: localhost
# Generation Time: Apr 14, 2004 at 08:12 AM
# Server version: 4.0.16
# PHP Version: 4.3.4
#
# Database : `photogalleryDB`
#
#
-----
#
# Table structure for table `albums`
# Albums are unique to a category,
# for example a specific ice climbing location in New
# Hampshire, or
# a mountaineering location like the Icefields in the
# Canadian Rockies, Alberta

CREATE TABLE `albums` (
  `albumID` int(5) NOT NULL auto_increment,
  `albumNAME` varchar(100) NOT NULL default '',
  `albumPIC` varchar(100) NOT NULL default '',
  `catID` int(5) NOT NULL default '0',
  `locID` int(5) NOT NULL default '0',
  PRIMARY KEY (`albumID`),
  UNIQUE KEY `albumNAME` (`albumNAME`),
  UNIQUE KEY `albumPIC` (`albumPIC`)
) TYPE=MyISAM AUTO_INCREMENT=3 ;

#
-----
#
# Table structure for table `categories`
# Acitivity type ... skiing, rock climbing, ice
# climbing etc

CREATE TABLE `categories` (
  `ID` int(5) NOT NULL auto_increment,
  `catID` int(5) NOT NULL default '0',
  `catNAME` varchar(100) NOT NULL default '',
  PRIMARY KEY (`ID`),
  UNIQUE KEY `catID` (`catID`)
) TYPE=MyISAM AUTO_INCREMENT=15 ;

#
-----
#
# Table structure for table `images`
# The guts of the database

```

```

CREATE TABLE `images` (
  `imgID` int(5) NOT NULL auto_increment,
  `catID` int(5) NOT NULL default '1',
  `albumID` int(5) NOT NULL default '1',
  `locID` int(5) NOT NULL default '1',
  `imgTITLE` varchar(200) NOT NULL default '',
  `thumbPATH` varchar(150) NOT NULL default '',
  `thumbNAME` varchar(100) NOT NULL default '',
  `popPATH` varchar(150) NOT NULL default '',
  `popNAME` varchar(100) NOT NULL default '',
  `copyright` int(4) NOT NULL default '0',
  PRIMARY KEY (`imgID`),
  UNIQUE KEY `popNAME` (`popNAME`),
  UNIQUE KEY `thumbNAME` (`thumbNAME`)
) TYPE=MyISAM AUTO_INCREMENT=27 ;

```

```
#
```

```
-----
#
```

```
# Table structure for table `location`
# State or Province
```

```

CREATE TABLE `location` (
  `ID` int(5) NOT NULL auto_increment,
  `locID` int(5) NOT NULL default '0',
  `locNAME` varchar(100) NOT NULL default '',
  PRIMARY KEY (`ID`),
  UNIQUE KEY `locID` (`locID`)
) TYPE=MyISAM AUTO_INCREMENT=26 ;

```

The concept involved in the table schema is to try to a certain degree to 'normalize' the database structure. What that means in a nutshell is that we don't want to end up with a database that has redundant data found in more than one place.

The table structure is independent of the logic that is used to display the results of queries necessary to add, edit, view and delete records. In this article, we are only interested in the code that will display the results of a SELECT query.

There has been more than enough conversation in forums and articles past about the advantages or lack there-of of including your images (binary data) in the database tables themselves. It boils down to personal preference. My personal feeling is that your images are best keep out of the database. Use the data in your tables to 'reference' the location of the folder on your server or hard drive where the images can be found.

The first bit of code necessary to get going is a connection script. It is a good idea to configure MySQL to grant the connection user a minimum set of rights. In this case we only need the right to query the database, so the user 'connect' has no rights in the mysql.user table and select rights only in the mysql.db table record that refers to the database 'photogalleryDB'. If someone sniffs out your password, what can they do? Not much with select rights. Take a look around like the rest of us... that's all.

```
<?php
// saved as the include file db_config.php
$db_name="photogalleryDB";
$conn = @mysql_connect("localhost", "connect",
"yourpassword")
    or die ("Could not connect to MySQL server");
$db = @mysql_select_db($db_name, $conn)
    or die("Could not select your database.");
?>
```

### *Albums and Categories*

Next up is the query that will select the available albums in a particular category. In this case we are interested in seeing gallery albums from the 'mountaineering' category. In our project, the CAT link is hard coded into the web page that deals with mountaineering and is just one of the menu links on that page.



The URL to query and display the result might look something like this  
[http://localhost/gallery/image\\_gallery.php?CAT=3](http://localhost/gallery/image_gallery.php?CAT=3)

The code needed to find the available albums in the chosen category is as follows.

```
<?php
require("../includes/db_config.php");
if(isset($_GET['CAT'])) {
    // clean up query string
    $CAT = htmlspecialchars($_GET['CAT']);
    // get image count in each album in the chosen
category
    $sql = "
SELECT
    categories.catNAME,
    albums.catID,
    albums.locID,
    albums.albumID,
    albums.albumNAME,
    albums.albumPIC,
    images.thumbPATH,
    location.locNAME,
COUNT(images.albumID) AS NUM
FROM categories, albums, images, location
WHERE images.catID = '$CAT'
    AND images.albumID = albums.albumID
    AND albums.locID = location.locID AND
```

```

categories.catID = '$CAT'
GROUP BY albums.albumNAME
ORDER BY albums.albumNAME ASC";

$result = @mysql_query($sql, $connection);
}
? >

```

The results of the above query will select the available albums in a particular category. In this case we have asked the database to tell us what albums are available in the 'mountaineering' category where \$CAT==3 The value of 3 represents a record in the 'categories' table called 'mountaineering'.



The resulting page gives us a choice of both vertical and horizontal display of the images of the 'Huntington Ravine' album. In our case there is only 1 album in which \$CAT == 3. If there were more, the page would show additional rows or 'albums' to choose from. The html and php code to display this table looks like this:

```

<table width="680" border="0" cellpadding="3"
cellspacing="0">
<tr>
<td valign="top">

<br clear="all">
<div style="margin-right:15px; margin-left:15px;
padding:8px;">
<table width="650" border="0" cellpadding="3"
cellspacing="0">
<?php
// some logic to deal with what ifs and people with
bad intentions
if(!isset($num)) {
print '<tr><td align="center" colspan="4">Your
query did not return any results</td></tr>';
}
if(!empty($num)) {
// build header row for display of images
?>
<tr>
<th>&nbsp;</th>

```

```

        <th>Gallery Name</th>
        <th>Category / Location</th>
        <th>Image Count</th>
    </tr>

<?php
while ($row = @mysql_fetch_array($result)) {
    $albumNAME =stripslashes($row['albumNAME']);
    $AID = $row['albumID'];
    $CAT = $row['catID'];
    $LOC = $row['locID'];
    $count = $row['NUM'];
    $category = $row['catNAME'];
    $location = $row['locNAME'];
    $path = $row['thumbPATH'];
    $intropic = $row['albumPIC'];
    // solitary picture seen in available albums in each
category

if(!empty($row)) {
    $display_block = "
        <tr>
            <td></td>
            <td width="335" align="left">
                <strong>$albumNAME</strong><br>
                <div style="margin-left:15px;">
                    <a
href="show_pictures_v.php?CAT=$CAT&AID=$AID&LOC=$LOC">Vertical
Display</a>
                </div>
                <div style="margin-left:15px;">
                    <a
href="show_pictures_h.php?CAT=$CAT&AID=$AID&LOC=$LOC">Horizontal
Display</a>
                </div>
            </td>
            <td width="165" align="center">$category /
$location</td>
            <td width="150" align="center">Contains:
<strong>$count</strong> images</td>
        </tr>";

print $display_block;
    }
}
}
?>
</table>
</div>

```

```

        <br clear="all">
    </td>
</tr>
</table>

```

What we are really interested in is the link that displays the images in the gallery in a horizontal format.

The URL in the code above looks like this:

```
http://localhost/gallery/show_pictures_h.php?CAT=3&AID=2&LOC=2
```

We start once again with the code needed to connect to the mysql server and request info from the query string attached to the URL. It is important to clean up query strings before you submit a request to the database. One of the simplest and most effective means to do so is to use the built in PHP 'htmlspecialchars' function which prevents html markup from adding dangerous code to your query. The function has more capabilities then presented here, but in general it does the following.

- & (ampersand) becomes '&'
- " " (double quote) becomes '&quot;';' when ENT\_NOQUOTES is not set.
- ' ' (single quote) becomes '&#039;';' only when ENT\_QUOTES is set.
- < (less than) becomes '&lt;';'
- > (greater than) becomes '&gt;';'

```

<?php
require("../../includes/db_config.php"); // connect
to mysql server from protected directory
if(isset($_GET['CAT']) && ($_GET['AID'])) {
    $CAT = htmlspecialchars($_GET['CAT']); // clean up
query string variable
    $AID = htmlspecialchars($_GET['AID']); // clean up
query string variable
    $LOC = htmlspecialchars($_GET['LOC']); // clean up
query string variable

    // get album name first to populate title bar
    $sql_T = "SELECT albums.albumNAME FROM albums WHERE
catID = '$CAT' AND albumID = '$AID' ";
    $sql_T_result = @mysql_query($sql_T, $connection)
    or die ("Could not execute your select albumNAME
request");
    $row_T = @mysql_fetch_array($sql_T_result);
    $NAME = stripslashes($row_T['albumNAME']);

    // get images from album
    $sql = "
SELECT
    albums.albumNAME,

```

```

        images.imgID,
        images.catID,
        images.albumID,
        images.locID,
        images.imgTITLE,
        images.thumbPATH,
        images.thumbNAME,
        images.copyright
    FROM images, albums
    WHERE images.catID = '$CAT' AND images.albumID =
'$AID' AND images.albumID = albums.albumID
    ORDER BY copyright, thumbNAME";

    $result = @mysql_query($sql, $connection);
    if(!$result) {
        print "Could not execute your select images
request";
        exit;
    }

    $num = @mysql_num_rows($result);
    // you choose how many columns you want to display in
each table row
    $thumbcols = 5;
    // quick and dirty formula to figure out how many
rows you will need
    $thumbrows = 1+ round($num / $thumbcols);
    }
? >

```

So now we get to the crux of our discussion. How do we get the query results to display horizontally?

Good question and one that took a day of tinkering to figure out. There were a few things we did know. First off, we decided how many images per row we wished to see. That value is stored in the `$thumbcols` variable. By dividing the `$num` variable that the query returns (number of rows in the result set) by the value of `$thumbcols`, we get the number of rows the html will display. In this case we just happen to have 11 images in the 'Huntington Ravine' album and since `$thumbcols == 5`, we need 3 rows. But, because the division returns a number closer to 2 than 3, the PHP `round` function rounds down to 2. By adjusting the variable definition to add 1 to the round results, we get around the closer to 2 than 3 problem.

But you say, what if the round function returns a whole number or a number closer to 3, won't there be an extra row added to the display? Not if we use a conditional to write out a table cell only if there is actually an image to display. This simple bit of code prints only what is needed to display all the images in the result set.

```

        if(!empty($image)) {
            // echo the actual data to the screen
            print

```

```

"<a
href="enlarge_image.php?ID=$ID&CAT=$catID&AID=$albumID">
  
</a>
<br clear="all">
$caption";
}
else {
print '&nbsp;';
}

```

Simple enough. Then at last we come to the code needed to put a result set row into a table cell and then write out five table cells per html row. We'll start with a bit of fail safe code in case we find ourselves arriving at `http://localhost/gallery/show_pictures_h.php` without a query string attached to the URL.

```

<table width="680" border="0" cellpadding="3"
cellspacing="0">
<tr>
<td valign="top" align="center">
<!-- gallery logo -->

<br clear="all">
<div style="margin-right:15px; margin-left:15px;
padding:8px; line-height:120%;">
<div align="center">
<strong>

<?php
// just in case no query string is part of the URL
request, the page will display an alert to pick a
category
// which is always displayed as part of the page
header
print
(isset($NAME) ? $NAME. ' - (Click to Enlarge)' :
'Please select a category to view');
?>

</strong>
</div>
<?php
print '<table width="650" border="0" cellpadding="5"

```



```

cellspacing="0">';
if(!empty($num)) {
    // if the query is successful print out a row of
table beta
    print '
        <tr><td colspan="5" bgcolor="#cdcdcd">
            <strong>DATABASE RETURNED => # of
Rows:'. $thumbrows. ' and # of Images:'. $num.'</strong>
        }
        </td>
    </tr>';
// function to display multiple table cells before
starting a new row
function display_table() {
    // make variables available outside of the function
    global $num, $result, $thumbrows, $thumbcols;
    // format the number of rows to be printed using a
loop
    for($r=1;$r<=$thumbrows;$r++) {
        print '<tr>';
        // format the number of columns to be printed in each
row using a 2nd for loop
        for($c=1;$c<=$thumbcols;$c++) {
            print '<td bgcolor="#999999" align="center"
valign="top">';
            $row = @mysql_fetch_array($result); // break out the
array of values from the returned query
            $ID = $row['imgID'];
            $catID = $row['catID'];
            $albumID = $row['albumID'];
            $location = $row['locID'];
            $albumNAME = $row['albumNAME'];
            $caption = stripslashes($row['imgTITLE']);
            $path = $row['thumbPATH'];
            $image = $row['thumbNAME'];
            $copyright = $row['copyright'];

if(!empty($image)) {
    // echo the actual data to the screen
    print"
        <a
href="enlarge_image.php?ID=$ID&CAT=$catID&AID=$albumID">
            
        </a>
        <br clear="all">
        $caption";
    }
else {
    print '&nbsp;';
}
}
}

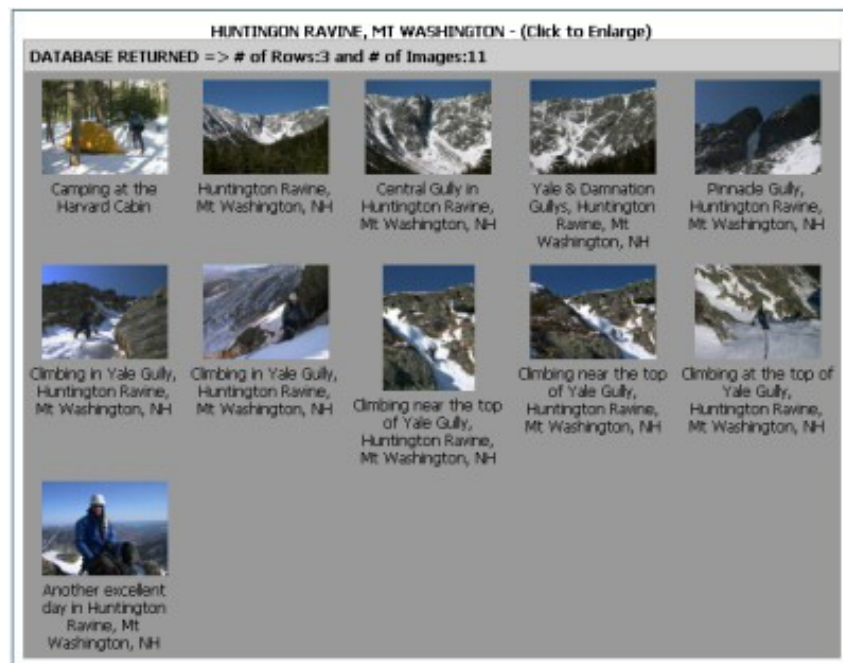
```

```

        print '</td>';
    }
    print '</tr>';
}
}
display_table(); // call function to do this wonderful
thing
print '</table>';
?>
    </div>
    <br clear="all">
    </td>
</tr>
</table>

```

Between the header and footer in `show_pictures_h.php`, the php code and html shown above will parse to look like this:



The trick, discovered only after repeated dead ends is the nested loop. The first for loop finds out how many rows to display and prints the `<tr>` tags. The second nested loop finds out how many `<td>` tags to print in each row. And because we already defined this to equal to 5, the inside loop prints out 5 table cells before giving control back over to the outside loop which then prints another row (if need be) and then lets the inside loop do its thing again until both loops are satisfied and no longer return a true value.

Because the data we want to display resides in the table cells, we don't break out the array values that contain all that data until we are inside the 2nd loop.

Cha ching! In our case we only wanted the thumbnail image and the caption. Each thumbnail however is surrounded with a hyperlink that requests something along these lines:

[http://localhost/gallery/enlarge\\_image.php?ID=22&CAT=3&AID=2](http://localhost/gallery/enlarge_image.php?ID=22&CAT=3&AID=2)

Clicking on the thumbnail image opens another page with a larger version of the image. Nothing special here... just another query that displays the results of the SELECT string parameters (imageID, catalogID and albumID) attached to the thumbnail hyperlink. The code behind the enlarge\_image.php page is almost identical to what we have looked at here. The only difference is in the html and the CSS used to display the one large image rather than loop thru an array of returned rows.



You may see all of this in action at [Mooney Mountain Guides](#). Please be aware that all mountaineering images displayed here are © 2004 by Art Mooney.

Hope this helps you with your PHP/MySQL projects and remember, keep it simple.