MYSQL RSS

Building a Search Engine with MySQL and PHP 5

By: Alejandro Gervasio
Rating: ★★★★☆ / 29
2007-07-31

- MySQL Security Tips (2009-11-05)
- Building a MySQL Abstraction Class with Method Chaining (2009-10-29)
- Mastering WHILE Loops for PHP and MySQL (2009-10-26)
- Building Dynamic Queries with Chainable Methods (2009-11-02)
- Using the Yahoo Site Explorer Inbound Links API (2009-11-11)
- Completing the MySQL Class with Method Chaining (2009-11-05)
- SEO Essentials: the Proper Web Server and Platform (2009-10-27)
- Gear6 Does Memcached Right (2009-10-14)

If you maintain a medium-sized, growing web site, you might find that it needs an internal proprietary search engine to improve your visitors' experience. This article, the first of three parts, will get you started with building such an engine using PHP and MySQL.

Introduction

Being one of the most popular and approachable programming languages used on the web these days, PHP truly provides web developers with the tools for building a vast arsenal of applications which could not be created a few years ago. In truth, database-driven web sites, shopping carts, pagination systems and form validation mechanisms are only a few examples of what can be done with this friendly scripting language. PHP has evolved significantly and now exposes a mature object-oriented model.

However, it's valid to notice that aside from the wealth of web-based applications that can be built quite easily with PHP, there's one in particular that's becoming very popular with many modern web sites as they grow in size. In this case, I'm talking about developing an internal search engine for any web site, by using the capacities provided by the powerful MySQL/PHP 5 tandem.

True to form, web-based search engines have existed for a long time and evolved steadily, mostly due to the continuous advances of large companies like Yahoo, Google, and many others in this huge and fascinating terrain.

Nevertheless, in this case I'm not so ambitious as to tackle the development of such a complex project. Instead, I'd like to focus my attention on a concrete situation that several PHP developers have to face on a frequent basis. They must deal with a medium-sized web site that requires the implementation of an internal search engine, so users can find the contents they're looking for more directly and easily, without the need to navigate across a certain number of irrelevant web pages. Sounds pretty logical, right?

From a theoretical point of view, building this kind of application might seem to be a pretty straightforward process that can be tackled with minor hassles. But in a real-world situation, developing an efficient search engine with PHP 5 and MySQL may be quite challenging, especially if you're a PHP developer making your first steps into the vast area of web development.

Thus, considering that more and more web sites are adding a proprietary search engine to their existing structure, in this series, which is comprised of three instructive tutorials, I'm going to teach you how to build a web-based search application that can be easily adapted to suit your personal needs. Of course, as you may have guessed, I'm going to use some basic MySQL database tables for storing the contents of a hypothetical web site, but this condition can be quickly modified to work with a different database server.

Having introduced the subject of this series of articles, it's time to start learning how to develop an extensible search engine by using the powerful MySQL/PHP 5 combination. Let's begin this instructive journey now!

A logical point to start building this MySQL-driven search application is with creating a basic front-end. It will be comprised of a simple online form into which users will be able to enter different search terms. Naturally, these search terms will first be inserted into a "SELECT" query, and then used to return to the client the corresponding database results, assuming the search has been successful.

Thus, keeping in mind this requirement, below I listed the signature of a simple (X)HTML file, called "form.htm." It simply displays the search online form that I mentioned a few lines above.

Here's the definition for this brand new file. Please take a look at it:

(definition of "form.htm" file)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />

<title>MySQL-based Search Engine</title>

<link href="default.css" rel="stylesheet" type="text/css" media="screen" />

<script language="javascript" type="text/javascript">

window.onload=function(){

if(document.getElementById&&document.
getElementsByTagName&&document.createElement){

var sfield=document.getElementsByTagName('form')[0].elements[0];

if(!sfield){return};

sfield.onfocus=function(){this.value=''};

sfield.onblur=function(){

if(!this.value){this.value='Enter your search term here'};

}

}

}

</script>
```

```
</head>

<body>

<h1>MySQL-based Search Engine</h1>

<div class="maincontainer">

<form method="get" action="processform.php">

<input type="text" name="searchterm" title="Enter your search
term here" value="Enter your search term here'
class="searchbox" />

<input type="submit" name="search" title="Search Now!
"value="Search" class="searchbutton" />

</form>

</div>

</body>

</html>
```

As you can see, the above (X)HTML file simply displays a typical text input box that lets users enter different search terms. These terms will be properly processed by MySQL to retrieve some database records. I decided to spice up the form in question with some JavaScript code to make it a bit more interactive.

So far, so good right? Now that you know how the basic structure has been created, please take a look at the signature of the following "default.css" CSS file, which is tasked with improving the look and feel of the online form.

Here's how this brand new CSS file looks:

(definition of "default.css" file)

```
body{

background: #ccc;

margin: 0;

padding: 0;

}

h1{

width: 375px;

padding: 10px;

margin-left: auto;

margin-right: auto;

background: #339;

font: normal 18px Arial, Helvetica, sans-serif;

color: #fff;

border: 1px solid #000;

text-align: center;

}

h2{
```

```
font: bold 18px Arial, Helvetica, sans-serif;
color: #339;
}
p{
font: normal 10pt Arial, Helvetica, sans-serif;
color: #000;
}
a:link,a:visited{
font: normal 10pt Arial, Helvetica, sans-serif;
color: #00f;
text-decoration: none;
}
a:hover{
color: #f00;
text-decoration: underline;
}
.maincontainer{
width: 375px;
padding: 10px;
margin-left: auto;
margin-right: auto;
background: #f0f0f0;
border: 1px solid #000;
}
.rowcontainer{
padding: 10px;
margin-bottom: 10px;
background: #ccf;
}
.searchbox{
width: 200px;
font: normal 12px Arial, Helvetica, sans-serif;
color: #000;
}
.searchbutton{
```

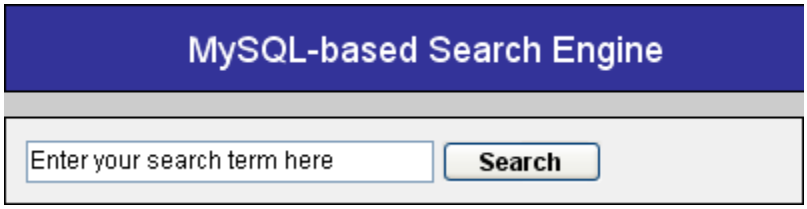
```
width: 80px;

font: bold 12px Arial, Helvetica, sans-serif;

color: #000;

}
```

As shown above, the previous CSS file declares some basic styles which are quite useful for providing the online search form with a decent visual appearance. And speaking of that, the screen shot below illustrates very clearly how the web form looks after applying to it the previous CSS styles:



Al right, at this stage I have built a primitive front-end for entering different search terms. These terms will be embedded into a SELECT query and then processed by MySQL to return to the client the corresponding search results when possible.

In this case, the searching procedure will be performed on the web server with the assistance of some PHP classes. In the section to come I'll be showing you the respective signatures for these classes.

Please click on the link below and keep reading to see how these brand new PHP classes will be defined.

As you might have guessed, developing the business logic that drives this MySQL-based search application is reduced to defining a couple of PHP classes. These classes will be responsible for performing some crucial tasks, such as connecting to MySQL and executing a specific SELECT statement, as well as returning to the browser the corresponding database results in accordance with the search terms entered on the online form that you saw previously.

To perform all of the aforementioned tasks, below I defined two basic PHP classes. You'll probably find them quite familiar, since I have used them with some of my previous articles on PHP development published here on the prestigious Developer Shed network.

The classes in question have been wrapped into one single file, called "mysql.php", whose signature looks like this:

(definition of "mysql.php" file)

```
<?php

// define 'MySQL' class

class MySQL{

private $conId;

private $host;

private $user;

private $password;

private $database;

private $result;

const OPTIONS=4;

public function __construct($options=array()){

if(count($options)!=self::OPTIONS){

throw new Exception('Invalid number of connection parameters');

}

foreach($options as $parameter=>$value){
```

```
if(!$value){

throw new Exception('Invalid parameter '.$parameter);

}

$this->{$parameter}=$value;

}

$this->connectDB();

}

// connect to MySQL

private function connectDB(){

if(!$this->conId=mysql_connect($this->host,$this->user,$this-
>password)){

throw new Exception('Error connecting to the server');

}

if(!mysql_select_db($this->database,$this->conId)){

throw new Exception('Error selecting database');

}

}

// run query

public function query($query){

if(!$this->result=mysql_query($query,$this->conId)){

throw new Exception('Error performing query '.$query);

}

return new Result($this,$this->result);

}

public function escapeString($value){

return mysql_escape_string($value);

}

}

// define 'Result' class

class Result {

private $mysql;

private $result;

public function __construct(&$mysql,$result){

$this->mysql=&$mysql;

$this->result=$result;
```

```
}
// fetch row
public function fetchRow(){
return mysql_fetch_assoc($this->result);
}
// count rows
public function countRows(){
if(!$rows=mysql_num_rows($this->result)){
return false;
}
return $rows;
}
// count affected rows
public function countAffectedRows(){
if(!$rows=mysql_affected_rows($this->mysql->conId)){
throw new Exception('Error counting affected rows');
}
return $rows;
}
// get ID form last-inserted row
public function getInsertID(){
if(!$id=mysql_insert_id($this->mysql->conId)){
throw new Exception('Error getting ID');
}
return $id;
}
// seek row
public function seekRow($row=0){
if(!is_int($row)||$row<0){
throw new Exception('Invalid result set offset');
}
if(!mysql_data_seek($this->result,$row)){
throw new Exception('Error seeking data');
}
}
```

```
]

?>
```

As illustrated above, the previous PHP file uses a pair of MySQL processing classes to perform a search process against one or more selected databases. Also, you should notice that the versatile and extensible structure offered by the above PHP classes makes it really easy to implement an internal search engine on any existing web site, a feature that speaks for itself about the expandability of this MySQL-based search application.

So far, so good. At this time you hopefully grasped the business logic that stands behind this search engine, since it only requires a few simple PHP classes to perform the corresponding searches on the MySQL server. So what's the next step?

Well, assuming that you may want to see how all the previously defined files can be linked with each other to implement a fully-functional search application, in the following section I'm going to develop an illustrative example to show you how to put this MySQL-based search engine to work quickly.

To learn how this hands-on example will be developed, please click on the link that appears below and keep reading.

As I stated in the section that you just read, implementing this MySQL-driven search engine is a matter of fetching the search terms entered on the online web form, and then embedding them into a SELECT query, and finally performing the query in question against one or more selected databases.

Logically, the last step involved in this process is to display the corresponding results (assuming that the query was successful) on the browser. Quite simple, right?

All right, based upon this description of how the search application must work, below I listed the definition of a new PHP file. It is called "processform.php" and is tasked first with performing a search process against a sample "USERS" database table, and then showing the result to the end user.

Given that, the signature of this brand new PHP file is as follows:

(definition for "processform.php" file)

```
<?php

// include MySQL-processing classes

require_once 'mysql.php';

try{

// connect to MySQL

$db=new MySQL(array
('host'=>'host','user'=>'user','password'=>'password',
'database'=>'database'));

$searchterm=$db->escapeString($_GET['searchterm']);

$result=$db->query("SELECT firstname, lastname,comments FROM
users WHERE MATCH(firstname,lastname,comments) AGAINST
('$searchterm')");

if(!$result->countRows()){

echo '<div class="maincontainer"><h2>No results were found. Go
back and try a new search.</h2></div>'."n";

}

else{

// display search results

echo '<div class="maincontainer"><h2>Your search criteria
returned '.$result->countRows().' results.</h2>'."n";

while($row=$result->fetchRow()){
```

```
echo '<div class="rowcontainer"><p><strong>First Name:
</strong>'.$row['firstname'].'<p><p><strong>Last Name:
</strong>'.$row['lastname'].'</p><p><strong>Comments:
</strong>'.$row['comments'].'</p></div>'."n";

}

}

echo '</div>';

}

catch(Exception $e){

echo $e->getMessage();

exit();

}

?>
```

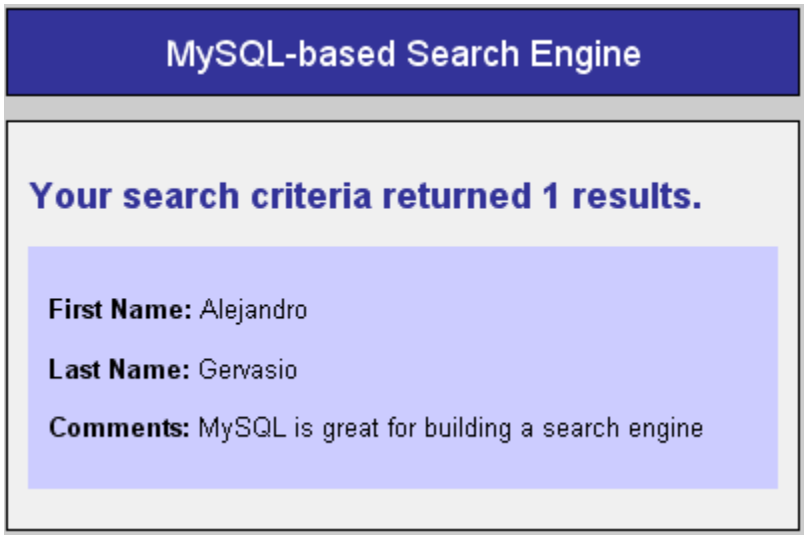
As you can see, the above PHP file uses the two MySQL processing classes to perform the pertinent search process against the "USERS" database table. In this case, I built the search query using the "MATCH" and "AGAINST" clauses to take advantage of full-text capabilities offered by newer versions of MySQL, but you may want to use a more conventional "LIKE" statement, particularly if your search queries are rather primitive.

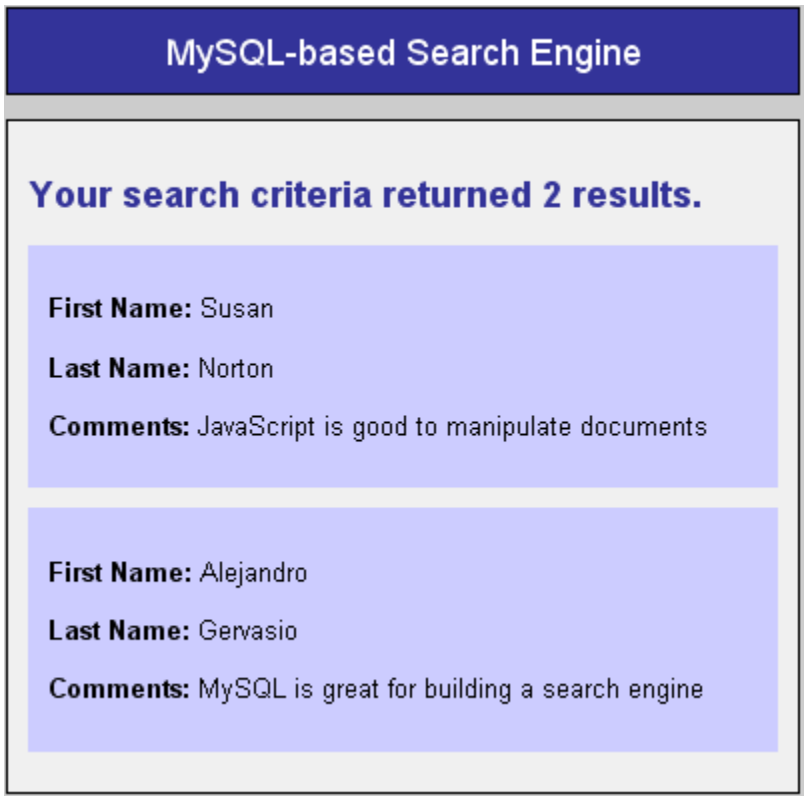
And finally, assuming that the aforementioned "USERS" database table has been previously populated with the following data:

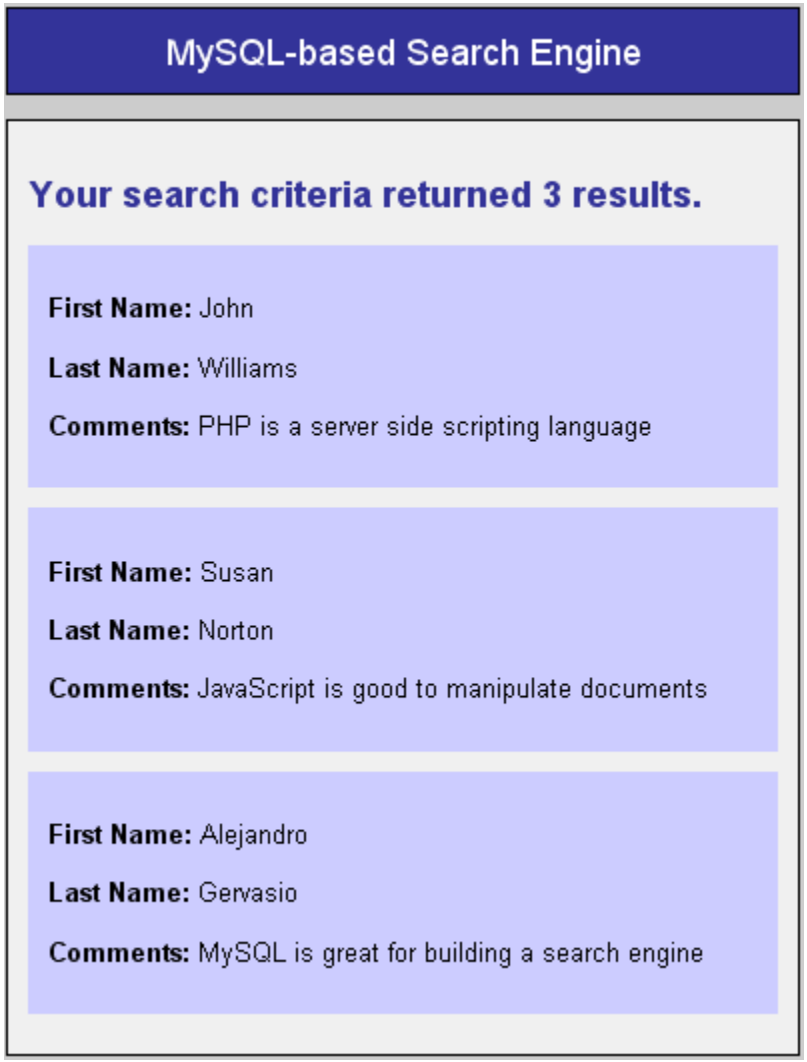
Id firstname lastname email comments

1 Alejandro Gervasio alejandro@domain.com MySQL is great for building a search engine
2 John Williams john@domain.com PHP is a server side scripting language
3 Susan Norton sue@domain.com JavaScript is good to manipulate documents
4 Julie Wilson julie@domain.com MySQL is the best open source database server

Please take a look at the following screen shots. They show the database results returned by this MySQL-driven search application when the search strings "Alejandro," "Alejandro+Susan" and finally "Alejandro+Susan+John" are entered into the corresponding online web form:







As demonstrated clearly by the above images, building an expandable search application using the powerful MySQL/PHP 5 combination is indeed a no-brainer process that can be performed with minor efforts from us. Feel free to use all the source files shown here to implement this search engine on your own web site, so you can provide users with the possibility of searching the site contents.

Final thoughts

In this first part of the series, you hopefully learned how to create an expandable search application using the capabilities offered by MySQL and PHP 5. Nonetheless, this instructive journey isn't finished yet, since the application lacks important features, such as the implementation of true Boolean searches and results paging.

All of these improvements will be introduced in the next part of the series, so you don't have any excuses to miss it!