

Maatkit Toolkit Tutorial: mk-table-checksum

Most production environments have a master/slave setup of servers. This provides the ability to easily perform backups, have warm fail-over if you have a hardware failure on the master, etc. Replication between the master and slave is fairly easy to set up, however as Baron points out [here](#), it is possible (and probably quite common) for for a master and slave to “drift” out of sync.

Fortunately, not only did he point out the problem, he provides a fix for the problem. Included in the [maatkit toolkit](#) is the mk-table-checksum tool (and the related tool mk-table-sync which we won't be discussing in this tutorial). It can tell you which tables on the master and and slave are out of sync and even tell you within a fair degree what rows are out of sync. Here is an example run:

```
mk-table-checksum -chunksize 500000 -replicate=test.checksum localhost
```

What are we doing here? Well, the chunksize option specifies that if tables are larger than 500,000 rows (in this case) that they should be chunked — broken apart — when processing. This will mean that a table doesn't get blocked for too long a period of time. The replicate option specifies that you are wanting the checksum tool to use the test.checksum table to store results in. This can be very helpful in providing an accurate picture of your results. The create command for this table should be:

```
CREATE TABLE `checksum` (  
  `db` char(64) NOT NULL,  
  `tbl` char(64) NOT NULL,  
  `chunk` int(11) NOT NULL,  
  `boundaries` char(64) NOT NULL,  
  `this_crc` char(40) NOT NULL,  
  `this_cnt` int(11) NOT NULL,  
  `master_crc` char(40) default NULL,  
  `master_cnt` int(11) default NULL,  
  `ts` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  
  PRIMARY KEY (`db`,`tbl`,`chunk`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

It doesn't have to be in the test database by the way. You can put it wherever you want. Your option. Just be sure to specify the right database when running the mk-table-sync command.

The final option is really easy. It just specifies the location of the master server. In this case, I ran the command on the master server itself so localhost was specified.

It will take a while for this command to run your databases are of any size. Once it is done executing and your slave has caught up (if it lagged behind the master) than you just run the following:

```
mk-table-checksum -replicate=test.checksum -replcheck 2 localhost
```

This will give you the results you want. Some sample output:

```
mk-table-checksum --replicate=test.checksum -replcheck 2 localhost
```

```
Differences on h=server.slave
```

DB	TBL	CHUNK	CNT_DIFF	CRC_DIFF	BOUNDARIES
database_1	table_1	0	0	1	`clientid` < 184723
database_1	table_2	0	0	1	`clientid` < 184721
mysql	db	0	-19	1	1=1
mysql	user	0	-1	1	1=1

This shows that there were some differences on the slave. In the mysql database, on the slave, there are nineteen fewer rows in the db table and one less row in the user database. This is because I have different permissions on my master and slave servers.

On database_1 I have two tables that are out of sync. The two table_1 tables (on the master and slave) have the same row count (shown by the **cnt_diff** column). However, the **crc_diff** column shows that there are differences in the contents of the columns on the two tables. The **boundaries** column show you which chunk of the table that the differences are located in. You can use this to figure out the exact problem yourself..or you can use the mk-table-sync tool to do this for you.

To finish the discussion on the mk-table-checksum tool I wanted to let you know about the custom udf (user defined function) that Baron programmed for use in the maatkit toolkit. Essentially the udf provides a replacement for the md5 hash function. This function provides an impressive speedup. On two servers I ran the mk-table-checksum tool with the default setup and then installed the udf and re-ran the checksum tool. Here are the results:

server	checksum w/md5	checksum w/fvn
db3	17m10.220s	5m0.090s
db6	112m21.719s	53m58.368s

Clearly it is a great benefit to use this udf if you are running table checksums. So how do you set it up? The source code is included in the tar.gz file that is distributed from the [maatkit website](#). Just download and unzip/untar. In the included files there is a udf directory with one file. To compile “just”:

```
gcc -fPIC -Wall -I/usr/include/mysql -shared -o fnv_udf.so fnv_udf.cc
```

The instructions are included in the source code. Once you have it compiled, just copy the resultant file (fnv_udf.so) to the /lib directory and then run the command:

```
mysql mysql -e "CREATE FUNCTION fnv_64 RETURNS INTEGER SONAME 'fnv_udf.so'"
```

This adds the function to your MySQL server setup. To test, log into mysql and make this call:

```
mysql> SELECT FNV_64('hello', 'world');
```

This should returns some results if everything is done properly. Once your MySQL server “sees” the new udf maatkit will begin using it automatically.

Baron has an excellent background post on the fvn udf here: [xaprb post](#)