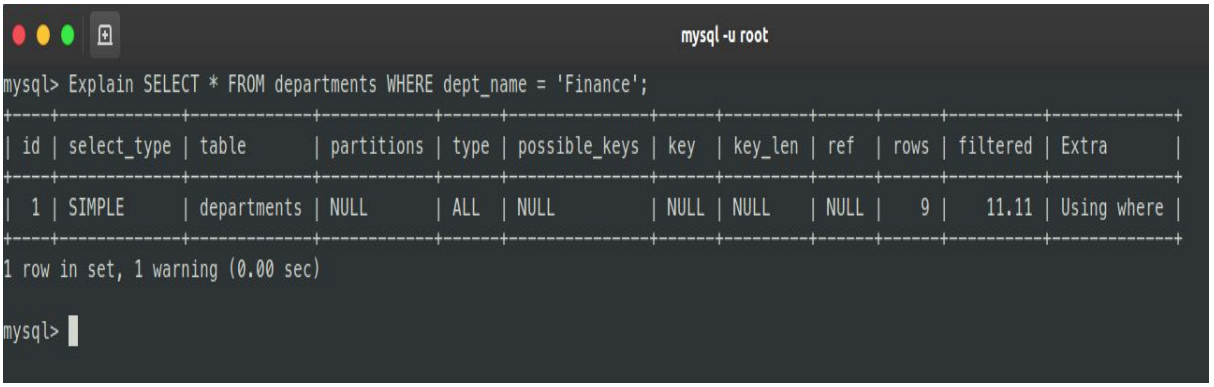
CO527 Advanced Database Systems

Lab: Query Optimization (lab 03)
Reg No: E/15/138
Name: M.M.M Irfan

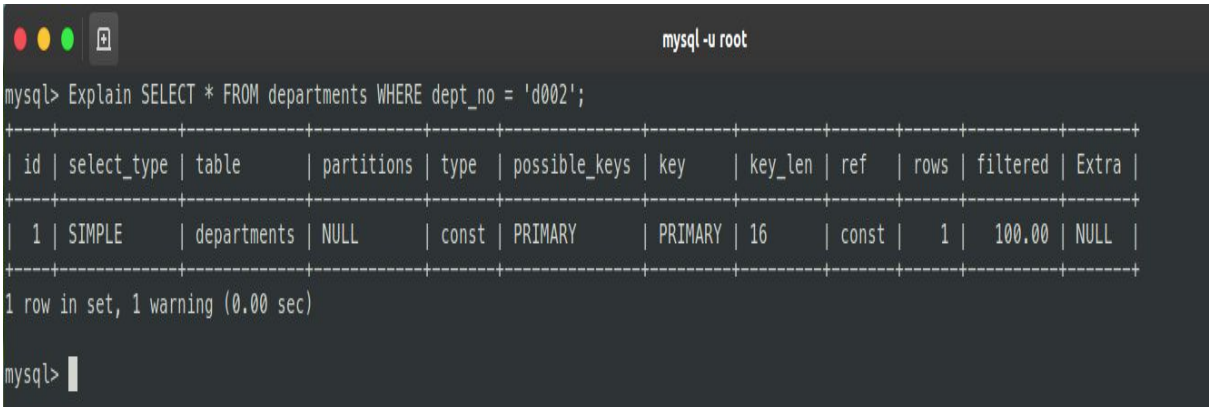
2. Troubleshooting with EXPLAIN

1. Use explain to analyze the outputs of the following two simple queries that use only one table access.

I. SELECT * FROM departments WHERE dept_name = 'Finance';



II. SELECT * FROM departments WHERE deptno = 'd002';



What conclusions you can draw from the results?

Using primary keys on where clause can increase the efficiency of the query, as it can be seen in the (II) part above, the query optimizer picks it up and we get the desired results.so, on the other hand, it can be said that having primary keys on tables and using it for selecting could increase the query execution performance.

2.

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

```
mysql> create table emplist select emp_no, first_name from employees;
Query OK, 300024 rows affected (4.06 sec)
Records: 300024  Duplicates: 0  Warnings: 0

mysql> describe emplist;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| emp_no     | int         | NO   |     | NULL    |       |
| first_name | varchar(14) | NO   |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql>
```

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

```
mysql> create table titleperiod select emp_no, title, datediff(to_date, from_date) as period FROM titles;
Query OK, 443306 rows affected (5.96 sec)
Records: 443306  Duplicates: 0  Warnings: 0

mysql> describe titleperiod;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| emp_no | int         | NO   |     | NULL    |       |
| title  | varchar(50) | NO   |     | NULL    |       |
| period | int         | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql>
```

Explain the statement with the select query

```
mysql>  EXPLAIN select first_name, period from emplist,titleperiod where titleperiod.period >4000 and emplist.emp_no = titleperiod.emp_no;
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-----------------------------------------------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                               |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-----------------------------------------------------+
|  1 | SIMPLE      | titleperiod | NULL     | ALL  | NULL          | NULL | NULL    | NULL | 441754 |    33.33 | Using where                                         |
|  1 | SIMPLE      | emplist   | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 299979 |    10.00 | Using where; Using join buffer (hash join)          |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-----------------------------------------------------+
2 rows in set, 1 warning (0.00 sec)

mysql>
```

- The select type is Simple SELECT (not using UNION or subqueries)
- Mysql is going to execute the where statement on the ***titleperiod*** table first.
- As the tables are in the unindexed state so there are no possible indexes to choose (so possible keys are NULL)
- Then it is going to execute the join statement with ***emplist*** table using a join buffer (hash join)

What could be the number of row combinations that MySQL would need to Check?

- In the **titleperiod** table, 441754 rows are going to be examined
- In the ***emplist*** table, 299979 rows are going to be examined

3. I), II), III) outputs

- Adding the primary key for ***emplist*** table

```
mysql>
mysql>  ALTER TABLE emplist ADD PRIMARY KEY (emp_no);
Query OK, 0 rows affected (7.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show index from emplist;
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| emplist |          0 | PRIMARY  |            1 | emp_no      | A         |      300015 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+---------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
1 row in set (0.01 sec)
```

- Adding a non-unique index on ***titleperiod*** table

```
mysql> create index idx_emp_no on  titleperiod (emp_no);
Query OK, 0 rows affected (5.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show index from titleperiod;
+-------------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table       | Non_unique | Key_name    | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-------------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| titleperiod |          1 | idx_emp_no  |            1 | emp_no      | A         |      299216 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-------------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
1 row in set (0.01 sec)
```

- Explain statement output after creating indexes

```
mysql>  EXPLAIN select first_name, period from emplist,titleperiod where titleperiod.period >4000 and emplist.emp_no = titleperiod.emp_no;
+----+-------------+-------------+------------+------+---------------+-------------+---------+-------------------------+--------+----------+-------------+
| id | select_type | table       | partitions | type | possible_keys | key         | key_len | ref                     | rows   | filtered | Extra       |
+----+-------------+-------------+------------+------+---------------+-------------+---------+-------------------------+--------+----------+-------------+
|  1 | SIMPLE      | emplist     | NULL       | ALL  | PRIMARY       | NULL        | NULL    | NULL                    | 300015 |   100.00 | NULL        |
|  1 | SIMPLE      | titleperiod | NULL       | ref  | idx_emp_no    | idx_emp_no  | 4       | company.emplist.emp_no  |      1 |    33.33 | Using where |
+----+-------------+-------------+------------+------+---------------+-------------+---------+-------------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)
```

It can be seen that now the order of execution has changed so that the ***emplist*** table is used at the beginning, the reason for that is it has a possible index to choose which is a primary key in this case.

And the new join type ***"ref"*** is used here for the ***titleperiod*** table, which is describing that all rows with matching index values are read from this table for each combination of rows from the previous tables. ref is used if the join uses only a leftmost prefix of the key or if the key is not a PRIMARY KEY or UNIQUE index (in other words, if the join cannot select a single row based on the key-value). If the key that is used matches only a few rows, this is a good join type.

And it can be seen that the ref column shows which columns or constants are compared to the index named in the key column to select rows from the table in this case, it is the ***emp_no*** column in the ***emplist*** table
Is it possible to optimize the query execution further? If so, what can be done?

The query is well optimized after adding the indexes, but we can optimize the size of the table ***titleperiod.*** because the column title is not used in any situation so projecting only the emp_no and period is enough for this query execution. But for our original requirement, we need the title when we search for assigned titles.

3. Query Rewriting Techniques

Using forced index

```
mysql> EXPLAIN SELECT first_name FROM emplist FORCE INDEX(PRIMARY) WHERE emp_no > 1000;
+----+-------------+---------+------------+-------+---------------+---------+---------+------+--------+----------+-------------+
| id | select_type | table   | partitions | type  | possible_keys | key     | key_len | ref  | rows   | filtered | Extra       |
+----+-------------+---------+------------+-------+---------------+---------+---------+------+--------+----------+-------------+
|  1 | SIMPLE      | emplist | NULL       | range | PRIMARY       | PRIMARY | 4       | NULL | 150007 |   100.00 | Using where |
+----+-------------+---------+------------+-------+---------------+---------+---------+------+--------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

Summary - "how *explain* helps in optimizing query execution"

When it comes to query tuning, EXPLAIN is one the most important tool in the DBA's arsenal. It helps us to find answers for questions like below,

- Why is a given query slow?
- What does the execution plan look like?
- How will JOINs be processed?
- Is the query using the correct indexes, or is it creating a temporary table?

With the help of EXPLAIN, we can see where we should add indexes to tables so that the statement executes faster by using indexes to find rows. We can also use EXPLAIN to check whether the optimizer joins the tables in an optimal order. Moreover, we can give a hint to

the optimizer to use a join order corresponding to the order in which the tables are named in a SELECT statement using (SELECT STRAIGHT_JOIN).

EXPLAIN can be very useful for spotting problems in our queries **early on**. There are a lot of problems that we only notice when our applications are in production and have big amounts of data or a lot of visitors hitting the database. If these things can be spotted early on using explain, there's much less room for performance problems in the future.

REFERENCES

- https://dev.mysql.com/doc/refman/5.7/en/using-explain.html
- https://dev.mysql.com/doc/refman/8.0/en/explain-output.html#explain-join-types