

## **Using MySQL as a repository for parameters**

### **or for results logging with LoadRunner 9.x**

## **Why do this in the first place?**

### **Parameterisation**

- MySQL gives us the ability to provide LoadRunner with dynamic data (and possibly even change it during a test.).
- In a recent engagement, LoadRunner was having problems when using large parameter files, even on our large load generator and controller servers (Quad core Windows 2003 servers with 6GB RAM). Some parameters were not evaluated properly and occasionally corrupt or partial strings were sent to the test system by LoadRunner.
- Script initialisation was taking a long time during script initialisation when the large parameter files were sent to the load generators by the controller. (Our Parameter files were often 30MB or more and our largest file was 258MB and contained more than three million rows.)

### **Results logging**

- LoadRunner summarises results and can display average response times at a minimum granularity of one second. Our client had a requirement to log all response times to determine whether poor performance in a one-second period was down to one very slow transaction or a number of slower than average transactions.
- It was necessary to be able to share detailed transaction information with the project developers. To help identify one transaction from many, we saved information to a MySQL results database. The records in this database contained the exact time of the transaction, details of the API under test, parameters used as well as information about the HTTP response code.

### **Script configuration**

- We were asked to provide the ability to test multiple APIs either simultaneously or independently. The application which we were testing behaves differently for different client types and so we were asked to cater for this in our scripts. We wrote scripts which would read run-time information from MySQL and set the proportion of transactions of each type accordingly.

In our scripts, we set the proportion of JSON/XML requests in a MySQL table and the LoadRunner scripts read this information at run-time and used the appropriate ratio when simulating client interactions with the test system. The test application was an online music application and using the same technique we could change the proportions of the search scripts so that they searched for artists/tracks/albums in the appropriate ratio.

## Reporting and analysis

- Running queries against the MySQL results database gave us the ability to compare multiple tests with one another, which is not possible using LoadRunner Analysis.
- Tools such as XML/SWF charts by maani.us allow us to produce configurable results charts for our client. [http://www.maani.us/xml\\_charts/](http://www.maani.us/xml_charts/)
- Excel and Access can be used to provide detailed reports and charts based on our test results which can be easily reconfigured by the client to display the results that they're interested in (drill-down).

## How to get started

### **First things first....install the MySQL database server.**

MySQL is the world's most popular open-source database and can be installed on either Windows or LINUX. LINUX versions of MySQL have the advantage of supporting more connections and being slightly more scalable. This is only likely to be important if you are planning on writing very large amounts of data to your results database. More than 2000 transactions per second are possible with 32-bit Windows MySQL, so you only really need to consider using LINUX if you're running at extreme load and want to keep all your test results in a table, or if you already have a MySQL server running on LINUX.

Installing on Windows – Either install WAMP - <http://www.wampserver.com/en/presentation.php> or install each of the subcomponents individually, Apache, MySQL and PHP.

Installing on Linux – Good instructions are available on the internet e.g. Guide to installing on Ubuntu. <http://www.supriyadisw.net/2006/12/lamp-installation-on-ubuntu>

For info:

LAMP = LINUX, Apache, MySQL and PHP,

WAMP = Windows, Apache, MySQL and PHP

Once you've completed your MySQL installation you need to make sure that it is configured to allow remote access so that LoadRunner can get to the database. By default connections are only allowed from the localhost connection. In our examples we have used the "root" user and set the password to "root".

### **Test the MySQL database**

Both WAMP and LAMP installations support PHPmyadmin. PHPmyadmin is a web-based administrative console for the database server. This should be installed and configured to have root-level access to the database. Once this is done, you are presented with a console where you can browse the contents of the database server and perform DBA functions such as creating tables, indexes, dropping tables and viewing table contents. It also includes a console where SQL commands can be entered and executed.

## Sample PHPMyadmin console

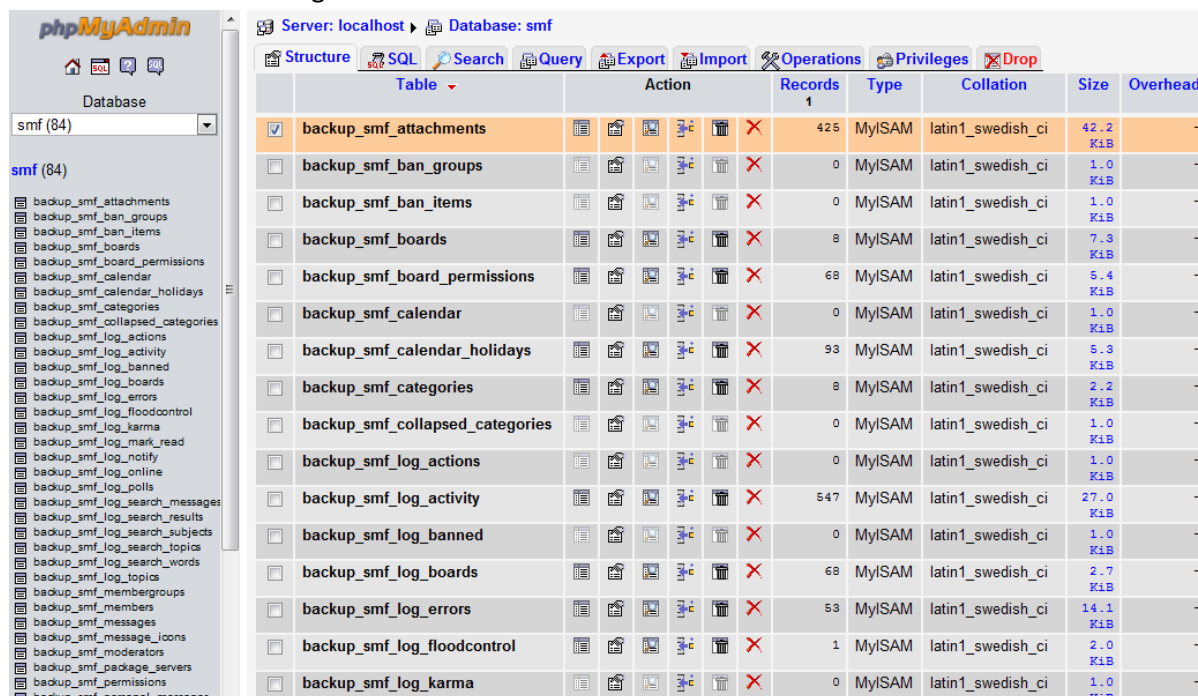
The screenshot shows the PHPMyAdmin console interface. On the left, a sidebar lists databases: information\_schema (28), joomla (36), mysql (32), omnifone (13), and smf (84). The main area features a top navigation bar with tabs: Databases, SQL, Status, Variables,Charsets,Engines,Privileges,Processes,Export, andImport. Below this, the 'MySQL localhost' section allows creating a new database and setting the MySQL connection collation to utf8\_general\_ci. The 'Interface' section shows language set to English, theme set to Original, and font size set to 82%. The right sidebar displays MySQL server information (localhost, version 5.1.36-community, user root@localhost) and web server information (Apache/2.2.11, PHP/5.3.0). A footer message states: 'The additional features for working with linked tables have been deactivated. To find out why click here.'

The tabs in the right hand pane of PHPMyadmin console allow you to perform various functions on the database as outlined in the table below.

Tab	Function
Databases	Displays the databases page where additional databases can be created.
SQL	Displays an SQL console where queries can be run against the server.
Status	Displays the run-time information page including performance statistics and server configuration information.
Variables	Displays the server variables and settings.
Charsets	Displays the character sets available on the server.
Engines	Displays configuration information for the database engines on the server.
Privileges	Displays a table showing user names and privileges for those users.
Processes	Displays the currently running processes on the server, this can be used to identify slow queries or transactions when the server is running.
Export	Contains options for exporting the entire database or individual tables to various file formats, including SQL. These exports can be used to rebuild databases on of the servers if necessary.
Import	Contains options for importing databases (See Export).

## Sample PHPMyAdmin console showing the tables contained within a database

Databases are listed in the left hand column. The database names are hyperlinks and clicking on a database name takes you to the page like the one below with a list of tables. Clicking on a table name allows you to browse the contents of that table and perform other functions such as altering table structure or inserting lines of data to the table.



The screenshot shows the PHPMyAdmin interface. On the left, a sidebar lists the database 'smf (84)' and its tables. The main area displays a table of tables for the 'smf' database. The table has columns: Table, Action, Records, Type, Collation, Size, and Overhead. The tables listed are backup\_smf\_attachments, backup\_smf\_ban\_groups, backup\_smf\_ban\_items, backup\_smf\_boards, backup\_smf\_board\_permissions, backup\_smf\_calendar, backup\_smf\_calendar\_holidays, backup\_smf\_categories, backup\_smf\_collapsed\_categories, backup\_smf\_log\_actions, backup\_smf\_log\_activity, backup\_smf\_log\_banned, backup\_smf\_log\_boards, backup\_smf\_log\_errors, backup\_smf\_log\_floodcontrol, and backup\_smf\_log\_karma.

Table	Action	Records	Type	Collation	Size	Overhead
backup_smf_attachments	[Icons]	425	MyISAM	latin1_swedish_ci	42.2 K1B	-
backup_smf_ban_groups	[Icons]	0	MyISAM	latin1_swedish_ci	1.0 K1B	-
backup_smf_ban_items	[Icons]	0	MyISAM	latin1_swedish_ci	1.0 K1B	-
backup_smf_boards	[Icons]	8	MyISAM	latin1_swedish_ci	7.3 K1B	-
backup_smf_board_permissions	[Icons]	68	MyISAM	latin1_swedish_ci	5.4 K1B	-
backup_smf_calendar	[Icons]	0	MyISAM	latin1_swedish_ci	1.0 K1B	-
backup_smf_calendar_holidays	[Icons]	93	MyISAM	latin1_swedish_ci	5.3 K1B	-
backup_smf_categories	[Icons]	8	MyISAM	latin1_swedish_ci	2.2 K1B	-
backup_smf_collapsed_categories	[Icons]	0	MyISAM	latin1_swedish_ci	1.0 K1B	-
backup_smf_log_actions	[Icons]	0	MyISAM	latin1_swedish_ci	1.0 K1B	-
backup_smf_log_activity	[Icons]	547	MyISAM	latin1_swedish_ci	27.0 K1B	-
backup_smf_log_banned	[Icons]	0	MyISAM	latin1_swedish_ci	1.0 K1B	-
backup_smf_log_boards	[Icons]	68	MyISAM	latin1_swedish_ci	2.7 K1B	-
backup_smf_log_errors	[Icons]	53	MyISAM	latin1_swedish_ci	14.1 K1B	-
backup_smf_log_floodcontrol	[Icons]	1	MyISAM	latin1_swedish_ci	2.0 K1B	-
backup_smf_log_karma	[Icons]	0	MyISAM	latin1_swedish_ci	1.0 K1B	-

## Configure LoadRunner to use MySQL

LoadRunner cannot communicate with MySQL databases "out of the box". A number of library files and a DLL need to be added to LoadRunner before communication is possible. These files need to be added on any system where that a script may be executed, either using vUser generator or the LoadRunner agent process/service.

The file LR\_MySQL.zip contains two folders, *bin* and *include*.

The contents of these folders should be extracted into the LoadRunner *bin* and *include* folders.

Depending on your LoadRunner installation, these folders are normally in

*C:\Program Files (x86)\HP\LoadRunner* or *C:\Program Files\HP\LoadRunner*.

- In your LoadRunner script you need to add the following line into the vuser\_init file above the init section, this makes the library file(s) are available to all subsequent LoadRunner actions. `#include "Ptt_MySQL.h"`
- You also need to add a function to load the DLL into memory which allows LoadRunner to connect to the MySQL database. `lr_load_dll("libmysql.dll");`  
This should be added within the vuser\_init function.

- You now need to add connection statements to your script to give LoadRunner the database connection details.

```
#define MYSQLSERVER      "SERVER_NAME"
#define MYSQLUSERNAME    "USER_NAME"
#define MYSQLPASSWORD    "PASSWORD"
#define MYSQLDB          "SCHEMA_NAME"
#define MYSQLPORT        "PORT"
```

Replace the variables in quotes with the relevant values for your server.

- Your vuser\_init files should now look something like this.

```
#include "Ptt_MySQL.h"

vuser_init()
{
    lr_load_dll("libmysql.dll");

    #define MYSQLSERVER      "127.0.0.1"
    #define MYSQLUSERNAME    "root"
    #define MYSQLPASSWORD    "root"
    #define MYSQLDB          "loadrunner_db"
    #define MYSQLPORT        "3306"

    return 0;
}
```

LoadRunner is now ready to connect to MySQL and execute queries to read, write or update your MySQL database.

### **Creating a sample database**

- Open PHPmyadmin. (<http://127.0.0.1/phpmyadmin>)
- Click on the "Databases" tab.
- Type in the name for your sample database. I use "loadrunner\_db" in this example.
- You will be prompted to create a new table on the database.
- Create two tables
  - one called "parameters" with three fields, username, password and secret.
  - one called "results" with three fields, time, duration and returned\_data.
  - set the field types/lengths to varchar /20 for this exercise.
  - the default storage engine (MyISAM) is suitable for this exercise.

This screen shot shows the options chosen to create the results table.

Server: localhost ▶ Database: loadrunner\_db ▶ Table: results

Field	time	duration	returned_data
Type	VARCHAR	VARCHAR	VARCHAR
Length/Values <sup>1</sup>	20	20	20
Default <sup>2</sup>	None	None	None
Collation			
Attributes			
Null	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index	---	---	---
AUTO_INCREMENT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Comments			

Table comments:

Storage Engine:

Collation:

PARTITION definition:

After creating a table. MySQL displays the SQL queries used to create the table. Rather than use the form shown in the screen shot above, it is possible to simply use these queries to create the tables required. The SQL commands can be either submitted at the command line or alternatively in the SQL tab within the PHPmyadmin console.

#### SQL used to create our sample tables.

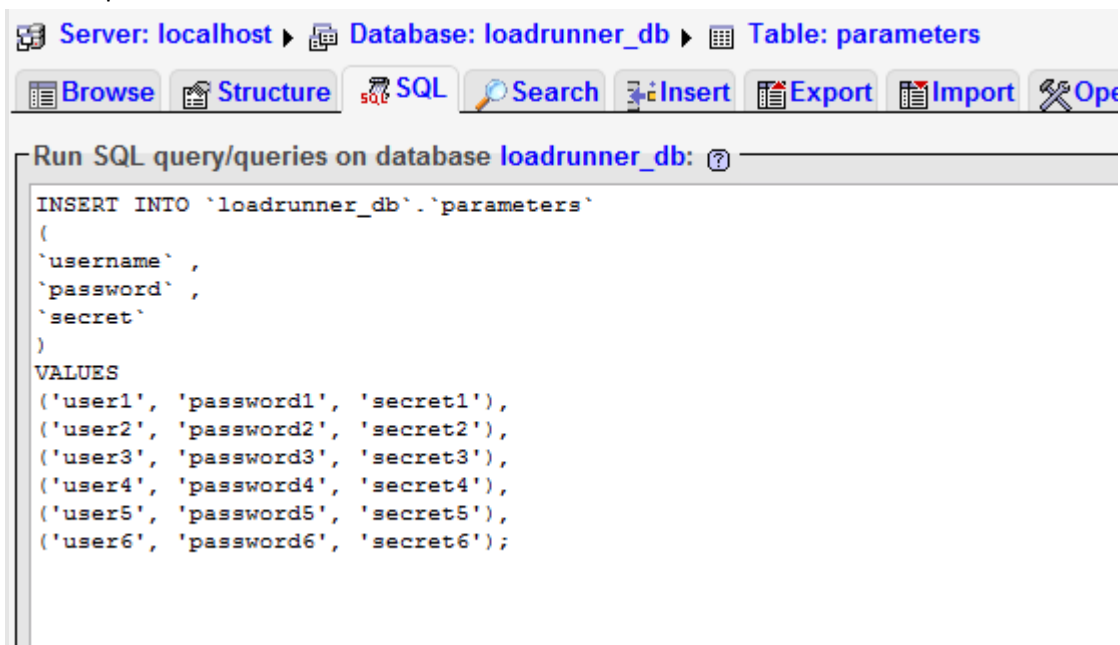
```
CREATE TABLE `loadrunner_db`.`parameters`
(
  `username` VARCHAR( 20 ) NOT NULL ,
  `password` VARCHAR( 20 ) NOT NULL ,
  `secret` VARCHAR( 20 ) NOT NULL
)
ENGINE = MYISAM ;
```

```
CREATE TABLE `loadrunner_db`.`results`
(
  `time` VARCHAR( 20 ) NOT NULL ,
  `duration` VARCHAR( 20 ) NOT NULL ,
  `returned_data` VARCHAR( 20 ) NOT NULL
)
ENGINE = MYISAM ;
```

To populate the parameter table with sample data, use this SQL.

```
INSERT INTO `loadrunner_db`.`parameters`  
(  
  `username` ,  
  `password` ,  
  `secret`  
)  
VALUES  
( 'user1', 'password1', 'secret1' ),  
( 'user2', 'password2', 'secret2' ),  
( 'user3', 'password3', 'secret3' ),  
( 'user4', 'password4', 'secret4' ),  
( 'user5', 'password5', 'secret5' ),  
( 'user6', 'password6', 'secret6' );
```

This screenshot shows the SQL code pasted into the PHPMyadmin console immediately before the insert is performed.



**But what if I want to insert lots of data?**

It is possible to insert large amounts of data into a MySQL table using the MySQL LOAD DATA INFILE statement. This statement reads row from a text file into a table at high speed. Details of the syntax for this command are available on-line in the MySQL documentation.

<http://dev.mysql.com/doc/refman/5.1/en/load-data.html>

Once the above sample code has been executed it is possible to see the contents of the parameters table by clicking on the table name in the left hand column (highlighted in orange in the screenshot below).

The screenshot shows the phpMyAdmin interface. On the left, the 'loadrunner\_db (2)' database is selected, and the 'parameters' table is highlighted in orange. The main panel shows the table's structure and data. The SQL query executed is:

```
SELECT *
FROM 'parameters'
LIMIT 0 , 30
```

The table displays 6 rows of data:

	username	password	secret
<input type="checkbox"/>	user1	password1	secret1
<input type="checkbox"/>	user2	password2	secret2
<input type="checkbox"/>	user3	password3	secret3
<input type="checkbox"/>	user4	password4	secret4
<input type="checkbox"/>	user5	password5	secret5
<input type="checkbox"/>	user6	password6	secret6

Below the table, there are options to 'Check All / Uncheck All' and 'With selected:'. The 'Show' button is set to 30 row(s) starting from record # 0, and the 'in' dropdown is set to 'horizontal' mode and repeat headers after 100 cells.



### Using LoadRunner to read information from a MySQL table

If we return to the LoadRunner script we can now start to insert functions which will connect to and read data from the MySQL table. A sample Action() statement below demonstrates the relevant syntax of for reading a row of data from MySQL.

See also the sample script [MySQL\\_SampleRead](#).

```
//Declare MySQL Elements
char sqQuery[512];           // The MySQL query to be executed
MYSQL *Mconn;               // The MySQL connection string
int MyRC;                   // The MySQL return code

Action()
{

//Create the connection string to connect to the server and relevant database
Mconn = lr_mysql_connect(MYSQLSERVER, MYSQLUSERNAME, MYSQLPASSWORD, MYSQLDB,
atoi(MYSQLPORT));

//Build an SQL query and save the query as sqQuery
sprintf(sqQuery, "select username, password, secret from loadrunner_db.parameters");

//Execute the SQL query (sqQuery) against the server and database defined as Mconn
lr_mysql_query(Mconn, sqQuery);

//Save the values returned in the first row of the table
//N.B. The row/cell values appear to be the "wrong way round"

lr_save_string(row[0][0].cell, "sUsername");
lr_save_string(row[1][0].cell, "sPassword");
lr_save_string(row[2][0].cell, "sSecret");

//Output the returned values
lr_output_message(lr_eval_string("Username: {sUsername}"));
lr_output_message(lr_eval_string("Password: {sPassword}"));
lr_output_message(lr_eval_string("Secret: {sSecret}"));

//Disconnect from MySQL
lr_mysql_disconnect(Mconn);

return 0;
}
```

### **A note about the data returned**

You can run any query from the 'lr\_mysql\_query' function. If you have run a select statement, you will be expecting some data to be returned. The data is returned in a multi-dimensional array and is referenced in your script by the function:

```
row[x][y].cell
```

where x is the column position (first column is 0) and y is the row position (first row is 0).

### **Randomising the data returned**

Whilst it may be useful to be able to return data from a multiple rows as shown above, it may be advantageous to select data at random from a table. To achieve this it is first necessary to know the number of rows in the table.

A random number less than or equal to the total number of rows in the table can then be used to identify which row to return to LoadRunner. An SQL query can be used to count the number of rows and return this information to LoadRunner. LoadRunner can then use this information to select a random row from the table. For tables with more than 32,768 rows it is necessary to use a function such as [fRandInt](#) to calculate a random number.

In the earlier example, the statement below is used to select the contents of a table and return it into the multi-dimensional array stored in LoadRunner. This is inefficient because this query returns the entire contents of the table to LoadRunner.

```
sprintf(sqQuery, "select username, password, secret from loadrunner_db.parameters");
```

It is possible to limit the number of rows returned by using the MySQL "LIMIT" statement.

#### **For example:**

These commands will return the first four rows (rows 0 through to 3) if executed sequentially.

```
sprintf(sqQuery, "select username, password, secret from loadrunner_db.parameters LIMIT 0,1");  
sprintf(sqQuery, "select username, password, secret from loadrunner_db.parameters LIMIT 1,1");  
sprintf(sqQuery, "select username, password, secret from loadrunner_db.parameters LIMIT 2,1");  
sprintf(sqQuery, "select username, password, secret from loadrunner_db.parameters LIMIT 3,1");
```

## Using LoadRunner to read random information from a MySQL table

The sample script [MySQL\\_SampleRandomRead](#) contains an example of this in practice.

- This query is used to return the number of rows in the parameter table.  
`sprintf(sqQuery, "SELECT COUNT(*) FROM loadrunner_db.parameters");`
- A random number function is used to calculate a random number between 0 and the final row in the parameter table.  
`iRand = fRandInt (1,(iNumRecords+1))-1;`
- Finally another query is used to return the random row.  
`sprintf(sqQuery, "select username, password, secret from loadrunner_db.parameters LIMIT %d,1",iRand);`
- 

### Sample script which reads a random record from the parameters table

```
//Declare MySQL Elements
char sqQuery[512];           // The MySQL query to be executed
MYSQL *Mconn;                // The MySQL connection string

//Declare variables used to choose a random row.
int iNumRecords;             // The number of records in the parameter table
int iMIN, iMAX;               // The minimum and maximum values for the random number function
int iRand;                   // The random number generated by the fRandInt function

Action()
{
    //Create the connection string to connect to the server and relevant database
    Mconn = lr_mysql_connect(MYSQLSERVER, MYSQLUSERNAME, MYSQLPASSWORD, MYSQLDB, atoi(MYSQLPORT));

    // Build and execute query to determine the number of rows in the parameters table
    sprintf(sqQuery, "SELECT COUNT(*) FROM loadrunner_db.parameters");
    lr_mysql_query(Mconn, sqQuery);

    //Save the returned value as a string called iNumRecords
    lr_save_string(row[0][0].cell, "sNumRecords");
    lr_output_message( "sNumRecords %s", lr_eval_string( "{sNumRecords}" ) );

    //Convert the string into an integer so that it can be used in the fRandInt function
    iNumRecords = atoi(lr_eval_string("{sNumRecords}"));
    lr_output_message( "iNumRecords %d", iNumRecords );

    //Choose a random row between 0 and iNumRecords
    iRand = fRandInt (1,(iNumRecords+1))-1;
    lr_output_message("Random number is : %d",iRand);

    //iRand=2;

    //Build an SQL query and save the query as sqQuery
    sprintf(sqQuery, "select username, password, secret from loadrunner_db.parameters LIMIT %d,1",iRand);

    //Execute the SQL query (sqQuery) against the server and database defined as Mconn
    lr_mysql_query(Mconn, sqQuery);

    //Save the values returned in the random row of the table
    //N.B. The row/cell values appear to be the "wrong way round"
    lr_save_string(row[0][0].cell, "sUsername");
    lr_save_string(row[1][0].cell, "sPassword");
    lr_save_string(row[2][0].cell, "sSecret");

    //Output the returned values
    lr_output_message(lr_eval_string("Username: {sUsername}"));
    lr_output_message(lr_eval_string("Password: {sPassword}"));
    lr_output_message(lr_eval_string("Secret: {sSecret}"));

    //Disconnect from MySQL
    lr_mysql_disconnect(Mconn);

    return 0;
}
```

### Using LoadRunner to write information to a MySQL table

In the example shown so far, simple SQL statements are used to read data from tables in the database. In the same way SQL statements can be used to insert data into MySQL database tables., as shown in the example SQL statement below.

```
sprintf(sqQuery, "insert into loadrunner_db.results (time, duration, returned_data)"
    "values (\\"%s\\",\\"%f\\",\\"%i\\")",
    lr_eval_string("{sTime}"),
    dDuration,
    iRand);

//Declare MySQL Elements
char sqQuery[512];           // The MySQL query to be executed
MYSQL *Mconn;               // The MySQL connection string

//Declare variables which can be sent to MySQL in this sample script
double dDuration;
int iMIN, iMAX, i, iRand;

Action()
{
    lr_start_transaction("0001_Example");

    lr_think_time(1);
    dDuration=lr_get_transaction_duration("0001_Example");

    lr_end_transaction("0001_Example",LR_AUTO);

    lr_save_datetime("%H:%M:%S", TIME_NOW, "sTime");
    iRand = fRandInt (1,1000000);

    //Create the connection string to connect to the server and relevant database
    Mconn = lr_mysql_connect(MYSQLSERVER, MYSQLUSERNAME, MYSQLPASSWORD, MYSQLDB, atoi(MYSQLPORT));

    //Build an SQL query and save the query as sqQuery
    sprintf(sqQuery, "insert into loadrunner_db.results (time, duration, returned_data)"
        "values (\\"%s\\",\\"%f\\",\\"%i\\")",
        lr_eval_string("{sTime}"),
        dDuration,
        iRand);

    //Execute the SQL query (sqQuery) against the server and database defined as Mconn
    lr_mysql_query(Mconn, sqQuery);

    //Disconnect from MySQL
    lr_mysql_disconnect(Mconn);

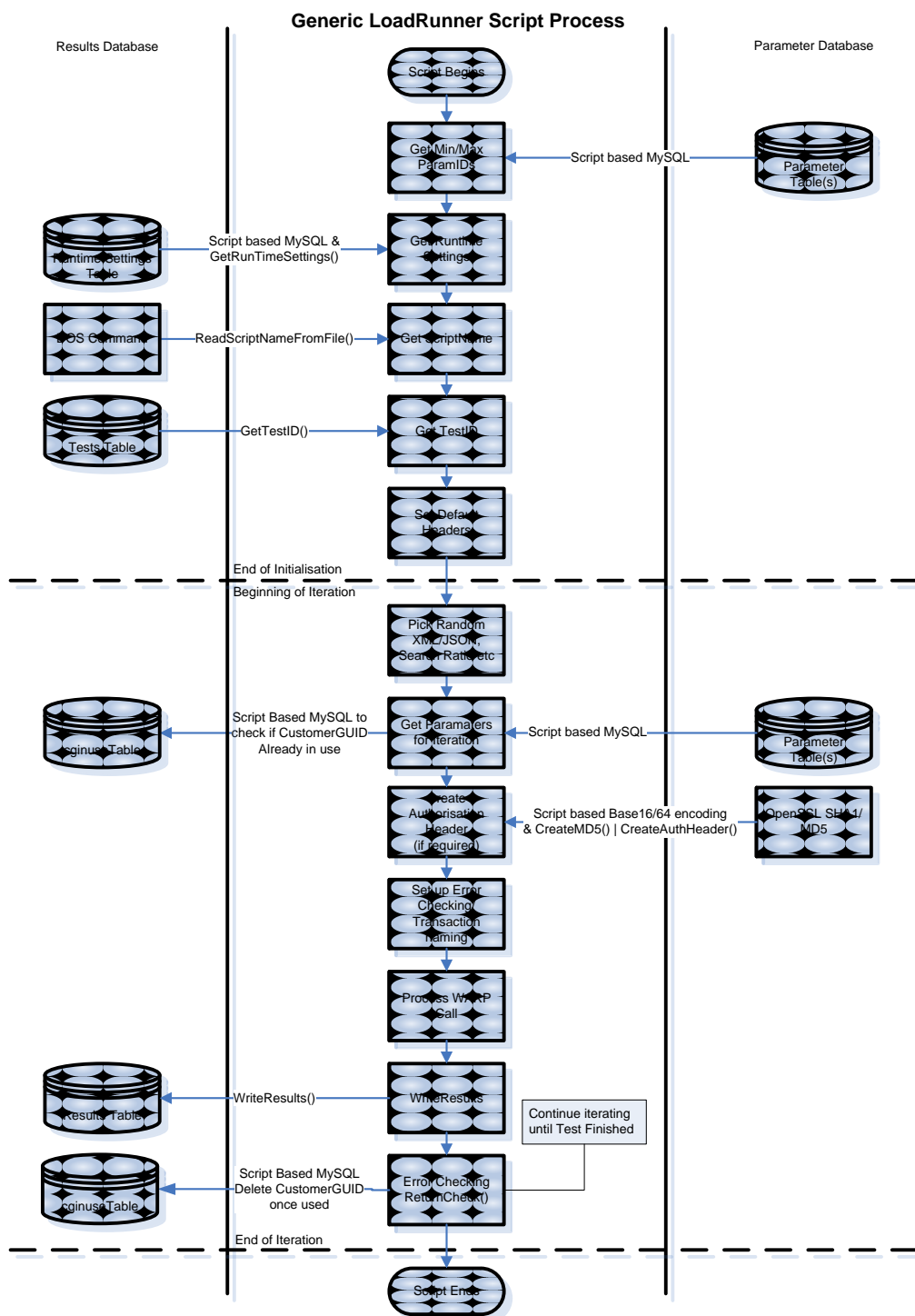
    return 0;
}
```

The sample script [MySQL\\_SampleWriteResults](#) contains a working example of this code and shows how the duration for a transaction, a random number (representing data returned from the test system and the current time could be stored in a results log).

## How far can you go using this technique?

It is possible to develop scripts which perform complex functions such as creating tables which do not already exist or using temporary tables containing details of parameters which are in use by virtual users to avoid problems in the application under test.

In our testing we have developed functions which read in run time settings from a table, read in parameter values, write test information to a results database and log all transactions to a database as described in the process flow below.



## Optimising performance of your MySQL databases

MySQL is a highly configurable database server. Both the LINUX and Windows versions can be configured by editing variables in configuration files. There is a large amount of documentation available on the Internet describing how to do this in detail, but here are a few pointers.

After you have been using MySQL for a period of time, usage statistics are visible in the "Status" tab of PHPmyadmin. This page contains details of the number of transactions processed by the server since start-up.

### The "Status" tab

Server: localhost

Databases SQL Status Variables Charsets Engines Privileges Processes

#### Runtime Information

[Refresh] [Reset] [?]

This MySQL server has been running for 0 days, 21 hours, 45 minutes and 45 seconds. It started up on Aug 05, 2010 at [SQL query] [InnoDB] [SSL] [Handler] [Query cache] [Threads] [Binary log] [Temporary data] [Delayed inserts] [Key cache]

**Server traffic: These tables show the network traffic statistics of this MySQL server since its startup.**

Traffic <sup>1</sup>		Ø per hour	Connections		Ø per hour	%
Received	127 KiB	5,972 B	max. concurrent connections	2	---	---
Sent	943 KiB	43 KiB	Failed attempts	0	0.00	0.00%
Total	1,070 KiB	49 KiB	Aborted	2	0.00 k	0.56%
			Total	359	16.50	100.00%

**Query statistics: Since its startup, 2,594 queries have been sent to the server.**

Total	Ø per hour	Ø per minute	Ø per second
2,594	119.20	1.99	0.00 k

Query type	Ø per hour	%	Query type	Ø per hour	%		
admin commands	0	0.00	0.00%	restore table	0	0.00	0.00%
assign to keycache	0	0.00	0.00%	revoke	0	0.00	0.00%
alter db	0	0.00	0.00%	revoke all	0	0.00	0.00%
alter db upgrade	0	0.00	0.00%	rollback	0	0.00	0.00%
alter event	0	0.00	0.00%	rollback to savepoint	0	0.00	0.00%
alter function	0	0.00	0.00%	savepoint	0	0.00	0.00%
alter procedure	0	0.00	0.00%	select	397	18.24	17.76%
alter server	0	0.00	0.00%	set option	362	16.63	16.20%
alter table	0	0.00	0.00%	show authors	0	0.00	0.00%
alter tablespace	0	0.00	0.00%	show binlog events	0	0.00	0.00%
analyze	0	0.00	0.00%	show binlogs	18	0.00 k	0.81%

As well as providing information about server use, this tab also advises on configuration settings which can be altered to improve performance.

### Example of advice given by MySQL to alter table cache value

Variable	Value	Description
Open_tables	0	The number of tables that are open.
Opened_tables	278	The number of tables that have been opened. If opened tables is big, your table cache value is probably too small.
Table_locks_immediate	444	The number of times that a table lock was acquired immediately.
Table_locks_waited	0	The number of times that a table lock could not be acquired immediately and a wait was needed. If this is high, and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

Settings such as "table cache value" can be altered in the *my.ini* file (Windows) or the *my.cnf* file (LINUX) on the database server. After changing settings the server needs to be restarted to start the server using the new settings.

### Database engines

MySQL has two commonly used database engines, MyISAM and InnoDB. The storage engine doesn't need to be set at a database level, each database can have tables which use either of the two storage engines.

MyISAM is the older of the two storage engines in MySQL, it uses table level locking and is more RAM and disk space efficient. This performs better when you want to do a full text search or when you want to do more "selects" than "inserts" on a table.

InnoDB is the newer storage engine it uses row level locking, this is better for our results tables since it performs better than MyISAM when you're doing more "inserts" than "selects".