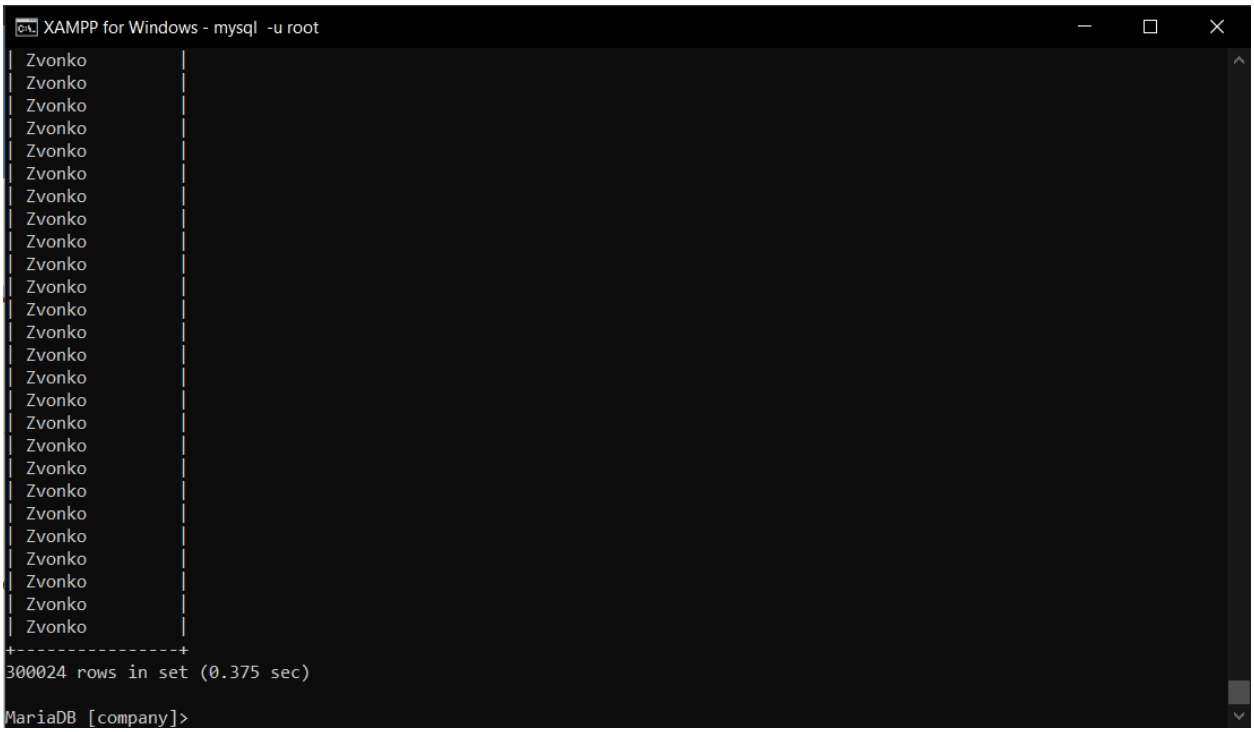
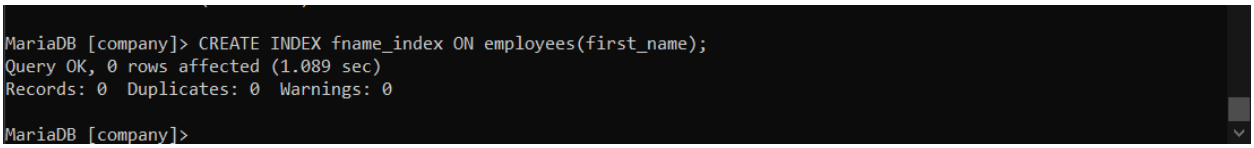
CO527: Advanced Database Systems
Lab 02 - Indexing

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

1. Assuming no indexes are used, record the query execution time for retrieving all the employees by first name in ascending order.



2. Create an index called fname_index on the first_name of the employee table. Retrieve all the employees by first name and record the query execution time. Observe the performance improvement gained when accessing with index.



When comparing query execution times with and without indexing, there is a noticeable difference.
The query with indexing took only 0.098 seconds to execute, compared to 0.375 seconds for the run without indexing. Thus, a notable optimization is seen in comparison.
3. Which indexing technique has been used when creating the above index?

```
MariaDB [company]> SHOW INDEX FROM employees;
+-----------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table     | Non_unique | Key_name    | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-----------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| employees |          0 | PRIMARY     |            1 | emp_no      | A         |      299290 |     NULL | NULL   |      | BTREE      |         |               |
| employees |          1 | fname_index |            1 | first_name  | A         |        2602 |     NULL | NULL   |      | BTREE      |         |               |
+-----------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
2 rows in set (0.002 sec)

MariaDB [company]>
```

4. Create a unique index on emp_no, first_name and last_name of employees table. Retrieve all the employees by emp_no, first_name and last_name.

```
2 rows in set (0.002 sec)

MariaDB [company]> CREATE UNIQUE INDEX emp_index ON employees (emp_no, first_name, last_name);
Query OK, 0 rows affected (0.496 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [company]>
```

```
| 499994 | Navin      | Argence    |
| 499995 | Dekang     | Lichtner   |
| 499996 | Zito       | Baaz       |
| 499997 | Berhard    | Lenart     |
| 499998 | Patricia   | Breugel    |
| 499999 | Sachin     | Tsukuda    |
+--------+------------+------------+
300024 rows in set (0.161 sec)

MariaDB [company]>
```

Without indexing, the query took 0.375 seconds; however, after the unique index was created, it only took 0.160 seconds. This suggests that query performance has slightly improved with indexing.
This might be because the database engine can find the rows more rapidly thanks to the unique index, which cuts down on the amount of time needed for searching and retrieval based on the indexed columns. It aids in query optimization for filtering or sorting based on emp_no, first_name, or last_name.
Though the query itself is simple and retrieves all employees by name and unique identifier, the degree of improvement may not be noteworthy in this instance. Comparing such queries to more complicated queries or ones involving enormous datasets, indexing may not be as beneficial in these cases.

5. Take the following 3 queries.

- select distinct emp_no from dept_manager where from_date>=’1985-01-01’ and dept_no>= ’d005’;
- select distinct emp_no from dept_manager where from_date>=’1996-01-03’ and dept_no>= ’d005’;
- select distinct emp_no from dept_manager where from_date>=’1985-01-01’ and dept_no<= ’d009’;

I. Choose one single simple index(i.e index on one attribute) that is most likely to speed up all 3 queries giving reasons for your selection.

An index on from_date would be the most appropriate single simple index that is likely to speed up all three queries because all three involve filtering based on this characteristic.

```
+---------+-----------------+--------------------+
300024 rows in set (0.161 sec)

MariaDB [company]> CREATE INDEX from_date_index ON dept_manager(from_date);
Query OK, 0 rows affected (0.023 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [company]>
```

II. For each of the 3 queries, check if MySQL storage engine used that index. If not, give a short explanation why not.

```
MariaDB [company]> EXPLAIN EXTENDED SELECT DISTINCT emp_no FROM dept_manager
    -> WHERE from_date>='1985-01-01' and dept_no>= 'd005';
+------+-------------+--------------+-------+------------------------+------------+---------+------+------+----------+-------------+
| id   | select_type | table        | type  | possible_keys          | key        | key_len | ref  | rows | filtered | Extra       |
+------+-------------+--------------+-------+------------------------+------------+---------+------+------+----------+-------------+
|    1 | SIMPLE      | dept_manager | index | PRIMARY,from_date_index | fk_emp_no3 | 4       | NULL | 24   |    48.61 | Using where |
+------+-------------+--------------+-------+------------------------+------------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.002 sec)

MariaDB [company]>
MariaDB [company]> EXPLAIN EXTENDED SELECT DISTINCT emp_no FROM dept_manager
    -> WHERE from_date>='1996-01-03' and dept_no>= 'd005';
+------+-------------+--------------+-------+------------------------+-----------------+---------+------+------+----------+-----------------------------------------+
| id   | select_type | table        | type  | possible_keys          | key             | key_len | ref  | rows | filtered | Extra                                   |
+------+-------------+--------------+-------+------------------------+-----------------+---------+------+------+----------+-----------------------------------------+
|    1 | SIMPLE      | dept_manager | range | PRIMARY,from_date_index | from_date_index | 19      | NULL | 2    |   100.00 | Using where; Using index; Using temporary |
+------+-------------+--------------+-------+------------------------+-----------------+---------+------+------+----------+-----------------------------------------+
1 row in set, 1 warning (0.002 sec)

MariaDB [company]>
MariaDB [company]> EXPLAIN EXTENDED SELECT DISTINCT emp_no FROM dept_manager
    -> WHERE from_date>='1985-01-01' and dept_no<= 'd009';
+------+-------------+--------------+-------+------------------------+------------+---------+------+------+----------+-------------+
| id   | select_type | table        | type  | possible_keys          | key        | key_len | ref  | rows | filtered | Extra       |
+------+-------------+--------------+-------+------------------------+------------+---------+------+------+----------+-------------+
|    1 | SIMPLE      | dept_manager | index | PRIMARY,from_date_index | fk_emp_no3 | 4       | NULL | 24   |   100.00 | Using where |
+------+-------------+--------------+-------+------------------------+------------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.002 sec)

MariaDB [company]>
```

6. Consider the queries you wrote for questions 2 - 10 in Lab 01 assignment. Give with short explanations, which attributes on which relations should be used for creating indexes that could speed up your queries.

2. Most likely, the last_name property from the employees table is used in the query. Since this query involves sorting or grouping by last names, creating an index on **last_name** could speed up the process.

3. Joining the departments and personnel tables and filtering by job names (such as "engineer") may be necessary for this query. An index on **title** in the employees table could help this query run more smoothly.

4. This query involves filtering based on gender and job titles. Creating an index on both **sex** and **title** in the employees table and titles table could help speed up this query.

5. This query likely involves joining the employees and departments tables and filtering based on salary ranges. Creating an index on **salary** in the employees table could be beneficial.

6. Filtering using the title "Senior" and potentially combining tables are possible steps in this query. The titles table's **title** index could be created to enhance performance.

7. Creating an index on **dept_name** in the departments table could speed up this query.

8. The purpose of this query is to analyze salaries across departments by combining the tables for employees and departments. Incorporating a **salary** index into the employees table may prove advantageous.

9. This query involves comparing salaries against the company average. Creating an index on **salary** in the employees table could help improve performance.

10. It is likely that this query will sort and aggregate data according to salary and job titles. It could be helpful to create an index in the salaries table based on **salary**.

7. Assume that most of the queries on a relation are insert/update/delete. What will happen to the query execution time if that relation has an index created?

It may affect query execution speed in a favorable or negative way.
Performance should be enhanced for queries that look for certain values or ranges inside the indexed columns. The database engine can swiftly find the pertinent rows thanks to indexes, saving it from having to search the whole table. Additionally, the existence of an index can greatly expedite the sorting and join key procedures for the indexed columns, hence cutting down on the total query execution time.
But there may also be drawbacks. In the event that the majority of queries on a table with indexing are insertion, update, and deletion queries, the database engine must update the

index in addition to changing the data. These procedures may be slowed down by this overhead in comparison to a table without indexes.
The index must be updated in accordance with any changes made to the data in the table. The performance of insert, update, and delete operations may be impacted by this maintenance expense, particularly when the size of the table and the number of indexes grow.
Additionally, there may be a storage overhead because the indexing column requires extra space, which adds up over time, particularly for big databases with several indexes. This can affect the overall performance of the system because of the higher I/O requirements.
Therefore, the performance of the query execution time may not improve if the majority of the queries are insertion, deletion, and update queries.