E/16/156

H.A.D.T.T. Jayathilaka

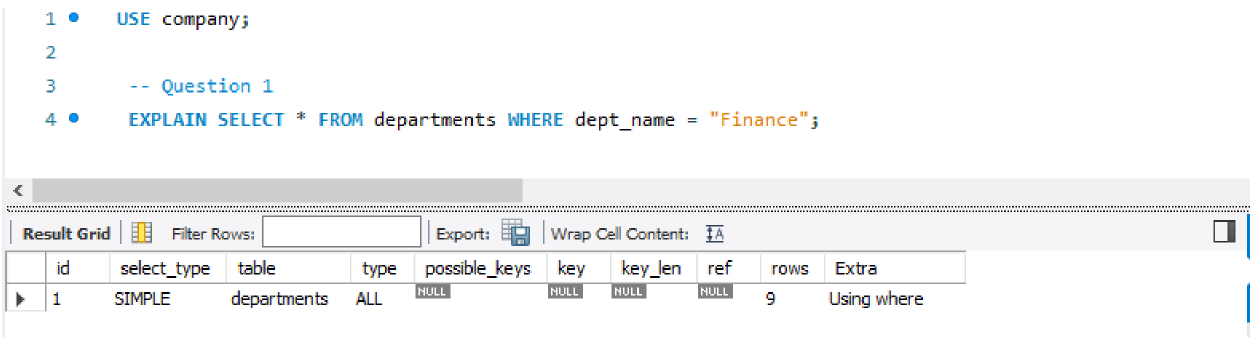
Department of Computer Engineering

University of Peradeniya

CO527 Advanced Database Systems

Lab Task :

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';**

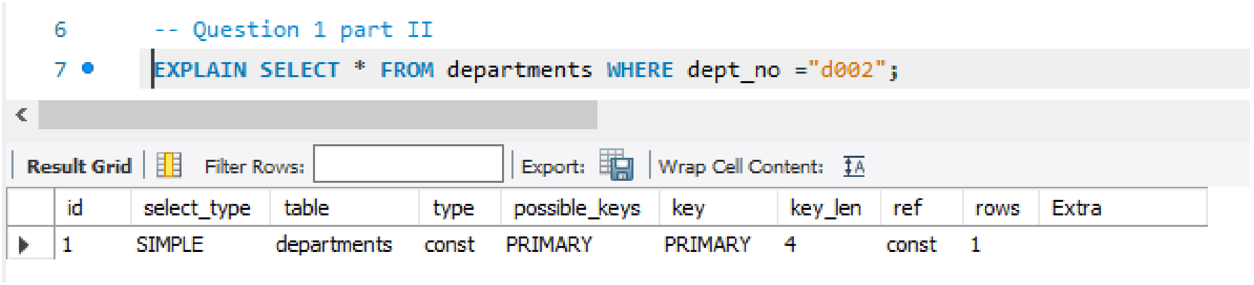


```
USE company;

 -- Question 1
 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
```

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



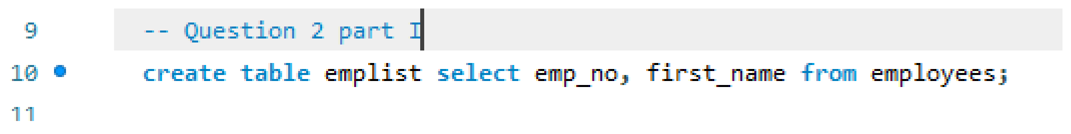
```
 -- Question 1 part II
EXPLAIN SELECT * FROM departments WHERE dept_no ="d002";
```

```
id  select_type  table        type   possible_keys  key      key_len  ref    rows  Extra
1   SIMPLE       departments  const  PRIMARY        PRIMARY  4        const  1
```

What conclusions you can draw from the results?

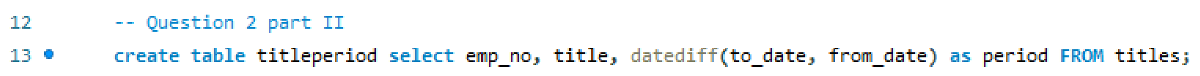
The first query has no keys that could be used for indexing. To display the findings, the compiler must browse through 9 rows. However, the second query's where clause uses the department number, which serves as the department table's primary key. "const" has been substituted for "ALL" as the type in the explanation tables. This indicates that the compiler found the needed record without having to search through all of the table's records. So, instead of the first query's nine rows, there is only one record. We can therefore say that the second query does 100% filtering**.** This shows how using the primary key to select rows will speed up query processing.

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



```sql
-- Question 2 part I
create table emplist select emp_no, first_name from employees;
```

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

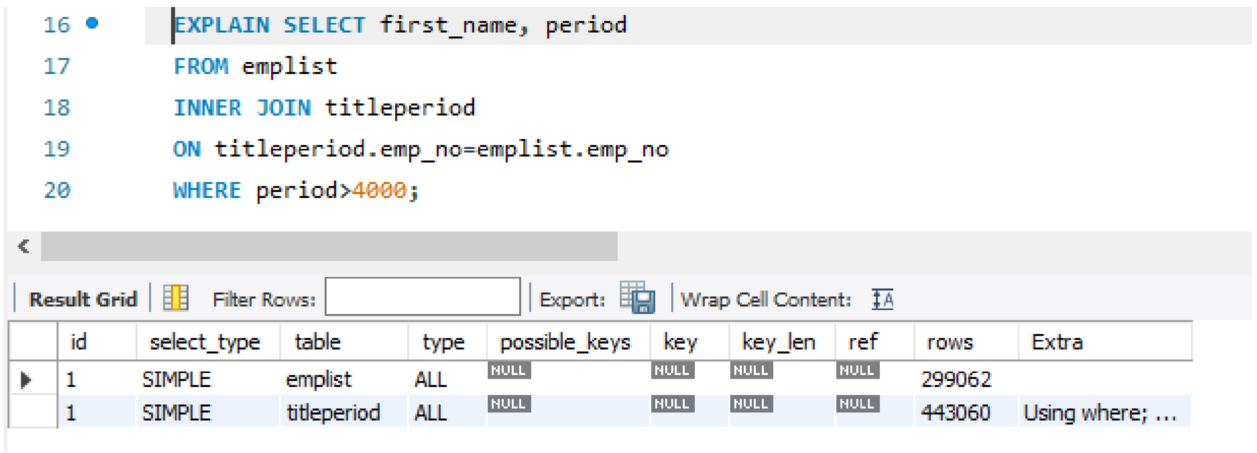


```sql
-- Question 2 part II
create table titleperiod select emp_no, title, datediff(to_date, from_date) as period FROM titles;
```

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

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

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



```sql
EXPLAIN SELECT first_name, period
FROM emplist
INNER JOIN titleperiod
ON titleperiod.emp_no=emplist.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	299062	
1	SIMPLE	titleperiod	ALL	NULL	NULL	NULL	NULL	443060	Using where; ...

"titleperiod" table and "emplist" table are involved so EXPLAIN plan two rows. The table access or joining methods are determined by the column 'type'. The fact that the access type in this case is "ALL" indicates that MySQL completely scanned the two tables. The table "titleperiod" was accessed for the first time using the ALL access type, as you can see in the EXPLIAN. In the following step, the ALL accessible type was used to access the table "emplist." It is obvious that MySQL did not use any indexes when executing the query in the above table because the column "key" is NULL in both rows. The "titleperiod" table's 443060 rows and the "emplist" table's 299062 rows were both scanned. The number of rows that the WHERE clause's conditions did not filter is represented by the values in the column "filtered." Due to the fact that it scanned a sizable amount of rows in both tables, this query was not optimized. On such tables, we build indexes in order to optimize this query.

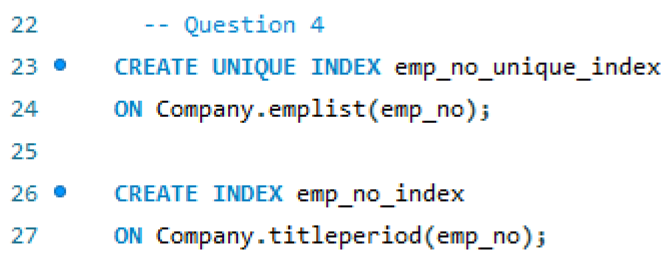
What could be the number of row combinations that MySQL would need to check?

The number of row combinations will be around 299388 x 441754.

3. **Good columns to index are those that you typically use for searching, grouping, or sorting records. The query does not have any GROUP BY or ORDER BY clauses, but it does use columns for searching:**
   - **The query uses emplist.emp_no and titleperiod.emp_no to match records between tables.**
   - **The query uses titleperiod.period to cut down records that do not satisfy the condition.**

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.**

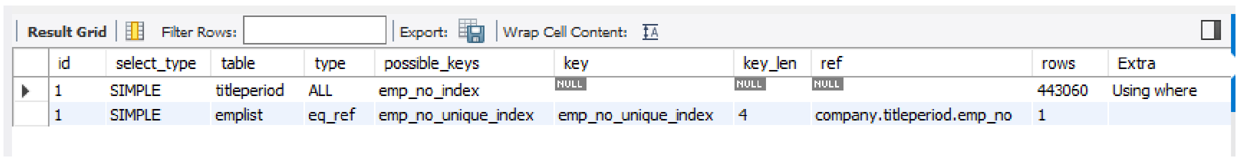
II. **In the titleperiod table, emp_no must be a non-unique index because multiple employees can share the same title:**



```
  -- Question 4
CREATE UNIQUE INDEX emp_no_unique_index
ON Company.emplist(emp_no);

CREATE INDEX emp_no_index
ON Company.titleperiod(emp_no);
```

III. **Analyse the outputs of EXPLAIN After creating the indexes. Is it possible to optimize the query execution further? If so, what can be done?**


Result Grid
Filter Rows:
Export:
Wrap Cell Content:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	titleperiod	ALL	emp_no_index	NULL	NULL	NULL	443060	Using where
1	SIMPLE	emplist	eq_ref	emp_no_unique_index	emp_no_unique_index	4	company.titleperiod.emp_no	1	

"titleperiod" table and "emplist" table are involved so EXPLAIN plan two rows. The table access or joining methods are determined by the column 'type'. The table "titleperiod" was the first accessed table utilizing the ALL access type, as can be seen in the EXPLIAN. Then the eq_ref accessible type was used to access the table "emplist." For every combination of rows from the preceding table, the database will access one row from this one. The actual index that MySQL chose to use is displayed in the column labeled "key." When the query was executed, MySQL utilized the index we constructed on the column "emp no" of the "emplist" table, "emp no uni index." Given that it did not scan the complete table "emplist," this query was optimized.

It is possible to optimize the query execution further by adding an index on the column ‘period’.