UNIVERSITY OF DAR ES SALAAM

DEPARTMENT OF COMPUTER SCIENCE

Database Driven Websites

A practical Approach to Design, Implementation, and Management

Lungo, J. H.

March, 2004

TABLE OF CONTENTS

1.0 Introduction

This practical manual is a collection of useful articles in order to guide students who are learning interactive website developments to complete their assignments and enhance their learning.

Recently, to keep up with recent web trends and keep the money flowing, many websites have adopted a dynamic approach to their site. Previously, only the big sites dared use anything but standard HTML pages, however, now everyone is using it to gain the most visitors and the most money. In this practical handout, I am going to focus on the basics of using MySQL (in simple terms, a database program) with PHP. Basic skills required is that you should have a prohrtamming knowledge in any languiage and you know how to create a database and add tables to it.

2.0 Programming in PHP

2.1 Introduction

PHP originally stood for **'Personal Home Pages'** but the meaning was changed as the language matured. Now it has a complicated but more descriptive name of **'PHP: Hypertext Pre-processor'**, which is a recursive acronym, For version 4 of PHP the engine was totally rewritten and became a super fast, and super easy language to use and learn. PHP is not what others consider a programming language, instead it is often called a scripting language or seen from the official PHP website, PHP.net:

"PHP is a server-side, cross-platform, HTML embedded scripting language."

This simply means that on a certain page you can switch in and out of PHP. This is a feature that many scripting languages do not offer yet it allows PHP to be super fast and reliable. A simple example of what this is is shown below in example.php:

```
<?php
// PHP Code Here
?>
HTML CODE HERE
<?php
// PHP Again
?>
HTML Again
```

Other scripting languages rquire you to use their print function to output the HTML and doing it the way PHP offers would cause some nasty errors. As you

may have noticed the file names for PHP end in .php but if using PHP3 then they can end in .php3. You also probably noticed that to inform the server that you are using PHP you must use **<?php** for an opening tab and use **?>** as a closing tag to tell PHP you've stopped using PHP. Lastly there is one line containing //. This is how you comment code in PHP as to make the code more readable for you and other coder. Other types of commenting include:

```
/*
Multi-Line Commenting
Like This One
*/

// Single Line Commenting
# After Code Single Line commenting like the following
function("foo"); # This Function has a value of 'foo'
```

2.2 Variables

A variable is one of the most important aspects of any language. It is an identifier to a certain value, which can be manipulated and deleted. Usually they have to be defined however PHP makes it simple and doesn't require type defintions. To set a value for a variable you choose a name and its value, or you can have an empty value. You then place a **dollar sign ($)** in front of the name and put an equal sign (=) after it. For Example:

```
$dog = "Grey Hound";
$x = 1;
```

Notice that numbers do not require quotations but strings do require them. You can use double or single quotations however there are some requirements to be met. Study the examples below, which explain what works and what doesn't.

```
$var = "This Works";
$var = 'This Works';
$var = Doesn't Work because there are no quotes;
$var = "This ' Works";
$var = "This " is bad because double quotes are inside double
quotes";
$var = "This \" works because the quote is escaped";
$var = 'This ' is bad because single quotes are inside single
quotes';
$var = 'This works \' because the quote is escaped';
```

As you can see you cannot have the same quote types in a string without escaping them. To escape characters you use a backslash and this applies to ", ' and $. This is extremely important because if not used it'll caused nasty parse errors on screen.

Finally an extremely important feature in PHP is seen in the above examples. All commands must end with a semi-colon unless it is a loop or condition statements. Once you've understood what you can already do with PHP, you can continue and start essential features every programming language contain. These include printing, conditionals, loops, includes and much more.

2.3 Printing

Printing in programming languages is not printing a page out onto paper but instead is outputting text onto the screen therefore the function to do this in PHP is **print()** or **echo().** There are a few variations on how to use these functions but always stick to one to make your code more organised. I tend to use echo because it is only four letters and it has a few more advanced features to it.

```
echo("Hi");
echo "Hi";
echo $var1,$var2;
echo "Hi " . $name;
echo "Hi $name";
echo("Hi " . $name);
echo("Hi $name");

print("Hi");
print "Hi";
print "Hi " . $name;
print "Hi $name";
print("Hi " . $name);
print("Hi $name");
```

Notice the line where **echo** has a feature that print doesn't. This feature allows you to print two or more variables very quickly by only separating them by a comma. Also notice that all variation can be with or without the brackets except for that one echo function. This is a matter of choice however I exclude brackets so that it is less to write.

From what we've learnt we can easily create pages that print variables and text, have comments in the code and work properly like the following example:

```
<?php
/* This is my first
Complete PHP script */

// Define Variables
$dog = "Bulldog";
$name = "John";
```

```
// Print Variables and formatted
echo "You have a $dog, $name<br>";
echo "You have a " . $dog . $name;
?>
```

In the above, the second echo contains a different format to the first that works. The second example contains **a period(.)** that is used to join variables and strings together. Usually this isn't necessary for strings, but some types of variables require this like arrays, which will be explained in more detail later.

```
echo "This is your " . $dog[0];
```

2.4 Conditions

Condition statements are also very important in every programming language as they can control output and input. If something then do something, is the typical set up of a condition statement. They use the if function using the following, non working examples:

```
if(Condition) {
Do Something
}

if(Condition) {
Do Something
}else {
Do Something Else
}

if(Condition) {
Do Something
} elseif(Condition) {
Do something
}
// More elseif's can be added and an else at the end
```

In the above, 'condition' can be as simple as checking a value to complicated string validation. If is always used at the start and must have a conditions. Elseif is optional and you can have multiples of them. Else is also optional but if there is one, it should be placed at the end. Where is says, 'Do Something' it means you can do whatever you want if the condition is met. Below is a complicated conditional statement.

```
<?php
if($x == 1) {
echo "x = 1";
} elseif($x == 2) {
```

```
echo "x = 2";
} else {
echo "x doesn't equal 1 or 2";
}
?>
```

This basically prints 'x = 1' if x equals one, 'x = 2' if x equals 2, and 'x doesn't equal 1 or 2' if x isn't either 1 or 2. Notice that in the condition == is used instead of =. This is to check if the value is the same and not to set it as a variable.

2.5 Includes

Includes are extremely simple in PHP and enable you to include files without using Server Side Includes (SSI). There are two options you can choose from in determining which function to use.

Include() - This includes the file and parses it for errors. An error will show up if it includes bad PHP. The most important feature about include is that it cannot get files from password protect directories.

Require() - Includes the code into the file and will cause errors on the including files page. This can completely stop pages from displaying. Most importantly is that this function can retrieve documents from password protected directories.

They are both used in the following format:

```
include("file"); OR include 'file'; for windows platform
require("file"); OR require 'file'; for windows platform
```

The 'file' can be an absolute path, a URL, or even a relative path which makes them both great functions. They cannot include from different servers or domains which requires the file() function. Example of includes:

```
include.php
<?php echo "Hi!"; ?>

require.php
<?php echo "Bye!"; ?>

main.php
<?php
include("include.php");
echo "<br>How are you?<br>";
require("require.php");
?>
```

OUTPUT:

```
Hi!
How are you?
Bye!
```

2.6 Loops and Arrays

Loops are functions that repeat a task for a set amount of times. There are three main types of loops that you need to know, as well as understand arrays which are explained. Arrays are variables that have multiple values by specifying an identifier to them. A typical array could contain:

```
$animals[0] = "Dog";
$animals[1] = "Cat";
$animals[2] = "Bird";
```

This array (which works) would have three values; dog, cat and bird. Value 0 is always the first in an array if you specify the array without giving them the identifier. This is important to remember with loops otherwise you may forget to output the first value in the array. A more simple way of setting up an array is below:

```
$animals = array("Dog","Cat","Bird");

$animals[] = "Dog";
$animals[] = "Cat";
$animals[] = "Bird";
```

There are more complicated arrays, which can have many values like animal -> dog, colour -> brown and age -> 2. However, these are too complex to be covered in this article. Now that we have this basic array, we can get data from it. You can either do it manually (don't) or use loops. The three methods of loops are using two different function; for() and while(). The syntax is as follows:

```
for(variable;condition;result) {
Do Something
}
/* Variable is setting a variable; $i = 1;
Condition is a condition like $i<100;
Result is what happens after the loop to the variable like $i++
*/
```

In the above the syntax of the for loop, the explanation is shown in the comments. It may seam confusing now but becomes simple later after using it a couple of times. Here is an extremely simple yet practical example of a for loop.

```
<?php
for($i=0;$i<10;$i++) {
echo "$i<br>";
}
// This prints the numbers 0 to 10 on a new line each time.
```

2.7 While Loops

Another loop is the while loop which is very similar in function but simpler code"

```
while(condition) {
do something
}
```

In the above change the condition to suit your needs and for your script. A typical while loop may be:

```
<?php
$i=0;
while($i<10) {
echo "$i<br>";
$i++;
}
?>
```

In the above we decide a variable and start the while loop. It check to see if $i has a value less that 10 and if so prints the value of $i on a new line. It then increases $i by 1 so it prints a different value the next time. This script would print the numbers 0 - 10 on a new line.

The second type of while loop is almost the same but the loop always occurs at least once. With the other methods of loops, they can all never loop. This is a good loop if you need something to happen at least once or more. Below is the syntax and a basic example.

```
do {
Do something
} while(Condition);

Example:
<?php
$i=0;
do {
echo "$i<br>";
$i++;
} while ($i == 2);
?>
```

In the example, $i equals zero and the condition states for the loops to run if $i equals 2. However, using this method it will echo 0 once because the condition is after the loop. It will also print 2 because $i is increased by one every time.

2.8 Using Loops

Most commonly loops are used to output values of arrays and that is all included below using the for() loop and the first while() loop method. The array must be set making sure that there is a value for the first identifier in the array(0). You also use the count() function to set a variable for the number of values in the array.

```
<?php
for($i=0;$i<count($animals);$i++) {
echo $animals[$i];
echo "<br>";
}
?>

<?php
$i=0;
while($i<count($animals)) {
echo $animals[$i];
echo "<br>";
$i++;
}
?>
```

Both the above will print the words; Dog, Cat and Bird each on separate lines. This type of function is extremely useful for inserting data into databases, validate arrays, and many more things. There is one more type of loop function however it is only needed for more advanced array work. It is called **foreach()**.

This article only touches on the features of PHP although this should be enough to grasp the basics of PHP and any other programming language and client side languages such as Javascript. Commenting, Printing, Arrays, Loops, Conditions and Includes can already help you to make decent dynamic web sites which makes your site a lot easier to update.

3.0 MySQL Database

3.1 Creating the database

While designing the database driven website, we know what data we want and what we want to do with that data. Now we need to create the database

to store the data. The first thing we need to do is to create an empty database, which will hold our table. For this tutorial I will name the database **library**. In MySQL you use the following command from the command line to create the new database.

```
mysqladmin -p create library
```

The "-p" flag is used so it will prompt you for your password to MySQL, most setups require a username and password. Read the MySQL documentation on how to setup the username and password using the mysqladmin tool. It is fairly straight forward, however you may need root access.

You can test connecting to the database using the mysql client. On the command line:

```
mysql -p library
```

You should get a mysql> prompt if it connects to the database, else it will return an error saying Unknown database 'library'

3.2 Creating the Database Table

Next we need to create the table in the database. Tables are created using SQL statements, and can be created using the mysql client tool or PHP. The nice thing about using a PHP script to create the table is you can save the script to for later use, so if something goes wrong you can recreate the tables, or just use it to refer to the database schema.

The SQL command to create a table is:

```
CREATE TABLE tablename (
column1 column1type,
column2 column2type,
etc.... )
```

The table we want to create is:

```
Table Name: book
Columns:    id          (integer - primary link key)
            title       (50 characters)
            publisher   (75 characters)
            description (text field - lots of text)
            category    (50 characters)
```

So the SQL code to create the categories table is:

```
CREATE TABLE book (
id INT NOT NULL AUTO_INCREMENT,
title VARCHAR(50),
publisher VARCHAR(75),
description TEXT,
category VARCHAR(50),
PRIMARY KEY(id) );
```

The **id** column is the primary key for this table. In order to be a primary key the column can not be null (NOT NULL). I also set it to automatically increment the number (AUTO_INCREMENT) so when each record is added the id will increase by one. The last line specifies that the id column will be the primary key for this table. A primary key is a unique number for that specific record or row of data.

The most common data types are INT, VARCHAR, TEXT and DATETIME. Look in the MySQL documentation about creating databases for information about other available data types in MySQL, and more specifics the data types it supports.

3.3 Executing SQL in PHP

The code to execute a SQL statement in PHP takes 3 steps.

1. Connect to database
2. Create Statement
3. Execute Statement

1.The command to connect to the database is:

```
$cid = mysql_connect($host,$usr,$pwd);
```

Where $host, $usr, and $pwd are previously specified. Host refers to the machine running MySQL and the username and password to connect to that MySQL machine. This command returns a connection id, which is used for to identify this connection in later queries.

2.Creating the SQL statement simply consists of assigning the SQL statement such as the one above to a string. I usually have it run over multiple lines concatenating the string together as I go, this makes it easier to read.

3.The command to actual send the database the SQL command is:

```
$result = mysql_db_query($db,"$SQL",$cid);
```

Where $db is the database to query, $SQL is the SQL statement, and $cid is the connection id created above. This returns a 1 if executed correctly, and undefined or false if an error occurred.

Putting all of this together, and adding a little error detection and displaying of the errors gives us the following script, which you can download and load on to your web server running PHP. Note: Opening the file through the web server will execute the script and create the database tables.

PHP Script to create tables (create_table.phps)

```
<?
   $usr = "--username--";
   $pwd = "--password--";
   $db = "library";
   $host = "localhost";

   # connect to database
   $cid = mysql_connect($host,$usr,$pwd);
   if (!$cid) { echo("ERROR: " . mysql_error() . "\n");    }

   # setup SQL statement
   # create links table
   $SQL = " CREATE TABLE book  (";
   $SQL = $SQL . " id INT NOT NULL AUTO_INCREMENT, ";
   $SQL = $SQL . " title VARCHAR(50), ";
   $SQL = $SQL . " publisher VARCHAR(75), ";
   $SQL = $SQL . " description TEXT, ";
   $SQL = $SQL . " category VARCHAR(50), ";
   $SQL = $SQL . " PRIMARY KEY(id) );";

   # execute SQL statement
   $result = mysql_db_query($db,"$SQL",$cid);

   # check for error
   if (!$result) { echo("ERROR: " . mysql_error() . "\n$SQL\n");    }

   # display result for table creation query
   # this should be 1 on a success
   echo ("Links Results RESULT = $result\n\n");

   mysql_close($cid);

?>
```

Problems: The most common problem when running this script is connecting to the database. Make sure you replace the username and password variables with your username and password setup to connect to your MySQL server. You can trouble shoot connection problems using the **mysql** client from the shell, use mysql -? for help.

4.0 Using PHP And MySQL

4.1 Introduction

This chapter is showing the basics of how to add, edit, delete and list records from a database table. The table format we'll be using is as follows and is the syntax that will create the table in phpMyAdmin or the MySQL command line:

```
CREATE TABLE addresses (
ID int(11) NOT NULL auto_increment,
FirstName varchar(50) NOT NULL,
LastName varchar(50) NOT NULL,
Address varchar(255) NOT NULL,
Email varchar(100) NOT NULL,
PRIMARY KEY (ID)
);
```

This table, addresses, will have five fields. These are ID, FirstName, LastName, Address and Email. Therefore, as you may have guess, this is a very simple address book that we are creating. We will be creating one file that does all the database connecting. This means that you only have to change the username and password in one file instead of many if you change hosts. Below is the file that we'll be using called connect.php:

```
<?php
$username = "bob";
$password = "password";
$host = "localhost";
$database = "addresses";

mysql_connect($host,$username,$password) or die("Cannot
connect to the database.<br>" . mysql_error());

mysql_select_db($database) or die("Cannot select the
database.<br>" . mysql_error());

?>
```

The above code doesn't really need explaining, although I will explain it. It simply, defines the variables for the database that you'll be using and then connects to the database. 'mysql_connect()' takes three arguments which are the host, database user and database password. The part after just prints out an error message. Later in the script, we connect to a certain database and print out an error if this fails also. Now with this created, we can easily included it in all our scripts to connect to the database.

4.2 Adding data

Adding data to your table is probably the most important so it's coming first. All we need to do, is collect the data from a form, which is submitted and the PHP script inserts the data into the database. Below is the code that does this which is explained below:

```
<?php
include("connect.php");

if(!empty($fname)) {
$fname = addslashes($fname);
$lname = addslashes($lname);
$email = addslashes($email);
$address = addslashes($address);

$sql = "INSERT INTO addresses SET FirstName='$fname',
LastName='$lname', Email='$email', Address='$address'";
$query = mysql_query($sql) or die("Cannot query the
database.<br>" . mysql_error());

echo "Database Updated.";
} else {
?>

<b>Add Address</b>
<form name="address" method="post" action="<?php echo
$PHP_SELF; ?>">
First Name: <input type="text" name="fname">
<br>
Last Name:
<input type="text" name="lname">
<br>
Email:
<input type="text" name="email">
<br>
Address:<br>
<textarea name="address"></textarea>
<input type="submit" name="Submit" value="Submit">
</form>

<?php
}
?>
```

Things to notice:

1. 'include("connect.php");' - This line includes the connect.php file which we'd created to connect to the database.

2. 'if(!empty($fname)) {' - This starts a conditional statement which checks if the form has been submitted. More specifically it checks whether or not the first name has a value. If it doesn't, it displays the form again. Otherwise it continues.

3. '$fname = addslashes($fname);' - The series of these add slashes (\) in front of ' and " characters. This is to stop PHP and MySQL from causing errors. It also keeps forms a little bit safer as many hack attempts can be stopped just be doing this.

4. '$sql = "INSERT INTO addresses SET FirstName='$fname', LastName='$lname', Email='$email', Address='$address'";' - This is one of the most important lines in this scripts as it sets a variable for the query. This query tells MySQL to insert a record into the table 'addresses' and insert the correct data into the correct fields.

5. Finally, '$query = mysql_query($sql) or die("Cannot query the database.
" . mysql_error());' executes the query that we created in the form as a variable and does the inserting. If it goes wrong, an error is printed on the page in an easy to read format and stops the script from executing further.

4.3 Editing database records

In many large scale operations, being able to edit the data you've placed into the database is an absolute must. This can help fix typos made, layout problems and much more.

```
<?php
include("connect.php");

if(!empty($fname)) {
$fname = addslashes($fname);
$lname = addslashes($lname);
$email = addslashes($email);
$address = addslashes($address);

$sql = "UPDATE addresses SET FirstName='$fname',
LastName='$lname', Email='$email', Address='$address'
WHERE ID='$id'";
$query = mysql_query($sql) or die("Cannot query the
database.<br>" . mysql_error());

echo "Database Updated.";
} else {

$sql = "SELECT * FROM addresses WHERE ID='$id'";
$query = mysql_query($sql) or die("Cannot query the
```

```
database.<br>" . mysql_error());
$result = mysql_fetch_array($query);
$FirstName = stripslashes($result["FirstName"]);
$LastName = stripslashes($result["LastName"]);
$Email = stripslashes($result["Email"]);
$Address = stripslashes($result["Address"]);
?> <b>Edit Address</b>
<form name="address" method="post" action="<?php echo
$PHP_SELF; ?>">
First Name: <input type="text" name="fname" value="<?php
echo $FirstName; ?>">
<br>
Last Name:
<input type="text" name="lname" value="<?php echo
$LastName; ?>">
<br>
Email:
<input type="text" name="email" value="<?php echo $Email;
?>">
<br>
Address:<br>
<textarea name="address"><?php echo $Address;
?></textarea>
<input type="submit" name="Submit" value="Submit">
<input type="hidden" name="id" value="<?php echo $id; ?>">
</form>

<?php
}
?>
```

This is very much the same as adding the data except with a few obvious changes. The first change we see is that at the top, the query has change from INSERT to UPDATE and a WHERE clause was added to the end. This clause tells MySQL to only update the records which fit the condition. We make sure that it fits where the ID equals the one we are editing. We currently don't have an ID although, when the listing is shown, it'll all become clearer.

If you look at the code you'll also notice the addition of some of the following lines:

```
$sql = "SELECT * FROM addresses WHERE ID='$id'";
$query = mysql_query($sql) or die("Cannot query the
database.<br>" . mysql_error());
$result = mysql_fetch_array($query);
$FirstName = stripslashes($result["FirstName"]);
```

These lines are getting the data that we want to edit. The select statement gets the data from the table or produces an error. The next lines places the data that we have collected into an array so that it is easily accessible. Finally, the last line sets a variable to a certain section of the array which relates to the field name in the database. To confuse you even more on this one I've joined the function stripslashes() at the same time. This removes the slashes we have previously added using addslashes().

Lastly, and most importantly, we have to show the data we've collected in the form as a default. To do this we add an attribute to the input tags like the following:

```
value="<?php echo $LastName; ?>"
```

This is added inside an input tag or in between the opening and closing tags of a textarea. This simply prints the variable out so that we can see the data we have collected from the database. Easy as pie!

4.4 Deleting Database Records

Deleting data is essential to a script like this. It helps you to easily remove old contacts or remove contacts that you've made an error on and don't want to edit. Deleting is also the simplest of all the actions as you can see below:

```
<?php
include("connect.php");

$sql = "DELETE FROM addresses WHERE ID='$id'";
$query = mysql_query($sql) or die("Cannot delete record.<br>"
. mysql_error());

echo "Record Deleted!";

?>
```

As you can see, it is very simple. We only have to create a query that deletes the record using a WHERE clause. The WHERE clause is very important otherwise the whole table can be deleted very easily and quickly. We only want to delete one record therefore we specify where ID='$id'. $id is set on the list of address pages that will be created next. The query then executes this variable and prints an error if it fails.

4.5 Listing database records

In order to edit and delete pages, we need to create a list of all the records and add links next to them so that we can pass the ID from one page to another. This is surprisingly simple and only requires a few lines of code.

There isn't much different from this script compared to the others and therefore doesn't require too much of an explanation:

```
<h2>List Records</h2>
<a href='add.php'>Add Record</a><br>
<?php
include("connect.php");

$sql = "SELECT * FROM addresses WHERE ID='$id'";
$query = mysql_query($sql) or die("Cannot query the
database.<br>" . mysql_error());
while($result = mysql_fetch_array($query)) {
$FirstName = stripslashes($result["FirstName"]);
$LastName = stripslashes($result["LastName"]);
$ID = $result["ID"];

echo "$FirstName $LastName [<a
href='edit.php?id=$ID'>Edit</a> | <a
href='delete.php?id=$ID'>Delete</a>]<br>";

}

?>
```

The first thing that you should notice is the new piece of code that we have that looks similar to another piece of code we've previously used:

```
New:
while($result = mysql_fetch_array($query)) {

Old:
$result = mysql_fetch_array($query);
```

The new piece of code here, cycles through all the data that the database gives us and places it into an array using a loop. By using a loop we can easily print out multiple records whereas the old method only allows us to print out one record which is what we needed for editing records.

The important line is this piece of code is the following:

```
echo "$FirstName $LastName [<a
href='edit.php?id=$ID'>Edit</a> | <a
href='delete.php?id=$ID'>Delete</a>]<br>";
```

This extremely long and weird looking line prints out the first name, last name and the links to edit and delete the database in a nice way. This line is what joins all the previous pages together and you've successfully created a

content management system for your address book. By editing this one line you can create a new page for public viewing to show all your addresses:

```
echo "<b>Name:</b>$FirstName
$LastName<br><b>Email:</b> $Email<br><b>Address:</b>
$Address<br><br>";
```

Using the above code formats the data into an easily readable way which can be used for the public viewing. Make sure to add the following in the correct place if you want to show the Email and Address:

```
$Address = stripslashes($result["Address"]);
$Email = stripslashes($result["Email"]);
```

With all your new found knowledge you can now go out and create much more advanced scripts and actually know what you're doing!

5.0 Cookies, and Header functions

5.1 Setting Cookies

cookie is a *collection of information, usually including a username and the current date and time, stored on the local computer of a person using the World Wide Web, used chiefly by websites to identify users who have previously registered or visited the site***.**

Cookies is a small file, located on the clients computer (usually in the cookie directory) that can be used for many things. These include counting the amount of times you visit the hosts website, what forms you've filled in so that you don't have to do it again and even to keep your username and password handy so you don't have to keep logging in. A cookie however can NOT identify you anymore than anyone else on the Internet which means that it can hold your IP address, as all websites do, and any private information that you give to the website. If you have a cookie on your computer and have not filled in personal information then they can not identify your name, age, address or any thing else.

Cookies are so relied on that PHP has a two methods of setting them but only one is used in this article. The setcookie() function provides and easy and quick way to set a little piece of data on a users computer to help them use your site more interactively. The setcookie() function uses the below syntax:

```
int setcookie (string name [, string value [, int expire [, string path [, string
domain [, int secure]]]]])
```

Anything in square brackets is optional but you have to follow the pattern which will be more obvious later and not fill in any random values. The first parts states the function is called **setcookie()** which is simple, the next part sets the name for the cookie. The next part is optional but if you are setting a cookie then it is definitely recommended to use because it is the information that the cookie will be storing. The next value is the expiry date which is made using the **time()** function. If you use the expiry date you must have a value. The next three settings are not needed for basics and most cookie work you'll do. Below is a basic file to create a cookie:

```
<?php
$name = "Bob";
setcookie("username",$name,time()+1209600);
?>
```

The above code uses the variable $name to set a value for a cookie called 'username'. This cookie has a expiry date of 'time()+1209600' which is a UNIX time value for the current time plus 1209600 second; that equals two weeks. This means that in two weeks this cookie with become inactive and will not work unless reinstated. To work out how long to make the expiry date, just find out the amount of second in that time period by multiply by 60 and another few numbers and then add it to time().

5.2 Reading from a Cookie

A cookies is pointless unless you can retrieve the value from it whenever the person visits your site. There are three main ways to do this but I only recommend one because with certain recommend settings in PHP, it can cause your script to malfunction using the bad method. Below are the three methods of retrieving the value from the cookie:

```
$user = $username // Not recommend
$user = $HTTP_COOKIE_VARS["username"]; //Recommended
$user = $_COOKIE["username"]; // Recommended but requires
PHP 4.1
```

The first method relies on PHP to search through every possible variable and finally find the cookie and can be used by name. However with 'register_globals' off in the PHP configuration file would cause the cookie to fail. Instead using the second will always allow your scripts to run. This gets the cookies name out of the specified cookies variables which makes it a lot faster and reliable. The third method is even better than the second although it requires PHP 4.1 and above to work, but in the future this will be the standard. This method is just shorter than the second and more secure.

Now that you can retrieve cookies you can use them practically as checking if a user has logged in before and if so let them into the admin area, or instead show them the login form. The example below is very insecure for obvious

reasons. It makes use of the **header()** function which allows many things such as browser redirection.

```
<?php
if($HTTP_COOKIE_VARS["username"] == "Bob") {
header("Location: admin.php");
} else {
header("Location: login.php");
}
?>
```

It is also helpful to be able to delete a cookie manually from your site. This is extremely simple and is almost the same as setting a cookie. All you do is set the same cookie but with no value and with an expiry date in the past. This forces the browser to delete the cookie from the users system. Below is how we'd delete our username cookie from the users system:

```
setcookie ("username", "", time()-60000);
```

As shown, the value is empty and the expiry date is the current time() minus 60000 seconds which is a large random number I added. Any negative number will work but due to variations in computer times, it is not recommended to use -1 but instead something higher like a day or two.

A lot of people have problems with setting cookies and usually is related to one thing that causes cookies to fail. *Cookies are sent in the header of the file therefore there cannot be any output before the cookie it set, modified or deleted otherwise it will produce a nasty error. Output include whitespace, HTML, of data outputted using echo, print or any other such function. You can however, still get the value of a cookie anywhere on the page which is extremely useful.*

6.0 Session Authentication

6.1 Introduction

Web site authentication applies to most Web sites on the Internet. Implementation of authentication schemes however can vary significantly from one site to another. Here is a one method of Web site authentication using PHP Sessions and a code submission by "ugehrig" with their submission, Sessions to store login information.

6.2 Why use session authentication?

Let's say you run the Web site www.myneatstore.com and you want your customers to be authenticated with a username and password before they are allowed to make purchases. To accomplish this, you could use a number of methods (the most obvious of which is the HTTP authentication headers). Unfortunately, to implement HTTP authentication you either have to properly configure it through your Web server (such as using a .htaccess file on an Apache Server) or duplicate the functionality through the use of the PHP header() function.

Although the HTTP mechanism is acceptable, even with a proper setup HTTP authentication has many shortcomings. These include:

- the lack of useful security features, such as inactivity timeout
- developer-friendly benefits, such as the ability to display secure/non secure data easily on the same page.
- developing session authentication, you can overcome most of these shortcomings, while at the same time providing a more useful and clean implementation.

6.3 How do sessions work?

To understand how session authentication works, it is first necessary to understand how sessions work. Sessions were designed to solve a common problem in Web development -- *the lack of an efficient and clean way to transfer large amounts of user-specific data from one script to another seamlessly*. For example, consider a series of scripts that require an array $foo to be passed from one script to another. Using "conventional" methods, this array would have to be converted into a valid string to be passed through a GET or POST method to the next script. This script would then have to convert the data back to its original array form. This task can quickly become extremely difficult, and at times impossible, as the amount of data and number of involved scripts increases.

The concept behind sessions basically involves associating a single unique ID string with a (usually) much greater amount of actual data. This data can be anything PHP allows and in practice could represent our example array $foo. When a user visits a session-enabled script, PHP checks for the existence of a session ID that may have been passed to the script (in the form of a cookie or GET method). If a session ID was passed, it is then checked for validity, and if valid the variables associated with that session ID are created.

If no session ID was provided, or the session ID provided was invalid, PHP creates a new ID that it then passes on to the browser (in either cookie form or by GET method). Through this mechanism, PHP ensures that in the event no valid session ID is given (as either a cookie or GET parameter to PHP when the page is requested), it is assigned one for future requests. In the event a valid session ID is given, all of the data associated with that session ID will be reloaded with identical values, and in the exact same state it was in when it was last accessed.
As a developer, this gives you the ability to write scripts seamlessly, even if they rely on variables that may have been created in other scripts executed independently of the current script.

6.4 How Session Authentication works

With this fundamental understanding of how sessions work, let's discuss how to develop an "authenticated" session. An authenticated session is nothing more than a session that contains variables that signify it as "authenticated". For example, consider a session containing a variable array called $authdata that by default is not defined (is_array($authdata) is false).

If the array does not exist, the user is presented with a login form. At this point, assuming the user presents a valid username and password, $authdata is then set. Because $authdata is passed from script to script in the session, the user will not be asked to log in again for as long as the session ID remains valid.
Conversely, a user can be unauthenticated by unsetting $authdata using the unset() statement on it.
Here's a step-by-step representation of this process:

- Check the value of your authentication variables (in our case $authdata)
- If the authentication variables fail, present a login page.

1. Confirm the login information provided by the user with the server records (such as a username/password file on the server)
2. Once confirmed, set appropriate authentication variables to reflect valid authentication

- If authentication variables prove to be valid, display secure content

If this seems confusing, it is strongly recommended that you experiment with sessions and their behavior to gain a better understanding of sessions before continuing with today's column.

6.5 The Script

Now it's time to look at our script and show the practical side of today's Code Gallery Spotlight column. Today's script is broken into two separate functions. The first, auth(), is the heart of the script and contains all of the necessary processing for the process to function and returns true if the session is to be considered authenticated. The second function, loginform(), is nothing more than a wrapper for the HTML needed to display the login form and will not be discussed further. Let's look at auth():

6.6 Code Flow

- Start session and Init variables
- Determine State (login check, auth check)

```
<?php
   function auth(
      $login = '',
      $passwd = '',
      $pass_file = 'password.txt'
   ) {
      session_start();
      global $PHP_SELF, $authdata;
      $check = ! empty( $login );
```

To begin the script, you need to first start the session by calling session_start(). When this function is called, the session process outlined earlier takes place and any variables that existed (such as $authdata) will automatically be created globally (which is then brought locally with the global statement). Once any registered variables have been created, you'll then check for the existence of the $authdata array that signifies the session is authenticated.

Next, the script must determine the mode it is in. This can be done by checking the value of the optional $check variable; if the variable is set to true, then login information was provided. $check itself is determined by examining the $login parameter to see if it is empty or not. It is necessary only to check $login (instead of $login and $passwd) because $login is the only variable guaranteed to have a value (a password may not exist). The

next step is to determine if the script needs to check a username and password and if so, perform the check.

- If $authdata already exists, session is authenticated so return true
- If a login name/password was provided
- Open the password file
- Read line by line from the password file
- Parse current line of the password file
- Compare given login data with login data in the password file
- If login data does not match any record in the password file, close it and return false and unset any previous authentication data
- Return false if no login information was found

```
if ( is_array( $authdata ) ) {
        return true;
    } elseif ( $check ) {
        $fp = fopen( $pass_file, 'r' );
        while ( !feof( $fp ) ) {
            $line = trim( fgets( $fp, 1000 ) );
            list( $l, $p ) = explode( ',', $line );
            if ( ($l == $login) && ( $p == $passwd ) ) {
                $authdata = array("login"=>$login);
                session_register( 'authdata' );
                fclose( $fp );
                return true;
            }
        }
        fclose( $fp );
        unset( $authdata );
        return false;

    } else {
        return false;
    }
```

Assuming that there was a $login and $passwd parameter passed to your auth() function, the $check variable mentioned earlier should be true, and the script will then proceed to check the login information provided with that which is stored in the password file. The password file itself is broken down into the following format:

<username>,<password>

```
<username>,<password>
<username>,<password>
```

The first step is to open the password file for reading using fopen(). Once opened, the next step is to read a single line from the file and trim any extra white space from the start and end of the line using the trim() function. Then, the line is broken into two variables, $l and $p, which represent the first two strings separated by a comma (the login and password respectively).

Next you'll check the username and password with the ones provided to the script when the function was called. If they match, create the $authdata session variable, register it as a session variable, close the password file and return true. If no match is found for the current line, repeat the process with the next line of the password file until a match is found, or the end of the file has been reached.

If no username and password combination in the file match the one provided to the function, they are to be assumed invalid and the function shall return false. Again, this is all done only if $check is set to true.

6.7 Using the script

Using this script is a fairly simple process, and is best illustrated through an example. Consider the following:

```
<?php
    function loginform($error = false) {
  ?>
  <HTML>
  <HEAD><TITLE>My Login Page</TITLE></HEAD>
  <BODY>
  <?php if($error) { ?>
  The login information you provided was invalid.
  Please log in again below:
  <?php } //end of error check ?>
  <FORM ACTION="<?php echo $PHP_SELF; ?>"
      METHOD=POST>
  Username: <INPUT TYPE="text" NAME="username"><BR>
  Password: <INPUT TYPE="password" NAME="password"><BR>
  <BR>
  <INPUT TYPE="submit" VALUE="Log in">
```

```
  </FORM>
  </BODY>
  </HTML>
  <?php
  } // end of function
if ( !auth($HTTP_POST_VARS['username'],    $HTTP_POST_VARS['password'
 ) ) {
   loginform( isset( $HTTP_POST_VARS['username'] ) );
  }
  echo "Welcome to the authorized site!";
?>
```

This script is started by calling auth() with no parameters. If the session is already valid, there is no need to continue, and the authorized page can be displayed.

If the session is not valid, continue on by checking for the existence of a username and password. If they are provided, call auth(), again passing these parameters. If no username and password is provided, or they are invalid instead of displaying the authorized page, the script calls loginform() which displays the login form as shown and the process repeats until the session is authenticated.

Once authenticated, all subsequent requests of this page by that session will automatically be allowed until the session is unauthenticated or expires.

Final Notes

As always, there is room for improvement in any script. For instance, today's Code Gallery Spotlight relies on plain-text username and password combinations. For better security, the crypt() function could have been used to encode any passwords to ensure that they remain secure. The script could also provide additional checks on $authdata (such as actually looking to see if there is a username and password keys within the array). Here is a final example that illustrates how a page can be both authenticated and unauthenticated simultaneously using session authentication. Because the login page is contained within a separate function, we can develop pages that only restrict certain content on that page to authenticated sessions. For example, consider the following:

```
<?php
      echo "This is non-authenticated";
      if(auth()) {
```

```
        echo "This is authenticated";
    }
?>
```

In this case, all users will be able to see the first echo statement. But only those who have be authenticated prior to the execution of this script will be able to see the second. There are many ways this script, and the concepts outlined today, can be used to improve not only your authentication, but many different aspects of your Web pages. Experiment with sessions and how they have been used here to accomplish user authentication -- and be sure to submit any useful code snippets to the code gallery!

7.0 Creating an Authentication System with Privilege Levels

7.1 Overview

This chapter will show you how to create the basic components of an authentication system. It will review how to construct a login process that validates the information stored in a database, and then how to tier access levels to tailor a site to the user logging in. (This is done by retrieving the access level assigned to a user in the database and then setting cookie information with the verifying information).

A system such as this is generally used for intranet access, but is equally useful for any application that requires different types of access for users. Good examples are Forum applications, with guests, registered users and forum moderators; also well as billing applications or other organizational software.

Definitions

- **Privilege Access**: A system that grants more or less access depending on the user's assigned access level.
- **Authentication**: A method by which the user's access level is retrieved while checking the validity of the username and password.
- **Cookie**: Information stored on the client system that allows persistent information to be exchanged between the server and client without having to re-authenticate.
- **Encryption**: Mathematical equations used to secure valuable information by encoding it.

7.2 Background Information

Authentication systems allow you to put information on the web and secure your data from unwanted access. In addition, a privilege access system allows information to be divided into sections with different access levels, ensuring that users see only the portions of the data that they are supposed to see.

For example, in a business situation, this would allow the Director to be able to see all of the employees' salary information, but would restrict the individual employees to seeing only their own salary information

Privilege access with authentication systems can be used for a wide variety of applications, and is easy to deploy when using PHP with a database such as mySQL.

7.3 Define the Database

The authentication system's database design uses a standard primary key. This means that the ID column will increment and insert a unique number to identify the record. This gives us a unique identifier for each record and is handy when updating and selecting records.

The table below shows the database fields along with their descriptions for the auth table:

Column	Description	Req'd?
username	The user's login name in plain text format	Y
password	The user's password in plain text format	Y
auth_level	The numerical representation of the access levels	Y
id	The numerical identifier for the database record	Y

Here are the MySQL definitions for the statistics:

Name	Type	Key
username	TEXT	
password	TEXT	

auth_level	INT	
id	INT	PRIMARY KEY

Note: When defining your table ensure that the fields USERNAME, PASSWORD, AUTH_LEVEL and ID are set with the NOT NULL Property.

Create the MySQL Statement

The following CREATE TABLE statement may be used to create the above table:

```
   CREATE TABLE auth (
  username VARCHAR(100) NOT NULL,
  password VARCHAR(100) NOT NULL,
  auth_level INT(11) NOT NULL,
  id INT(11) NOT NULL auto_increment,
  PRIMARY KEY  (id)
)
```

7.4 Validate the Data

When you begin entering user information, verify that MySQL contains the data and that all fields are being populated with data. To do this, at the MySQL command line, type:
SELECT * FROM auth LIMIT 4;
This will give you a nice, short slice of the database information you have been collecting. It should look something like this:

mysql> SELECT * FROM auth LIMIT 4;

username	**password**	**auth_level**	**id**
guest	test	1	1
user	test	2	2
editor	test	3	3
administrator	test	4	4

Now you're ready for the scripts and the rest of the tutorial.

7.5 How the Scripts Work

In this tutorial, two scripts will be used:

- **auth.php:** This script will illustrate how to display a login screen, how to retrieve information from a database,and how different levels of access can be displayed and separated from one another. This script also illustrates the use of cookies to setup persistent authentication between the client system and the host server
- **next.php:** is used to illustrate the persistent authentication between the client system and the host server by displaying the login information without the user having to re-login (by simply clicking to another portion of the application). This script will also illustrate how to log-out of the system by destroying the cookies that were set in auth.php.

7.5.1 Script Overview

Auth.php

This script contains the main portion of the code and functionality that we need. It performs the authentication, verifies and then retrieves the appropriate access level for the specific user and sets up the cookies responsible for a persistent authentication connection.

The auth.php script also illustrates how different tiers of access are then displayed, using basic if/else statements, but illustrates how one could retrieve information from a database or setup a series of scripts to perform these same tasks. By performing all of these tasks, the auth.php script does everything except illustrate the persistent connection and giving the user a way to log out.

One of the methods that can be expanded on is, as mentioned above, displaying information for the various access levels. This can include database retrieval, and script separation or includes.

Another important thing to consider when tailoring this tutorial to your application is security. Passwords and other sensitive information should be encrypted with the relevant PHP functions and then compared against a known value, thereby providing a heightened level of security.

For the purpose of this tutorial we will simply use if/else statements and plaintext passwords.

Once you've finished reading this script, it should be very obvious how this script can be customized to create a powerful authentication front end.

7.5.2 Code Flow

- Display the HTML needed for the initial form to obtain a username and password.
- Check the database to validate the username/password values and retrieve the access level assigned to that user.
- Set the persistent authentication via a cookie containing the username and access level.
- Display the logged-in user's access level, and provide a link to next.php.

Ensure that the database connection runs only after the username and password information have been submitted. The username and password information are submitted at the end of this script but are executed first.

```
if ($submit) {

   $db = mysql_connect("localhost", "username", "password");
   mysql_select_db("auth", $db);
$sql = "
SELECT auth_level
FROM auth
WHERE username = '$username'
AND password = '$password'
";
   $result = mysql_query($sql);
   while ($row = mysql_fetch_array($result)) {
      $auth_level = $row["auth_level"];
   }
```

Retrieve the authorization level from the table where the username and password are equal to the values submitted.

```
   if (!mysql_num_rows($result)) {

      echo "You are not Authorized for access.";
```

If the information doesn't match (i.e. returns no records) then an appropriate message is displayed.

```
      } else {
setcookie('username', $_POST['username'], (time()+2592000), '/', '', 0);
```

```
setcookie('auth_level', $_POST['auth_level'], (time()+2592000), '/', '', 0);
    }
```
If the information does match a registered user,, set their persistent authentication with cookies for both username and access level.

Now begin the if/else statements to display the appropriate information for their access level. This could be supplemented by database retrievals, various script redirects or includes. If left in it's current form and expanded, a number of switches could be used instead of the if/else statements, to provide a more robust output method.

```
    if ($auth_level == "1") {
            echo "You are logged in as a Guest.<br />
<a href='next.php'>Click here for options</a>
";
        } elseif ($auth_level == "2") {
            echo "You have Member level access.<br />
<a href='next.php'>Click here for options</a>
";
          } elseif ($auth_level == "3") {
            echo "You have Editor level access.<br />
<a href='next.php'>Click here for options</a>
";
    } elseif ($auth_level == "4") {
            echo "You have Administrative access.<br />
<a href='next.php'>Click here for options</a>
";
    }
```

For each access level, the level is displayed as well as the link to the next.php script. For expansion, the access level displayed could also be stored in a database for dynamic retrieval.

If username and password have not been submitted, then the original form is displayed.

```
<form method="POST" action="<?php echo $GLOBALS ['PHP_SELF'];?>">
Name:<input type="text" name="username"><br />
Password: <input type="password" name="password"><br />
<input type="submit" name="submit" value="Login">
</form>
```

Next.php

This is the second script in our tutorial and is directly linked to by a simple HTML statement after the auth.php logs a user in. This script is intended to illustrate two items that are essential to any discussion about an authentication system.

First, it illustrates that the authentication connection is persistent. That means that without having to re-enter your username and password, without having to put that information in the link as variable information, and without having to submit form variables, your authentication is maintained by information stored on your computer.

This information is stored in cookies. Cookies enable websites to personalize and “remember” visitors even after they’ve visited different sites. (It’s cookies that let Amazon.com provide suggested selections for you when you arrive at their site.0 In our case, we are storing the username and access level information in cookies so that from page to page, your access information is retained.

Second, this script illustrates how to destroy the connection by forcing the cookies that were created in auth.php to expire, thereby logging the user out and deleting the information stored on their computer, in case another person uses the same computer afterwards.

Code Flow

- Display the username and access level to the user.
- Display a log-out form button.
- If the button is clicked, set the same cookies that were set in auth.php to no value and their expiration for one hour ago.

```
If the log-out button is clicked, destroy the cookies and tell the user that
they have logged out
if ($submit) {
    setcookie('username', '', time() - 3600);
    setcookie('auth_level', '', time() - 3600);
    echo "You've successfully logged out.";
} else {
Make sure that if someone tries to access the page without access rights,
they can’t execute the script
    if (!$_COOKIE['username']) {
echo "You are not Authorized.";
```

```
exit;
}
?>
```

If the user has authorization, display their username and access level when they first arrive from the auth.php link and give them the option to log-out .

```
<u>Your User Name</u>: <b><?php echo $_COOKIE['username'];?></b>
<br />
<u>Your Access Level</u>:
<b><?php echo $_COOKIE['auth_level'];?></b>
<br /><br />
<form method="POST" action="<?php echo $GLOBALS ['PHP_SELF'];?>">
<input type="submit" value="Logout" name="submit">
</form>
```

The Scripts

The code below can be copied and loaded into the auth.php and next.php scripts. .All portions of the script that require customization have been outlined above. The code comments indicate where the auth.php script ends and the next.php script begins.

Note: The PHP elements are found within the <?php and ?> marks. The code contains comments preceded by // or /* and*/marks.

```
//begin auth.php



<?php

// Make sure that the database connection runs only
// after the username and password information have
// been submitted. The username and password
// information are submitted at the end of this script
// but are executed first.

if ($submit) {
        $db = mysql_connect("localhost", "username", "password");
    mysql_select_db("auth", $db);
$sql = "
SELECT auth_level
FROM auth
```

```
WHERE username = '$username'
AND password = '$password'
";
   $result = mysql_query($sql);
   while ($row = mysql_fetch_array($result)) {
      $auth_level = $row["auth_level"];
   }
    // retrieve the authorization level from the table
   // where the username and password are equal to
   // the values submitted

   if (!mysql_num_rows($result)) {
            echo "You are not Authorized for access.";
            // if the information doesn't match, i.e. returns
      // no records, then they shouldn't be there.

   } else {
      setcookie('username', $_POST['username'], (time()+2592000), '/', '',
0);
      setcookie('auth_level', $_POST['auth_level'], (time()+2592000), '/', ''
, 0);
   }

   // otherwise, set their persistent authentication
   // with cookies for both username and access level

   // Now begin the if/else statements to display the
   // appropriate information for their access level.
   // This could be supplemented by database retrievals,
   // various script redirects or includes.

   if ($auth_level == "1") {

      echo "You are logged in as a Guest.<br />
<a href='next.php'>Click here for options</a>
";

   } elseif ($auth_level == "2") {

      echo "You have Member level access.<br />
<a href='next.php'>Click here for options</a>
```

```
";

    } elseif ($auth_level == "3") {

        echo "You have Editor level access.<br />
<a href='next.php'>Click here for options</a>
";

    } elseif ($auth_level == "4") {

        echo "You have full Administrative access.<br />
<a href='next.php'>Click here for options</a>
";
    }

    // for each access level, the level is displayed
    // as well as the link to the next.php script.
    // For expansion, the access level displayed could
    // also be stored in a database for dynamic retrieval.


} else {

    // if they haven't submitted their username and
    // password, then the original form is displayed

?>

<form method="POST" action="<?php echo $GLOBALS ['PHP_SELF'];?>">
Name:<input type="text" name="username"><br />
Password: <input type="password" name="password"><br />
<input type="submit" name="submit" value="Login">
</form>
<?php

// close the script

}
?>

// begin next.php
```

```
<?php

// if the log-out button is clicked, destroy the cookies
// and tell the user that they have logged out.

if ($submit) {
    setcookie('username', '', time() - 3600);
    setcookie('auth_level', '', time() - 3600);
    echo "You've successfully logged out.";
} else {

    // Make sure that if someone is accessing
    // the page without access, that they can't
    // execute the script

    if (!$_COOKIE['username']) {
echo "You are not Authorized.";
exit;
}

    // otherwise, display their username and access level
    // when they first arrive from the auth.php link
    // and give them the option to log-out

?>
<u>Your User Name</u>:
<b><?php echo $_COOKIE['username'];?></b><br />
<u>Your Access Level</u>:
<b><?php echo $_COOKIE['auth_level'];?></b>
<br /><br />
<form method="POST" action="<?php echo $GLOBALS ['PHP_SELF'];?>">
<input type="submit" value="Logout" name="submit">
</form>

<?php

// close the script

}
?>
```

8.0 Search Engine: Using MySQL Full-text Searching

8.1 Overview

Before the advent of the search engine, users had to search manually through dozens – or hundreds – of articles and tidbits to find the ones that were right for them. Nowadays, in our more user-centered world, we expect the results to come to the user, not the other way around. The search engine gets the computer to do the work for the user. Using directories to group articles by category is a great way to help people to navigate through many articles. At some point, however, someone will want to find *all* the articles that pertain to a certain topic that may not have a directory of it's own, or may span many directories. This is what the search engine is for.

8.2 Definitions

- **MySQL** – An Open Source database that is used by many PHP developers for it's support and speed, as well as because it's free.
- **Full-text** – Built in functionality in MySQL that allows users to search through certain tables for matches to a string.
- **Boolean Search** – A search which allows users to narrow their results through the use of Boolean operators.
- **Boolean Operators** – A deductive logical system by which a user can narrow results through the use of AND, OR, XOR, and other operators.

8.3 Synopsis

Let's start with a quick review of our situation:

We have a database that contains articles. We might create a table of database contents using a statement like this:

CREATE TABLE articles (body TEXT, title VARCHAR(250), id INT NOT NULL auto_increment, PRIMARY KEY(id);

Let's say we have about 100 of these articles, covering various topics: MySQL, PHP, and various other topics of that sort. How do the users find the tutorials they want? Remember, we need to bring the results to the user. This is going to be a search engine operation.

8.4 Initial Ideas

When I started to work with my first database which was only a tenth of the size, my MySQL query went something like this:

```
SELECT * FROM articles WHERE body LIKE '%$keyword%';
```

This was slow and inefficient. Every time someone searched for an article, they got far too many results, and as the database grew the system became downright shameful.

So what is the solution? It's right here: **Full-text Searching**.

The Solution: Setup

Full-text Search is a feature introduced to MySQL in version 3.23.23. I started out with an update to my table:

```
ALTER TABLE articles ADD FULLTEXT(body, title);
```

This set ups our Full-text index. The (body, title) part tells us that we can search the body and title for keywords later on. We'll find out how to use this later, once we've overcome a potential problem.

In my original database BLOB was my datatype for the body of the article. What's the problem, you ask? BLOBs are meant primarily for binary data. What use is searching binary data? MySQL has been programmed not to index BLOB datatypes for Full-text searching. If you try to index BLOB datatypes, you get an Error 140.

The fix for this is simple:

```
ALTER TABLE articles MODIFY body TEXT;
```

That switches datatype from BLOB to TEXT, thus making a useful column for searching.

The Solution: Actually Doing Something

How do we get results? Let's jump right in and try it out:

```
<?php
   MySQL_connect("hostname", "username", "password");
   MySQL_select_db("our_db");
   $query = "
      SELECT * FROM articles
      WHERE MATCH(title, body) AGAINST ('PHP')
   ";
   $sql = MySQL_query($query);
   /* output results */
?>
```

What will this give us? Well, let's go over Full-Text first.

According to the MySQL manual, Full-text is a "natural language search"; it indexes words that appear to represent the row, using the columns you specified. As an example, if all your rows contain "MySQL" then "MySQL" won't match much. It's not terribly unique, and it would return too many results. However, if "MySQL" were present in only 5% of the rows, it would return those rows because it doesn't appear too often to be known as a keyword that's very common.

MySQL also does something pretty useful. It creates a score. This score is usually something like .9823475 or .124874, but always larger than zero. It can range up above 1, and I have seen it at 4 sometimes.

MySQL will also order a row by its score, descending.

Another useful tidbit: If you use *MATCH() AGAINST()* Change the document style for this to "Inline Code" twice in a query, as we will, there is no additional speed penalty. You might expect that because you are executing the same search twice the query would take twice as long, but in fact MySQL remembers the results from the first search as it runs the second.

So, let's talk about the actual query: We are taking every column from articles, and searching "title" and "body" for *$keyword* This is also Inline Code. Pretty simple.

And if we want to display the score too:

```
<?php
   /* connect to MySQL (same as always) */
   $query = "
```

```
      SELECT *,
         MATCH(title, body) AGAINST ('PHP') AS score
      FROM articles
      WHERE MATCH(title, body) AGAINST('PHP')
   ";
   $sql = MySQL_query($query);
   /* display the results... */
?>
```

8.5 More about Basic Searching

When most people meet up with a search box they don't type in only one word. Not knowing the backend, they just type in as many words as they feel like!

MySQL realizes this and deals with it. If I were you, the only thing I would do is remove the commas that might be there, using *str_replace*. MySQL will take all the words, split them up, and then match using a natural language search.

As a secondary note, you should never send input directly from the user to the MySQL prompt because any number of characters could terminate your MySQL query and begin another dastardly statement. This is presuming you replace PHP with a *$keyword* in the above script.

Example: Basic Searching Application

Let's launch straight into the code. This bare bones application will search for a phrase or a keyword that the user inputs:

```
<?php
   /* call this script "this.php" */

   if ($c != 1) {
?>
<form action="this.php?c=1">
<input type="text" name="keyword">
<input type="submit" value="Search!">
</form>
<?php
   } else if ($c==1) {
      MySQL_connect("hostname", "username", "password");
```

```
      MySQL_select_db("database");
      $sql = "
         SELECT *,
            MATCH(title, body) AGAINST('$keyword') AS score
            FROM articles
         WHERE MATCH(title, body) AGAINST('$keyword')
         ORDER BY score DESC
      ";
      $res = MySQL_query($sql);
?>
<table>
<tr><td>SCORE</td><td>TITLE</td><td>ID#</td></tr>
<?php
      while($row = MySQL_fetch_array($rest)) {
         echo "<tr><td>{$sql2['score']}</td>";
         echo "<td>{$sql2['title']}</td>";
         echo "<td>{$sql2['id']}</td></tr>";
      }
      echo "</table>";
   }
?>
```

What does this script do? First, it checks *$c* to see if user input has been sent. If it has not, the form is displayed. If it has, the script moves onwards.

The same query that we've been using is used here: we match against what the user inputs. We then draw a table and display it in [semi-]pretty form.
The ORDER BY score DESC
Code Inline makes sure that the best scores (the most accurate matches) are shown first.

Important note: Never use this simple script in any production form because I have done absolutely no error checking. The *$query* variable provides an easy opening for an intruder to input something nasty into your query that might destroy your data.

8.6 Advanced Boolean Searching

If you need more options in your MySQL searching, or you haven't finished your coffee yet, keep reading. The *advanced* search engine tutorial begins here.

Before we get started into the magic of bool, I recommend you do a quick *SELECT version();*

Code Inlineon your MySQL server. I spent several hours battling my computer until I read this line in the MySQL manual:

As of Version 4.0.1, MySQL can also perform Boolean full-text searches using the IN BOOLEAN MODE modifier.

Whoops; 4.0.1 is the newest, alpha release of MySQL Still check for this... it should by MySQL. If you're looking to use this on a production server, I'd strongly recommend against that decision. I found I was using a 3.23.23, and I had to set up an experimental MySQL server to use the bool functions that it offers now.

Overall, I was very pleased with the performance of the new bool searching; the scoring system is changed, but one can still manage. Within 15 minutes of upgrading, I had a simple bool search page up and running on my articles database.

Boolean: The Basic Technical Aspect

The only thing you change to use Boolean mode is the *AGAINST()* part of your query. You add *IN BOOLEAN MODE* to the very end of it, and place the arguments right before it. E.g. to search for all the articles that contain the word PHP, you could write:

*SELECT * FROM articles WHERE MATCH(title, body) AGAINST('PHP' IN BOOLEAN MODE)DR[10]*

That will find all the articles that contain the word "PHP" somewhere in them. It's a fairly simple search. If you were to get more complex, and wanted everything that has to do with PHP, *but not with MySQL*, then you could execute this statement:

SELECT * FROM articles WHERE MATCH(title, body) AGAINST('+PHP-MySQL' IN BOOLEAN MODE);

There are more modifiers that one can use to search with, and I will quote

from the MySQL manual since I see no point in typing out a synopsis of the manual:

A Basic Boolean Searching Application

Again, we'll start with the code straight off:

```
<?php
  /* call this script "advs.php" */
  if(!$c) {
?>
<form action="advs.php?c=1" method=POST>
<b>Find Results with: </b><br>
Any of these words: <input type="text" length=40 name="any"> <br>
All of these words: <input type="text" length=40 name="all"> <br>
None of these words: <input type="text" length=40 name="none"> <br>
<input type="submit" value="Search">
</form>
<?
  } else if($c) {
  MySQL_connect("hostname", "username", "password");
     MySQL_select_db("database");
  if((!$all) || ($all == "")) { $all = ""; } else { $all = "+(".$all.")"; }
  if((!$any) || ($any == "")) { $any = ""; }
  if((!$none) || ($none == "")) { $none = ""; } else { $none = "-
(".$none.")"; }
  $query = "
     SELECT *,
       MATCH(title, story) AGAINST ('$all $none $any' IN BOOLEAN MODE) A
S score
       FROM compsite
     WHERE MATCH(title, story) AGAINST ('$all $none $any' IN BOOLEAN M
ODE)";
    $artm1 = MySQL_query($query);
    if(!$artm1) {
      echo MySQL_error()."<br>$query<br>";
    }
    echo "<b>Article Matches</b><br>";
    if(MySQL_num_rows($artm1) > 0) {
      echo "<table>";
       echo "<tr><td>Score </td><td>Title </td><td>Body</td></tr>";
```

```
          while($artm2 = MySQL_fetch_array($artm1)) {
          $val = round($artm2['score'], 3);
          $val = $val*100;
          echo "<tr><td>$val</td>";
          echo "<td>{$artm2['title']}</td>";
          echo "<td>{$artm2['body']}</td></tr>";
        }
    echo "</table>";
  }
  else {
    echo "No Results were found in this category.<br>";
  }
  echo "<br>";
  }
```

After we get the input from the form, *$c* Code Inlineis set to 1 and we start the real work.

First we check our input. If it's empty, we leave it empty, if it's not, we append the proper + or - to it. The parentheses are to allow for the user typing more than 1 word in a given field.

```
$query = "
   SELECT *
      MATCH(title, story) AGAINST ('$all $none $any' IN BOOLEAN MODE) AS
 score
   FROM compsite
   WHERE
      MATCH(title, story) AGAINST ('$all $none $any' IN BOOLEAN MODE)";
```

That's the final query that we use. *$all*, *$none* Code Inline, and $any have already been prepared for the query, and they are inserted. Score is returned as a column to order them by (if we wanted to do that), and from there on, we just have to output the results.

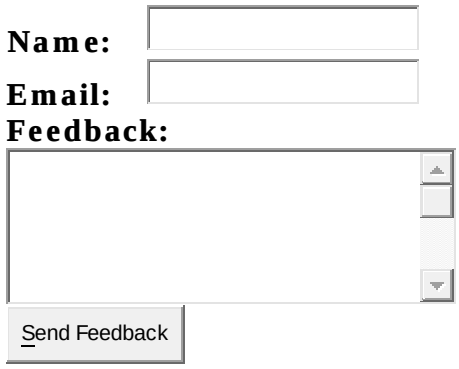
```
  if(MySQL_num_rows($artm1) > 0) {
    echo "<table>";
   echo "<tr><td>Score </td><td>Title </td><td>Body</td></tr>";
        while($artm2 = MySQL_fetch_array($artm1)) {
```

```
        $val = round($artm2['score'], 3);
        $val = $val*100;
        echo "<tr><td>$val</td>";
        echo "<td>{$artm2['title']}</td>";
        echo "<td>{$artm2['body']}</td></tr>";
      }
    echo "</table>";
```

That's the output code. If there's less than 1 row to output, we send a "no records found" message out.

9.0 Send Mail Using PHP

An easy way to get feedback on a site is to have it emailed to you. This script shows you how to create a Feedback form with the response being emailed to you using PHP. The first thing we need is our HTML form. Keeping it simple I'll only ask for Name, Email and their Feedback. Use your browser's view source if you would like the HTML code.



PHP Code
The PHP code to send email is simply one PHP function **mail()**. The basic format for this function is:

```
mail("to", "subject", "message");
```

For example (this is straight out of the online help that comes with PHP):

```
mail("joecool@example.com", "My Subject", "Line 1\nLine 2\nLine 3");
```

PHP does need to be installed and setup properly for mail to work. On a Unix-type setup, such as Linux, PHP will use your local sendmail program to send

it. Windows machines use SMTP. If you need to you can edit your **php.ini** file and set the appropiate SMTP host you use to send email.

Here's the code to send the email:

```
<?
if ($REQUEST_METHOD == "POST") {


// Just to be on the safe side - I'll strip out HTML tags

// (scripting code may mess with some email clients)
$realname = strip_tags($realname);
$email = strip_tags($email);
$feedback = strip_tags($feedback);

// setup variables
$sendto = "$email";
$subject = "Send Mail Feedback Using PHP";
$message = "$realname, $email\n\n$feedback";

// send email
mail($sendto, $subject, $message);

}
?>
```

10.0 Uploading Files

It is common to upload files on the websites. This happens especially if we want to submit files in specific file format search as Word or PDF. Using the following scripts, a file is copied from a user machine and saved on the web sever in a directory called “download”

Create Upload Form

First we need to create the form to upload the image. The MAX_FILE_SIZE variable needs to be set to the maximum allowable file size (in bytes) for upload. This is set using a hidden field and for this example will set to 50,000 bytes (approx. 50 kb).

```
<form action="<?=$SCRIPT_NAME; ?>" method="POST"
enctype="multipart/form-data">
<input type="hidden" name="MAX_FILE_SIZE" value="25000"> Upload
<font size="1">Click browse to upload a local file</font>
<input type="submit" value="Upload Image">
</form>
```

```
<?php
// author: Juma Lungo
$_FILES['userfile']['size'];
//The size, in bytes, of the uploaded file.

$_FILES['userfile']['tmp_name'];
//The temporary filename of the file in which the uploaded file was stored on the server.

$_FILES['userfile']['error'];
//The error code associated with this file upload. ['error'] was added in PHP 4.2.0
?>
                    <?php
// In PHP earlier then 4.1.0, $HTTP_POST_FILES  should be used instead of $_FILES.
if (is_uploaded_file($_FILES['userfile']['tmp_name'])) {
        $filename=$_FILES['userfile']['name'];
   copy($_FILES['userfile']['tmp_name'], "download/$filename");
} else {
  echo "The file you are uploading is may be exceeds the maxmum file size: ie
5000000bytes, check it: " . $_FILES['userfile']['name'];
}
/* ...or... */
move_uploaded_file($_FILES['userfile']['tmp_name'], "download/$filename");
?>
```

11.0 Advanced Form Processing with PHP and Javascript

11.1 What Is Javascript, and What Is Not?

Javascript is popular client side scripting language used for controlling objects on the web page. It's code is embedded and sent to the client's browser in the HTML. As long as the client does not make a new request from the server, Javascript will stay and run on the client. In the same way, Javascript's effects stay on the client side until submission to the server. That's why it's client-side. And you cannot access client machine's resources like filesystem or data sources. That's why it's called scripting language, not programming. Javascript deals with objects on the web page only.

Javascript is NOT a security tool. Since your source is sent to the client, who can talk about security? Javascript is used simply to save server bandwidth. "Client-side scripting" and "security" are a contradiction in terms.

Rules are well and fine so long as people obey them. Javascript sets rules for users without any evil intention. But, an unscrupulous and knowledgeable intruder won't mind breaking a few rules in an effort to find a gate, which you forgot to close.

11.2 Utilizing Javascript to Reduce Traffic

Javascript is capable of controlling HTML objects on a page, including form objects. You can minimize network traffic between server and client by utilizing Javascript as a pioneer.

Consider this case: You have a web page including a form with username and password fields, and both cannot be shorter than 6 characters. You have two alternatives: Let Javascript check this constraint first, client side, then have PHP take over, server side; or have everything dealt with by PHP server side. Evaluate the pseudo alternatives when user entered 5 characters in username field:

Javascript + PHP	**PHP**
1. Server sends HTML including Javascript to client. 2. Client fills out form. 3. Javascript checks form and warns user. 4. Client corrects form. 5. Client sends form back to server.	1. Server sends HTML to client. 2. Client fills out form. 3. Client sends form back to server. 4. PHP checks form and finds error. 5. Server sends HTML and error message to client. 6. Client corrects form. 7. Client sends form back to server.

There are two steps (1,5) causing traffic between server and client when Javascript is utilized before PHP, while there are four (1,3,5,7) with PHP only.

Does server workload matter when utilizing Javascript? As for PHP's CGI nature, every time your script called, an instance of php's executable is created, using up memory and processor resources. As the execution finishes, the file dies and resources get freed. Executing a few PHP script won't usually overload the server unless there are huge numbers of php instances running at the same time. The time taken to execute PHP scripts is measured in miliseconds, and you can observe resource consumption using Task Manager (on Windows platforms), or the command ps –ef | grep php (on Unix platforms).

You should take server performance into account,

- When you're expecting hundreds of visitors online at the same time
- When you have very limited server resources (for hardware, hosting etc. reasons)

If you choose average web server software and hardware, performance is not an issue. Therefore, Javascript makes no difference to your server's workload. The biggest factor affecting on your site's performance is still bandwidth.

11.3 What if Javascript has Gone Away?

Although it's can't be claimed that Javascript is available on every browser, certainly every modern browser supports it. However, even with modern browsers, scripting can be manually disabled for security reasons. What happens if Javascript is not enabled, or is circumvented in any way, or if someone sent data by hand instead of filling out your form?

When coping with form processing, you have to ensure that,

- the form is displayed on the client browser and filled out,
- Javascript has gone over the form, and checked that no error occured.

If the client browser does not support Javascript, the form will also be displayed, but JS procedures won't be included. Against these non-JS enabled (or purposely disabled) browsers, you can use the <NOSCRIPT> HTML tag. This tag specifies HTML to be displayed in browsers that do not

support scripting. Using this tag, you will exclude users with very old fashioned or text-based browsers, plus those who manually disabled scripting.

A second thing to remember: Do not use a SUBMIT type for the button which submits the form:

```
<INPUT TYPE="submit" VALUE="OK" onClick="check_form(this);">
```

If you do, the form will be submitted regardless of your JS procedure's return value(s). It's better to use a BUTTON object and set it's onClick event to a related JS function:

```
<INPUT TYPE="button" VALUE="OK" onClick="check_form(this);">
```

This will call the associated Javascript function to check the form, and this function will submit it only if no error occured.

To ensure that the form is 'really' filled out, rather than sent by another application or manually, you may use a noisy image created online as a form checking technique. (See the sign-up page on hotmail.com). Zend.com has a tutorial on Securing Forms with Noisy Images

11.4 Passing Data From PHP To Javascript

Sometimes you may need to use some data back on client side. There are 2 ways of passing data from PHP to Javascript. The first way is directly adding PHP code in the <SCRIPT> tag, appropriate to Javascript syntax:

```
<?php
$user_id = $_GET["uid"];
?>
font color="#007700"><SCRIPT LANGUAGE="Javascript1.2">
   var user_id = <?php echo($user_id);?> ;
   alert("Your user ID is:" + user_id);
</SCRIPT>
```

A second, similar, way is to insert a HIDDEN type input field into the page, and set it's value with PHP. Javascript is able to access this element's value.

```
<?php
$user_id = $_GET["uid"];
?>
<SCRIPT LANGUAGE="Javascript1.2">
   //You can directly access an object's value with id.
   alert("Your user ID is:" + p2j.value);
</SCRIPT>
```

```
<BODY BGCOLOR= "#FFFFFF">
<INPUT TYPE= "hidden" ID= "p2j" VALUE="<?php echo( $user_id);?> ">
```

11.5 Using Regular Expressions

Tips & Tricks (Processing)

When evaluating a user's input, length is not the only criterium you have to check. Remember that a space is also counted as a character. For better auditing;

- Replace multiple spaces with a single space using replace regexp . This is even more important when you explode the input into an array, using space as delimiter. Multiple spaces will result in keys with empty values on the resulting array. (See sample code 1 and 2.)
- Trim string inputs before using them in comparisons. Javascript has no trim function. See function below. (See sample code 5.)
- Check the data types of fields. PHP has very useful functions for this purpose: is_numeric(), is_string(), is_integer() etc.
- Quotes and some other special characters (percent sign, underscore, semicolon etc.) may cause problems on SQL commands. Since JS has no similar function, you'd better use PHP's addslashes() function before querying SQLs. This function adds slashes as an escape character for those special characters, including null. Not only characters, but some keywords are reserved and meaningful for databases systems. Like CREATE, DROP, ALTER, DELETE, GRANT, REVOKE.
- As a global programming rule, try to handle all exceptions in code and never let PHP show an error message. Some error messages may reveal key information (like physical paths) to intruders. You can also display messages like different system messages to confuse novice intruders, such as:

$db_conn = @**mysql_query**('localhost', 'u', 'p') or die('ADO Connection Error 0x80000922: Microsoft Jet Engine couldn\'t be started'); // What the heck ADO is doing there J?

```
$recs = @mysql_query($sql, $db_conn) or die ('ADO Recordset Error 0x800A0BCD: Either BOF or EOF is True, or the current record has been deleted. Requested operation requires a current record.');
```

- Receiving data with a method that you don't expect indicates an intrusion attempt. Unexpected variables are not funny either. Prefer using $_POST['Variable'] syntax instead of $Variable or $_REQUEST['Variable']. Specify which method are you expecting data in. Occasionally you may actually expect more than one. If that is the case, put them in order:

```
// If POST data is set, use it, otherwise use GET data.
$name = isset($_POST['name']) ? trim($_POST['name']) :
trim($_GET['name']);
```

- If a checkbox is not checked, the isset() function will return false when processing. Therefore, before checking the value of a checkbox, always check if it's set.

```
if (isset($_POST['read_licence_aggreement'])) {
   if($_POST['read_licence_aggreement']== "Y") {
         echo "User has accepted licence aggreement.";
   }
}
```

Tips & Tricks (Designing Form)

- Always include MAXLENGTH attribute in text fields to limit the size of field. This will save your script from an unexpected error.
- If you want a field's value to be displayed but without allowing the user to change it, use READONLY boolean attribute set and also set it's TABINDEX attribute's value to -1. (Using HIDDEN type of input will not display this field on form.)

```
<INPUT TYPE="text" NAME="price" READONLY TABINDEX="-1">
```

- If you place multiple checkboxes on a form with the same name, PHP will accept only the value of the last check box (in source order). In case you need all the values as an array, put brackets "[]" (without quotes) after NAME attributes of each one. (See sample code 6.)

Sample Codes

1. Replacing strings using regular expressions in PHP.

```
<?php
// This command will replace multiple spaces with a space.
$result = ereg_replace("\s+", " ", $_POST['sentence']);
echo('Old String:'.$_POST['sentence'].'\n');
echo('New String:'.$result.'\n');
?>
```

2. Replacing strings using regular expressions in Javascript.

```
<SCRIPT LANGUAGE="Javascript1.2">
var r, re;
//String to be checked.
var s = "23th University Summer Olympic Games
is going to be organized by city of Izmir in 2005.";
// Regular expression. Masks at least 1 space on global scope.
re = /\s+/g;

// Replace space(s) with a space.
r = s.replace(re, " ");

//See the results
alert("Old string:\n"+s);
alert("New string:\n"+r);
</SCRIPT>
```

Note: Do not forget to add “g” (global search) after the regular expression’s closing slash, if you want to replace ALL matches. Otherwise, the replace function will only replace the first match.

3. Matching strings with regular expressions in Javascript.

```
<SCRIPT LANGUAGE="Javascript1.2">
// First way of defining regular expressions
// See parameters after second slash?
// i: ignore case g:global search
// This regexp means exact 5 numeric characters
var re_zip1 = /[0-9]{5}/ig;
```

```
// ...and second.
var re_zip2 = new RegExp("/[0-9]{5}/", "ig");

// Testing. Attention please.
alert(re_zip1.test("35140"));  // true. All numeric and 5 digits
alert(re_zip1.test("3510"));   // false. All numeric but 4 digits.
alert(re_zip2.test("a5140"));  // false. 5 digits but not all numeric.
</SCRIPT>
```

4. Common HTML and Javascript source checking the form before submission.

```
<HTML>
<HEAD>
   <TITLE>Sample Form</TITLE>
</HEAD>

<SCRIPT LANGUAGE="JavaScript1.2">
   // Form object
   var f=document.forms(0);
   // Boolen to track if error found
   var foundErr;
   // Form element index number which the first error occured.
   var focusOn;

   function check_form() {
      foundErr = false; focusOn = -1;
      // Username field must be at least 6 chars.
      if (f.user.value.length<6) {
         alert ("Username too short.");
         foundErr = true; focusOn = 0;
      }

      // Password field must be at least 6 chars.
      if (f.pass.value.length<6) {
         alert("Password too short");
         foundErr = true;
         if (focusOn==-1) focusOn=1;
      }

      //Has any error occured?
```

```
        if (foundErr) {
           //Yes. Focus on which the first occurred.
           f.elements.focus(focusOn);
        } else {
           // No. Submit the form.
           f.submit();
        }
     }
</SCRIPT>

<BODY BGCOLOR="#FFFFFF">
   <FORM ACTION="check.php" METHOD="post">
   <TABLE BORDER="0" WIDTH="100%" CELLSPACING="0"
CELLPADDING="0">
      <TR>
         <TD WIDTH="30%" ALIGN="right">Username</TD>
         <TD WIDTH="70%">: <INPUT TYPE="text"
NAME="user"></TD>
      </TR>
      <TR>
         <TD ALIGN="right">Password</TD>
         <TD>: <INPUT TYPE="password" NAME="pass"></TD>
      </TR>
      <TR>
         <TD> </TD>
         <TD><INPUT TYPE="button" VALUE="Submit"
onClick="check_form();"></TD>
      </TR>
   </TABLE>
   </FORM>
</BODY>
</HTML>
```

5. Sample trim function in Javascript.

```
<SCRIPT LANGUAGE="Javascript1.2">
function trimmer(pVal) {
   TRs=0;
   for (i=0; i<pVal.length; i++) {
      if (pVal.substr(i,1)==" ") {TRs++;} else {break;}
   }
```

```
    TRe=pVal.length-1;
    for (i=TRe; i>TRs-1;i--) {
       if (pVal.substr(i,1)==" ") {TRe--;} else {break;}
    }

    return (pVal.substr(TRs, TRe-TRs+1));
}
</SCRIPT>
```

6. Retrieving Multiple Checkbox Values As An Array If brackets are missing on names;

```
...
<INPUT TYPE="checkbox" NAME="s" VALUE="A" CHECKED>Checkbox
1<BR>
<INPUT TYPE="checkbox" NAME="s" VALUE="B">Checkbox 2<BR>
<INPUT TYPE="checkbox" NAME="s" VALUE="C">Checkbox 3<BR>
...
```

processing script

```
<?php
echo '<PRE>('.(gettype($_POST['s']).') ';
print_r($_POST['s']);
?>
```

will produce:

(string) A

But if you add [] after each element's name,

```
...
<INPUT TYPE="checkbox" NAME="s[]" VALUE="A"
CHECKED>Checkbox 1<BR>
<INPUT TYPE="checkbox" NAME="s[]" VALUE="B">Checkbox 2<BR>
<INPUT TYPE="checkbox" NAME="s[]" VALUE="C">Checkbox 3<BR>
...
```

the same PHP code will produce the output below:

```
(array)
Array (
   [0] => A
   [1] => B
   [2] => C
)
```

from which we can see that all three boxes are checked.

12.0 Bibliography

Jim Ferrara (2002). Using MySQL Full-text Searching. Available from: http://www.zend.com/zend/tut/tutorial-ferrara1.php. Accessed [March 8, 2004]

Juma Lungo (2003). Development of Interactive Website: Exploration, Design, Implementation and Evaluation of Website. Available from: http://www.ifi.uio.no/~systarb/hisp/PublicationList.php. Accessed [March 8, 2004]

Marcus Kazmierczak (2003). Web Database Tutorial (PHP & MySQL). Available from: http://www.blazonry.com/scripting/linksdb/index.php. Accessed [March 8, 2004]

Patrick Delin (2002). Creating an Authentication System with Privilege Levels using PHP. Available from: http://www.zend.com/zend/tut/tutorial-delin4.php. Accessed [March 8, 2004]

Peter McNulty (2001). Using PHP and MySQL. Available from: http://www.codingclick.com/article.php/aid/21. Accessed [March 8, 2004]

Tobias Ratschiller (2000). Session Handling with PHP. Available from: http://www.zend.com/zend/tut/session.php. Accessed [March 8, 2004]