webmonkey/programming/

PHP/MySQL Tutorial

by Graeme Merrall

Unless you've been living on Mars for the last six to eight months, you've heard of open source software (OSS). This movement has got so much momentum that even the big boys are taking notice. Companies like Oracle, Informix, and a host of others are releasing their flagship database products for that poster child of the OSS movement, Linux.

Having a massively complex RDBMS (relational database management system) is all well and good if you know what to do with it. But perhaps you are just getting into the world of databases. You've read Jay's article and you want to put up your own data-driven Web site. But you find you don't have the resources or desire for an ASP server or some pricey database. You want something free, and you want it to work with Unix.

Enter PHP and MySQL. These two make up what must be the best combination for data-driven Web sites on the planet. You needn't take my word for it. An unofficial Netcraft survey shows that PHP usage has jumped from 7,500 hosts in June 1998 to 410,000 in March 1999. That's not bad. The combination was also awarded Database of the Year at Webcon98, where it received a lovely tiara.

MySQL is a small, compact database server ideal for small - and not so small - applications. In addition to supporting standard SQL (ANSI), it compiles on a number of platforms and has multithreading abilities on Unix servers, which make for great performance. For non-Unix people, MySQL can be run as a service on Windows NT and as a normal process in Windows 95/98 machines.

PHP is a server-side scripting language. If you've seen ASP, you'll be familiar with embedding code within an HTML page. Like ASP, PHP script is processed by the Web server. After the server plays with the PHP code, it returns plain old HTML back to the browser. This kind of interaction allows for some pretty complex operations.

In addition to being free (MySQL does have some licensing restrictions though), the PHP-MySQL combination is also cross-platform, which means you can develop in Windows and serve on a Unix platform. Also, PHP can be run as an external CGI process, a stand-alone script interpreter, or an embedded Apache module.

If you're interested, PHP also supports a massive number of databases, including Informix, Oracle, Sybase, Solid, and PostgreSQL - as well as the ubiquitous ODBC.

PHP supports a host of other features right at the technological edge of Internet development. These include authentication, XML, dynamic image creation, WDDX, shared memory support, and dynamic PDF document creation to name but a few. If that's not enough, PHP is easy to extend, so you can roll your own solution if you're programming savvy.

Finally, since both efforts are collaborative in nature, there's always plenty of support from documentation and mailing lists. Bugs are fixed rapidly, and requests for features are always heard, evaluated, and if feasible, implemented.

Enough talk! Let's go over what we're going to cover in this tutorial.

> Lesson 1 is going to cover the installation of these products on both Unix and Windows systems. If you don't need to worry about that (you're working on your ISP's machine, perhaps), jump right to the first example scripts, where the magic starts.
>
> In Lesson 2 we'll look at some more complex scripting goodies, including looping, form input, and sending data from and to the database.
>
> Lesson 3 will cover validation and techniques for making your PHP scripts smart and clean.

Let's roll.

Installing MySQL

Let's jump straight in, grab ourselves a copy of these great packages, and get hacking! This isn't simple stuff. There are lots of options available to you for obtaining, compiling, and installing the software. Let's deal with MySQL first, as we'll need it before we get PHP going.

MySQL central is http://www.mysql.com/. As befits a program of its stature, there are a zillion mirrors located all over the globe, so do the Internet a favor and pick the one closest to you.

You've got plenty of choices at this point. If you're a do-it-yourselfer, then grab the source code. If you're not that brave, there are some precompiled binaries for other platforms already available for download.

In addition, there is a shareware version of MySQL for Windows users. It is an older version of MySQL. If you want the latest version, you'll have to purchase a license. There are also ODBC drivers that let your applications talk to MySQL. Various other exciting bits and pieces are lurking about on the site, too, so take a look.

The precompiled Unix versions and the Windows version are as simple as unpacking and going, and they don't require much explanation. So let's compile from the source code. Windows users, please keep in mind that you need to run `mysqld` in the `mysql/bin` directory.

Download the compressed file into your source directory and uncompress and untar it using gzip and tar. The fast way of doing this is to type:

```
gunzip < mysql-xxxx.tar.gz | tar xvf -
```

The `xxxx` is where you put the version number. This will create a directory called mysql-xxxx, which contains all the source files. Move to that directory by typing `cd mysql-xxxx` and check out the various README and INSTALL files. They're lifesavers in sticky situations.

MySQL comes with a handy configuration script. Simply type `./configure` and let things take care of themselves. If you need to specify what happens and where, typing `./configure --help` gives you a list of options to choose from. For example, if you're compiling on a machine with little memory, you can opt for the `--with-low-memory` flag. I like MySQL to install in one handy directory tree rather then in various locations on my machine, so I specify an install location with the `--prefix` flag.

You can also specify lots of other options, such as what to compile and what to skip.

Let's assume that we want everything under `/usr/local/mysql` on our server. This means we'd type `./configure --prefix=/usr/local/mysql`.

The configure script will run and inspect your system and then build the necessary files to successfully compile. If it fails, you'll usually get a helpful error message saying why. Quite often, you'll find the script will fail when it's looking for threading libraries. Check that you've got MIT-pthreads installed on your machine, and if not, add them. Linux users will have to download LinuxThreads. These are critical libraries that allow MySQL to multithread (i.e., run multiple versions of itself). Recent distributions of Linux may already have these libraries installed.

If everything goes according to plan, simply type `make` and go get a coffee. MySQL is a complex program and takes some time to compile. If you get an error, check the documentation to see if there is anything specific that you've missed for your particular OS.

Next, type `make install` and all the necessary files will be installed in all the necessary spots. Now you're almost ready to roll! If you are a MySQL virgin and you've never installed MySQL before, you need to create the default permissions, so type ... `scripts/mysql_install_db` to set these up.

That's it. We're ready to roll. All we need to do is add the ability to start and stop the server at boot-up and shutdown times. And yes, there's a script for that as well. Typing `mysql.server start` starts the server, and `mysql.server stop` stops the server. It's kind of obvious, really. To start the server manually (so you can play without rebooting) enter the root directory in your MySQL installation (/usr/local/mysql) and type `bin/safe_mysqld &`.

You're halfway there. Now on to PHP.

Installing PHP

Phew! Hopefully you've got MySQL all up and running by now. That was almost fun! Now for PHP ... This process is slightly easier, but the array of options is dazzling. Don't be daunted, though. You can always go back later and recompile PHP to add or remove options as needed.

The home of PHP is http://www.php.net/. The PHP site is a mine of information, from project listings to bug reports. As with MySQL, you should choose a nearby mirror. Obviously you'll want the Downloads section to get PHP. I'll be taking you through an installation of PHP3. To learn how to tackle PHP4, read Webmonkey Julie's detailed PHP4 installation instructions.

Your range of options here is a little more limited. A few precompiled binaries are available, but these are experimental. If you're on anything except a Windows platform, grab the source code and compile it yourself.

But first let's cover Windows. When using PHP, a common practice is to develop on a Windows machine and then run your site on a Unix server. It may end up that you will do this yourself, which means you need to be proficient in installing on both platforms.

Let's grab the Windows binary and uncompress it using our favorite Zip decompression tool into a directory on your C drive called php3. The supplied README file deals with the installation in some detail, but here's the *Reader's Digest* version: If you want to install PHP to a folder other than `c:\php3`, you'll need to edit the .inf file that comes with PHP.

In the php3 directory, you'll find a lot of .dll files. Take all the .dll files that don't begin with `php_` and move them into your `\windows\system` directory. Then rename php.ini-dist to php3.ini and move it into your `\windows` directory. If you open up that file, you'll see there are lots of interesting things to change. For now just "uncomment" the line:

```
        extension=php3_mysql.dll
```

If you're using Apache for Win32, set up Apache to recognize and parse PHP files. Depending on the version of Apache you're using, you'll need to add the following to either the httpd.conf or srm.conf file:

```
ScriptAlias /php3/"c:/path-to-php-dir/"
AddType application/x-httpd-php3 .php3
Action application/x-httpd-php3"/php3/php.exe"
```

Or if you're using IIS or PWS, right-click on `php_iis_reg.inf` and select 'Install'. You'll need to reboot for IIS to see this change.

OK, now that Windows is out of the way, let's get to Unix. Of course, we'll be compiling from source code. As with MySQL, download and unpack the source code. Again, PHP comes with a configure script. You can't get away with going for defaults here, though. Run `./configure -help | more` to see pages and pages of new and interesting options. You have to decide between compiling as a CGI or as an Apache module. If you are using the Apache Web server and you are able to recompile it, use the module: It's faster and easier to use. Otherwise, you can go with the CGI version. We also need to compile in MySQL support.

For now we'll assume that we're running the module with MySQL support. If you want to add other options or other libraries, you can do this later. Type:

```
        ./configure --with-apache=/path/to/apache/dir --with-mysql=/usr/local/mysql
```

Skip the -with-apache option if you're creating a CGI version. The configure process will run and produce the relevant system files. Now simply type `make` again.

It's time for another coffee. If you start feeling a bit nervous and shaky at this point, don't worry about it. We all get a little anxious during our first PHP install. Have some more coffee.

If you've created a CGI version, you're now ready to roll. Simply copy the resulting executable file into your CGI file. For Apache module users, type `make install` to copy files to your Apache directory. From there, follow the instructions to add a module to Apache and recompile.

You'll need to tell your Web server how to process pages through the PHP program now. If you're not using Apache, you'll need to check your Web server documentation on how to get it to process documents with a .php3 extension. Apache 1.3.x users can simply add `AddType application/x-httpd-php3 .php3` to the httpd.conf or srm.conf file. If you're using the CGI version, you'll need to add the following before `AddType`:

```
        ScriptAlias /php3/"/path-to-php-dir/" AddType application/x-httpd-php3 .php3
        Action application/x-httpd-php3"/php3/php"
```

That's it. With any luck, you've now got MySQL running and PHP functioning. Don't forget to check the FAQs and documentation if you get stuck. Also try the mailing lists.

Now that we've managed all that, lets put this stuff in motion!

Your First Script

You'll be glad to know that the really tricky stuff is behind you. Installation of software is always a black hole because so much changes from system to system. But with any luck your database is up and running, and PHP is compiled and installed with our Web server and able to recognize documents with .php3 extensions.

Let's dive right in and write our first script. Create a text file containing the following:

```
<html>

<body>


<?php

$myvar = "Hello World";

echo $myvar;

?>


</body>

</html>
```

Now call up the URL, for instance, http://myserver/test.php3. You should see a page containing the text "Hello World." If you get an error message, check the PHP documentation to see if you set things up properly.

That's it! That's your first PHP script. If you view the HTML source for the page, you'll see that there is only the text. Hello World

That's because the PHP engine has examined the page, processed any code blocks that it found, and returned only HTML.

The first thing you'll notice about the script above are the delimiters. These are the lines that start `<?php`. This indicates the start of a block of PHP code, and `?>` indicates the end of the block. The power of PHP is that these can be placed anywhere - and I mean anywhere - in your code in any number of ways. Later we'll see some interesting uses for these, but for now let's keep it simple. If you wish, you can also configure PHP to use short tags, <?, and ?>, but these are not XML compliant, so be careful. If you're making the switch from ASP, you can even configure PHP to use the `<%` and `%>` delimiters.

Another thing you'll notice is the semicolon on the end of each line. These are known as separators and serve to distinguish one set of instructions from another. It is feasible to write an entire PHP script on one line and separate the portions with semicolons. But that would be a mess, so we'll add a new line after each semicolon. Just remember that each line must end in a semicolon.

Finally, you see that the word `myvar` begins with a dollar sign. This symbol tells PHP that this is a variable. We assigned the words "Hello World" to the variable $myvar. A variable can also contain numbers or an array. Either way, all variables start with the dollar sign symbol.

The real power of PHP comes from its functions. These are basically processing instructions. If you add up all of the optional add-ins to PHP, there are more than 700 functions available. So there's quite a bit you can do.

Now let's add some MySQL to the picture.

Load Up a Database

So now we're ready to plug in MySQL. One handy way of knowing what options are available in PHP and what's going on in your server is to use the `phpinfo()` function. Create a script with the following:

```
<html>

<body>


<?php

phpinfo();

?>


</body>

</html>
```

Save and view this script through your Web server. You'll see a page filled with useful and interesting information like this. This info tells all about your server, internal Web server environment variables, the options that are compiled, and on and on. In the first section, Extensions, look for a line beginning with MySQL. If this is missing, then for some reason MySQL hasn't made it into PHP. Go back and review the installation steps and check the PHP documentation to see if you missed anything.

If MySQL is there, then you're ready to roll.

Before we can get data out of MySQL, we have to put data in it. There's really no easy way to do it at this stage. Most PHP scripts come with what's known as a dump file that contains all the data required to create and populate a MySQL database. The ins and outs of this process are really outside the scope of this tutorial, so I'll just do it for you.

MySQL uses its own user permissions table. At setup, a default user (root) is automatically created with no password. It's up to the database administrator to add other users with various permissions, but I could write a whole other article on that, so we'll stick with using the root user. If you set up your own server and database, it's vital that you assign a password to the root user.

Anyway, let's get on with the database. For Win32 users, I'm sorry, but this requires some DOS work. You'll have to use a DOS window or type everything in the Run window. Don't forget to type in the path to the location of the MySQL/bin directory with your commands. Unix users can work from the MySQL bin directory, but you may have to start each command with ./ so the programs run.

The first thing we need to do is create the actual database. From the command line, type:

```
mysqladmin -u root create mydb
```

That creates a database called "mydb." The flag tells MySQL that we're doing this as the root user.

Next we'll add some data using everyone's favorite example, the employees database. We're going to need that dump file I mentioned earlier. If you're interested in how it goes together, review the manual that comes with MySQL or check out http://www.turbolift.com/mysql/.

Copy and paste the following text to a file and save it in MySQL's bin directory. (I'll call the file mydb.dump.)

```
CREATE TABLE employees (  id tinyint(4) DEFAULT '0' NOT NULL AUTO_INCREMENT,  first varchar(20),  las

INSERT INTO employees VALUES (2,'John','Roberts','45 There St , Townville','Telephonist');

INSERT INTO employees VALUES (3,'Brad','Johnson','1/34 Nowhere Blvd, Snowston','Doorman');
```

If the lines wrap, make sure that each insert statement is on a new line. Now we'll insert it into the mydb database. From the command line, type:

```
mysql -u root mydb < mydb.dump
```

You shouldn't get any errors doing this. If you do, check for incorrect line wrapping.

Pull It Back Out

OK, now we've got our data in the database. Let's do something with it. Copy and paste the following into a text file and save it in your Web server document tree with a .php3 extension.

```
<html>

<body>

<?php

$db = mysql_connect("localhost", "root");

mysql_select_db("mydb",$db);

$result = mysql_query("SELECT * FROM employees",$db);
```

```
printf("First Name: %s<br>\n", mysql_result($result,0,"first"));

printf("Last Name: %s<br>\n", mysql_result($result,0,"last"));

printf("Address: %s<br>\n", mysql_result($result,0,"address"));

printf("Position: %s<br>\n", mysql_result($result,0,"position"));

?>

</body>

</html>
```

Let's explain what happens here. The `mysql_connect()` function opens a link to a MySQL server on the specified host (in this case it's localhost) along with a username (root). If you needed to specify a password, you'd add it here as well. The result of the connection is stored in the variable $db.

`mysql_select_db()` then tells PHP that any queries we make are against the mydb database. We could create multiple connections to databases on different servers. But for now, let's leave it to this.

Next, `mysql_query()` does all the hard work. Using the database connection identifier, it sends a line of SQL to the MySQL server to be processed. The results that are returned are stored in the variable `$result`.

Finally, `mysql_result()` is used to display the values of fields from our query. Using $result, we go to the first row, which is numbered 0, and display the value of the specified fields.

The syntax of the `printf` function may seem a little odd if you haven't used Perl or C before. In each of the lines above, `%s` indicates that the variable in the second half of the expression (e.g., `mysql_result($result,0,"position")`) should be treated as a string and printed. For more on `printf`, see the PHP documentation.

So there we have it. We successfully complied, installed, and configured MySQL and PHP, and we've executed a simple script to retrieve some information. In Lesson 2, we'll do some clever stuff to display multiple records and even send data to and from the database.

Come on back, now.

Graeme Merrall works in New Zealand for KC Multimedia, an Internet/intranet design company. While sometimes being forced to use ASP, he enjoys spending his time coding in PHP, keeping the grass trimmed, and scaring his work mates with song lyrics repeated out of context.