CO527 Advanced Database Systems

S.THINESH
E/15/366

Introduction

When a SELECT query does not perform well as you think it should, use the EXPLAIN statement to ask the MySQL server for information about how the query optimizer processes the query. When you issue a query, the MySQL Query Optimizer tries to devise an optimal plan for query execution. EXPLAIN is one of the most powerful tools available for understanding and optimizing inefficient queries.You can use EXPLAIN by placing the keyword EXPLAIN in front of SELECT, DELETE, INSERT, REPLACE, and UPDATE query as of MySQL5.6.3. Earlier versions permit only SELECT statements for this. You can also add the keyword EXTENDED after EXPLAIN in your query for additional information about the execution plan.

Troubleshooting with EXPLAIN

1.

Consider a case where you are running the following SQL queries from a database without an index:

1. SELECT * FROM departments WHERE deptname = ’Finance’;

The above queries will force MySQL server to conduct a full table scan (start to finish) to retrieve the record that we are searching.

Luckily, MySQL has a special '**EXPLAIN**' statement that you can use alongside select, delete, insert, replace and update statements to analyze your queries.

Once you append the query before an SQL statement, MySQL displays information from the optimizer about the intended execution plan.

If we run the above SQL one more time with the explain statement, we will get a full picture of what MySQL will do to execute the query:

```
mysql> explain SELECT * FROM DEPARTMENTS WHERE dept_name='Finance';
+----+-------------+-------------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table       | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | DEPARTMENTS | ALL  | NULL          | NULL | NULL    | NULL |    9 | Using where |
+----+-------------+-------------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

As you can see in the first query, the optimizer has displayed very important information that can help us to fine-tune our database table. First, it is clear that MySQL will conduct a full table scan because key column is '**NULL**'. Second, MySQL server has clearly indicated that it's going to conduct a full scan on the 9 rows in our database.

2. SELECT * FROM departments WHERE dept_no = 'd002';

We can clearly see that the number of rows to scan will be 1.Once you append the query before an SQL statement, MySQL displays information from the optimizer about the intended execution plan.

If we run the above SQL one more time with the explain statement, we will get a full picture of what MySQL will do to execute the query:

```
mysql> explain SELECT * FROM DEPARTMENTS WHERE dept_no='d002';
+----+-------------+-------------+-------+---------------+---------+---------+-------+------+-------+
| id | select_type | table       | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+----+-------------+-------------+-------+---------------+---------+---------+-------+------+-------+
|  1 | SIMPLE      | DEPARTMENTS | const | PRIMARY       | PRIMARY | 4       | const |    1 | NULL  |
+----+-------------+-------------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.00 sec)
```

The optimizer has displayed very important information that can help us to fine-tune our database table. First, it is clear that MySQL will scan table by using key column is '**PRIMARY**'. Second, MySQL server has clearly indicated that it's going to conduct a scan on the 1 rows in our database.

2.

I. create table emplist select emp_no, first_name from employees;

II. create table titleperiod select emp_no, title, datediff(to_date, from_date) as period FROM titles;

The following SQL query will find the details of employees ” Which employees have worked for more than 4000 days for an assigned title”.

```
SELECT first_name, title, period
        FROM emplist JOIN titleperiod ON emplist.emp_no = titleperiod.emp_no
        WHERE period > 4000;
```

```
EXPLAIN SELECT first_name, title, period
        FROM emplist JOIN titleperiod ON emplist.emp_no = titleperiod.emp_no
        WHERE period > 4000;
```

```
mysql> EXPLAIN SELECT first_name,title,period
    -> from emplist join titleperiod on emplist.emp_no =titleperiod.emp_no
    -> where period > 4000;
+----+-------------+-------------+------+---------------+------+---------+------+--------+----------------------------------------------------+
| id | select_type | table       | type | possible_keys | key  | key_len | ref  | rows   | Extra                                              |
+----+-------------+-------------+------+---------------+------+---------+------+--------+----------------------------------------------------+
|  1 | SIMPLE      | emplist     | ALL  | NULL          | NULL | NULL    | NULL | 299425 | NULL                                               |
|  1 | SIMPLE      | titleperiod | ALL  | NULL          | NULL | NULL    | NULL | 442146 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------------+------+---------------+------+---------+------+--------+----------------------------------------------------+
2 rows in set (0.02 sec)
```

The first thing we notice, is that it can include more than one row. The query we're analyzing involves two tables in the process, which are joined using an inner join. Each of these tables gets represented in a different row in the execution plan above. As an analogy to the coding world, you can look at the concept of an inner join as very similar to a nested loop. MySQL chooses the table it thinks will be best to start the journey with (the outer "loop") and then touches the next table using the values from the outer "loop".

What can we learn from this specific EXPLAIN plan?

- MySQL chooses to start with the emplist table. The EXPLAIN output shows that it has to go through 299425 rows, which is about all the rows in the table. That's nasty.
- The Extra column indicates that MySQL tries to reduce the amount of rows it inspects is null.
- Looking at the WHERE clause, we can see there is a condition, *period>4000*, which looks very selective and should drastically reduce the amount of rows to inspect. The titleperiod table contains 443308 records, while only 442146 rows are returned when applying the condition. which means that it's very selective. In this case, it would be best if MySQL would have started the execution using the titleperiod table, to take advantage of this selective condition.
- Looking at the possible_keys values for both tables, we can see that MySQL doesn't have a lot of optional indexes to choose from. More precisely, there is no index in the possible_keys that contains the columns mentioned in the WHERE clause.

Therefore, we'll add the following index. Index starts with the column mentioned in the WHERE clause. The index for the titleperiod table also includes the joined column.

ALTER TABLE emplist ADD PRIMARY KEY(emp_no);

CREATE INDEX emp_index ON titleperiod(emp_no);

```
mysql> EXPLAIN SELECT first_name,title,period
    -> from emplist join titleperiod on emplist.emp_no =titleperiod.emp_no
    -> where period > 4000;
+----+-------------+-------------+------+---------------+---------+---------+-----------------------+--------+-------------+
| id | select_type | table       | type | possible_keys | key     | key_len | ref                   | rows   | Extra       |
+----+-------------+-------------+------+---------------+---------+---------+-----------------------+--------+-------------+
|  1 | SIMPLE      | emplist     | ALL  | PRIMARY       | NULL    | NULL    | NULL                  | 299918 | NULL        |
|  1 | SIMPLE      | titleperiod | ref  | emp_ind       | emp_ind | 4       | company.emplist.emp_no |      1 | Using where |
+----+-------------+-------------+------+---------------+---------+---------+-----------------------+--------+-------------+
2 rows in set (0.00 sec)
```

What changed?

- The first we start with the titleperiod table. It uses the new index to filter out the rows and estimates to filter all but 1 records, which are then joined to the second table, *emplist*
- The second change we see, by looking at the *key* column, is that indexes are used for lookups and filtering in both tables.
- By looking at the key_len column, we can see that the composite index for the *titleperiod* table is used in full - 4 bytes, which covers both the columns.
- The last important change we see is the amount of rows MySQL estimates it needs to inspect in order to run evaluate the query. It estimates it needs to inspect 299918 * 1 = 299918 rows, which is great, comparing to the millions of records in the original execution path.

So looking at the execution duration now, we can see a drastic improvement, from a very slow query which never actually returned.

References:

https://www.eversql.com/mysql-explain-example-explaining-mysql-explain-using-sta ckoverflow-data/

https://dev.mysql.com/doc/refman/5.7/en/using-explain.html

https://dzone.com/articles/how-to-optimize-mysql-queries-for-speed-and-perfor