MySQL PHP Tutorial

Ferry Boender

1. Introduction

The PHP scripting language's power truly comes to life when used in conjunction with a powerful database like MySQL. User authentication, guest-books, forums, you name it, and it's probably easiest to do when using a database. This tutorial will try and explain, in depth, how to setup MySQL and PHP, how you can get PHP to access data in the database, and how you can effectively use that data in you web-pages. It uses mostly examples which are related to a forum. You can look at this forum and download it's sourcecode from:

forum (http://www.electricmonk.nl/data/projects/small_programmings/forum.tar.gz)

2. Setting up PHP and MySQL

This section does not go into the details on exactly how to setup PHP and MySQL, just how to get MySQL working from within PHP. So in order to follow this tutorial you should already have PHP and MySQL installed on your system.

Got that? Okay. First of all, you'll have to be sure you can use php's MySQL features. How can you tell if you can? Just create the following script:

```
<?
// phpinfo.php
phpinfo();
>
```

Put it somewhere in your server-path (where it can be called from your browser) and point your browser to it. You should now get a page witch contains loads of information concerning PHP. If you do not get this info, but some kind of error instead, there's probably something wrong with your PHP setup. Please check the proper documentation from PHP to get further help on that topic.

You should search the page for a section called 'mysql'. It should probably have some information like this:

```
MySQL

MySQL support Enabled
--
Active persistent links 0
Active links...
```

etc.

Is it there? Okay, you're all set. Proceed to the next section of this tutorial.

If it isn't, then you probably don't have the mysql.so module installed and configured. You should check your hard-disk to see if a file called 'mysql.so' exists somewhere on it. It's most probably somewhere in the /usr/lib/php4 directory. An easy (but unreliable) method to see if you've got it,is typing 'locate mysql.so' at the command prompt. Can't find it? Then you don't have it installed on your system. Please use your Linux distribution's package manager to install it, or recompile PHP from source with the --with-mysql option.

If you've got the mysql.so module, but it still doesn't work, then it's probably not loaded into PHP automagically. There are two ways to load it:

Always load it by default:

edit the php.ini file (/etc/php/php.ini, or /etc/php/apache/php.ini) en find the

```
; Dynamic Extensions
```

section. In this section add the following line:

```
extension=mysql.so
```

restart apache, and reload the phpinfo.php script to see if the mysql section shows up.

Load it on demand:

PHP allows you to load a module on demand. This is how it is done:

```
<?
// phpinfo2.php
dl(mysql.so);
php_info();
?>
```

The mysql section should now show up when you load the above script in your browser.

If you still can't get it to work, I suggest you consult your PHP installation manual on how to get it to work.

3. How it works

In this section I will try to explain how data is retrieved from the database and used in MySQL. It's purely theoretical, so if you just want to get on with the actual code example stuff, then quickly scroll down.

I presume that you already have a MySQL server and some kind of example database to which to connect. If you don't, just read this section anyway, and the read the 'Using MySQL' section.

Here's the steps which need to be performed in order to retrieve data from a MySQL database:

- 1. Create a connection to the MySQL database.
- 2. Select the database you wish to access.
- 3. Send a query to the database.
- 4. Retrieve some results from the database.
- 5. Use those results in your page.

That's it. Usually you repeat steps 3 through 5 a couple of times, but only onceconnect to MySQL and select the correct database.

Okay. Let's walk through these steps, a little more in depth this time:

- 1. Create a connection to the MySQL database.

  The connection will require a username, password and a host on which the MySQL server runs. For me, this usually is 'localhost'. The username and password depend on which database I want to access. When creating databases in MySQL, never ever give all the separate databases the same username and password. This is Stupid(r).

- 2. Selecting the database is easy. Just supply the database's name.
- 3. Send a query to the database.

  This one is a little trickier. It requires knowledge of MySQL (or general SQL) queries in order to operate. If you have no knowledge of SQL queries whatsoever, I suggest you read some kind of MySQL tutorial. Later on in this manual I will quickly explain some common SQL queries, but it's not really in depth. Anyway, the query is a string with which you tell the MySQL server what information in the database you want to retrieve, change or remove. Here's a small example of a query:

  "SELECT * FROM users WHERE name="John";

  This will retrieve all fields from the table 'users' in which the field 'name' contains 'John'. Easier said: We want John's user information.

  Please note that this does not actually retrieve the data. It just tells the MySQL server to 'prepare' the data so we can retrieve it at the next step.

- 4. Retrieve some results from the database.

  Now we retrieve the results, which the server 'prepared' for us. This can mean a lot of things, since there are a lot of results we can retrieve. For instance, we can retrieve the field we requested from the database (* , which means everything, or in this case 'all fields'). But we could also just look if the database found any user information for the requested user 'John'.

- 5. Use those results in your page.

Well, of course it all depends on what you were retrieving. If you would like to know if the user 'John' actually existed within the 'users' table, you could output some information about that. Or you could show 'John's user information. Whatever.

4. Using MySQL.

Now for some information about the use of MySQL.

In order to retrieve data from a database, there must first be a database. There's number of programs you can use to manipulate databases in MySQL. One of them, and in my opinion the best, is the command-line driven 'mysql' client. if you feel more comfortable using a graphical client, be my guest, but if I ever need to manipulate a database through a telnet or ssh connection, I'd rather be using a command-line client.

When you installed the MySQL server, you probably also installed the MySQL command-line client (if you are using some sort of package manager). From now on, I will assume you are using this client.

Okay, now let's see how we can create a new database, and a user to match.

4.1. Getting info on databases and tables.

Fire up the MySQL client by typing the following:

```
[todsah@sharky]~$ mysql -u todsah
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 464 to server version: 3.22.32-log

Type 'help' for help.

mysql> _
```

Here we are at the MySQL command-prompt. Please note that if your MySQL account has a password, you should append a -p option to the 'mysql -u todsah' line.

First, we'll check out the server to see if some databases already exist:

```
mysql> show databases;
+------------+
| Database   |
+------------+
| forum      |
| intranet   |
| mysql      |
| news       |
+------------+
4 rows in set (0.26 sec)
```

Okay, so we've got 4 databases here. We can select one with the following command:

```
mysql> use forum;
Database changed
```

We are now working on the 'forum' database. Please note that you should NEVER FORGET THE SEMI-COLON (;) If you've ever used any real programming language, you'll be used to the semi-colon thingy.

Right! Now let's see what tables this 'forum' database contains:

```
mysql> show tables;
+-----------------+
| Tables in forum |
+-----------------+
| posts           |
+-----------------+
1 row in set (0.01 sec)
```

a table 'posts'? Let's refresh my memory and look how it's build up:

```
mysql> describe posts;
+-------------+----------------------+------+-----+-----------+----------------+
| Field       | Type                 | Null | Key | Default   | Extra          |
+-------------+----------------------+------+-----+-----------+----------------+
| id          | mediumint(8) unsigned |      | PRI | 0         | auto_increment |
| parent      | mediumint(8) unsigned | YES  |     | 0         |                |
| authorname  | varchar(40)          | YES  |     | anonymous |                |
| authoremail | varchar(100)         | YES  |     | NULL      |                |
| date        | datetime             | YES  |     | NULL      |                |
| subject     | varchar(255)         | YES  |     | NULL      |                |
| text        | text                 | YES  |     | NULL      |                |
+-------------+----------------------+------+-----+-----------+----------------+
7 rows in set (0.00 sec)
```

Ah yes! That's right. Let me explain to you.

id:

> This field uniquely defines each row in the database (For your information, a row is made up out of fields, a table is made up out of rows, and a database if made up out of tables) The 'mediumint(8) unsigned' tells us, it's a numerical field, which can't have negative numbers (unsigned).The MySQL documentation tells us: MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]
>
> A medium-size integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215. (source: http://www.mysql.com/documentation)
>
> The 'null' part (all the YES parts under the Null header) of the table above shows, that this field dares not to be empty, a number should always be assigned to it. This is automatically done by MySQL thanks to the 'auto-increment' part.

parent:

Since this is a forum database, this field lets us know to which other post (id) the current row replies. It's Type is the same as the 'id' field, since it refers to it.

authorname:

varchar(40), a variable length string (char) of up to 40 characters. Default value (if not specified) is 'anonymous'.

date:

This holds the date on which the post was made. I'll get back to dates later.

text:

A field with the Type 'text': A BLOB or TEXT column with a maximum length of 65535 (2^16 - 1) characters. Says the MySQL manual. So we can insert 65535 characters of text into this field.

4.2. Creating a database and user.

Let's pretend for a moment that the above database does not yet exist. We'll create it using the following steps:

- Create a new file, using your favorite editor (I use joe), name it something like forum.mysql. This file will contain a couple of commands, just as we would enter them in the MySQL console.

Here's the contents of the file:

```
❶ CREATE DATABASE forum;
❷ GRANT SELECT,INSERT,DELETE,UPDATE ON forum.* TO forum@localhost IDENTIFIED BY 'secretpw';
❸ USE forum;

❹ CREATE TABLE posts
    (
❺   id MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
      parent MEDIUMINT,
      authorname VARCHAR(40) DEFAULT 'anonymous',
      authoremail VARCHAR(100),
      date DATETIME,
      subject varchar(255),
      text TEXT
❻ );
```

❶ Here's where we tell MySQL to create a new database. If the database already exists, you'll get an error. In that case, you can delete the old database by typing:

```
mysql> DROP DATABASE forum;
```

Be careful with that. It deletes all data.

❷ This line creates a new user, which can access all tables in the forum database. It only has the rights to manipulate data within the tables themselves. This is what the rights mean:

SELECT

Retrieve data from tables.

INSERT

Insert new rows of data into tables.

DELETE

Deleting of rows from the tables.

UPDATE

Changing existing data in the tables.

The forum@localhost part identifies the username, and from which hosts that user may use the database. In this case, the user forum may only connect to the database from the same machine on which the server runs.

secretpw' is his password. Hence, he should connect like this:

```
[todsah@sharky]~$ mysql -u forum -h localhost -p
Enter password:
Welcome....
```

Please note, that in some cases, the user root does not yet have the rights necessary to GRANT other users. Why this sometimes is the case, is a mystery. To fix this, you can let the root user first give itself the right to GRANT permissions on other users. I know, it sounds strange, but it worked for me (and believe me, I spent quite some time figuring THAT one out) Here's how to do that:

```
GRANT all privileges ON *.* TO root@localhost identified by 'secret' WITH GRANT OPTION;
```

❸ Tell MySQL we want to work on the 'forum' database.

❹ Okay MySQL client. We are now gonna create a new table in the 'forum' database called 'posts'.

❻ the column specifications end here. After this, you can repeat 5 through 6 in order to create extra tables

- Now we must tell MySQL that it should run the commands we just entered into this file. The reason we are using a file, is because if you accidently made a typo in one of the commands, and you happened to be using the command-line client, you can start all over again. This inconvenience will not happen if we use a file. Alright. Lets tell mysql to parse the file. It's done like this:

```
[todsah@sharky]~$ mysql -u root -p < forum.mysql
Enter password:
[todsah@sharky]~$ _
```

No output.. that means no errors. That's a Good Thing. It doesn't neccasery have to done by the 'root' user. The user just needs to be able to add databases and grant rights to users. Lets check out if it worked:

```
[todsah@sharky]~$ mysql -u forum -h localhost -p
Enter password:
Welcome...
mysql> describe forum.posts;
```

You should now see your created table.

There you have it. We've created a database, a table and a user to match. Ain't it grand?

4.3. Miscellaneous MySQL queries.

This section explains some basic MySQL queries. First I'll summarize what we've already learned:

`CREATE DATABASE forum;`

Creates a new database forum, if it doesn't yet exists.

`GRANT ...`

Tells MySQL to create/update the rights of a user on certain tables/databases.

`SHOW databases;`

MySQL shows which database are available.

`USE forum;`

We want to work on the forum database.

`SHOW tables;`

MySQL shows which tables are available in a database.

`CREATE TABLE posts ( ... );`

Creates a new table named 'posts'.

`DESCRIBE posts;`

MySQL describes the table posts for us.

`SELECT * FROM users WHERE name="John";`

Retrieved all information about 'John' from the table users.

`DROP DATABASE forum;`

Completely DELETE's the database named 'forum'.

4.3.1. Retrieving data from tables (SELECT).

As we have seen, the SELECT statement allows us to retrieve data from a table. Let's dissect the statement:

```
SELECT [columns] FROM [table] WHERE [specification];
```

example:

```
mysql> SELECT * FROM posts WHERE id=1;
```

retrieves all fields from table posts where the 'id' field is '1'.

example:

```
SELECT id,authorname,authoremail FROM posts;
```

retrieves the fields id, authorname and authoremail from all rows in the table posts;

But the SELECT statement can do lots more:

```
SELECT [columns] FROM [table] WHERE [specification] ORDER BY [field [ASC / DESC]] LIMIT [from, rows];
```

The new stuff:

```
ORDER BY [field [ASC / DESC]]
```

This allows us to sort the results either ascending or descending. Note that only the retrieved results are sorted, not the entire table.

```
LIMIT [from, rows]
```

Here we can specify how many results we want. I.e: Lets take a guest-book. We don't want the whole table on one page. We want to be able to 'browse' through the entries, say, 15 at a time. Then here's what we do: LIMIT 0,15. On the next page, we do: LIMIT 15,15. Then, on the next page we do: LIMIT 30,15. That last one will show 15 rows, leaving out the first 30 (which we've already seen).

example:

```
SELECT * FROM posts ORDER BY date LIMIT 0,15;
SELECT * FROM posts ORDER BY date LIMIT 15,15;
```

or, for a reversed list:

```
SELECT * FROM posts ORDER BY date DESC LIMIT 0.15;
```

The WHERE specification may also contain wild-cards. In order to use wild-cards, you must add the keyword LIKE to the statement:

```
SELECT * FROM posts WHERE subject LIKE '%forum%';
```

This will retrieve all fields from each row that has the text 'forum' in the subject. You can also search only the beginning or end of a field like this:

```
SELECT * FROM posts WHERE subject like 'forum%';
SELECT * FROM posts WHERE subject like '%forum';
```

If you want to combine multiple specifications, you can do it the following way:

```
SELECT * FROM posts WHERE (subject like '%forum%') AND (authorname='toDSaH');
```

4.3.2. Inserting data into tables (INSERT).

What good is a nice database, if you can't fill it with data? For that we use the INSERT statement. There's two ways to do that:

```
INSERT INTO [table] [col1,col2] VALUES ( [val1, val2] );
```

or

```
INSERT INTO [table] SET col1=val1, col2=val2;
```

In the first one, you will need to make sure that the 'col' and 'val' part are in the correct order.

Some notes:

- For character column types, the values should be encapsulated with quotes (' , or ") like this: 'John'.
- id fields (Primary key, auto-increment, that kind of stuff) shouldn't be specified when inserting data. They automatically get assigned a value by MySQL. That doesn't mean that you can't. Some specific cases require you to set the id manually.

example:

```
INSERT INTO posts VALUES (",0,'toDSaH','todsah@arise.nl', 2000-10-12 22:38:20,'Test','Hello, this is a test.
```

See? The first field/value/column is left empty, because it's automagically assigned a unique number by MySQL.

example:

```
INSERT INTO posts SET authoremail='todsah@arise.nl', date=2000-10-12 22:38:20, text='Hello, this is a test.'
```

In this example the columns id, authorname and subject are not specified. Therefor they will get default values from MySQL. I.e authorname will become 'anonymous' by default.

4.3.3. Changing data in tables (UPDATE).

Most humans are only human, so they make mistakes. It is therefor a good thing that you can alter information that's already in the database. It's done like this:

```
UPDATE [table] SET col1=val1, col2=val2 WHERE [specification];
```

Pretty straight forward right?

example:

```
UPDATE forum SET authorname='anonymous' WHERE authorname='toDSaH';
```

This will change all ’authorname’ fields which have the value ’toDSaH’ to ’anonymous’.

4.3.4. Deleting data from tables (DELETE).

Deleting a row from a table is easy:

```
DELETE FROM posts WHERE id=1;
```

Deleting multiple rows from a table is just as easy:

```
DELETE FROM posts WHERE id < 10;
```

will delete all rows which id is lower than 10.

5. Using a MySQL database in PHP

Now that we know how MySQL works, were gonna use that knowledge in PHP. The reason that you should know all those MySQL statements, is that PHP functions use these statements to retrieve data from the MySQL server.

5.1. Retrieving data from the database.

Here’s a small example on how to retrieve information from a database:

```
<?
❶ dl ("mysql.so");

❷ $db = mysql_connect (’localhost’, ’forum’, ’secretpw’;
❸ mysql_select_db(’forum’, $db);

❹ $result = mysql_query("SELECT * FROM posts ORDER BY date",$db);
❺ if (!$result) { echo "SELECT query failed"; }

❻ while ($row = mysql_fetch_row($result)) {
❼ echo "ID:           " . $row[0] . "<br>\n";
    echo "Replies to:   " . $row[1] . "<br>\n";
    echo "Author:       " . $row[2] . "<br>\n";
    echo "Author email: " . $row[3] . "<br>\n";
    echo "Date:         " . $row[4] . "<br>\n";
    echo "Subject:      " . $row[5] . "<br>\n";
    echo "Text:         " . $row[6] . "<br>\n";
}
?>
```

Well, that’s easy, ain’t it? Remember, the line numbers are there just for the convenience of it. They’re not actually part of the code. Let’s dissect it:

❶ Dynamically load the mysql.so module. You'll only have to do this if it isn't loaded automatically by PHP.

❷ Create a connection the server at localhost using username 'forum' and password 'secretpw'. The connection is then assigned to the variable $db. This can be compared to: mysql -u forum -h localhost -p

❸ We want to work on the forum database, use the connection $db which we made at line 04. This can be compared to: USE forum;

❹ Tell the server to which we are connected ($db) that we want the results from the query "SELECT * ... BY date". Please note that at this point we don't actually have the results yet. We've only told the database to prepare the data for us. We also specify the database connection variable ($db). In this case, it's not really necessary, because we've only got one database connection open. But we'll do it anyway, to prevent future mistakes.

❺ If the SELECT query fails, show an error message. If you want to be complete in your error checking, you should also make sure that the while statement (line 10 through 18) do not run unless the SELECT statement succeeded.

❻ $row = mysql_fetch_row($result) will retrieve the results from the query we prepared at line 07. The while statement means: retrieve data as long as there's some left. The results are stored in the $row variable. Note: It's only one row at a time. The next row of results are fetched at the next call of mysql_fetch_row($result)

❼ Here we output the data we fetched from the database. Each array element in $row resembles 1 field in the row. They are number in the same order as how we CREATEd the table.

Okay, so it works. Great. But what if we change the order of the fields in the database? For instance, we DROP the entire table, and create a new one, in which we want to store the author's homepage? The new table would look like this:

```
+-------------+----------------------+------+-----+-----------+----------------+
| id          | mediumint(8) unsigned |      | PRI | 0         | auto_increment |
| parent      | int(10) unsigned      | YES  |     | 0         |                |
| authorname  | varchar(40)           | YES  |     | anonymous |                |
| authoremail | varchar(100)          | YES  |     | NULL      |                |
| authorurl   | varchar(100)          | YES  |     | NULL      |                |
| date        | datetime              | YES  |     | NULL      |                |
etc...
```

If we look at the PHP example, we'll notice that $row[4] used to be the date, but now it's the 'authorurl' field. Instead, 'date' is now $row[5]. Time for a complete code rewrite? Yes! But this time, we're going to do it another way:

```
while ($row = mysql_fetch_array($result)) { ❶
echo "ID:              " . $row["id"]          . "<br>\n";
echo "Replies to:      " . $row["parent"]      . "<br>\n";
echo "Author:          " . $row["authorname"]  . "<br>\n";
echo "Author email:    " . $row["authoremail"] . "<br>\n";
echo "Author homepage: " . $row["authorurl"]   . "<br>\n";
echo "Date:            " . $row["date"]        . "<br>\n";
echo "Subject:         " . $row["subject"]     . "<br>\n";
echo "Text:            " . $row["text"]        . "<br>\n";
}
```

Notice the differences? Instead of fetching the results using mysql_fetch_ROW, we are now using mysql_fetch_ARRAY. The difference between these functions is, that we can now use names to identify the fields. So instead of saying: $row[2] we now say $row["authorname"]. The mysql_fetch_array() function is

somewhat slower than mysql_fetch_row(), but it will give you less problems in the long run. If you're absolutely certain that the table structure will _never_ change, you may just as well use the mysql_fetch_row() function. Now, if we only want to list all author's and subjects of all posts, we don't have to retrieve all the other information, that would be a waste of CPU cycles. Here's how we do it:

```
?>
dl ("mysql.so");

$db = mysql_connect ('localhost', 'forum', 'secretpw';
mysql_select_db('forum', $db);

$result = mysql_query("SELECT author,subject FROM posts ORDER BY date",$db);
if (!$result) { echo "SELECT query failed"; }

while ($row = mysql_fetch_array($result)) {
echo "Author:       " . $row["author"]  . "<br>\n";
echo "Subject:      " . $row["subject"] . "<br>\n";
}
<?
```

The difference is in the SELECT query. In the first examples I used 'SELECT * FROM' and now I'm using 'SELECT author,subject FROM'.

5.2. Inserting data into the database.

Ok, here's some more small examples:

```
<?
dl ("mysql.so");

$db = mysql_connect ('localhost', 'forum', 'secretpw';
mysql_select_db('forum', $db);

$result = mysql_query("INSERT INTO posts SET parent=1,
authorname='toDSaH', authoremail='todsah@arise.nl',
subject='Wassup dog?',text='This is the text field'",$db);

if (!$result) { echo "INSERT query failed"; }
?>
```

Here we insert some new data into the table. Not all the fields are specified, so they will get their default values from MySQL. Of course the mysql_query line shouldn't be spread over multiple lines in the real code. There's also another way to insert data into the database:

```
<?
dl ("mysql.so");

$db = mysql_connect ('localhost', 'forum', 'secretpw';
mysql_select_db('forum', $db);

$result = mysql_query("INSERT INTO posts VALUES ('','0','Anonymous',
'anonymous@nohost.com',now(),'My post','This is my post'",$db);
```

```
if (!result) { echo "INSERT query failed"; }
?>
```

In this example, we don't specify any columns, so we have to specify ALL values for each column. Since MySQL doesn't know which column we are refering to, this query type is quite error sensitive. I suggest you only use the first INSERT example.

5.3. Deleting data from the database.

To delete data from the database we use the following:

```
$result = mysql_query("DELETE FROM posts WHERE author='anonymous'",$db);
```

I've left out the rest of the program, because you should be able to create that part yourself by now. This query deletes all anonymous posts.

Another example:

```
$result = mysql_query("DELETE FROM posts WHERE id < 11",$db);
```

This will delete all the posts who's ID is smaller than 11.

Okay, now let's say you want to delete all posts which contain nasty words:

```
$result = mysql_query("DELETE FROM posts WHERE text LIKE '%fuck%'",$db);
```

This query will delete all posts who's text contain the word 'fuck' (LIKE '%fuck%') The percent-sign (%) is a wildcard which matches one or more characters. If you leave one of the percent-signs out, you could only search for text at the beginning or end of the 'text' field contents.

5.4. Changing data in the database.

If you want to change data in the database you can do it like this:

```
$result = mysql_query("UPDATE posts SET author='Anonymous coward' WHERE
author='anonymous'",$db);
```

This will change all author fields which are 'anonymous' into 'Anonymous coward'. You may specify multiple SET's by comma-delimiting them.

5.5. Storing dates and times in MySQL tables.

MySQL has it's own internal column format for storing date and time stamps. But I've found that they are hard to use in PHP, since their format is fixed. Although I might be wrong, and the internal MySQL DATETIME column is sufficient, I've found another solution, which works just fine for me.

I use a INT column type for date and time storage. In this I save a number, which represents the date and time. This is called a timestamp. You can get the timestamp for a date and time in PHP by using the mktime() function. Here's the description of the function from the PHP manual:

```
int mktime (int hour, int minute, int second, int month, int day, int year [, int is_dst])
```

So **`mktime (22,48,0,11,27,2000)`** will return the timestamp for `27 December 2000, 22:48 : 975361680`

You can reverse a timestamp with the `date()` function. It's description:

```
string date (string format [, int timestamp])
```

`date ("d F Y, G:i", 975361680);` will return: `27 November 2000, 22:48`

The `date()` function's format parameter can contain a large number of possible way's to describe a date/time. This is a great advantage. If you only want to show the year of a timestamp, you could just use **`date("Y",975361680)`**, which will only return "`2000`".

The other advantage is, that the MySQL database can be easily sorted by date using timestamps. In fact, MySQL internally also uses timestamps for it's DATETIME column type. Yet another advantage is, that it's easy to substract two timestamps from each other, and show how many days apart they are, for instance. If you want your dates to be displayed in your own language, use the following:

```
<?
setlocale(LC_TIME,"dutch");
print strftime("%A, %e %B %Y (%R)");
?>
```

Will print: `zondag, 15 januari 2001 (11:51)`. Please refer to PHP's Date and Time functions (http://www.php.net/manual/en/ref.datetime.php) section for more date and time related functions. See the setlocale function in the manual to learn more about using different languages in your php script.

6. References.

Here are some references which might come in handy:

General MySQL documentation (http://www.mysql.com/documentation/)

General PHP documentation (http://www.php.net/manual/)

MySQL related PHP documentation (http://www.php.net/manual/ref.mysql.php)

7. Copyright / Author

Author:

Ferry Boender
De Cotelaer 28
3772 BP
Barneveld
The Netherlands
<f (DOT) boender (AT) electricmonk (DOT) nl>

7.1. Document changes

This document has undergone the following changes:

Revision History

Revision 0.5 ? Revised by: FB
Rewrote the whole thing to docbook SGML.
Revision 0.4 ? Revised by: FB

Added information about dates in different languages to section 5.5.
Revision 0.3.1 ? Revised by: FB
Fixed a small typo (!#$result -> !$result) in section 5.2, thanx to crasp
Revision 0.3 ? Revised by: FB
Added a reference to the sourcecode for the forum database which is used as an example throughout this tutorial. More justificatior
Revision 0.2.2 ? Revised by: FB
Fixed some more dumb mistakes ($rows, $row). Did some justification of text.
Revision 0.2.1 ? Revised by: FB
Fixed some really nasty mistakes. (SORT BY, doh!)
Revision 0.2 ? Revised by: FB
Added more MySQL queries from PHP. Split up section 5 into sub-sections. Added a little note about creating users. Added date ar
Revision 0.1 ? Revised by: FB
Initial document