Source : https://cryptkcoding.com/blog/2012/04/06/how-to-optimize-mysql-join-queries-through-indexing/

How to optimize MySQL JOIN queries through indexing

Posted on April 6, 2012 by cryptk

This is a pretty broad topic, and there is no way that I will be able to cover every facet of using indexes in your schema and queries in anything short of a small book, but the first step is getting your feet wet. I will be covering one of the most common causes for queries to run slowly, lack of indexes. This can cause simple queries to run slowly, but it has an exponentially increasing negative impact on performance when used on more complex queries, such as ones that use joins. In order to grasp this topic, it would be extremely helpful if you already have a basic grasp of how to use MySQL. Topics such as creating tables, and running queries should be pretty easy for you already. If you expand your basic grasp with some information on how to optimize those queries using indexing, continue on after the break.First, lets cover our sample dataset. On this MySQL server we have a database named training, and within it are two tables, one named `user_auth_1000` and one named `user_meta_1000`. The auth table stores an id which is setup with AUTO_INCREMENT, a username and an encrypted password, the meta table has columns for a user ID number, a real name, a city and a secret word. Here is the creation syntax for each:

user_auth_1000:

```
CREATE TABLE `user_auth_1000` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` char(50) DEFAULT NULL,
  `password` char(41) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=1001 DEFAULT CHARSET=latin1
```

user_meta_1000:

```
CREATE TABLE `user_meta_1000` (
  `uid` int(11) NOT NULL DEFAULT '0',
  `real_name` varchar(50) DEFAULT NULL,
  `city` varchar(50) DEFAULT NULL,
  `secret_word` varchar(50) DEFAULT NULL
) ENGINE=MyISAM AUTO_INCREMENT=1001 DEFAULT CHARSET=latin1
```

The actual content of the rows is just random words, but that isn't important as this is just a sample dataset, it could easily be real world data and these concepts would still ring true. What is important is that both tables have 1000 rows of data. For this setup I have the query cache in mysql completely disabled. Having the query cache enabled would mask the problem of improper indexing for the most part, but it would not solve the problem, merely cover up some of the symptoms. Having it enabled though would lead to requiring allocating a potentially excessive amount of RAM towards that cache though, which is obviously less than optimal, that RAM could be much better used in other places (such as running your code).

Here is the query that we will be using throughout this walkthrough:

```
SELECT auth.username, auth.password, meta.secret_word FROM user_auth_1000 aut
h LEFT JOIN user_meta_1000 meta ON auth.id = meta.uid;
```

It is a pretty basic join query, but is is more than enough to show the problems. First lets see how long it takes to run that query as the table schema's sit now…

```
1000 rows in set (0.28 sec)
```

0.28 seconds, only a quarter of a second, that's pretty quick… or is it… Many people who are new to MySQL do not realize that database queries have a different concept of time than most people. While 0.28 seconds is very fast for you and I, in the MySQL world, that may as well be a year. To drive this point home, I set up a simple PHP script which just runs this query against MySQL and displays the output. Lets use apache-bench to hit that page with 100 concurrent connections for a total of 1000 connections.

```
Concurrency Level:      100
Time taken for tests:   73.415 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      197780000 bytes
HTML transferred:       197588000 bytes
Requests per second:    13.62 [#/sec] (mean)
Time per request:       7341.460 [ms] (mean)
Time per request:       73.415 [ms] (mean, across all concurrent requests)
Transfer rate:          2630.87 [Kbytes/sec] received
```

```
Connection Times (ms)
min  mean[+/-sd] median   max
Connect:        0    1   1.7      0       6
Processing:   391 7226 1653.0   7313   12608
Waiting:      387 7222 1652.9   7310   12605
Total:        397 7226 1652.3   7313   12613

Percentage of the requests served within a certain time (ms)
50%   7313
66%   7450
75%   7591
80%   7696
90%   8123
95%  10361
98%  11539
99%  11835
100%  12613 (longest request)
```

It took over a minute to serve this very simple page 1000 times. And trust me when I tell you that 100 concurrent connections is not much at all, Apache and MySQL should be able to handle this with ease. What's even worse is that during this test, MySQL was using over 370% CPU… that's a lot considering that this machine is a quad-core, so the max is 400%… doesn't leave much for running my code…

MySQL is a very fast database system, when you use it properly, so, lets see if we can find out what is going wrong here… First, we are going to profile the query to get some time information. At a MySQL prompt we will run the following:

```
mysql> SET PROFILING=1;
mysql> SELECT auth.username, auth.password, meta.secret_word FROM user_auth_1
000 auth LEFT JOIN user_meta_1000 meta ON auth.id = meta.uid;
mysql> SHOW PROFILE FOR QUERY 1;
```

The output of that last line will be something like the following:

```
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| starting             | 0.000117 |
| Opening tables       | 0.000045 |
| System lock          | 0.000010 |
| Table lock           | 0.000013 |
| init                 | 0.000033 |
| optimizing           | 0.000009 |
| statistics           | 0.000022 |
| preparing            | 0.000018 |
| executing            | 0.000006 |
| Sending data         | 0.279235 |
| end                  | 0.000017 |
| query end            | 0.000007 |
| freeing items        | 0.000051 |
| logging slow query   | 0.000005 |
| cleaning up          | 0.000006 |
+----------------------+----------+
```

We can see that 0.27 seconds were spent sending the data. This is a little misleading. The executing portion is only the time it takes for mysql to execute the query, and it returns the data as it gets it. So the 0.27 seconds in "Sending data" is the amount of time it took to both retrieve the data from the database and deliver it to the requester. It's a little more complex than that, but that's the gist of it.

Now that we know where the time is being spent, lets see if we can figure out why it is taking so much time there… for this we will use EXPLAIN:

```
mysql> EXPLAIN SELECT auth.username, auth.password, meta.secret_word FROM use
r_auth_1000 auth LEFT JOIN user_meta_1000 meta ON auth.id = meta.uid;
```

Which gives the following output:

```
+----+-------------+-------+------+---------------+------+---------+------+--
----+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | r
ows | Extra |
+----+-------------+-------+------+---------------+------+---------+------+--
----+-------+
|  1 | SIMPLE      | auth  | ALL  | NULL          | NULL | NULL    | NULL | 1
000 |       |
|  1 | SIMPLE      | meta  | ALL  | NULL          | NULL | NULL    | NULL | 1
000 |       |
+----+-------------+-------+------+---------------+------+---------+------+--
----+-------+
```

We can see from this, that there are no indexes being used! In this case, it's not surprising, if you look back at the create table syntax, there aren't even any indexes in place. The really important thing to understand here, and the first part of the EXPLAIN output that I always look at, is the rows column. This tells you how many rows were considered when running the query. both the query against the auth table, as well as the query against the meta table have 1000 rows considered. The really important thing to realize here is that for every row considered in the auth table, 1000 rows in the meta table are being considered. Using the following formula we can determine the total number of row considerations:

```
(table1rows * table2rows) + table1rows
```

which in this case is:

```
(1000 * 1000) + 1000 = 1,001,000
```

MySQL is considering over 1 million rows to run this query. All of a sudden the speed of MySQL really shows up… It reviewed over 1 million pieces of data in about a quarter of a second… how long would it take you to do the same.

Lets see if we can give MySQL a helping hand though. The important thing to know about indexing a JOIN query is that the columns that the tables are joined on need to be indexed in order to prevent this huge increase in row considerations. In this case, the columns that the JOIN is being performed on are the id column on the auth table and the uid column on the meta table. Lets take a look at the keys that are in place on those tables:

user_auth_1000:

```
mysql> show indexes from user_auth_1000;
+----------------+------------+----------+--------------+-------------+------
-----+-------------+----------+--------+------+------------+---------+
| Table          | Non_unique | Key_name | Seq_in_index | Column_name | Colla
tion | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+----------------+------------+----------+--------------+-------------+------
-----+-------------+----------+--------+------+------------+---------+
| user_auth_1000 |          0 | PRIMARY  |            1 | id          |
A         |        1000 |     NULL | NULL   |      | BTREE      |         |
+----------------+------------+----------+--------------+-------------+------
-----+-------------+----------+--------+------+------------+---------+
```

user_meta_1000:

```
mysql> show indexes from user_meta_1000;

Empty set (0.00 sec)
```

So we are all covered on the auth table, it has an index already in place on the id table, but there are no indexes on the meta table. Lets create one now:

```
mysql> create index `uid` on user_meta_1000(`uid`);

Query OK, 1000 rows affected (0.02 sec)

Records: 1000  Duplicates: 0  Warnings: 0
```

Now that we have that index in place, lets check out that EXPLAIN again:

```
mysql> EXPLAIN SELECT auth.username, auth.password, meta.secret_word FROM use
r_auth_1000 auth LEFT JOIN user_meta_1000 meta ON auth.id = meta.uid;

+----+-------------+-------+------+---------------+------+---------+---------
---------+------+-------+

| id | select_type | table | type | possible_keys | key  | key_len | re
f               | rows | Extra |

+----+-------------+-------+------+---------------+------+---------+---------
---------+------+-------+

|  1 | SIMPLE      | auth  | ALL  | NULL          | NULL | NULL    | NUL
L              | 1000 |       |

|  1 | SIMPLE      | meta  | ref  | uid           | uid  | 4       | training
.auth.id |    1 |       |

+----+-------------+-------+------+---------------+------+---------+---------
---------+------+-------+
```

And now we will re-run the profile to see the performance realized!

```
mysql> SET PROFILING=1;

mysql> SELECT auth.username, auth.password, meta.secret_word FROM user_auth_1
000 auth LEFT JOIN user_meta_1000 meta ON auth.id = meta.uid;

mysql> SHOW PROFILE FOR QUERY 1;
```

And the output of that is:

```
+--------------------+----------+

| Status             | Duration |

+--------------------+----------+

| starting           | 0.000100 |

| Opening tables     | 0.000024 |
| System lock        | 0.000009 |
| Table lock         | 0.000014 |
| init               | 0.000032 |
| optimizing         | 0.000008 |
| statistics         | 0.000032 |
| preparing          | 0.000019 |
| executing          | 0.000006 |
| Sending data       | 0.007865 |
| end                | 0.000007 |
| query end          | 0.000007 |
| freeing items      | 0.000040 |
| logging slow query | 0.000005 |
| cleaning up        | 0.000005 |
+--------------------+----------+
```

Wow, that really sped things up! Before we were spending 0.27 seconds in sending data, now it is only 0.007 seconds! And finally, lets hit that page with apache-bench again using the exact same setup as before, but this time the query is indexed.

```
Concurrency Level:      100
Time taken for tests:   4.337 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      197780000 bytes
HTML transferred:       197588000 bytes
Requests per second:    230.55 [#/sec] (mean)
Time per request:       433.747 [ms] (mean)
Time per request:       4.337 [ms] (mean, across all concurrent requests)
Transfer rate:          44529.33 [Kbytes/sec] received

Connection Times (ms)
min  mean[+/-sd] median   max
Connect:        0    1   2.9      0      11
Processing:    15  414  80.8    425     643
```

```
Waiting:       13  410  80.7    421     641
Total:         21  415  78.7    425     644


Percentage of the requests served within a certain time (ms)
50%    425
66%    440
75%    450
80%    459
90%    484
95%    507
98%    533
99%    553
100%    644 (longest request)
```

We served all 1000 requests in under 5 seconds, if you remember from earlier it took 73 seconds before we indexed that query. Our page which was taking about 7.3 seconds to load earlier when under the load of 100 concurrent connections now loads in about a half a second in the same time period!

I do plan to write more posts on the subject of tuning up mysql, this one is great if you already know what your problem queries are, the next one in the series will be on how to find those problem queries. Stay tuned…