Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, Volume 21, No. 1, ISSN 1726-9679
ISBN 978-3-901509-73-5, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2010
Make Harmony Between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals of DAAAM International

USING MYSQL QUERY CACHE TO SPEED UP QUERY PERFORMANCE

BOICEA, A[lexandru]; PETCU, R[obert]; RADULESCU, F[lorin] & TRIFAN, I[onut]

Abstract: *This paper presents how to use query cache from MYSQL to improve your database response from a query. This query cache caches the results of SELECT queries and the most frequently used database queries will run much faster, because the data results will be read from the cache instead of having to run the query again. The testing was made on an database backup dumps from Wikimedia through execution of stored procedure PL/SQL which is processing the fields table from this database.*
Keywords: *database, query, cache, mysql, stored procedure*

1. INTRODUCTION

As we know, speed is always the most important element in developing aplications, especially for those aplications with high traffic database (http://www.techiecorner.com). MySQL Query Cache is a powerful feature which when it is used correctly can give big performance gains on MySQL instance (http://www.dbtuna.com).

The query cache stores the text of a SQL query statement (or stored procedure) together with the corresponding results that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again. A result set generated by one client can be sent to the same query issued by another client because the query cache is shared among sessions (http://dev.mysql.com/doc/refman/5.1 /en/query-cache.html).

The query cache can be very useful in an environment where you have tables that do not change very often and for which the server receives many identical queries (http://www.mysql performanceblog. com). To show if the query cache is enabled and what parameters are set we can use the next SQL statement:

```
SHOW VARIABLES LIKE '%query_cache%'
```

Fig. 1. SQL statement

An example result for the above query is below, which shows that the query cache engine is available, but the query cache size is set to zero and therefore nothing will be cached, and the query cache engine will not actually be used.

Variable_name	Value
have_query_cache	YES
query_cache_limit	1048576
query_cache_min_res_unit	4096
query_cache_size	0
query_cache_type	ON
query_cache_wlock_invalidate	OFF

Fig. 2. Query cache variables and their values

2. MYSQL QUERY CACHE CONFIGURATION

First we need to have MySQL 4.0.1 or higher to use query cache and then to see if query cache is enabled (http://www.petefreitag.com). We can do this by selecting the *have_query_cache* variable from VARIABLES. Several other system variables control query cache operation, variables that can be set in an option file (my.ini from mysql directory) or on the command line when starting mysqld (http://www.databasejournal. com).

To set the size of the query cache first set the *query_cache*_size system variable. If we set this variable to zero the query cache will be disabled. When the *query_cache_size* is a nonzero value, you have to keep in mind that the query cache needs a minimum size of 40KB to allocate its structures (http://ronaldbradford.com/ blog).

The sql statement to set the size of the query cache is:

```
mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_size | 41984 |
+------------------+-------+
```

Fig. 3. Set query_cache_size

The *query_cache_size* value is aligned to the nearest 1024 byte block so the value reported may be different from the value that we assign. After we set this variable the *query_cache_type* variable influences the way query cache works. This variable can be set to the following values:

- a value of 0 or OFF prevents caching or retrieval of the cached results;
- a value of 1 or ON allows caching except of those statements that begin with SELECT SQL_NO_CACHE;
- a value of 2 or DEMAND causes caching for only those statements that begin with SELECT SQL_CACHE (http://www.cyberciti.biz).

Setting the GLOBAL *query_cache_type* value determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the SESSION *query_cache_type* value. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

Fig. 4. Set query_cache_type

We can also manipulate query cache with the following MySQL statements:

- *RESET QUERY CACHE* – remove all the queries from query cache;

- *FLUSH QUERY CACHE* – defragment the query cache memory

3. TEST PROCEDURE

For testing we used some dump tables downloaded from Wikimedia (http://dumps.wikimedia.org/backup-index.html). The database name used is cache_test were we have imported the following tables: *imagelinks* with 31.546 rows, *langlinks* with 228.864 rows, *pagelinks* with 5.887 rows, *page_restrictions* with 14 rows and *template_links* with 537.887 rows.

To select data from this tables we have build some stored procedures. The first procedure is called select_data() and has four parameters: table_db1, column_db1, table_db2 and column_db2. This procedure selects the columns of the tables introduced as parameters.

Procedure simple_select() has no parameters and it selects columns *pl_from, pl_namespace, pl_title* from pagelinks table, columns *tl_from, tl_namespace, tl_title* from templatelinks where the column *pl_from* and *tl_from* are equal. Procedure call_all() calls the above procedures with predefined parameters. Last procedure (select_two()) will select data from the largest database tables (*langlinks and templatelinks*).

4. TEST RESULTS

The results from our test are splited in two parts: tests done with query cache disabled and tests done with query cache enabled.

To achieve the table below each procedure was called by five consecutive times to make a good estimation for the response time (response time is measured in seconds).

Procedure	t1	t2	t3	t4	t5
select_data	1.516	1.594	2.203	1.547	1.453
simple_select	3.094	3.218	3.000	3.093	3.344
call_all	4.891	4.953	4.547	4.609	4.812
select_two	82	79	78	79	80

Tab. 1. Test results with query cache disabled

Procedure	t1	t2	t3	t4	t5
select_data	1.500	1.359	1.344	1.328	1.343
simple_select	3.125	3.078	3.032	3.022	3.110
call_all	4.563	4.329	4.485	4.390	4.312
select_two	85	80	78	80	78

Tab. 2. Test results with query cache enabled

To highlight differences between the time response of the procedures with query cache disabled and query cache enabled we got the values from above tables, we calculated the average of given values for three of the four procedures and then we have built the chart in Figure 5.

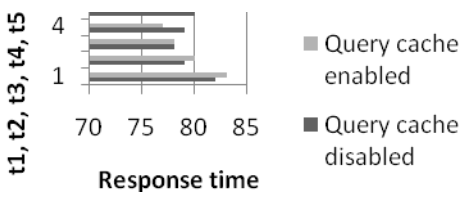

Fig. 5. Response time chart

Because the big diference between time responses from the first three procedures and the *select_two()* procedure we decided to make an other chart (Figure 6). In this chart we will have all the values from the tables with query cache disabled and the table with query cache enabled.

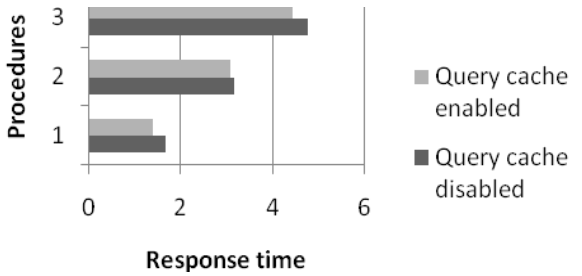

Fig. 6. Response time for *select_two()*

5. CONCLUSIONS

Analyzing the results of testing it can be easily observed the difference between the response time when we have query cache disabled and when it is enabled.

The results set must be equal or smaller than the *query_cache_limit* and we can't use MySQL version 5.0 because querys running from stored procedures, functions or triggers are not cached (http://rackerhacker.com).

Thus, the procedures that make selects from tables with ten thousands of records has differences of tens microseconds when using query cache, the procedures which make select from tables with hundred of thousands records differences between using query cache disabled and query cache enabled is of seconds. This fact draws us to the conclusion that we can obtain higher performance in MySql applications if we use the query cache enabled and we have set the query_cache_size variable to a value greather than 40KB.

Also, query cache can be useless if it is used in an application were we have a lot of inserts or updates on the tables of database but, when a table is modified(insert, updates, etc.) the cache automatically expires. Other reasons which will make queries un-cacheable are:

- use of functions, such as CURRENT_DATE, RAND and user defined functions;
- queries that uses bind variables.

6. REFERENCES

Freitag, P. (2005); *The MySQL Query Cache*, *Available from:* http://www.petefreitag.com, *Accessed on: 2010-05-14*

Hacker, R. (2007); *MySQL's query cache explained*, *Available from:* http://rackerhacker.com, *Accessed on: 2010-05-14*

Bradford, R. (2009); *Using the Query Cache effectively*, *Available from:* http://ronaldbradford.com/blog, *Accessed on: 2010-05-26*

*** *(2009)* http://www.dbtuna.com, *MySQL Query Cache Performance*, *Accessed on: 2010-05-26*

*** *(2007)* http://www.cyberciti.biz, *Enable the query cache in MySQL to improve performance*, *Accessed on: 2010-05-13*

*** *(2006)* http://www.mysqlperformanceblog.com, *MySQL Performance Blog*, *Accessed on: 2010-05-01*

*** *(2006)* http://dev.mysql.com/doc/refman/5.1/en/query-cache.html, *The MySQL Query Cache*, *Accessed on: 2010-05-01*

*** *(2006)* http://www.techiecorner.com, *Turn on MySQL query cache to speed up query performance?*, *Accessed on: 2010-05-03*

*** *(2003)* http://www.databasejournal.com, *MySQL's Query Cache*, *Accessed on: 2010-05-03*