Web Database

Step-by-Step Guide

April 22, 1999

By Ying Zhang
yzhang@sfu.ca

Contents

Revisions

April 13, 1999

- updated the entire how-to
- covers Apache 1.3.6, PHP 3.0.7 and MySQL 3.22.21
- created PDF and Postscript versions
- HTML version works with babelfish (but the examples will get garbled up)

January 7, 1999

- initial version of the document

Credits

Thanks to all the people who emailed me with their feedback, suggestions, and questions! I'm really glad my howtos have been able to help people out, please keep on visiting my site!

Thanks to these fine folks for their wonderful products:

- The Apache Group (http://www.apache.org/contributors)
- The PHP Development team (http://www.php.net/credits.php3)
- T.c.X DataKonsult AB (http://www.mysql.com/info.html)

These are all excellent products and they work amazingly well with each other!

Introduction

This guide walks you through installing a web server, an SQL database server, and a server-side scripting tool that ties everything together. Some of the more popular tools for doing this are Apache, MySQL, and PHP3.

This is what you will have accomplished after successfully completing this guide:

- setup the MySQL database server
- setup the Apache web server
- setup the PHP 3.0 Hypertext Preprocessor for server-side-scripting
- create a simple web enabled database

This guide is meant as an introductory guide to get you started in the world of server-side-scripting and web databases. It will help you get up and running with the aforementioned products, and hopefully give you a better understanding of how this stuff all works.

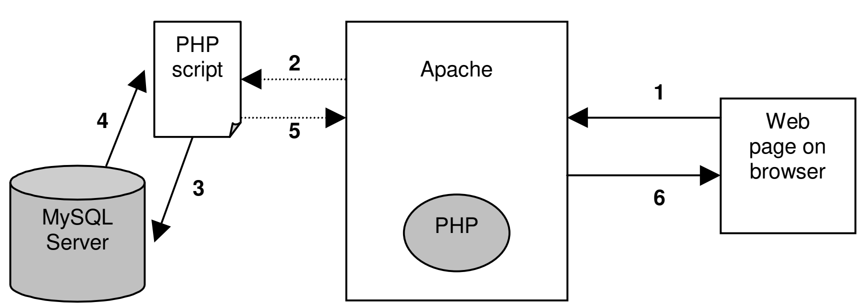
Disclaimer

I cannot guarantee the accuracy or correctness of the information I present in this howto. I am in no way associated with the various products mentioned in this guide, what I describe here is basically a log of what I had to learn. Do not blame me if something goes wrong, there are no warranties so use this information at your own risk!

On a brighter note, if you have any questions, comments, or suggestions, please feel free to send me email at yzhang@sfu.ca. I will try my best to answer your question or point you to better resources, all I ask is that you follow this guide as closely as possible before sending off an email.

How It Works

It is helpful to have a feeling for what goes on behind the scenes, so here is an over simplification of how things would work,. This diagram isn't really correct but it should be enough for now:



So let's set the scenario. We have a web page that pulls some data out of a database. John Doe requests this page from his browser, this request is sent to the web server which in turn calls a PHP script. The PHP script is executed by the PHP preprocessor, it pulls data from the database and the results are massaged by the rest of the PHP script. The output of all this is HTML that gets sent back to the user's browser.

Got that? Let's examine this step by step:

1. John Doe clicks on a link to from his web browser, his web browser sends a request for **http://www.foo.com/foofoo.php3**.

2. Apache gets the request for **foofoo.php3**. It knows that **.php3** files are handled by the PHP preprocessor, so it tells PHP to deal with it.

3. **foofoo.php3** is a PHP script that contains commands. One of these commands is to open a connection to a database and grab some data. PHP knows how to talk to the database, so it does it's thing.

4. The data comes back from the database, and foofoo.php3 does something to format the data. Typically this would be to make it look pretty before formatting it into HTML.

5. The HTML goes back to Apache.

6. Apache sends this back to John Doe's browser, as the response to his request. John Doe now sees a pretty web page containing some information from a database.

Again, that's not 100% correct but it's enough to understand what goes on :) Now that we have a basic understanding of what we are trying to accomplish, let's get on to installing the software.

Requirements

In writing this document, I will assume that you already have:

- a basic understanding of Unix commands, HTML, and SQL
- a working TCP/IP network
- a working Linux box on which you will install the software
- the necessary packages to compile programs in Linux
- you don't already have MySQL, Apache, or PHP installed

This guide was written with RedHat Linux in mind (that's what I use) so if that doesn't apply to you, please make changes as necessary for your setup. I will try to point out differences as I go along.

Getting MySQL

First let's grab MySQL, you will find it at http://www.mysql.com. The latest version as of this writing is 3.22.21. From their download page, get the precompiled binaries:

- Binary RPM containing the server for i386 machines
  MySQL-3.22.21-1.i386.rpm

- Binary RPM containing client programs for i386 machines
  MySQL-client-3.22.21-1.i386.rpm

- Binary RPM containing include files and libraries for development for i386 machines
  MySQL-devel-3.22.21-1.i386.rpm

If you're not using RedHat, download the precompiled binaries (mysql-3.22.21-pc-linux-gnu-i686.tar.gz). More on this as it becomes important.

Getting Apache

Next thing to get is the Apache source distribution. You will find it at their homepage http://www.apache.org, the latest version as of this writing is 1.3.6.

I chose not to use the Apache RPM package that comes with RedHat because it's an older version (1.3.3) and a bunch of things have changed since then. If you want to use the Apache RPM, you will have to make appropriate changes to my examples to suite your configuration.

Getting PHP

Finally, get the PHP source distribution from their homepage http://www.php.net. The latest version as of this writing is 3.0.7.

There is an RPM package for PHP from their web site, but it is for version 3.0.5. Also, MySQL support isn't compiled in, so it's easier for you to use the source and compile it yourself.

Getting GD (optional)

You can also get the GD library if you want to be able to create and manipulate GIF files on the fly. If you have the GD library, PHP can be compiled to take advantage of it.

Their homepage is at http://www.boutell.com/gd, download it from there or grab the RPM if you're using RedHat. You'll need gd-1.3-3.i386.rpm and gd-devel-1.3-3.i386.rpm. These are on your RedHat 5.2 CD. You can also find it on http://www.rpmfind.net/RPM.

Other Packages

There are many other things you can compile into Apache and PHP. Doing this is beyond the scope of this document, you can always add support for these things later. We will keep things simple to get through this guide.

Installing MySQL

Okay, we actually get to do something now! Assuming you've downloaded everything into /tmp, do this to install MySQL. If you downloaded the RPM's, the installation is really easy. If you downloaded the binary tarball (because you don't have RedHat), it's a wee bit more involved.

RPM Installation

You must be root to install an RPM:

```
$ cd /tmp
$ su
# rpm -Uvh MySQL*
```

That should have installed the three MySQL RPM's that you downloaded. The RPM installation is the recommended way to go if you are running RedHat, since everything is done for you.

Binary Installation

Installing the binary distribution is a little bit more involved that installing the RPM. We have to manually do some things that the install scripts in the RPM would have done for us.

Installing the Binary Distribution

If you downloaded the binary distribution, it has a name like **mysql-3.22.21-pc-linux-gnu-i686.tar.gz**. Extract this into /usr/local, you must do this as root:

```
$ cd /usr/local
$ su
# tar -zxvf /tmp/mysql-3.22.21-pc-linux-gnu-i686.tar.gz
```

After it extracts everything, a directory called mysql-3.22.21-pc-linux-gnu-i686 will be created. We make a symlink to this directory and give it a friendlier name:

```
# ln -s mysql-3.22.21-pc-linux-gnu-i686 mysql
```

Next time there is a new version of MySQL, you can just extract the source distribution to a new directory and change the symlink.

Creating a MySQL User

Now we will create a user account to run the MySQL server daemons and to own all the MySQL files. Use Linuxconf or useradd to add a user called **mysql**. No one should be logging into this account, so disable logins.

Preparing MySQL

First let's change the ownership of MySQL directories and files to be owned by the mysql user and the root group:

```
# cd /usr/local
# chown -R mysql:root mysql-3.22.21-pc-linux-gnu-i686 mysql
```

Now we have to run a little script that creates the initial MySQL database, do this a the **mysql** user. This is the only time we use this account directly:

```
# su mysql
$ cd mysql
$ scripts/mysql_install_db
$ exit
```

If that didn't give you any error messages, you're well on your way.

Make MySQL Start Automatically

There is a startup script in the binary distribution of MySQL called mysql.server. Copy this to /etc/rc.d/init.d:

```
# cd /etc/rc.d/init.d
# cp /usr/local/mysql/support-files/mysql.server mysql
```

Next, make it executable:

```
# chmod +x mysql
```

Finally, run chkconfig to add MySQL to your system startup services:

```
# /sbin/chkconfig --del mysql
# /sbin/chkconfig --add mysql
```

Testing MySQL

MySQL comes with a sample database called test and its internal database that keeps track of permissions and users, so let's fire it up and see if everything is working so far. First start up MySQL:

```
# /etc/rc.d/init.d/mysql start
```

If that worked, you should be able to something like:

```
Starting mysqld daemon with databases from /var/lib/mysql
```

If you installed the binary distribution, the programs are located in **/usr/local/mysql/bin**, you may want to this to your path. Start the client by running:

```
# mysql
```

You should see the following:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.22.21

Type 'help' for help.

mysql>
```

Next, list the installed databases by typing **show databases**:

```
mysql> show databases;
```

You should see:

```
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)
```

If you did, then it's working!! Exit MySQL by typing **exit**:

```
mysql> exit;
Bye
```

Changing the Admin Password

The first thing to do after everything works is to change the administrator password. Do this by running **mysqladmin** (remember that it may not be in your path):

```
# mysqladmin –u root password newpassword
```

This sets the password for the user **root** to **newpassword**. You probably don't want to use that, so substitute it with something clever.

Installing Apache

Now we will install Apache. Assuming you downloaded it into /tmp, do the following (note you probably shouldn't be root while doing this):

```
$ cd /tmp
$ tar -zvxf apache_1.3.6.tar.gz
$ cd apache_1.3.6
$ ./configure --sysconfdir=/etc/httpd \
             --datadir=/home/httpd \
             --logfiledir=/var/log/httpd \
             --enable-module=most \
             --enable-shared=max \
             --disable-rule=WANTHSREGEX
```

This tells Apache to store it's configuration files in /etc/httpd. The data files directories (HTML documents, CGI-BIN directory) will be rooted in /home/httpd. Next, we tell Apache we want most of the modules enabled, and that they should be built as DSO modules.

DSO stands for Dynamic Shared Object. By compiling Apache with DSO support, you are able to add and remove modules from Apache without having to recompile it. This is really handy, read all about it at the Apache DSO page (http://www.apache.org/docs/dso.html), if you're interested.

Next thing to do is to compile this thing. Do this by running make:

```
$ make
```

That should take a little while. When it finishes, you'll have to su to root to copy the files to their final destinations:

```
$ su
# make install
```

If all goes well, you should see a bunch of messages tell you some information about Apache and how it was installed on your computer. If you're using RedHat, make a symlink from **/usr/local/apache/bin/apachectl** to **/etc/rc.d/init.d/httpd**:

```
# ln -s /usr/local/apache/bin/apachectl /etc/rc.d/init.d/httpd
```

Next, add these lines to that file (some where around the top, maybe after the 2rd line):

```
# chkconfig: 2345 10 90
# description: Activates/Deactivates Apache Web Server
```

Finally, run chkconfig to add Apache to your system startup services:

```
# /sbin/chkconfig --del httpd
# /sbin/chkconfig --add httpd
```

Configuring Apache

Starting from Apache 1.3.4, the three configuration files: srm.conf, httpd.conf, and access.conf have been consolidated into httpd.conf. Naturally, this carries on to 1.3.6, so load it into your favorite editor and change these directives:

- Port 80
- DirectoryIndex index.html index.shtml index.cgi index.phtml index.php3
- AddType application/x-httpd-php3 .php3 .phtml
- AddType application/x-httpd-php3-source .phps

Scan through the files and change any other directives that you feel should be changed. For detailed information about the directives, visit the Apache Website and look at the server documentation.

Testing Apache

If you're using RedHat, run this:

```
# /etc/rc.d/init.d/httpd start
```

Otherwise run this:

```
# /usr/local/bin/apachectl start
```

If it said that Apache was started, you should be able to use any web browser to connect to your box. If all goes well, you will see a Welcome to Apache page. Also, check that the httpd processes are running:

```
# ps ax | grep httpd
```

You should see a bunch of processes like:

```
14362  ?  S    0:00 /usr/local/apache/bin/httpd
14364  ?  S    0:00 /usr/local/apache/bin/httpd
14365  ?  S    0:00 /usr/local/apache/bin/httpd
14366  ?  S    0:00 /usr/local/apache/bin/httpd
14367  ?  S    0:00 /usr/local/apache/bin/httpd
14368  ?  S    0:00 /usr/local/apache/bin/httpd
```

Now let's turn off Apache and install PHP. If you're using RedHat, run this:

```
# /etc/rc.d/init.d/httpd stop
```

Otherwise run this:

```
# /usr/local/bin/apachectl stop
```

Installing PHP

With Apache installed and working, let's now turn our attention to PHP. Assuming you downloaded it to /tmp, let's get to work on it (you don't need to be root yet):

```
$ cd /tmp
$ tar -zxvf php-3.0.7.tar.gz
$ cd php-3.0.7
$ ./configure --with-apxs=/usr/local/apache/bin/apxs \
              --with-config-file-path=/etc/httpd \
              --with-mysql \
              --with-gd \
              --with-zlib \
              --with-system-regex
```

This configures PHP to compile itself as an Apache DSO module. Configuration files are to be stored in /etc/httpd (along with your Apache configuration files).

The above configuration command should work fine if you've been following this howto exactly. If you have things installed in different locations:

- Change **--with-apxs=/usr/local/apache/bin/apxs** to point to whatever path the apxs script is in. If you installed Apache from RPM, it will probably be in **/usr/sbin/apxs**

- If you installed MySQL from the binary distribution, you should change **--with-mysql** to **--with-mysql=/usr/local/mysql**

- If you don't have the GD library, take out the line that says **--with-gd**

After running that configure script, you can build PHP by running make:

```
$ make
```

If that completed without errors, you will have to su to root and install PHP:

```
$ su
# make install
```

Voila, PHP is magically installed into Apache, it will get enabled next time Apache is started.

Configuring PHP

Let's configure PHP, start by copying the PHP configuration file into /etc/httpd.

```
# cp php3.ini-dist /etc/httpd/php3.ini
```

The defaults in this file are good enough, but I like the debug messages to be more verbose. Change the error_reporting directive from the default value of 7 to 15:

```
error_reporting = 15
```

You can now start Apache again by running one of:

```
# /etc/rc.d/init.d/httpd start
# /usr/local/apache/bin/apachectl start
```

No errors means it started up with PHP enabled. Everything working so far? Excellent! Let's do a quick little test page with PHP to make sure it is indeed working.

Testing PHP

Create a file in /home/httpd/htdocs (or /home/httpd/html if you installed Apache from the RedHat RPM) called test.php3:

```
<html>
<head>
<title>My First PHP Page</title>
</head>
<body bgcolor=#ffffff>
<? echo "Hello World"; ?>
</body>
</html>
```

Save this file, and try to load it up from your web browser. For example:

```
# lynx http://localhost/test.php3
```

You should see a page come up with the words Hello World. If so then that's you're laughing! Now let's get started with a simple web database sample.

Web DB Example Part 1

In the next sections, we will go through the steps involved in creating a web enabled database. First we create the database, and populate it with some data. After that is done, we create some PHP scripts that talk to the database and see it all work.

Building the Database

Make sure the MySQL server processes are running. If they aren't start them manually by running:

```
# /etc/rc.d/init.d/mysql start
```

Now we start the MySQL client as the administrator, this time we will see a password prompt:

```
# mysql -u root -p
Enter password: newpassword
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13 to server version: 3.22.21

Type 'help' for help.

mysql>
```

Creating the database

Create a database called example with the command **create database example;**. When using the MySQL client, remember to put a semicolon (;) after each command:

```
mysql> create database example;
Query OK, 1 row affected (0.03 sec)

mysql> use example;
Database changed
```

Create a new table

Create a table in the example database called mytable:

```
mysql> CREATE TABLE mytable (
    -> name CHAR(30),
    -> phone CHAR(10)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

Here's a handy tip, that will save you some typing. Use the up arrow and down arrow keys to recall back the previous/next lines you typed in.

Adding some data

Now let's populate this with some data, we will insert a few entries into the table. Enter in the following INSERT statments:

```
mysql> INSERT INTO mytable VALUES ("Homer Simpson", "555-1234");
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO mytable VALUES ("Bart Simpson", "555-4321");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO mytable VALUES ("Lisa Simpson", "555-3214");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO mytable VALUES ("Marge Simpson", "555-2314");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO mytable VALUES ("Maggie Simpson", "555-3142");
Query OK, 1 row affected (0.00 sec)
```

Make sure everything is there by issuing with a SELECT statement:

```
mysql> SELECT * FROM mytable;
+----------------+----------+
| name           | phone    |
+----------------+----------+
| Homer Simpson  | 555-1234 |
| Bart Simpson   | 555-4321 |
| Lisa Simpson   | 555-3214 |
| Marge Simpson  | 555-2314 |
| Maggie Simpson | 555-3142 |
+----------------+----------+
5 rows in set (0.00 sec)
```

Looking good so far? Excellent.

Creating a Database User

We've created the database and put some data in there, now we must create a user account. This user will have access to this database. We create the user and grant privileges with the GRANT commant:

```
mysql> GRANT usage
    -> ON example.*
    -> TO webuser@localhost;
Query OK, 0 rows affected (0.15 sec)
```

This creates a new user called webuser. This user can connect only from localhost, and he has the ability to connect to the example database. Next we have to specify what operations webuser can perform:

```
mysql> GRANT select, insert, delete
    -> ON example.*
    -> TO webuser@localhost;
Query OK, 0 rows affected (0.00 sec)
```

This gives webuser the ability to execute SELECT, INSERT, and DELETE queries on every table of the example database. Our work is done here, exit the MySQL client:

```
mysql> exit
Bye
```

Web DB Example Part 2

We've created the database, now let's make the PHP scripts that form the guts of our web database example.

Creating the PHP Scripts

To keep things really simple, we will just create two scripts: one that lists all the entries in the database, and one that allows us to add new entries.

index.php3

Create a new directory called example in your web directory. Note, if you installed the Apache 1.3.3 RPM, your home pages are stored in /home/httpd/html, not htdocs.

```
# cd /home/httpd/htdocs
# mkdir example
```

Next, create a file called index.php3 in this directory. It should contain:

```
<html>
<head><title>Web Database Sample Index</title>
</head>

<body bgcolor=#ffffff>
<h1>Data from mytable</h1>
<?
mysql_connect("localhost", "webuser", "");
$query = "SELECT name, phone FROM mytable";
$result = mysql_db_query("example", $query);

if ($result) {
    echo "Found these entries in the database:<ul>";
    while ($r = mysql_fetch_array($result)) {
          $name = $r["name"];
          $phone = $r["phone"];
          echo "<li>$name, $phone";
    }
    echo "</ul>";
} else {
    echo "No data.";
}
mysql_free_result($result);
?>

<p><a href="add.php3">Add new entry</a>
</body>
</html>
```

add.php3

Next, we create add.php3 in the same directory. This script does two things, first it will prompt the user for information to add to the database. Second, it will add this information to the database. This second function is normally put in a separate file, but it is so easy to do that we cram them both into one PHP script:

```
<html>
<head><title>Web Database Sample Inserting</title>
</head>

<body bgcolor=#ffffff>
<?
mysql_connect("localhost", "webuser", "");

if (isset($name) && isset($phone)) {
    $query = "INSERT INTO mytable VALUES ('$name', '$phone')";
    $result = mysql_db_query("example", $query);

    if ($result) {
         echo "<p>$name was added to the database</p>";
    }
}
?>

<h1>Add an entry</h1>
<form>
Name: <input type=text name='name'><br>
Phone: <input type=text name='phone'><br>
<input type=submit>
</form>

<p><a href="index.php3">Back to index</a>
</body>
</html>
```

That's it, nice and simple. Now let's test it.

Web DB Example Part 3

All the work is done, let's see if it works! Make sure everyone has read permissions (chmod ugo+r) for these files, otherwise you'll get an Access Denied error message from Apache. You should now be able to use your web browser to access what you've just created.

Assuming you're using lynx:

```
$ lynx http://localhost/example
```

This should bring up a page that lists all the entries you added to the database earlier. For example:

```
Found these entries in the database:
     * Homer Simpson, 555-1234
     * Bart Simpson, 555-4321
     * Lisa Simpson, 555-3214
     * Marge Simpson, 555-2314
     * Maggie Simpson, 555-3142

   Add new entry
```

Select the Add new entry link, and you should go to a page that allows you to enter data:

```
Name: ____________________
Phone: ____________________
Submit

Back to index
```

Enter something and select the Submit link. This submits the filled-out form back to the same PHP script. This time, there are values in the two fields name and phone, so these get added to the database.

You should see a message telling you something was added to the database. Select the Back to index link to confirm that your new entry is infact there.

If everything worked, then congradulations! You've completed this howto and built your first web database application! It was pretty easy wasn't it?

Conclusion

This concludes this step-by-step howto. You've now had a peek at the power and relative ease of building a web database from scratch. We've only scratched the surface, so at this point you should go on to the Apache, PHP, and MySQL homepages to read up on their documentation.

Of particular interest is the MySQL documentation. You should find out and understand the limitations and compatibility issues of using this product. While it has excellent performance and really easy to use, it has it's limitations. For example, as of this writing there is no support for nested selects.

You will also find a lot of tutorials by searching on the Internet. Some good places to start are:

- Developer's Shed (http://www.devshed.com)
- PHP Homepage (http://www.php.net)

By now, you should have a pretty good understanding of how everything fits together. Now you should be able to get a good feel for any other tool you decide to use, because the underlying concepts are pretty much the same.

I hope you've found this document useful!