CO527: Advanced Database Systems
Lab 03 - Query Optimization

Ranage R.D.P.R.
E/19/310

1. Use explain to analyze the outputs of following two simple queries which use only one table access.
I. SELECT * FROM departments WHERE deptname = 'Finance';

```
MySQL localhost:3306 ssl company SQL > EXPLAIN SELECT * FROM department WHERE dept_name = 'Finance';
+----+-------------+------------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table      | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+------------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | department | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    9 |    11.11 | Using where |
+----+-------------+------------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.0013 sec)
```

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

```
MySQL localhost:3306 ssl company SQL > EXPLAIN SELECT * FROM department WHERE dept_no = 'd002';
+----+-------------+------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table      | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | department | NULL       | const | PRIMARY       | PRIMARY | 16      | const |    1 |      100 | NULL  |
+----+-------------+------------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.0008 sec)
```

What conclusions you can draw from the results?

1. Query 1 (dept_name = 'Finance'):
   The query uses a simple table scan (type: ALL), meaning it examines all rows in the departments table.
   There are no possible keys or indexes that can be utilized for this query (possible_keys: NULL, key: NULL).
   The WHERE clause is applied directly to the table (Using where).
   The number of rows examined is 9.
   Conclusion: This query is not utilizing any indexes, resulting in a full table scan, and could potentially benefit from an index on the dept_name column to improve performance.
2. Query 2 (dept_no = 'd002'):
   The query uses a const access method, which indicates a single-row lookup based on a constant value.
   The PRIMARY key is used for this query (possible_keys: PRIMARY, key: PRIMARY), and the lookup is based on the primary key index.
   The key length is 16 bytes, matching the length of the primary key.
   Only one row is examined.
   Conclusion: This query benefits from the presence of the primary key index on the dept_no column, resulting in efficient single-row lookup.

2. Start by creating the initial tables emplist and titleperiod as follows. These derived tables need to contain only the columns involved in the query.

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;

```
MySQL localhost:3306 ssl company SQL > create table emplist select emp_no, first_name from employee;
Query OK, 300024 rows affected (4.0564 sec)

Records: 300024  Duplicates: 0  Warnings: 0
MySQL localhost:3306 ssl company SQL > create table titleperiod select emp_no, title, datediff(to_date, from_date) as period FROM title;
Query OK, 443306 rows affected (6.1473 sec)

Records: 443306  Duplicates: 0  Warnings: 0
```

Now write the query that gives the desired information in the required format.

```
Records: 443306  Duplicates: 0  Warnings: 0
MySQL  localhost:3306 ssl  company  SQL > select e.first_name, t.period
                                       -> from emplist e JOIN titleperiod t ON e.emp_no = t.emp_no
                                       -> WHERE t.period > 4000
                                       -> LIMIT 20;

+------------+---------+
| first_name | period  |
+------------+---------+
| Georgi     | 2926512 |
| Bezalel    | 2922821 |
| Parto      | 2923065 |
| Chirstian  | 2923067 |
| Kyoichi    | 2922781 |
| Anneke     | 2925011 |
```

Analyze the output of applying EXPLAIN to the above query explaining each value. Note that the tables are in their initial unindexed state.

```
MySQL  localhost:3306 ssl  company  SQL > EXPLAIN select e.first_name, t.period from emplist e JOIN titleperiod t ON e.emp_no = t.emp_no WHERE t.period > 4
000 LIMIT 20;
+----+-------------+-------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                        |
+----+-------------+-------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
|  1 | SIMPLE      | t     | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 442668 |    33.33 | Using where                                  |
|  1 | SIMPLE      | e     | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 299751 |       10 | Using where; Using join buffer (hash join) |
+----+-------------+-------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------+
2 rows in set, 1 warning (0.0008 sec)
```

The query that involves two tables, t and e, and its execution process is explained in the EXPLAIN result. A full table scan of both tables is performed; this is indicated by the type: ALL, indicating that each table's rows are looked over. Moreover, the tables are in their original unindexed state because there are no potential keys and no indexes are used. As a result, the query only uses sequential scans of all the tables, which can result in less than ideal speed, especially when dealing with big datasets. Furthermore, the WHERE clause is used in both tables, suggesting that filtering is done while the query is being executed. The fact that Table e uses a join buffer for a hash join operation is noteworthy since it implies that the join between the two tables is processed by a hash join algorithm. Nevertheless, since the optimizer expects a large number of rows to be analyzed, the absence of indexes and the reliance on complete table scans suggest possible performance bottlenecks. Thus, it's best to construct the right indexes on the columns that are used in join conditions and filtering predicates in order to improve query performance. Queries can be executed more quickly and with greater efficiency when pertinent columns are indexed. This allows the query optimizer to make use of more effective access techniques, including index scans.

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

No. of row combinations = titleperiod table rows x emplist table rows

= 442929 x 299715

= 132,752,465,235

3.

I. Create indexes on the columns used to join the tables. In the emplist table, emp_no can be used as a primary key because it uniquely identifies each row.

```
MySQL  localhost:3306 ssl  company  SQL > alter table emplist
                                       -> Add primary key(emp_no);
Query OK, 0 rows affected (1.5009 sec)
```

II. In the titleperiod table, emp_no must be a non-unique index because multiple employees can share the same title.

```
MySQL localhost:3306 ssl company SQL > alter table titleperiod
                                    -> add index(emp_no);
Query OK, 0 rows affected (1.3945 sec)

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

III. Analyse the outputs of EXPLAIN After creating the indexes.

```
MySQL localhost:3306 ssl company SQL > EXPLAIN select e.first_name, t.period from emplist e JOIN titleperiod t ON e.emp_no = t.emp_no WHERE t.period > 4
000 LIMIT 20;
+----+-------------+-------+------------+------+---------------+--------+---------+------------------+--------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key    | key_len | ref              | rows   | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+--------+---------+------------------+--------+----------+-------------+
|  1 | SIMPLE      | e     | NULL       | ALL  | PRIMARY       | NULL   | NULL    | NULL             | 299545 |      100 | NULL        |
|  1 | SIMPLE      | t     | NULL       | ref  | emp_no        | emp_no | 5       | company.e.emp_no |      1 |    33.33 | Using where |
+----+-------------+-------+------------+------+---------------+--------+---------+------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.0034 sec)
```

Is it possible to optimize the query execution further? If so, what can be done?

Optimizing Table 'e' Access: Ensure that the query filters rows based on the primary key of table 'e' to utilize the primary key index efficiently. If the primary key is not used for filtering, consider whether it is necessary for the query. If not, the primary key index can potentially be dropped to save space and improve write performance.

Index Usage on Table 't': Since the 'emp_no' column of table 't' is being used for filtering, ensure that the index on this column is optimized. Consider reviewing the index definition and column data types to ensure they align with the query requirements. Additionally, monitor the index's fragmentation and consider rebuilding it periodically for optimal performance.

Query Rewriting Techniques

Using explain to analyze queries give you clues about ways the query might be improved. You can modify the query and then run explain again to see output changes. The following query rewriting techniques can be useful.

1. USE/FORCE INDEX - To force MySQL to use an index.
2. IGNORE INDEX - To tell MySQL to ignore an index.

```
MySQL localhost:3306 ssl company SQL > 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 |    1 |      100 | Using where |
+----+-------------+---------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.0085 sec)
```

```
MySQL localhost:3306 ssl company SQL > EXPLAIN SELECT STRAIGHT_JOIN 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 |    1 |      100 | Using where |
+----+-------------+---------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.0014 sec)
```

Therefore, based on the information provided, we can conclude that MySQL's EXPLAIN function is a strong tool for SQL query optimization. It offers details on how a specific query is carried out by MySQL.

- Understanding the execution of queries
- locating bottlenecks
- making use of indexes
- joining operations efficiently
- optimizing select queries.