Searching with PHP and MySQL

By Cal Henderson

So you have a PHP/MySQL website, and you want to let users search for things. Easy!

```
$search_term_esc = AddSlashes($search_term);

$sql = "SELECT * FROM Content WHERE content_body LIKE '%$search_term_esc%'";
```

Great. Only not really. If I enter '%' or '_' as my search term, it'll match every piece of content you have. And if someone enters 'e' then it'll match every article with an 'e' in it. And if someone enters ' foo bar' it'll match the phrase, not the two words. It looks like we have a lot to learn.

Step one is breaking the terms apart. We could do this with a simple explode, but wouldn't it be cool if we could also allow phrases by accepting quotes? One easy way to achieve this is with a simple three step routine:

Convert whitespace between brackets into something non-whitespacey. Split on whitespace. Convert the 'something' back to the whitespace it was, for each token. For the first stage, we can use my favourite feature of regular expressions - the 'e' (evaluation) flag.

```
$terms = preg_replace("/\"(.*?)\"/e", "search_transform_term('\$1')", $terms);

function search_transform_term($term){
 $term = preg_replace("/(\s)/e", "'{WHITESPACE-'.ord('\$1').'}'", $term);
 $term = preg_replace("/,/", "{COMMA}", $term);
 return $term;
}
```

So what happens here exactly? The first preg_replace() call find all pair of brackets and passes their contents to the search_transform_term() function for processing. We then use a further two preg_replace() calls to replace any whitespace with a holder token, and the same with commas. We process commas incase users enter comma seperated terms, instead of space seperated ones.

With this done, we're ready to split on whitespace and commas. We'll use preg_split() for this:

```
$terms = preg_split("/\s+|,/", $terms);
```

For each term, we then want to replace the holding tokens back with their original contents.

```
$term = preg_replace("/\{WHITESPACE-([0-9]+)\}/e", "chr(\$1)", $term);
$term = preg_replace("/\{COMMA\}/", ",", $term);
```

So now we have a full function for splitting up search terms:

```
function search_split_terms($terms){

 $terms = preg_replace("/\"(.*?)\"/e", "search_transform_term('\$1')", $terms);
 $terms = preg_split("/\s+|,/", $terms);

 $out = array();

 foreach($terms as $term){
```

```
   $term = preg_replace("/\{WHITESPACE-([0-9]+)\}/e", "chr(\$1)", $term);
   $term = preg_replace("/\{COMMA\}/", ",", $term);

   $out[] = $term;
  }

  return $out;
 }

 function search_transform_term($term){
  $term = preg_replace("/(\s)/e", "'{WHITESPACE-'.ord('\$1').'}'", $term);
  $term = preg_replace("/,/", "{COMMA}", $term);
  return $term;
 }
```

On to stage two - doing a useful search with each term. Instead of using MySQL's LIKE operator, we'll use it's much more powerful friend, RLIKE, which lets us use regular expressions.

MySQL's regular expressions are quite perl-like, but not entirely. The first thing we need is an escaping function, since some of the characters in the search term might contain a MySQL regular expression meta-character.

```
 function search_escape_rlike($string){
  return preg_replace("/([.\[\]*^\$])/", '\\\$1', $string);
 }
```

This function inserts a slash before each meta-character that MySQL uses. Next we need to make sure each term matches on word boundaries so that 'foo' matches 'a foo a' but not 'a food a'. In MySQL syntax, these are '[[:<:]]' and '[[:>:]]'. We'll write a function to turn our list of terms into a list of regular expressions.

```
 function search_db_escape_terms($terms){
  $out = array();
  foreach($terms as $term){
   $out[] = '[[:<:]]'.AddSlashes(search_escape_rlike($term)).'[[:>:]]';
  }
  return $out;
 }
```

So now we're ready to build some SQL. I'm assuming a single content table with a single field to match. You can figure out how to expand it to something more complicated yourself.

```
 $terms = search_split_terms($terms);
 $terms_db = search_db_escape_terms($terms);

 $parts = array();
 foreach($terms_db as $term_db){
  $parts[] = "content_body RLIKE '$term_db'";
 }
 $parts = implode(' AND ', $parts);
```

```
$sql = "SELECT * FROM Content WHERE $parts";
```

With this function, the search string 'foo bar' generates this SQL:

```
SELECT * FROM Content WHERE content_body RLIKE '[[:<:]]foo[[:>:]]'
 AND content_body RLIKE '[[:<:]]bar[[:>:]]';
```

You can replace the 'AND' with an 'OR' if you want to match pages which match any of the terms, rather than all.

Maybe you want to rank the results by how many times the search terms appear in them - that's what users often expect.

```
$terms_rx = search_rx_escape_terms($terms);

$rows = array();

$result = mysql_query($sql);
while($row = mysql_fetch_array($result, MYSQL_ASSOC)){

 $row[score] = 0;

 foreach($terms_rx as $term_rx){
  $row[score] += preg_match_all("/$term_rx/i", $row[content_body], $null);
 }

 $rows[] = $row;
}

uasort($rows, 'search_sort_results');


function search_rx_escape_terms($terms){
 $out = array();
 foreach($terms as $term){
  $out[] = '\b'.preg_quote($term, '/').'\b';
 }
 return $out;
}


function search_sort_results($a, $b){

 $ax = $a[score];
 $bx = $b[score];

 if ($ax == $bx){ return 0; }
 return ($ax > $bx) ? -1 : 1;
}
```

Here we use preg_match_all() to find the number of matches for each term in the content. Our

search_rx_escape_terms() turns each of the terms into a preg style regular expression. After stashing each of the matching results in $rows, along with its' score, we use usort() to sort the results using a custom function.

We have a final touch left - displaying the list of search terms back to the user with the results. For this, we want to show quoted terms and display them in a "pretty" list:

```
 $terms_html = search_html_escape_terms($terms);

function search_html_escape_terms($terms){
 $out = array();

 foreach($terms as $term){
  if (preg_match("/\s|,/", $term)){
   $out[] = '"'.HtmlSpecialChars($term).'"';
  }else{
   $out[] = HtmlSpecialChars($term);
  }
 }

 return $out;
}

function search_pretty_terms($terms_html){

 if (count($terms_html) == 1){
  return array_pop($terms_html);
 }

 $last = array_pop($terms_html);

 return implode(', ', $terms_html)." and $last";
}
```

Understanding the above function is left an an exercise for the reader ;)

So now we have our full suite of search code - let's bring it all together:

```
 function search_split_terms($terms){

 $terms = preg_replace("/\"(.*?)\"/e", "search_transform_term('\$1')", $terms);
 $terms = preg_split("/\s+|,/", $terms);

 $out = array();

 foreach($terms as $term){

  $term = preg_replace("/\{WHITESPACE-([0-9]+)\}/e", "chr(\$1)", $term);
  $term = preg_replace("/\{COMMA\}/", ",", $term);

  $out[] = $term;
```

```
 }

 return $out;
}

function search_transform_term($term){
 $term = preg_replace("/(\s)/e", "'{WHITESPACE-'.ord('\$1').'}'", $term);
 $term = preg_replace("/,/", "{COMMA}", $term);
 return $term;
}

function search_escape_rlike($string){
 return preg_replace("/([.\[\]*^\$])/", '\\\$1', $string);
}

function search_db_escape_terms($terms){
 $out = array();
 foreach($terms as $term){
  $out[] = '[[:<:]]'.AddSlashes(search_escape_rlike($term)).'[[:>:]]';
 }
 return $out;
}

function search_perform($terms){

 $terms = search_split_terms($terms);
 $terms_db = search_db_escape_terms($terms);
 $terms_rx = search_rx_escape_terms($terms);

 $parts = array();
 foreach($terms_db as $term_db){
  $parts[] = "content_body RLIKE '$term_db'";
 }
 $parts = implode(' AND ', $parts);

 $sql = "SELECT * FROM Content WHERE $parts";

 $rows = array();

 $result = mysql_query($sql);
 while($row = mysql_fetch_array($result, MYSQL_ASSOC)){

  $row[score] = 0;

  foreach($terms_rx as $term_rx){
   $row[score] += preg_match_all("/$term_rx/i", $row[content_body], $null);
  }

  $rows[] = $row;
 }
```

```
 uasort($rows, 'search_sort_results');

 return $rows;
}

function search_rx_escape_terms($terms){
 $out = array();
 foreach($terms as $term){
  $out[] = '\b'.preg_quote($term, '/').'\b';
 }
 return $out;
}

function search_sort_results($a, $b){

 $ax = $a[score];
 $bx = $b[score];

 if ($ax == $bx){ return 0; }
 return ($ax > $bx) ? -1 : 1;
}

function search_html_escape_terms($terms){
 $out = array();

 foreach($terms as $term){
  if (preg_match("/\s|,/", $term)){
   $out[] = '"'.HtmlSpecialChars($term).'"';
  }else{
   $out[] = HtmlSpecialChars($term);
  }
 }

 return $out;
}

function search_pretty_terms($terms_html){

 if (count($terms_html) == 1){
  return array_pop($terms_html);
 }

 $last = array_pop($terms_html);

 return implode(', ', $terms_html)." and $last";
}


#
# do the search here...
#
```

```
 $results = search_perform($HTTP_GET_VARS[q]);
 $term_list =
search_pretty_terms(search_html_escape_terms(search_split_terms($HTTP_GET_VARS[q])));


 #
 # of course, we're using smarty ;)
 #

 $smarty->assign('term_list', $term_list);

 if (count($results)){

  $smarty->assign('results', $results);
  $smarty->display('search_results.txt');
 }else{

  $smarty->display('search_noresults.txt');
 }
```

And there we go. Not quite as easy as our very first example, but alot more useful for your users.

If you're not totally sick of searching yet, then you might like to add some search term highlighting.

Note: The code in this article is designed for small sized tables. A medium to large dataset will make for very heavy processing - regular expressions are not CPU cheap and can't be indexed. For large scale websites, you need a proper enterprise-level solution. Xapian is a good place to start. For medium sized tables, MySQL's own FULLTEXT indexes can do some fancy things. The downsides are that you'll need to use MySQL 4.1 if you have UTF-8 data and your tables will have to be MyISAM, which has it's own set of issues.

(end of article)