By Adrian Portsmouth

Table of Contents

Introduction

As a PHP and MySQL Developer I browse around many forums and one of the more common questions being asked is ’How do I Add Next / Prev Links to my Pages?’. Even I asked this very question a long time ago when I was beginning to learn PHP. I decided to write this article because despite learning how to do this myself, I never really found a decent article on the Internet that was geared more towards a beginner and that was suitable for PHP Installations that are now configured with Register Globals Off.
I have attempted to write an article that explains even the basic areas, that walks through every line of the code explaining what it does and why it is there, so this is what I have attempted to do with this article. Hopefully by the time you reach the end you will feel confident enough to write your own code without even using this article as a reference guide.
The article is based on a MySQL database and assumes a basic knowledge of constructing SQL Queries. I will go through connecting and disconnecting from the database and some other clauses and statements. This should be fairly easy to port to other database platforms if required.

Getting Started

So lets get started going through the various parts of the code. At the end I will display it all together and you should be able to read and understand what it is doing.
The first thing to do is set the variables that the script will need in order to work correctly. The variables may change at some point, so we put them at the top of the script where they are easy to find if we need to change them. If they are used multiple times within the script we will only need to edit them in one place.

```
// Set Script Variables
$DB_Host="localhost";
$DB_Name="MyDataBase";
$DB_User="MyUserName";
$DB_Pass="MyPassword";
$Per_Page=10;
```

As I said these variables will be used in various stages of the script. We will now go through them one by one. $DB_Host should contain the host name where your MySQL resides, in most cases this is ’localhost’. The next three variables are used to connect to your MySQL database; $DB_Name will be the name of the database, $DB_User should contain your MySQL username and $DB_Pass should contain your MySQL password. Finally $Per_Page, this variable defines how many results will be displayed on each page. In this article we are going to display 10 results per page, but you can change this figure to any number that suits your system and layout.

Making a Connection

Now we have set the variables the next stage is to open a connection to MySQL, this should be done as follows:

```
// Open MySQL Connection
$Connection=mysql_connect($DB_Host, $DB_User, $DB_Pass);
```

We use the mysql_connect() function to open the connection using the variables we defined at the beginning of the script. The link identifier will then be assigned to the variable $Connection which will be used later in the script to obtain our result sets and also when we close the connection to MySQL.
The next step is to run a count query without any limits thus giving us the total number of results, we will use this total number in conjunction with our $Per_Page variable to create the Next / Prev links as appropriate.

```
// Run The Query Without a Limit to get Total result
$SQL="SELECT COUNT(*) AS Total FROM Products WHERE Description LIKE
'".$_REQUEST['Keyword']."'";
$SQL_Result=mysql_db_query($DB_Name, $SQL);
$SQL_Result_Array=mysql_fetch_array($SQL_Result);
$Total=$SQL_Result_Array['Total'];
```

The first line is the SELECT statement which tells MySQL what data we are looking for. In this article we are selecting data from a table called 'Products' (Full Table Schema can be found at the end of this page). We use the COUNT(*) function and assign the result AS Total. In some tutorials you may have noticed that instead they use the mysql_num_rows() function. We opted for the COUNT() function due to it's speed. Our WHERE clause indicates that this table has a column called 'Description'. In other words a product description. And we are searching the Description column for all records that contain something LIKE the value of $_REQUEST['Keyword']. We therefore assume that $_REQUEST['Keyword'] is passed to this page via a post form or through the URL, so what we are simulating in this article is a Keyword search.
The second line actually runs the query on the database using the mysql_db_query() function, we first pass the database name and then the SQL query itself. This runs the query and the result identifier is assigned to the $SQL_Result variable. In the third line we use the mysql_fetch_array() function to retrieve the data from the Result. In the fourth line we assign the value to the Total variable.
Now that we have the total result we can reset the SQL variable with a new query and begin to append to the original SQL statement. In order to do this we need to run an if statement that checks the value of $Result_Set, we have not yet touched on this variable yet so I will now explain.
The variable $Result_Set is used to define where to start retrieving results, we pass this variable across through the Next / Prev URL's which we create later in the script and when it reaches this point in the script we use the value in a LIMIT clause. This is probably a little confusing for beginners so I will take us away from the article and explain in a little more depth the LIMIT clause.

Limiting the Dataset

When we issue a LIMIT clause to MySQL we pass across two parameters, you can pass just a single parameter but we need to use them both for this type of script. This is an example of an SQL statement with a LIMIT clause:

```
// Example, this is NOT used in our script
$SQL="SELECT * FROM Products WHERE Description LIKE
'".$_REQUEST['Keyword']."' LIMIT 10, 20";
```

Lets say for example sake that this query when run without the LIMIT clause gives us 30 records in our result, by entering the LIMIT clause it is telling MySQL to start at row 10 and retrieve 20 records. So it will skip records 1 through to 9 and return 10 through to 30.
Ok back to the article, here is the if statement and SQL we will be using:

```
// Create a new SELECT Query with the ORDER BY clause and without
the COUNT(*)
$SQL="SELECT * FROM Products WHERE Description LIKE
'%".$_REQUEST['Keyword']."%' ORDER BY ProductID";
// Append a LIMIT clause to the SQL statement
if (empty($_GET['Result_Set']))
{
$Result_Set=0;
$SQL.=" LIMIT $Result_Set, $Per_Page";
}else
{
$Result_Set=$_GET['Result_Set'];
$SQL.=" LIMIT $Result_Set, $Per_Page";
}
```

Because we no longer need the original SELECT query we can assign a new one using the same $SQL variable, our new query above no longer contains the COUNT() function because we actually want to return the data from the table and it also contains an ORDER BY clause using the ProductID, it is important to include the ORDER BY clause to ensure that each time a next or prev link is clicked, the query sorts the data in the same order. Without an ORDER BY statement is pretty much useless creating a page system.
Ok, so the first line of the if statement simply says that if $_GET['Result_Set'] is not present, it has not been passed to the script. Next we run the first {} loop, if it is present then we run the else {} loop. Because we need to pass two parameters with the LIMIT clause the first thing we do in the first of the loops (Line Three) is to assign $Result_Set a value of 0. As per our previous example this means the LIMIT clause will return results from row 0 of the result set. The next line appends our LIMIT clause to our SQL statement, this is done by using a period prior to the equals sign. This tells PHP that we are not creating or overwriting the existing variable but appending to it. (Notice the space between the first double

quote and the first letter of LIMIT) So if we were running this script right now and $Result_Set has no value, then it will go through the first of our loops and our variable $SQL will now look like this:

```
$SQL="SELECT * FROM Products WHERE Description LIKE ’$Keyword’ LIMIT
0, 10";
```

The else {} loop will come into play when the variable $_GET[’Result_Set’] does have a value (If someone has clicked on one of our Next / Prev links) in this instance we set $Result_Set with the value passed in the URL.
Now that we have completed appending to our SQL statement we now need to run the statement like so:

```
// Run The Query With a Limit to get result
$SQL_Result=mysql_db_query($DB_Name, $SQL);
$SQL_Rows=mysql_num_rows($SQL_Result);
```

This code should look familiar as we used it earlier when we ran the first SQL statement therefore I do not need to explain the functions again but moreover the logic. This code will run our query and then count the results, you are probably thinking, ’Why count them when we have specifically said how many results to select?’ Well, the reason we need to count them is in-case there are an odd number of results. If our original query returns 53 results, and the user has clicked our Next buttons five times and we are displaying 10 records per page, then we need to know that we only need to display 3 records on this the last page.

Display the Data

We should now display our results to the user, this is the area where you will need to make a lot of changes in order to display your data, but I will include an example based on our article’s database of products:

```
// Display Results using a for loop
for ($a=0; $a < $SQL_Rows; $a++)
{
$SQL_Array=mysql_fetch_array($SQL_Query);
$Product=$SQL_Array[’Name’];
$Description=$SQL_Array[’Description’];
echo "$Product - $Description<BR><BR>";
}
```

The first line of this code sets the conditions of the loop, $a starts off with the value of 0, while the value of $SQL_Rows is greater than the value of $a then the loop will run, however the last condition ’$a++’ changes the value of $a so that each time it goes through the loop it adds 1 to the value of $a. Thus eventually the value of $a will match the value of $SQL_Rows and the loop will no longer run. The three lines of code within the loop start by retrieving the data from the result set and assign the values to a variable before echoing them out to the screen. The next step is to produce the Next / Prev links and display them as the script dictates. We also need to pass across the URL anything which is included in the SQL Statement, in this case $_REQUEST[’Keyword’] and also $Result_Set which we will define during the creation of the Next / Prev links, you will need to modify the links if you pass any other variables across which are used in the SQL Statement. For example if you add a product type to the table and your search form then you will need to append it to each of the links below so that it is carried through the URL for the next page to use in the query.

```
// Create Next / Prev Links and $Result_Set Value
if ($Total>0)
{
if ($Result_Set<$Total && $Result_Set>0)
{
$Res1=$Result_Set-$Per_Page;
echo "<A
HREF=\"test.php?Result_Set=$Res1&Keyword=".$_REQUEST[’Keyword’]."\"><;<;
Previous Page</A> ";
}
// Calculate and Display Page # Links
$Pages=$Total / $Per_Page;
if ($Pages>1)
{
for ($b=0,$c=1; $b < $Pages; $b++,$c++)
{
```

```
$Res1=$Per_Page * $b;
echo "<A
HREF=\"test.php?Result_Set=$Res1&Keyword=".$_REQUEST['Keyword']."\">$c</A> \n";
}
}
if ($Result_Set>=0 && $Result_Set<$Total)
{
$Res1=$Result_Set+$Per_Page;
if ($Res1<$Total)
{
echo " <A
HREF=\"test.php?Result_Set=$Res1&Keyword=".$_REQUEST['Keyword']."\">Next
Page >></A>";
}
}
}
```

The first thing to take notice of in this code are the three echo statements, these make up the URL that will be printed. In this code we have named the file result.php that means you should name the file you are creating result.php, if you name it something else then you should change the name in all of the above echo statements.
As you can probably gather this is the more complex part of this script, the first line starts us off with an if statement, this checks to see if we have any records at all in our result. So if the value of $Total is not greater than 0 we obviously have no records in our result and we will not need to display Next / Prev links. If we do have some results then the second if statement (line 3) will determine if we need to display a Prev link, obviously if this is the first page of the result set then we don't want to display a Prev Link. The condition of this if statement checks to see if the value of $Result_Set is less than the value of $Total and greater than 0 then we will display a Prev link, if $Result_Set is not less than $Total and greater than 0 then we are on the first page of our results and do not need a Prev link. Line 5 of the code is the math which creates the value of $Result_Set for the Prev Link if it is being displayed. This simply takes the value of $Result_Set and subtracts the value of $Per_Page which we defined at the top of the script, we assign this value to the variable $Res1 and then echo the value in the URL as the new $Result_Set value for the Prev link (line 6). (Remember to append any variables you are passing through the scripts to all of the URL's that we echo out).
Line 9 is the start of the calculation for displaying page numbers for our result set, so a user can go directly to the page they think is most relevant to what they want. This is done by first calculating how many pages we need to display, we use the value of $Total and divide it by the value of $Per_Page. We then start a new if statement and check that there are more than 1 pages because we don't need to display any page numbers at all if there is no more than one page. The next step is a rather complex for statement (line 12). We define two variables $b and $c, both of these variables increment by 1 each time the loop is processed so the only difference is that $b starts at 0 and $c starts at 1. The reason being that we need a zero value in order to loop the correct number of times against the value of $Pages but we don't want our page number display to start at 0, instead we want it to start at 1 so we use the $c variable as the number we display in our page. The $Res1 calculation in this if statement is very simple, we take the value of $b and multiply it by the value of $Per_Page before using it in our URL.
The fourth if statement first checks that $Result_Set is equal or greater than 0, this indicates that we need a

Next link, it also checks that the value of $Result_Set is less than the value of $Total as we don’t want to display a link if there are no more results to display. Both of these must be true before a Next link will be displayed. The script then creates a value for $Res1 which will be passed across the URL in the Next link, again some simple math, we take the value of $Result_Set and add the value of $Per_Page. So if we are currently displaying results 20 through to 30 in our next page we need to start from 30 and display through to 40 (if there are that many records). So our $Result_Set is currently set to 20 and our $Per_Page is set to 10, add them together and we now have 30 assigned to $Res1. We then run one more if statement to check that the new value of $Res1 is less than the value of $Total before displaying the Next link. If $Res1 now equals 30 in our example situation and $Total only equals 29 then we do not want to display a link, but the fourth if statement will come through as true, so the fifth if statement is there to prevent the link being displayed in this situation.
The last thing we do in our script is close the MySQL connection, you should remember to do this each time you open a connection to MySQL. This is done as follows:

```
// Close Database Connection
mysql_close($Connection);
```

Notice how we use the variable $Connection which we assigned earlier in the script.

Conclusion

And that is all there is to it. The complete script using our example database will look like this (In this article we have named this file result.php, this is significant in echoing for the next prev and numbered links):

```
<?
// Set Script Variables
$DB_Host="localhost";
$DB_Name="MyDataBase";
$DB_User="MyUserName";
$DB_Pass="MyPassword";
$Per_Page=10;
// Open MySQL Connection
$Connection=mysql_connect($DB_Host, $DB_User, $DB_Pass);
// Run The Query Without a Limit to get Total result
$SQL="SELECT COUNT(*) AS Total FROM Products WHERE Description LIKE
’%".$_REQUEST[’Keyword’]."%’";
$SQL_Result=mysql_db_query($DB_Name, $SQL);
$SQL_Result_Array=mysql_fetch_array($SQL_Result);
$Total=$SQL_Result_Array[’Total’];
// Create a new SELECT Query with the ORDER BY clause and without
the COUNT(*)
$SQL="SELECT * FROM Products WHERE Description LIKE
’%".$_REQUEST[’Keyword’]."%’ ORDER BY ProductID";
// Append a LIMIT clause to the SQL statement
if (empty($_GET[’Result_Set’]))
{
$Result_Set=0;
$SQL.=" LIMIT $Result_Set, $Per_Page";
}else
{
$Result_Set=$_GET[’Result_Set’];
$SQL.=" LIMIT $Result_Set, $Per_Page";
}
// Run The Query With a Limit to get result
$SQL_Result=mysql_db_query($DB_Name, $SQL);
$SQL_Rows=mysql_num_rows($SQL_Result);
// Display Results using a for loop
for ($a=0; $a < $SQL_Rows; $a++)
{
$SQL_Array=mysql_fetch_array($SQL_Result);
$Product=$SQL_Array[’Name’];
$Description=$SQL_Array[’Description’];
echo "$Product - $Description<BR><BR>";
}
// Create Next / Prev Links and $Result_Set Value
if ($Total>0)
{
if ($Result_Set<$Total && $Result_Set>0)
{
$Res1=$Result_Set-$Per_Page;
echo "<A
HREF=\"test.php?Result_Set=$Res1&Keyword=".$_REQUEST['Keyword']."\"><;<;
Previous Page</A> ";
}
// Calculate and Display Page # Links
$Pages=$Total / $Per_Page;
if ($Pages>1)
{
for ($b=0,$c=1; $b < $Pages; $b++,$c++)
{
$Res1=$Per_Page * $b;
echo "<A
HREF=\"test.php?Result_Set=$Res1&Keyword=".$_REQUEST['Keyword']."\">$c</A> \n";
}
}
if ($Result_Set>=0 && $Result_Set<$Total)
{
$Res1=$Result_Set+$Per_Page;
if ($Res1<$Total)
{
echo " <A
HREF=\"test.php?Result_Set=$Res1&Keyword=".$_REQUEST['Keyword']."\">Next
Page >></A>";
}
}
}
// Close Database Connection
mysql_close($Connection);
?>
```

To create the database table we have used in this article you can use the following MySQL:

```
CREATE TABLE Products (
ProductID int(10) auto_increment NOT NULL PRIMARY KEY,
Name varchar(100),
Description text
);
```