Query Optimization

E/15/202

Introduction

Query optimization is a very important part when executing queries. When a SELECT query do not perform as good as we expect we can ask the MYSQL server for more information about how the query optimizer performs this. For that we use the EXPLAIN key word with the SELECT query. This keyword is helpful in understanding inefficient queries. By understanding the inefficiencies in the query we can take measures to make them perform well. It may include indexes, primary keys or etc. We can use EXPLAIN keyword by placing it in front of SELECT, DELETE, INSERT, REPLACE and UPDATE as of MYSQL 5.6.3. Earlier versions had permitted only SELECT. For additional information we can use EXTEND keyword after the EXPLAIN keyword.

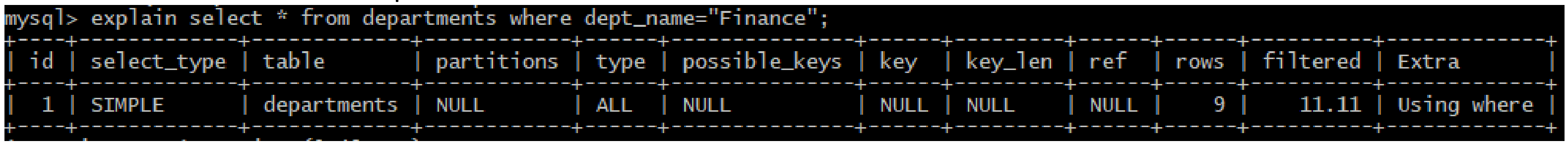
Troubleshooting with EXPLAIN

To understand how the EXPLAIN key word, let’s look at it with some simple examples. Consider the two queries below. (In this example I have used the database *Company* which we created on the previous lab)

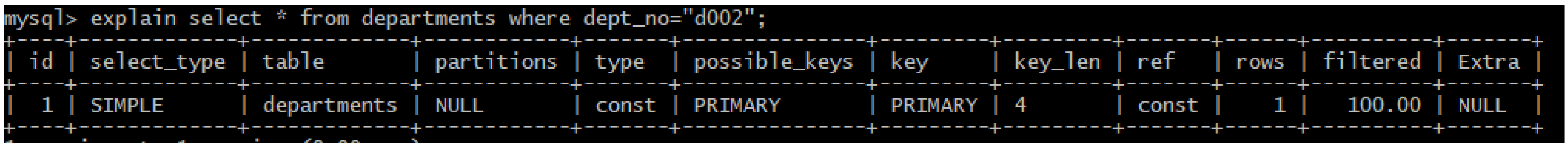
Example :-

1) SELECT * FROM departments WHERE dept_name = ’Finance’;
2) SELECT * FROM departments WHERE dept_no = ’d002’;

The above two queries use only one table access. The output of using EXPLAIN keyword on the above two queries is as follows.



```
mysql> explain select * from departments where dept_name="Finance";
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table       | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | departments | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    9 |    11.11 | Using where |
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------------+
```

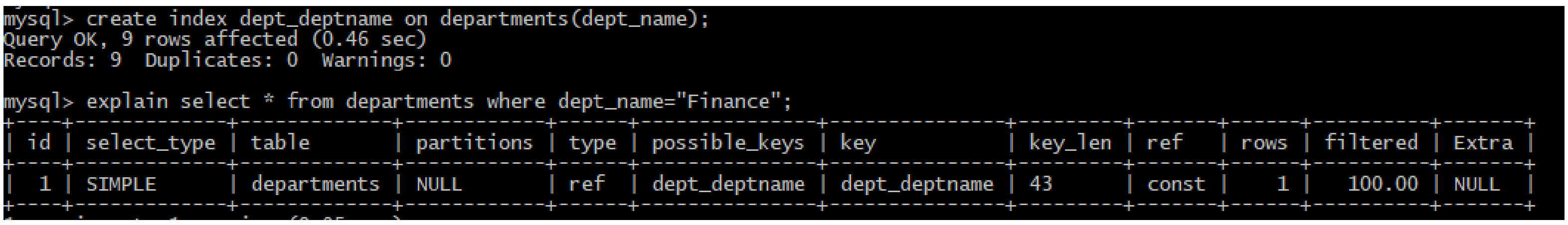


```
mysql> explain select * from departments where dept_no="d002";
+----+-------------+-------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table       | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | departments | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | NULL  |
+----+-------------+-------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
```

Both queries are simple (no any sub queries or unions). When the *type* of first query is ALL the *type* of second query is const. That means, the first query has to scan the entire table which has about 9 rows. MYSQL estimates using where clause in first query reduce this amount by 11.11%, which is still bad.

For the second query MYSQL uses the 4 bytes (key_len is 4) from PRIMARY KEY (key), which is the *dept_no*. So it has to examine only one row and the filtered percentage is 100%. So it is much more efficient than the first query.

As you can see the second query is in its’ optimized version. So we are only left with the first query to optimize. To optimize it I choose to make an index on dept_name. Because dept_name is in the WHERE clause, so making and index on it can make a big difference.



```
mysql> create index dept_deptname on departments(dept_name);
Query OK, 9 rows affected (0.46 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> explain select * from departments where dept_name="Finance";
+----+-------------+-------------+------------+------+---------------+---------------+---------+-------+------+----------+-------+
| id | select_type | table       | partitions | type | possible_keys | key           | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------------+------------+------+---------------+---------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | departments | NULL       | ref  | dept_deptname | dept_deptname | 43      | const |    1 |   100.00 | NULL  |
+----+-------------+-------------+------------+------+---------------+---------------+---------+-------+------+----------+-------+
```

Now the first query uses the newly created index on dept_name (key field shows the used index). So type field has changed to ref which is not scanning the whole table. So it has upgraded. Now the filtered percentage has also increased.

But since this table has only small number of rows (only 9 rows) these changes is not much of an effect. Both the queries can perform the task in a very short amount of time.
So what we can conclude from these observations is that creating index on an attribute which is used in the WHERE clause can make the query more optimized. But these do not show much of an effect if the number of rows in a table are very less.

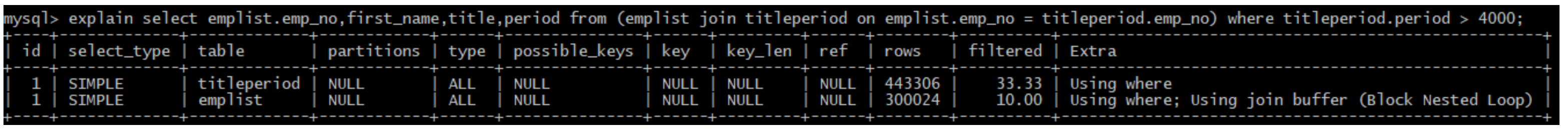
Using JOIN operation can increase the amount of server processing. In that case EXPLAIN command is especially important. Let's look at it with a small example. (used the previous database *Company*)
Example:-
We want to display the employees who have worked for more than 4000 days for an assigned title. In this we display their employee number, first name, title and the period they worked. So first, we create two tables as below.

1) CREATE TABLE emplist SELECT emp_no, first_name FROM employees;
2) CREATE TABLE titleperiod SETECT emp_no, title, datediff(to_date, from_date) AS period FROM titles;

Note that the two tables are in their initial unindexed state.
Now these two tables are joined with the employee number to get the final quey.

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

Let's use the EXPLAIN keyword to analyze this query.



```
mysql> explain select emplist.emp_no,first_name,title,period from (emplist join titleperiod on emplist.emp_no = titleperiod.emp_no) where titleperiod.period > 4000;
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
| id | select_type | table       | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                              |
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
|  1 | SIMPLE      | titleperiod | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 443306 |    33.33 | Using where                                        |
|  1 | SIMPLE      | emplist     | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 300024 |    10.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
```

The above query does not have any sub queries or unions. It has only one query, so the id for both rows is one.
This query uses two tables, so using EXPALIN sows these two tables in two rows.
The two rows are performed as simple queries as they do not include any sub queries or unions. So the select_type is SIMPLE.
From the 2 joined tables MYSQL has chosen the table it thinks will be best to start the journey with (which is *titleperiod* in this case) and then go to the next table (which is *emplist*) using the values obtained from first table.
For both the 2 tables the *type* field is ALL. That means both of them have to go through all the rows which are included in their tables.

MYSQL has not found any possible keys that can be used in each of the table. It happens because we have not defined any primary keys in both of the tables when creating them.
So there are no any keys to use in this particular query.
Rows field show the number of rows in each of the tables. MYSQL has to check **300024*443306** numbers of rows in each and every table.
It shows that although it uses where clause to reduce the number of rows in titleperiod table still the estimated percentage is 33.33%, which is bad. And in the emplist estimated percentage is also very bad. It is 10%, although it uses the where clause and the join operation to reduces the number of rows.
In this query MYSQL has to inspect a large number of rows, which makes the performance really bad.

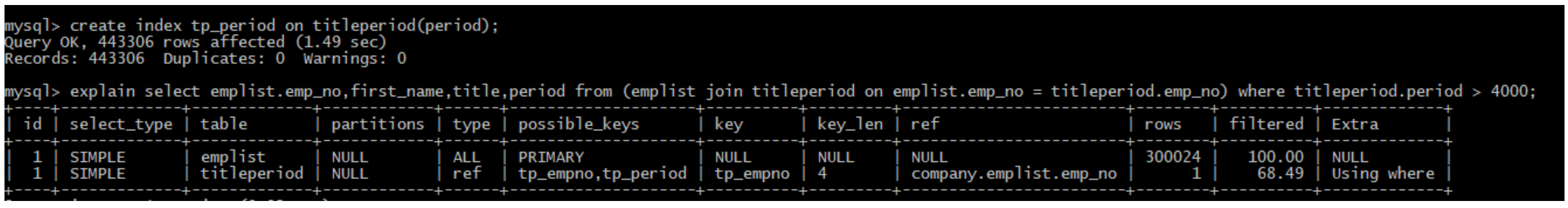
By looking at the way how query optimizer works we can take measures to minimize the processing time. That is making indexes. Creating indexes on columns that are used for searching, grouping or sorting will make our query more optimized. The above query does not include any GROUP BY or ORDER BY clauses. But it does use columns for searching.

It uses emplist.emp_no and titleperiod.emp_no to match record between tables and titleperiod.period to cut down records that do not satisfy the condition.
In the emplist table, emp_no can be used as a primary key because it uniquely identifies each row. But In the titleperiod table, emp_no must be a non-unique index because multiple employees can share the same title.

So we can make the emp_no column of emplist table as the PRIMARY KEY.
ALTER TABLE emplist add PRIMARY KEY(emp_no);

We can make two indexes on columns emp_no and period of titleperiod table. Beacause those two columns are used for searching.
CREATE INDEX tp_empno ON titleperiod(emp_no);
CREATE INDEX tp_period ON titleperiod(period);

After adding those changes let’s use EXPLAIN on the query.



```
mysql> create index tp_period on titleperiod(period);
Query OK, 443306 rows affected (1.49 sec)
Records: 443306  Duplicates: 0  Warnings: 0

mysql> explain select emplist.emp_no,first_name,title,period from (emplist join titleperiod on emplist.emp_no = titleperiod.emp_no) where titleperiod.period > 4000;
+----+-------------+-------------+------------+------+----------------------+----------+---------+------------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type | possible_keys        | key      | key_len | ref                    | rows   | filtered | Extra       |
+----+-------------+-------------+------------+------+----------------------+----------+---------+------------------------+--------+----------+-------------+
|  1 | SIMPLE      | emplist     | NULL       | ALL  | PRIMARY              | NULL     | NULL    | NULL                   | 300024 |   100.00 | NULL        |
|  1 | SIMPLE      | titleperiod | NULL       | ref  | tp_empno,tp_period   | tp_empno | 4       | company.emplist.emp_no |      1 |    68.49 | Using where |
+----+-------------+-------------+------------+------+----------------------+----------+---------+------------------------+--------+----------+-------------+
```

As you can see it has changed the order of tables to be used. But in the above case when the titleperiod is used first it has to check 443306 rows. But by choosing this, it’s only 300024 rows.
Although we have created two indexes as tp_empno and tp_period only the tp_empno was used in this case. But it shows when using WHERE clause to reduce the number of rows the estimated percentage has gone higher (68.49%). So creating the index on tp_period has actually taken in to effect in optimizing this query.
Not only that but also the type filed of titleperiod has changed as ref which is a total upgrade rather than the scanning the whole table.

And adding PRIMARY KEY on emplist table has changed the filtered percentage to 100% although it has to scan the entire table.
So now the query only has to check only 300024*1 rows which is an improvement.
And now this query is somewhat more effective than previous one.

Query Rewriting Techniques

EXPLAIN keyword help us to get an idea of the measurers that we can take to make our query optimize. After the modifications we can again use the EXPLAIN command to see changes it had made. But sometimes the indexes that we have created were not included by the MYSQL query optimizer. If we really want to include those indexes, MYSQL allows us to force them to the query. Not only that but also we can tell MYSQL to ignore an index.
For example:-
EXPLAIN SELECT * FROM employees **FORCE INDEX(fname_index)** ORDER BY first_name ASC;
EXPLAIN SELECT * FROM departments **IGNORE INDEX(PRIMARY)** WHERE dept_no = ’d002’;

Other than FORCE or IGNORE we can force the MYSQL to join tables in a particular order, begin the query with SELECT STARIGHT_JOIN rather than SELECT. This instruction tells the MYSQL query optimizer to join the tables from left to right in the order they are listed in the query.

Conclusion

As with the above examples the EXPLAIN keyword has been very helpful in taking decisions about the performance of the queries. The different fields in the table given, when using EXPLAIN on those queries, shows how the indexes and primary keys are being used. And we can get an idea of how the MYSQL scan the tables in the query, whether it has to scan the entire table or a part of it. So we can take measures to reduce the amount of rows it has to scan, in order to reduce the processing time. And it shows an estimated filtered percentage when it uses the extra information to reduce the number of rows. Most importantly, MYSQL do not take a long time to show all of this information. If there were no such keyword, then we have to look at the processing time after the execution of a query. Which is a very impractical way, when there are large number of rows in a table, and the processing time is very high. But having this one keyword has made our work very easy.