Remote Database Table Copier

By Stephen Junker

Table of Contents

Overview

While most sites use PHP for little more than selecting and displaying information dynamically on various web pages, the language provides a very robust and flexible interface for databases in general and MySQL specifically, and that interface can be used for much more than simple information display. This fact is not news to most PHP developers, since the database connectivity offered by PHP is widely touted as one of its greatest strengths. Unfortunately, much of the functionality provided for database access is rarely used.

One place where PHP's advanced database access features are used are in any of the several existing script−based MySQL database administration tools such as phpMyAdmin and mysql−admin. These applications attempt to provide a convenient script−based alternative to MySQL's native command line access.

In addition to these tools, the advanced database functionality in PHP affords the developer a great deal of latitude in developing highly useful tools for database manipulation across the internet. This brings me to the real subject of this column −− the script−based server−to−server database table copier.

I would guess that most developers who lack an infinite amount of time to study and play with the tools of the trade approach technology on an as−needed basis. They look for solutions to specific problems as they are needed, and never get an opportunity to explore the possibilities made available through the growing variety of tools available. Fortunately, I recently had a problem that allowed me to explore some concepts that I had thought about, but never been able to test or explore due to the familiar time constraints, which led to the database copier.

The problem seemed reasonably simple: I had built a demonstration of a web−based application on one of my servers, allowing my client to rent the space there until they were able to set up their own hosting arrangement. This was not a problem, though the site was collecting more and more data the longer they waited to move. When the day finally came, the new ISP informed me that, for security reasons, I could not be given shell access to the web space, that I could have only FTP access to the web root.

> How was I to build tables? "Through our convenient web interface."
> How was I to load the legacy data? "Through our convenient web interface."
>
> Load thousands of rows through the web interface? I think not.
> Opportunity to tinker? I think so.

My solution was something that I had imagined while looking through the PHP manual and wondering why you would ever need to use the connection identifier returned by the mysql_connect() function. You would need it if you need to connect to more than one database server or database in the context of a single script−−and that would be terribly useful if you ever needed to copy information from one database to another, even if they were on separate servers.

So, I had a plan: I would create a single script−based tool for the sole purpose of copying data from a MySQL database on any host to a database on a separate host, and do so without having to dump the data to a text file from the source database to be uploaded to the target. As the project progressed, it occurred to me that I would need a few more features to be really useful, like the ability to build target tables on the fly, based on the

structure of the source tables.

The purpose of this article is to describe the methods used to create the database copying tool, and outline features of the technologies used in its creation. Specifically, I focus on the following topics: MySQL permissions for remote connections; PHP support for simultaneous connections to multiple databases; building dynamic INSERT and CREATE TABLE statements with PHP.

The result of that labor was an interesting insight into a number of different possibilities and limitations of PHP, MySQL, and the internet. . . and a script−based tool for copying databases from one server to another.

Note: Throughout the article, I will refer to two databases as the "source" and "target" databases, meaning the databases that the data will be copied from and to respectively. Also, I often use the term "database" to describe both databases and database servers (see previous sentence). I believe the context will make my usage clear, and I will try to make the distinction explicit where necessary.

The Permissions Problem

One problem with this database table copying scheme is that you need to have remote access permissions on at least one of the two servers you're going to access, unless the two databases are on the same server. Most ISPs will set up MySQL databases so they can only be accessed from a specific account from the local server (localhost), and with good reason. This makes perfect sense for most hosting situations where the databases are used to serve data to web pages from a local machine.

However, in this case, either one or both of the MySQL servers will require permission for a remote login from the machine that's running the database copier script. It doesn't matter whether it's the source, the target, or a completely separate machine that's running the script; you will need to be able to set up a remote access permission on whatever machines you need to access that are not running the script.

To copy data, you will need the following permissions.

Permission	Database	Reason
SELECT	Source	Read source data
INSERT	Target	Insert rows in target table
DELETE	Target	Option to clear table before copying
CREATE	Target	Option to create table before copying

MySQL has an excellent security system that can be set up to specify the commands allowed for a given user from a specific host address, and the databases and tables accessible to that user. In addition, each logon is password protected.

To set up a remote logon permission to a MySQL server, you will need to have permission to grant rights to other users on your MySQL server. (If you're using an ISP, it's unlikely that you have grant rights on the server, unless you have a private server space and are running your own MySQL server.) The syntax of the grant command for MySQL is as follows:

```
GRANT [command list] on [database].[table] to [user]@[host]
identified by '[password]';
```

If I were accessing a database at some other server from my personal domain (junkman.org) for access to all tables in the 'demo' database, the command on the remote MySQL server would look something like this:

```
GRANT SELECT, INSERT ON demo.* TO demo@junkman.org IDENTIFIED
BY 'devshed';
```

Keep in mind that you will need to grant remote access permissions on the server that will NOT be running the copier script, because the server that's running the script will be connecting to localhost, which should already have appropriate permissions to select, insert, delete, and create. Also, the user name specified is only for the purpose of authentication on the MySQL server, and is not related to any UNIX or NT authentication on the machine where the server is running.

I would strongly recommend reading the MySQL manual (available in a variety of formats from www.mysql.com) on the subject of permissions and security. Do not take the subject lightly, or make exceptions for ease of use (i.e.: GRANT ALL_PRIVILEGES ON *.* TO root@*;). You (and your clients) would hate to log in someday and find all of your data corrupted or missing!

For experts only: If you need to copy data between ISP−hosted server on which you only have FTP permissions, you can do the following:
1) Set up a MySQL server on your home machine, if you haven't already
2) Allow remote access permissions to the local MySQL server from both ISPs.
3) Set up your router or firewall to allow requests on port 3306 to your local server
4) Run the script at the source ISP to copy data from the source to your local server
5) Run the script at the destination ISP to copy the data from your local server to the target server.

Connecting the World, Two Servers at a Time

Everyone who's reading this article has probably used PHP to connect to a single database, return data to a pointer variable and display it in an HTML wrapper. You've probably also used PHP to take data from an HTML form and insert that data into a table using SQL insert statements. What you likely have not done is performed both of those operations on separate databases nearly simultaneously in the same script.

To perform this trick, we're going to need to make use of the database connection identifier, something which is likely ignored in simpler pages. One very convenient feature of PHP is the assumption that if you've only opened up one database connection, you're going to use it to perform any further database operations and you don't need to specify it in database functions. However, that connection identifier is available if you need perform operations on two databases at once.

To use the connection identifier, simply set the result of your mysql_connect() function to a variable:

```
$source_cnx = mysql_connect("source_db", "uname", "pass");
```

You can then use this connection identifier in any subsequent database related calls such as mysql_select_db() and mysql_query(). It's important to remember to call mysql_close() for every connection that's opened, as well. (This isn't a complete list of functions that can take the connection identifier argument, but they are the most commonly used. For a complete list, please see the PHP manual at www.php.net.) Because the connection identifier is an optional argument in these functions, it is typically the last argument in the function call.

After setting up your first connection, you would set up another in the same way, using a different connection identifier variable:

```
$target_cnx = mysql_connect("target_db", "uname", "pass");
```

Using this information we can set up the shell of connections for a multiple database connection script with some error checking as follows:

```
if (!($srcCnx = mysql_connect($srcHost, $srcUname, $srcPass)))
echo "Unable to connect to $srcHost.<BR>";
if (!mysql_select_db($srcDB, $srcCnx))
```

```
echo "Unable to open database $srcDB on $srcHost.<BR>";
if (!($tgtCnx = mysql_connect($tgtHost, $tgtUname, $tgtPass)))
echo "Unable to connect to $tgtHost.<BR>";
if (!mysql_select_db($tgtDB, $tgtCnx))
echo "Unable to open database $tgtDB on $tgtHost.<BR>";

. . . (perform SQL transactions on databases) . . .

mysql_close($srcCnx);
mysql_close($tgtCnx);
```

Copying the Data

After establishing the connections, the next logical step is to move the data from one place to another. This means "SELECT"−ing the data from the source table, and "INSERT"−ing it into the target table. (Assume for now that the target table already exists and has the correct structure to handle the data−−we'll deal with the concept of dynamically building the correct table structures in the next section.)

This can be done with the code shown below (assuming that you've created the connections with the code from the previous section). Rather than provide extensive commentary outside of the code, I've added commenting in the code where appropriate to show the function of each section.

```
// Return a recordset pointer to all rows in the source table
$srcRst = mysql_query("select * from $srcTable;", $srcCnx);

$fieldList = "";
$columns = mysql_num_fields($srcRst);

// Build the field list for our INSERT statement by iterating
through the column names
for ($i = 0; $i < $columns; $i++) {
$tempField = mysql_field_name($srcRst, $i);
$fieldList .= $tempField . ", ";
}

// Chop off the trailing comma and space (I'm sure there are
more elegant ways of doing this.)
$fieldList = substr($fieldList, 0, strlen($fieldList) − 2);

// Iterate through the rows, creating an insert statement for
each row
while ($srcRow = mysql_fetch_row($srcRst)) {
$insertSQL = "insert into $tgtTable";
$insertSQL = " ($fieldList)";
$insertSQL = " VALUES ";

// Build a comma−separated value list to match the field list
generated above
$insertSQL = "(";
// Iterate through the columns again, this time grabbing the
value from each column
// and building a value list. If values are null, we want to
explicitly insert the null
// value, because dates will be set to all zeroes, if they're
set to an empty string
for ($i = 0; $i < $columns; $i++) {
```

```
if ($srcRow[$i] == null)
$insertSQL .= "null, ";
else
$insertSQL .= "'" . $srcRow[$i] . "', ";
}

// Again, chop off the comma and space at the end
$insertSQL = substr($insertSQL, 0, strlen($insertSQL) − 2);
$insertSQL .= ");";

// Execute the finished SQL insert statement on the target
table
mysql_query($insertSQL, $tgtCnx);
}

// Free memory from the source recordset.
mysql_free_result($srcRst);
```

Building the insert statement on the fly is made significantly easier by MySQL's flexibility with field types when entering data into tables. In my experience with other databases, specifically Access and Visual FoxPro, specific formatting and quoting syntax must be followed for the inserts to work correctly. An ODBC version of this database copying tool would be wonderful, but it would be difficult to deal with the quirks in SQL syntax for each database engine or ODBC driver.

One other thing I should mention is that copying data this way is probably not appropriate for large tables of more than, say, 10,000 rows. To copy larger sets of data, you would likely have to adjust the timeout settings on your web server, if you have access to it. The timeout for most servers is between 30 and 90 seconds before scripts fail due to timeout, and that may not be enough time to copy exceptionally large tables. I've successfully tested the script with tables of up to 5,000 rows without any problems.

Adding Some Options

Copying data from one place to another is great, but our script would be much more useful if it could not only copy the data for us, but build the tables that will store the data before copying. By adding that feature, as well as a few others, we have a tool that can be used in a number of different scenarios.

Building the table on the fly is similar to building a row of data, except that you've got to know the data type and length of each field that you're going to build. PHP has some excellent functionality for obtaining this information not only from a table definition, but from a recordset.pointer as well. The functions are used to dynamically build an SQL create statement to build the table which is executed against the database, as shown below:

```
// Select a recordset with all rows from the source table
$srcRst = mysql_query("select * from $srcTable;", $srcCnx);

$columns = mysql_num_fields($srcRst);

// Iterate through each column, getting the name and type of
each
for ($i = 0; $i < $columns; $i++) {
$tempField = mysql_field_name($srcRst, $i);
$tempType = mysql_field_type($srcRst, $i);

if ($tempType == "string") {
// If the type is string, get the size of the field to size
correctly.
$createSQL .= "$tempField VARCHAR (" .
mysql_field_len($srcRst, $i) . "), ";
}
else
$createSQL .= "$tempField $tempType, ";
}

// Chop the trailing comma and space
$createSQL = substr($createSQL, 0, strlen($createSQL) - 2);

// Attach the field list to the other parts of the create
statement
$createSQL = "CREATE TABLE $srcTable ( $createSQL );";

// Run the query, create the table
mysql_query($createSQL);
```

Unfortunately, this script has limitations. PHP's mysql_field_type() function does not return exact field type information about each field. Instead, it will return general type information like "string", "int", "real", and "blob". Thus, any varchar, char, or text fields are returned as the string type. While giving enough information to be functional, it's not entirely accurate. In this script, all "string" fields are interpreted as VARCHAR type, allowing flexibility without taking up a tremendous amount of space.

Other fields are handled without respect to column size, which could cause a problem for date and blob types. Admittedly, this is one aspect of the project that could be improved, but this is a start, and was acceptable for my purpose at the time.

The finished script also contains some additional functionality, which I will not explain in great detail here. It will suffice to explain the purpose of each option, and let the sample code speak for itself.

- Empty a target table before copying data, which was very useful during script testing

- The option to not copy data, which would allows a user to just create tables based on existing structures or empty tables using the option above without copying any data

- 'Verbose' mode, which simply echoes each SQL statement to the browser before being executed. This should be used carefully on large tables, which would generate thousands or hundreds of thousands on insert statements.

All of these options can be seen in the completed script presentation at the end of the article. They are controlled by checkboxes on the control form, which turn into boolean switches in the part of the script that actually does the copying.

Summary

As a summary, I present a working script that contains the functionality discussed above, paired with a form interface to collect the data needed for the script to work, like server information, table names, and option settings. The form and processing scripts, which could be separated into two separate files, are actually contained in a single file and controlled by a ‘phase’ variable. While a lengthy discussion of the merits and drawbacks of ‘phased’ scripting could be subject of another article, I mention it here only to clear up any confusion about the structure of the finalized script.

I refrain from calling this a finished product, because there are a number of areas for improvement, and it is my hope that this will serve as a starting point, or at least open up some possibilities that you may not have conceived. With this tool, and the foregoing discussion, I have tried to present some advanced features of PHP and its MySQL functionality such as multiple simultaneous database connections, MySQL database permissions for remote access, building "create table" and insert statements on the fly from existing data, and obtaining and using information about tables and fields.

Sample Code

```
<?
// #!/usr/local/bin/php (put this line as first line for
Interland)
if (empty($phase)) {
setcookie("srcHost", "");
setcookie("srcDB", "");
setcookie("srcUname", "");
setcookie("srcPass", "");
setcookie("tgtHost", "");
setcookie("tgtDB", "");
setcookie("tgtUname", "");
setcookie("tgtPass", "");
}

if ($phase == "setHost") {
setcookie("srcHost", $srcHost);
setcookie("srcDB", $srcDB);
setcookie("srcUname", $srcUname);
setcookie("srcPass", $srcPass);
setcookie("tgtHost", $tgtHost);
setcookie("tgtDB", $tgtDB);
setcookie("tgtUname", $tgtUname);
setcookie("tgtPass", $tgtPass);
}
?>
<HTML>
<HEAD>
<TITLE>Remote Database Copier</TITLE>
</HEAD>
<BODY>
<?
if (empty($phase)) {
?>
<FORM method="post" action="<? echo $SCRIPT_NAME; ?>">
<TABLE border=0 cellpadding=5 cellspacing=0>
<TR><TD colspan=2><B>Source:</B></TD></TR>
<TR><TD>Hostname:</TD><TD><INPUT NAME="srcHost"
TYPE="text"></TD></TR>
<TR><TD>Database:</TD><TD><INPUT NAME="srcDB"
TYPE="text"></TD></TR>
<TR><TD>Username:</TD><TD><INPUT NAME="srcUname"
TYPE="text"></TD></TR>
```

```
<TR><TD>Password:</TD><TD><INPUT NAME="srcPass"
TYPE="password"></TD></TR>
<TR><TD colspan=2><B>Target:</B></TD></TR>
<TR><TD>Hostname:</TD><TD><INPUT NAME="tgtHost"
TYPE="text"></TD></TR>
<TR><TD>Database:</TD><TD><INPUT NAME="tgtDB"
TYPE="text"></TD></TR>
<TR><TD>Username:</TD><TD><INPUT NAME="tgtUname"
TYPE="text"></TD></TR>
<TR><TD>Password:</TD><TD><INPUT NAME="tgtPass"
TYPE="password"></TD></TR>
<TR><TD colspan=2 align=center><INPUT TYPE="submit"></TD></TR>
</TABLE>
<INPUT type="hidden" name="phase" value="setHost">
</FORM>
<?
}
if ($phase == "setHost" || $phase == "copy") {
if (!($srcCnx = mysql_connect($srcHost, $srcUname, $srcPass)))
echo "Unable to connect to $srcHost.<BR>";
if (!mysql_select_db($srcDB, $srcCnx))
echo "Unable to open database $srcDB on $srcHost.<BR>";
if (!($tgtCnx = mysql_connect($tgtHost, $tgtUname, $tgtPass)))
echo "Unable to connect to $tgtHost.<BR>";
if (!mysql_select_db($tgtDB, $tgtCnx))
echo "Unable to open database $tgtDB on $tgtHost.<BR>";
}

if ($phase == "setHost") {
if (!($srcRst = mysql_list_tables($srcDB, $srcCnx))) {
echo "Unable to get table list for $srcHost.<BR>";
echo mysql_error($srcCnx);
}
if (!($tgtRst = mysql_list_tables($tgtDB, $tgtCnx))) {
echo "Unable to get table list for $tgtHost.<BR>";
echo mysql_error($tgtCnx);
}
?>
<FORM method="post" action="<? echo $SCRIPT_NAME; ?>">
<TABLE border=0 cellpadding=5 cellspacing=0>
<TR><TD colspan=2><B>Source: (<? echo "$srcHost : $srcDB";
?>)</B></TD></TR>
<TR><TD>Table:</TD>
<TD><SELECT NAME="srcTable">
<? while ($srcRow = mysql_fetch_row($srcRst))
echo "<OPTION value=\"$srcRow[0]\">$srcRow[0]</OPTION>";

?> </SELECT></TD></TR>
<TR><TD colspan=2><B>Target: (<? echo "$tgtHost : $tgtDB";
```

```
?>)</B></TD></TR>
<TR><TD>Table:</TD>
<TD><SELECT NAME="tgtTable">
<? while ($tgtRow = mysql_fetch_row($tgtRst))
echo "<OPTION value=\"$tgtRow[0]\">$tgtRow[0]</OPTION>";
?> </SELECT></TD></TR>
<TR><TD colspan=2><B>Options:</B></TD>
<TR><TD>Create Table?</TD><TD><INPUT type="checkbox"
name="tgtCreate"></TD></TR>
<TR><TD>Clear Table?</TD><TD><INPUT type="checkbox"
name="tgtClear"></TD></TR>
<TR><TD>Copy Data?</TD><TD><INPUT type="checkbox"
name="tgtCopy" CHECKED></TD></TR>
<TR><TD>Verbose?</TD><TD><INPUT type="checkbox"
name="verbose"></TD></TR>
<TR><TD colspan=2 align=center><INPUT TYPE="submit"></TD></TR>
</TABLE>
<INPUT type="hidden" name="phase" value="copy">
</FORM>
<?
mysql_free_result($srcRst);
mysql_free_result($tgtRst);
}

if ($phase == "copy") {
$srcRst = mysql_query("select * from $srcTable;", $srcCnx);

$fieldList = "";
$columns = mysql_num_fields($srcRst);
for ($i = 0; $i < $columns; $i++) {
$tempField = mysql_field_name($srcRst, $i);
$tempType = mysql_field_type($srcRst, $i);

$fieldList .= $tempField . ", ";

if ($tempType == "string")
$createSQL .= "$tempField VARCHAR (" .
mysql_field_len($srcRst, $i) . "), ";
else
$createSQL .= "$tempField $tempType, ";
}

$fieldList = substr($fieldList, 0, strlen($fieldList) - 2);
$createSQL = substr($createSQL, 0, strlen($createSQL) - 2);

if ($tgtCreate) {
$createSQL = "CREATE TABLE $srcTable ( $createSQL );";
if ($verbose) echo $createSQL . "<BR>";
mysql_query($createSQL);
```

```
$tgtTable = $srcTable . "";
}

if ($tgtClear) {
mysql_query("delete from $tgtTable;", $tgtCnx);
echo "Table $tgtTable cleared.<BR>";
}

if ($tgtCopy) {
while ($srcRow = mysql_fetch_row($srcRst)) {
$insertSQL = "insert into $tgtTable ($fieldList) VALUES (";
for ($i = 0; $i < $columns; $i++) {
if ($srcRow[$i] == null)
$insertSQL .= "null, ";
else
$insertSQL .= "'" . $srcRow[$i] . "', ";
}
$insertSQL = substr($insertSQL, 0, strlen($insertSQL) - 2);
$insertSQL .= ");";
if ($verbose) echo "$insertSQL<BR>";
mysql_query($insertSQL, $tgtCnx);
}
echo "Table copied.<BR>";
}

echo "<A HREF=\"$SCRIPT_NAME?phase=setHost\">Another
Table</A><BR>";
echo "<A HREF=\"$SCRIPT_NAME?phase=\">Another
Database</A><BR>";
mysql_free_result($srcRst);
}

if ($phase == "setHost" || $phase == "copy") {
mysql_close($srcCnx);
mysql_close($tgtCnx);
}
?>
</BODY>
</HTML>
```