MySQL Performance

Sang Shin
www.javapassion.com
"Learning is fun!"

Topics

- Query cache
  - Overview
  - System variables
- Indexing
  - Overview
  - Example query
  - Strategy
  - Runtime counters
- EXPLAIN
- Slow query log

Query Cache:
Overview

What is Query Cache?

- The query cache stores the text of a SELECT statement together with the corresponding result that was sent to the client
  - If an identical statement (case sensitive) is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again
- The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client

Why Query Cache?

- The query cache is extremely useful in an environment where you have tables that do not change very often and for which the server receives many identical queries
  - This is a typical situation for many Web servers that generate many dynamic pages based on database content

How Does Query Cache Work?

- When a table is changed, any relevant entries in the query cache are discarded
  - Prevents the query cache from being stale
- The query has to be exactly the same including the case
  - "SELECT * from person" is considered a different query from "select * from person"

Query Cache:
System Variables

Configuration variables - You can set

- query_cache_size
  - The amount of memory allocated for caching query results. The default value is 0, which disables the query cache.
- query_cache_type
  - 0 or OFF: Query caching is turned off
  - 1 or ON: Query caching is turned on
  - 2 or DEMAND: Query caching is done on demand basis
- query_cache_limit
  - Maximum result set size. Result sets larger than this value will not be cached

Configuration variables

```
mysql>  SHOW variables LIKE '%query_cache%';
+------------------------------+-------------+
| Variable_name                | Value       |
+------------------------------+-------------+
| have_query_cache             | YES         |
| query_cache_limit            | 1048576     |
| query_cache_min_res_unit     | 4096        |
| query_cache_size             | 1048576     |
| query_cache_type             | ON          |
| query_cache_wlock_invalidate | OFF         |
+------------------------------+-------------+
6 rows in set (0.00 sec)
```

Runtime counter variables

- Qcache_inserts
  - Total number of times queries are cached
- Qcache_queries_in_cache
  - Number of queries that are currently present in the query cache
  - Will be reset to 0 when query cache is flushed
- Qcache_hits
  - Number of times queries in the cache are hit
  - High number of this indicates that the cache is being used effectively

Runtime counter variables

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Qcache_free_blocks            | 0     |
| Qcache_free_memory            | 0     |
| Qcache_hits                   | 0     |
| Qcache_inserts                | 0     |
| Qcache_lowmem_prunes          | 0     |
| Qcache_not_cached             | 0     |
| Qcache_queries_in_cache       | 0     |
| Qcache_total_blocks           | 0     |
+-------------------------------+-------+
8 rows in set (0.00 sec)
```

Counter variables after 1st query

```
mysql> select * from employees where employee_id = 5;
+---------------+---------+------+-----------+
| employee_id | ename | d_id | salary     |
+---------------+---------+------+-----------+
|                5 | jones    |     3 | 5000.00 |
+---------------+----------+------+-----------+

mysql>  SHOW STATUS LIKE 'Qcache%';
+-------------------------------+-------------+
| Variable_name                 | Value       |
+-------------------------------+-------------+
| Qcache_free_blocks            | 1           |
| Qcache_free_memory            | 1038376 |
| Qcache_hits                   | 0           |
| Qcache_inserts                | 1           |
| Qcache_lowmem_prunes  | 0           |
| Qcache_not_cached             | 0           |
| Qcache_queries_in_cache | 1           |
| Qcache_total_blocks           | 4           |
+-------------------------------+-------------+
```

Counter variables after 2nd Identical query

```
mysql> select * from employees where employee_id = 5;
+---------------+----------+------+-----------+
| employee_id | ename | d_id | salary     |
+---------------+----------+------+-----------+
|                5 | jones    |     3 | 5000.00 |
+---------------+----------+------+-----------+

mysql>  SHOW STATUS LIKE 'Qcache%';
+-------------------------------+-------------+
| Variable_name                 | Value       |
+-------------------------------+-------------+
| Qcache_free_blocks            | 1           |
| Qcache_free_memory            | 1038376 |
| Qcache_hits                   | 1           |   <--- Cache is hit
| Qcache_inserts                | 1           |
| Qcache_lowmem_prunes          | 0           |
| Qcache_not_cached             | 0           |
| Qcache_queries_in_cache       | 1           |
| Qcache_total_blocks           | 4           |
+-------------------------------+-------------+
```

Selective Caching

```
// Using SQL_NO_CACHE means "don't cache this query" when
// query_cache_type is set to DEMAND
mysql> select SQL_NO_CACHE * from employees where employee_id = 1;
+-------------+-------+------+---------+
| employee_id | ename | d_id | salary  |
+-------------+-------+------+---------+
|           1 | jack  |    1 | 3000.00 |
+-------------+-------+------+---------+
1 row in set (0.00 sec)

// Using SQL_CACHE means “cache this query” when
// query_cache_type is set to DEMAND
mysql> select SQL_CACHE * from employees where employee_id = 1;
+-------------+-------+------+---------+
| employee_id | ename | d_id | salary  |
+-------------+-------+------+---------+
|           1 | jack  |    1 | 3000.00 |
+-------------+-------+------+---------+
1 row in set (0.00 sec)
```

Indexing:
Overview

What is and Why Indexing?

- A logically separate table that contains sorted values of a column
  - You can have multiple indexes for multiple columns of a table
- Prevents a full table scan
  - Without indexing, a full table scan might be needed to find a match
- Inserting or deleting a row from an indexed column is more efficient
  - Since you do not have to move around the rows with full number of columns

Indexing in Joined Query

- Indexes are even more valuable when you're running queries involving joins on multiple tables
  - In a single-table query, the number of values you need to examine per column is the number of rows in the table
  - In a multiple-table query, the number of possible combinations exponentially increases because it's the product of the number of rows in all tables involved

Where Indexing is Used in MySQL

- To speed up searches for rows matching terms of a WHERE clause or rows that match rows in other tables when performing joins
- For queries that use the MIN() or MAX() functions, the smallest or largest value in an indexed column can be found quickly without examining every row.
- MySQL can often use indexes to perform sorting and grouping operations quickly for ORDER BY and GROUP BY clauses

Indexing:
Example Query

Example Query Case

- Suppose we have 3 tables, t1, t2, t3
  - Each table contains a column c1, c2, and c3
  - Each table contains 100 rows
- Suppose we have the following query
  - SELECT t1.i1, t2.i2, t3.i3
  - FROM t1, t2, t3
  - WHERE t1.c1 = t2.c2 AND t2.c1 = t3.c3;

When tables are NOT Indexed

- The possible number of rows to scan is
  - 100 * 100 * 100 = 1 million
- It is because there is no knowing which row of a table contains which value without scanning them all. So in order to find the one that matches the WHERE clause, you might have to read all rows of all three tables
  - 100 * 100 * 100 = 1 million

When tables are Indexed

- The number of rows to scan is
  - 100 for the first table, t1, and 1 in the 2nd table, t2, and 1 in the 3rd table, t3
- Using indexes of t2 and t3, for each row of t1, you can find quickly find which one in t2 and t3 are matches

Indexing:
Strategy

Which columns to Index?

- Index columns that you use for searching, sorting, or grouping
  - Columns that appear in WHERE clause
  - Columns named in JOIN clauses
  - Columns that appear in ORDER BY or GROUP BY clauses
- Do not index columns that appear only in the output column list following the SELECT keyword

Example: Indexing Strategy

```
SELECT
    c1                      /* t1.c1 is not a good candidate for indexing */
FROM
    t1 LEFT JOIN t2
    ON t1.c2 = t2.c3    /* t1.c2 and t2.c3 are good candidate for indexing */
WHERE
    c4 = 2000;              /* t1.c4 is a good candidate for indexing */
```

Column Cardinality

- The cardinality of a column is the number of distinct values that it contains
  - For example, a column that contains the values 1, 3, 7, 4, 7, 3 has a cardinality of four (1, 3, 4, 7).
- Indexes work best for columns that have a high cardinality relative to the number of rows in the table (that is, columns that have many unique values and few duplicates).

Indexing:
Runtime counters

Indexing Runtime Counters

- Handler_read_first:
  - Number of times the first entry was read from an index. If this is high, it suggests that the server is doing a lot of full index scans.
- Handler_read_key:
  - Number of requests to read a row based on a key. If this is high, it is a good indication that your queries and tables are properly indexed.
- Handler_read_rnd:
  - Number of requests to read a row based on a fixed position. This will be high if you are doing a lot of queries that require sorting of the result.

EXPLAIN

What is EXPLAIN?

- Used to obtain information about how MySQL executes a SELECT statement
- When you precede a SELECT statement with the keyword EXPLAIN, MySQL displays information from the optimizer about the query execution plan. That is, MySQL explains how it would process the SELECT, including information about how tables are joined and in which order.

What is EXPLAIN?

- With the help of EXPLAIN, you can see where you should add indexes to tables to get a faster SELECT that uses indexes to find rows.
- You can also use EXPLAIN to check whether the optimizer joins the tables in an optimal order.

EXPLAIN Fields

- table
  - Indicates which table the output is about (for when you join many tables in the query)
- type
  - Indicates which type of join is being used. From best to worst the types are: system, const, eq_ref, ref, range, index, all
- possible_keys
  - Indicates possible indexes that can be used by MySQL
- key
  - Indicates which index (among the ones in possible_keys) is actually used

EXPLAIN Fields (Continued)

- key_len
  - The length of the key used. The shorter the better.
- ref
  - Tells which column, or a constant, is used
- rows
  - Number of rows MySQL believes it must examine to get the data
  - With indexing, smaller number is expected
- extra Extra info
  - The bad ones to see here are "using temporary" and "using filesort"

EXPLAIN for non-indexed Query

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employees
         type: ALL
possible_keys: d_id
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 5
        Extra: Using where
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: departments
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4
        Extra: Using where; Using join buffer
2 rows in set (0.00 sec)
```

Note that without indexing, the number of rows MySQL is expected to read is the total number of rows of each table, 5 for the employees table and 4 for the departments table.

EXPLAIN for Indexed Query

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employees
         type: ref
possible_keys: ename,d_id
          key: ename
      key_len: 257
          ref: const
         rows: 1
        Extra: Using where
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: departments
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: mydb.employees.d_id
         rows: 1
        Extra:
2 rows in set (0.05 sec)
```

Note that with indexing, the number of rows MySQL is expected to read is 1 for both employees and departments tables.

Slow Query Log

Slow Query Log

- MySQL has a feature that allows you to log slow running queries to a file
- These queries are the candidates for optimization

Configuring Slow Query Log

```
mysql> SHOW VARIABLES LIKE '%long_query%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)

mysql> SET GLOBAL slow_query_log=ON;
Query OK, 0 rows affected (0.13 sec)

mysql> SHOW VARIABLES LIKE '%slow%';
+--------------------+-----------------------------------------------------
| Variable_name      | Value
+--------------------+-----------------------------------------------------
| log_slow_queries   | ON
| slow_launch_time   | 2
| slow_query_log     | ON
| slow_query_log_file | C:\ProgramData\MySQL\MySQL Server 5.1\Data\ShinLaptop-slow.log |
+--------------------+-----------------------------------------------------
4 rows in set (0.03 sec)
```

Configuring Slow Query Log

```
C:\Users\sang>type "C:\ProgramData\MySQL\MySQL Server 5.1\Data\ShinLaptop-slow.log"
C:\Program Files (x86)\MySQL\MySQL Server 5.1\bin\mysqld, Version: 5.1.44-community
(MySQL Community
 Server (GPL)). started with:
TCP Port: 3306, Named Pipe: MySQL
Time                 Id Command    Argument
# Time: 100329 15:15:08
# User@Host: root[root] @ localhost [127.0.0.1]
# Query_time: 0.040000  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0
use mydb;
SET timestamp=1269890108;
DROP TABLE IF EXISTS employees;
# Time: 100329 15:15:09
# User@Host: root[root] @ localhost [127.0.0.1]
# Query_time: 0.030000  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0
SET timestamp=1269890109;
DROP TABLE IF EXISTS departments;
# User@Host: root[root] @ localhost [127.0.0.1]
# Query_time: 0.100000  Lock_time: 0.000000 Rows_sent: 0  Rows_examined: 0
SET timestamp=1269890109;
...
```

Thank you!

Sang Shin
http://www.javapassion.com
“Learning is fun!”