Doing More With phpMyAdmin (Part 1)

By Harish Kamath

2003−10−27

Printed from DevShed.com

URL: http://www.devshed.com/Server_Side/PHP/Doing_More_phpMyAdmin/

The Silver Bullet

Every RDBMS worth its salt comes with one or more database administration tools that SQL developers can use to perform common tasks like executing queries, setting user privileges and viewing server status. For example, Oracle developer tend to swear by Quest Software's TOAD, while Microsoft SQL Server devotees use the ubiquitous Enterprise Manager and SQL Query Analyzer.

But what about MySQL? Does MySQL come with any GUI−based tools to speed up development? Is there a way for SQL newbies to work with a MySQL database without getting into the nitty−gritty of SQL syntax? What about for developers to easily interact with and manipulate a MySQL database or table, obtain E−R diagrams of table relationships and page through the contents of a table without having to resort to the command prompt? Or for administrators to quickly and easily assign user privileges and track down and kill rogue processes?

Yes, yes, yes and yes − phpMyAdmin is the answer to all the questions above.

What started as a simple project to manage a MySQL database via a Web interface is today a comprehensive one−stop shop for managing and maintaining multiple MySQL databases on a single server (or across multiple servers, if needed) without much fuss. And with thousands of enthusiastic developers downloading it every day, it is no surprise that it consistently ranks as one of the most popular open−source projects on SourceForge.

If you're an SQL purist, you're probably scoffing right now − like most purists, you believe in the power of the SQL command line for your daily tasks. But if you're new to SQL in general and MySQL in particular, or if you're a DBA who needs to get a particular task done quickly and don't have the time to learn the specifics of MySQL's command syntax, you'd probably be quite interested to know that phpMyAdmin can turn a lot of otherwise−complex tasks into simple point−and−click operations: creating databases and tables, inserting table data, running SQL queries, setting user privileges, viewing reports of server usage and importing/exporting tables from one server to another.

Over the course of this two−part article, I'm going to take you on a tour of one of the most popular Web−based tools for managing a MySQL database server. The first part of this article lays the foundation, explaining how to obtain the software, install and configure it for secure access, and use it for tasks such as managing multiple servers, manipulating user privileges, viewing reports on server activity, and exporting MySQL records into different formats. The second part explains the more advanced aspects of the application, including using it for transformations, maintaining a history of all the SQL queries executed in the phpMyAdmin session, defining relations between tables to create JOINs automatically, creating E−R diagrams in PDF format, and bookmarking important queries for future reference.

I will assume in this two−part series that you're familiar with phpMyAdmin, and have used it in the past for routine tasks like table creation and data manipulation. In case you haven't, read the first few pages of this tutorial, which teach you how to install and configure it, take a short break to play around with it and get familiar with common tasks (it's really, really simple to use), and then come back to find out more about what you can do with it.

Let's get going!

Start Me Up

In order to get started with phpMyAdmin, you'll need a working PHP installation (I'm currently running PHP 4.2.3), a MySQL database server (I'm running MySQL 4.0.14) and a copy of the latest release (currently version 2.5.3) of the package; you can download a copy from http://www.phpmyadmin.net/

The developers of phpMyAdmin have been considerate enough to maintain two versions of this popular software, one for servers that have been configured to parse PHP files with the ".php3" extension (a widely observed legacy) and another for those using the newer ".php" extension. You should select the version that is compliant with your Web server's configuration.

Setting up the package is pretty simple. The first step is to uncompress the source archive into a directory under your

Web server (mine is called "medusa" here) root.

```
$ cd /usr/local/apache/htdocs/

$ tar −xzvf /tmp/phpMyAdmin−2.5.3.tar.gz
```

This should create a directory named "phpMyAdmin/".

After checking to make sure that your Web and MySQL servers are up and running (you'd be surprised how often even experienced developers forget to do this at times), you can try accessing the application, by popping open your Web browser and navigating to the URL http://localhost/phpMyAdmin/. Here's what you might see.

"So, what's that error all about," you ask?

Simple − the software isn't configured yet. Let's do that next.

phpMyAdmin is configured via a single configuration file, "config.inc.php" (or "config.inc.php3"), located in the application folder. Open it in your favourite text editor and take a quick look at it. If you read the comments, you'll see that it allows you to configure almost every aspect of the application.

The first thing you usually need to configure is the $cfg['PmaAbsoluteUri'] variable − this stores the Web server URL to the phpMyAdmin installation directory. Assuming an installation directly under the Web server root, you can set this as follows:

```
$cfg['PmaAbsoluteUri'] = 'http://localhost/phpMyAdmin/';
```

While this will get rid of the error message displayed on the application's opening page, you will still not be able to view databases or tables on the MySQL server (go on, try it and see for yourself). This is because the application does not yet know where to look for the MySQL server, or have access privileges for the same. Move on to the next section of the configuration file, which allows you to define the server configuration variable $cfg['Servers'].

I'll begin with showing you how to configure phpMyAdmin for a single server. In order to do this, you will need three values: the host name or IP address of the machine hosting the MySQL server, the user name to access the server (note that this is *not* the same as the UNIX user name) and the corresponding password. If the server is running on a non−default port, you'll need that information too.

If you're running phpMyAdmin as an administrator, you can set the user name to "root", or an equivalent privileged user in MySQL. If, on the other hand, you're configuring a "limited−view" phpMyAdmin, which is to be used only by a specific user, you can use that user's user name and password instead.

Once you have all this information, make sure that your configuration file contains the following (I'll assume here that the server is "localhost", the user is "root", and the root password is "secret", with the server running on its standard port):

```
$cfg['Servers'][$i]['host'] = 'localhost'; // MySQL hostname
$cfg['Servers'][$i]['port'] = ''; // MySQL port − leave blank for default port
$cfg['Servers'][$i]['socket'] = ''; // Path to the socket − leave blank for default
socket
$cfg['Servers'][$i]['auth_type'] = 'config'; // Authentication method
$cfg['Servers'][$i]['user'] = 'root'; // MySQL user
$cfg['Servers'][$i]['password'] = 'secret'; // MySQL password
```

A small note here on the "auth_type" parameter above. I have set this value to "config", which indicates that phpMyAdmin should use the credentials provided in this configuration file to access the database server. Note that this is a potential security hole, as the database server becomes vulnerable if the configuration file falls into the wrong hands. I'll be explaining how to solve this problem shortly.

Once you've made your changes to the file, save it and try navigating back to http://localhost/phpMyAdmin/ in your browser. This time, if you've configured it correctly, you should be able to see a list of databases on the MySQL server in the drop−down box at the top left corner. Select a database from the list and you'll see a list of tables present in it.

Cool, huh? This is just the beginning.

Locking the Doors

I've already written about the simplest authentication mechanism in phpMyAdmin − the "config" option, wherein the user name and password to access the MySQL server are stored in the phpMyAdmin configuration file. At noted previously, this method is not very secure, due to the risk of an unauthorized user gaining access to the configuration file.

In order to control access to the application and thereby the RDBMS, phpMyAdmin supports two alternative mechanisms: cookie−based authentication and HTTP−based authentication. Both are pretty straightforward, and simple to set up.

In order to use either of these authentication mechanisms, you need to first create a special MySQL user (I'll call this user "admin" in the examples that follow) with restricted permissions on the MySQL grant tables. phpMyAdmin's documentation assists in this process by listing the queries needed:

```
GRANT USAGE ON mysql.* TO admin@localhost IDENTIFIED BY "j823kfg2ld";

GRANT SELECT (
    Host, User, Select_priv, Insert_priv, Update_priv, Delete_priv,
    Create_priv, Drop_priv, Reload_priv, Shutdown_priv, Process_priv,
    File_priv, Grant_priv, References_priv, Index_priv, Alter_priv,
    Show_db_priv, Super_priv, Create_tmp_table_priv, Lock_tables_priv,
    Execute_priv, Repl_slave_priv, Repl_client_priv
    ) ON mysql.user TO admin@localhost;

GRANT SELECT ON mysql.db TO admin@localhost;

GRANT SELECT ON mysql.host TO admin@localhost;

GRANT SELECT (Host, Db, User, Table_name, Table_priv, Column_priv)

        ON mysql.tables_priv TO admin@localhost;
```

This set of queries creates a MySQL "admin" user and gives that user the ability to use and perform SELECT queries on all tables in the "mysql" database.

Next, you need to tell phpMyAdmin about the new user. Open the configuration file, locate the "controluser" and "controlpass" variables for your server, and update the server configuration block so it looks like this:

```
$cfg['Servers'][$i]['host'] = 'localhost'; // MySQL hostname
$cfg['Servers'][$i]['port'] = ''; // MySQL port − leave blank for default port
$cfg['Servers'][$i]['socket'] = ''; // Path to the socket − leave blank for default
socket
$cfg['Servers'][$i]['auth_type'] = 'config'; // Authentication method
$cfg['Servers'][$i]['user'] = ''; // MySQL user
$cfg['Servers'][$i]['password'] = ''; // MySQL password
$cfg['Servers'][$i]['controluser'] = 'admin'; // MySQL control user settings
$cfg['Servers'][$i]['controlpass'] = 'j823kfg2ld'; // access to the grant tables
```

Notice that I have removed the user name and password previously set for the "user" and "password" parameters.

Now that the special control user has been created, you can choose to use either cookie− or HTTP−based authentication.

1. To use cookie−based authentication, change the value of the "auth_type" parameter in $cfg['Servers'] from "config" to "cookie", and also set a value for the special $cfg['blowfish_secret'] variable. Updated, your configuration should now look like this:

   ```
   $cfg['blowfish_secret'] = 'abracadabra';
   ```

```
$cfg['Servers'][$i]['host'] = 'localhost'; // MySQL hostname
$cfg['Servers'][$i]['port'] = ''; // MySQL port − leave blank for default port
$cfg['Servers'][$i]['socket'] = ''; // Path to the socket − leave blank for
default socket
$cfg['Servers'][$i]['auth_type'] = 'cookie'; // Authentication method
$cfg['Servers'][$i]['user'] = ''; // MySQL user
$cfg['Servers'][$i]['password'] = ''; // MySQL password
$cfg['Servers'][$i]['controluser'] = 'admin'; // MySQL control user settings
$cfg['Servers'][$i]['controlpass'] = 'j823kfg2ld'; // access to the grant tables
```

Now, open phpMyAdmin in a new window and try accessing the server − you should see a neat little username and password prompt.

Enter your MySQL user name and password − not the "admin" user created above, but the one you normally use for development. You should now be able to access all the databases that you have privileges for.

2. To use HTTP−based authentication, change the value of the "auth_type" parameter in $cfg['Servers'] from "config" to "http"

```
$cfg['Servers'][$i]['host'] = 'localhost'; // MySQL hostname
$cfg['Servers'][$i]['port'] = ''; // MySQL port − leave blank for default port
$cfg['Servers'][$i]['socket'] = ''; // Path to the socket − leave blank for
default socket
$cfg['Servers'][$i]['auth_type'] = 'cookie'; // Authentication method
$cfg['Servers'][$i]['user'] = ''; // MySQL user
$cfg['Servers'][$i]['password'] = ''; // MySQL password
$cfg['Servers'][$i]['controluser'] = 'admin'; // MySQL control user settings
$cfg['Servers'][$i]['controlpass'] = 'j823kfg2ld'; // access to the grant tables
```

and try to access the phpMyAdmin application. This time around, you should see the browser−generated password prompt typical of HTTP based authentication, as seen here.

Enter your MySQL user name and password − not the "admin" user created above, but the one you normally use for development. You should now be able to access all the databases that you have privileges for.

Now, I'm sure you're wondering − why did you even bother creating that "admin" user if you never use its credentials anywhere? The answer is both simple and elegant − phpMyAdmin uses that "admin" account to check whether the user/password credentials you provide to the cookie− or HTTP−based authentication system are valid. Since the application only requires read access to the MySQL grant tables to validate this information, only restricted privileges are assigned to the "admin" user.

Using this type of two−tiered authentication mechanism not only leaves the task of managing and authenticating MySQL users where it should be − with MySQL − but it also reduces the risk of unauthorized database manipulation by users who get their hands on the phpMyAdmin configuration file. Since the file now only contains credentials for the limited−access "admin" user, a hacker won't be able to do much with it (other than read the MySQL grant tables, itself not such a great security risk since MySQL automatically encrypts grant table passwords).

The More the Merrier

You can also use phpMyAdmin to manage multiple database servers using the same instance of phpMyAdmin. Go back to the phpMyAdmin configuration file, and take a closer look at the server configuration block you manipulated earlier.

If you keep reading, you'll notice that the first configuration block is followed by many similar blocks, each one configured by default with empty values.

```
$i++;
$cfg['Servers'][$i]['host'] = '';
$cfg['Servers'][$i]['port'] = '';
```

```
$cfg['Servers'][$i]['socket'] = '';
$cfg['Servers'][$i]['connect_type'] = 'tcp';
$cfg['Servers'][$i]['compress'] = FALSE;
$cfg['Servers'][$i]['controluser'] = '';
$cfg['Servers'][$i]['controlpass'] = '';
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = '';
$cfg['Servers'][$i]['only_db'] = '';
$cfg['Servers'][$i]['verbose'] = '';
$cfg['Servers'][$i]['pmadb'] = ''; // 'phpmyadmin' − see scripts/create_tables.sql
$cfg['Servers'][$i]['bookmarktable'] = ''; // 'PMA_bookmark'
$cfg['Servers'][$i]['relation'] = ''; // 'PMA_relation'
$cfg['Servers'][$i]['table_info'] = ''; // 'PMA_table_info'
$cfg['Servers'][$i]['table_coords'] = ''; // 'PMA_table_coords'
$cfg['Servers'][$i]['pdf_pages'] = ''; // 'PMA_pdf_pages'
$cfg['Servers'][$i]['column_info'] = ''; // 'PMA_column_info'
$cfg['Servers'][$i]['history'] = ''; // 'PMA_history'
$cfg['Servers'][$i]['verbose_check'] = TRUE;
$cfg['Servers'][$i]['AllowDeny']['order']
    = ''; $cfg['Servers'][$i]['AllowDeny']['rules']
    = array();
```

In order to manage more than one server with phpMyAdmin, therefore, all you need to do is configure more servers, by setting values for each block. In case you run out (the default configuration file for phpMyAdmin supports 2 servers other than the primary one), just replicate an empty block (like the one) above and set values for it in the normal manner.

Once multiple servers have been configured, phpMyAdmin will first display a list of servers, and allow you to select a server from the list for use, as seen here.

Note that phpMyAdmin does not automatically connect to all the configured servers on startup; it only attempts a connection when a user selects a server.

In case you need to disable access to a server temporarily, simply set the "host" parameter of the corresponding configuration block to empty, and the server will be ignored by phpMyAdmin

You can also have phpMyAdmin automatically attempt a connection to any one of the listed servers via the $cfg['ServerDefault'] variable in the configuration file. For example,

```
$cfg['ServerDefault'] = 2;
```

A Perfect State

phpMyAdmin comes with a whole set of reporting tools that a database administrator will find extremely useful. It allows you to monitor the status of the server, view a list of active processes (and even kill them), and view statistics on server usage. All this information is easily available from the main page of the application, via a series of links.

Of these, the most interesting is probably the first one, which lets you view detailed status information on the current server state. This section provides information on the server uptime, together with a report on various server statistics. This report is broken down into a number of different areas, each one dealing with a different aspect of the server.

- **Server traffic:** This shows the amount of traffic the server has received since it was last started, including the number of bytes sent and received, and the per−hour traffic. It also reveals the client connections per hour, together with a breakdown of how many succeeded, how many failed, and how many were aborted. Example.
- **Query statistics:** In case you're interested in what users are doing on the server, this section will be enlightening − in addition to calculating the total number of queries processed by the server since startup, it provides a listing (over 50 categories!) of the main query types, together with the number of queries intercepted by the server in each type, and a percentage calculation of the relative quantum of those queries within the total set of queries. Example.

- **Status variables:** This section provides a list of various MySQL runtime variables, including information on the MySQL query cache, the number of threads active, the number of current client connections, the number of running queries and the number of open tables. Example.

  For developers building applications around the new InnoDB transactional table type, the "InnoDB Status" link on this page will hold a great deal of attraction − it provides detailed information on the performance of the InnoDB table handler, buffer pool and memory allocation, and can be used to view and fine−tune application performance. Here's a snippet of the output from this report (if you're not a geek, this will make your eyes water − you have been warned!).

  In addition to obtaining status information on the server, you can also obtain other types of information about the MySQL server (these sections are all available from the main page of the application).

- **Server variables:** This report lists the current values of all the MySQL configuration variables (although it doesn't yet allow you to change any). Example.

- **Active processes:** This report provides a list of all active processes on the MySQL server, and even allows you to kill any of them with a single click. This can come in handy when you need to kill rogue processes that could bring down the server by using more resources then they should; however, always exercise caution when using this function, as you might inadvertently disrupt complex transactions in progress on the server if you don't know what you're doing!

The Privileged Few

You may or may not know this, but the entire MySQL security system consists of five tables (the so−called "grant tables").

```
+−−−−−−−−−−−−−−−−+
| columns_priv
| db
| host
| tables_priv
| user
+−−−−−−−−−−−−−−−−+
```

Each of these has a different role to play in deciding whether a user has access to a specific database, table or table column. Access rules may be set up on the basis of username, connecting host, or database requested.

When a user requests a connection to the database server from a specific host, MySQL will first check whether there is an entry for the user in the grant tables, if the user's password is correct, and if the user is allowed to connect from that specific host. If the check is successful, a connection will be allowed to the server.

Once a connection is allowed, every subsequent request to the server − SELECT, DELETE, UPDATE and other queries − will first be vetted to ensure that the user has the security privileges necessary to perform the corresponding action. A number of different levels of access are possible − some users may only have the ability to SELECT from the tables, while others may have INSERT and UPDATE capabilities, but not DELETE capabilities.

Normally, changes to the grant tables are made using the special GRANT and REVOKE commands (you've already seen these in action when creating the "admin" user a few pages back). However, if you don't really want to use command−line SQL to accomplish user privilege changes, phpMyAdmin includes a Web−based interface for the same that is both friendly and easy to use.

This privilege module is accessible from the main application page, via the "Privileges" link. By default, this module provides a report of all the users MySQL currently knows about, together with a list of the privileges assigned to each one.

Adding a new user is accomplished via the "Add a new user" link, which allows you to create a new user record in the

"users" table and simultaneously assign privileges to that user. When setting user privileges, phpMyAdmin allows you to set privileges for a specific user, or create generic rules that apply to any user. Example.

The privileges that may be assigned to a user are broken down into three categories: data manipulation privileges like SELECT and INSERT, data creation privileges like CREATE and DROP, and administrative privileges like RELOAD and LOCK TABLES. Assigning privileges is as simple as checking boxes on a Web form − no complex syntax needed.

Also new in MySQL 4.x is the ability to assign resource limits − the maximum number of queries, connections and changes per hour − per user; phpMyAdmin supports this as well.Example.

For existing users, the phpMyAdmin privilege module allows you to change user passwords, alter privileges and resource limits, assign privileges on a per−database basis and (I really, really like this feature!) copy an existing user's profile to create a new user with identical privileges but a different name.

As you can imagine, this can come in very handy when you need to quickly create a user "like joe", but don't want to go through the hassle of checking Joe's privileges and manually assigning them to a new user.

In and Out

One of the most common tasks a database administrator performs is the frequent backup of the data in a database. phpMyAdmin simplifies this task via its "Export" module, which makes it possible to export the structure and contents of a database or table to a variety of different formats, either for backup or to migrate data from one database to another.

This module is accessible from the main application page, via the "Export" link. Here's what it looks like.

Using this module is simplicity itself − all you have to do is select the databases to be exported, the format of the output file, and whether you would like the output file to contain the table definitions, the table contents or both. A number of different output formats are available − you can have your data exported as regular SQL queries, as comma−separated values, or as LaTEX−formatted data. You can also choose to either view the exported output directly in your browser (useful if you're trying to quickly obtain information on the structure of a table) or save it to a file for archival or import into another application.

Here's an example of what the output of this module looks like − I've exported it as SQL, with both table definition and contents retained in the final dump.

As with all good export modules, the output file created by phpMyAdmin can be imported back into the application to recreate the database. Simply select the appropriate database (or create a new one as needed), switch to the SQL tab of the database module, and give phpMyAdmin the location of the output file. Example.

The application will now automatically take care of detecting the file format and importing its contents into the selected database.

Mood Ring

Finally, on a lighter note − are you tired of the grey−and−blue colour scheme that phpMyAdmin ships with by default? Well, you can change it to match your mood, simply by editing the appropriate parameters in (you guessed it) the phpMyAdmin configuration file.

A quick glance at the list of options available, and you'll see that just about the entire interface can be customized. In fact, this is just the opportunity for budding artists (like me) to make their lives a little more colourful...

```
$cfg['LeftBgColor'] = '#FF6600'; // background color for the left frame
$cfg['RightBgColor'] = '#00FF33'; // background color for the right frame
$cfg['RightBgImage'] = ''; // path to a background image for the right frame
$cfg['LeftPointerColor'] = '#334455'; // color of the pointer in left frame
$cfg['Border'] = 0; // border width on tables
$cfg['ThBgcolor'] = '#657833'; // table header row colour
$cfg['BgcolorOne'] = '#FFFF00'; // table data row colour
$cfg['BgcolorTwo'] = '#FFCC99'; // table data row colour, alternate
$cfg['BrowsePointerColor'] = '#45AA23'; // color of the pointer in browse mode
$cfg['BrowseMarkerColor'] = '#FF33FF'; // color of the marker (visually marks row)
```

Here's the result.

And on that rather garish note, I'll say goodbye for now. Make sure you come back for the concluding part of this article, in which I'll be discussing some of the more advanced features available in phpMyAdmin. These include transformations, which allow you to convert ordinary fields into more powerful entities; database dictionaries, which let you store information about tables and their columns and relationships in PDF format for your database; and bookmarks, which let you save important or frequently−used queries for future use.

Note: All examples in this article have been tested on MySQL 4.0.14. Examples are illustrative only, and are not meant for a production environment. Developer Shed and Melonfire provide no warranties or support for the source code described in this article.