Hands-on Lab: Improving Performance of Slow Queries in MySQL

Estimated time needed: 45 minutes

In this lab, you will learn how to improve the performance of your slow queries in MySQL, which can be particularly helpful with large databases.

Objectives

After completing this lab, you will be able to:

1. Use the `EXPLAIN` statement to check the performance of your query
2. Add indexes to improve the performance of your query
3. Apply other best practices such as using the `UNION ALL` clause to improve query performance

Software Used in this Lab

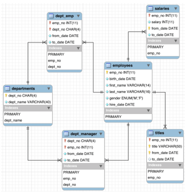
In this lab, you will use MySQL. MySQL is a Relational Database Management System (RDBMS) designed to efficiently store, manipulate, and retrieve data.

To complete this lab, you will utilize the MySQL relational database service available as part of the IBM Skills Network Labs (SN Labs) Cloud IDE. SN Labs is a virtual lab environment used in this course.

Database Used in this Lab

The Employees database used in this lab comes from the following source: https://dev.mysql.com/doc/employee/en/ under the CC BY-SA 3.0 License.

The following entity relationship diagram (ERD) shows the schema of the Employees database:



The first row of each table is the table name, the rows with keys next to them indicate the primary keys, and the remaining rows are additional attributes.

Exercise 1: Load the Database

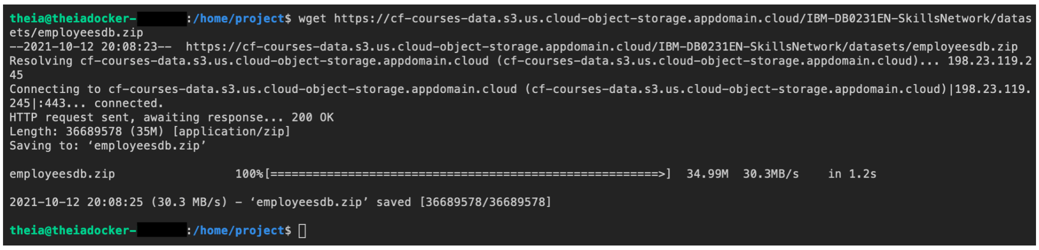
Let's begin by retrieving the database and loading it so that it can be used.

1. In the menu bar, select `Terminal > New Terminal`. This will open the Terminal.

   To download the zip file containing the database, copy and paste the following into the Terminal:

   ```
   wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0231EN-SkillsNetwork/datasets/employeesdb.zip
   ```

   Copied!

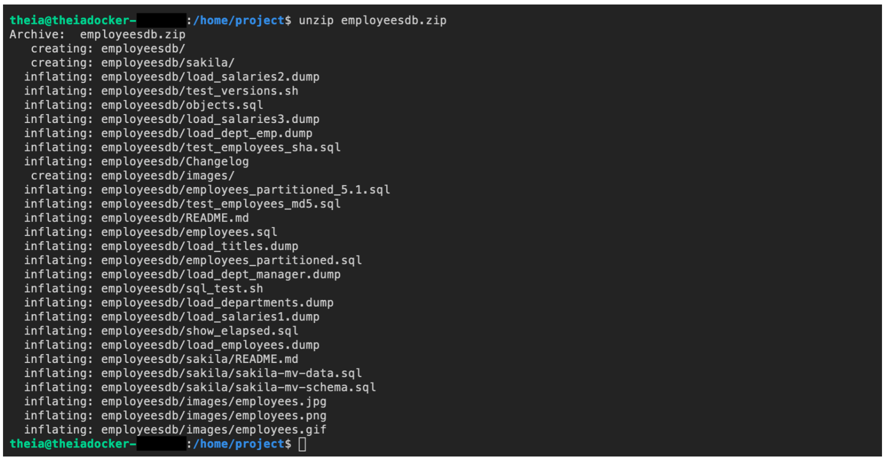
   

   ```
   theia@theiadocker-[redacted]:/home/project$ wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0231EN-SkillsNetwork/datas
   ets/employeesdb.zip
   --2021-10-12 20:08:23--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0231EN-SkillsNetwork/datasets/employeesdb.zip
   Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.2
   45
   Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.
   245|:443... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 36689578 (35M) [application/zip]
   Saving to: ‘employeesdb.zip’

   employeesdb.zip                100%[====================================================>]  34.99M  30.3MB/s    in 1.2s

   2021-10-12 20:08:25 (30.3 MB/s) - ‘employeesdb.zip’ saved [36689578/36689578]

   theia@theiadocker-[redacted]:/home/project$
   ```

2. Next, we'll need to unzip its contents. We can do that with the following command:

   ```
   unzip employeesdb.zip
   ```

   Copied!

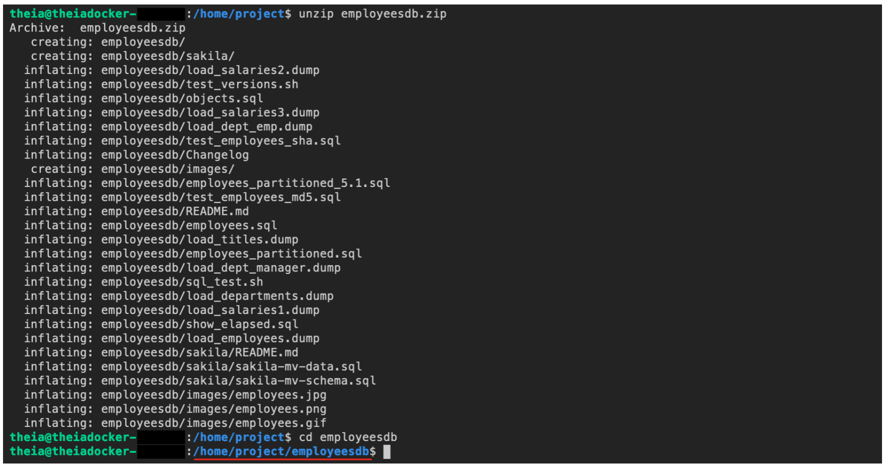
   ```
   theia@theiadocker-[redacted]:/home/project$ unzip employeesdb.zip
   Archive:  employeesdb.zip
      creating: employeesdb/
      creating: employeesdb/sakila/
     inflating: employeesdb/load_salaries2.dump
     inflating: employeesdb/test_versions.sh
     inflating: employeesdb/objects.sql
     inflating: employeesdb/load_salaries3.dump
     inflating: employeesdb/load_dept_emp.dump
     inflating: employeesdb/test_employees_sha.sql
     inflating: employeesdb/Changelog
      creating: employeesdb/images/
     inflating: employeesdb/employees_partitioned_5.1.sql
     inflating: employeesdb/test_employees_md5.sql
     inflating: employeesdb/README.md
     inflating: employeesdb/employees.sql
     inflating: employeesdb/load_titles.dump
     inflating: employeesdb/employees_partitioned.sql
     inflating: employeesdb/load_dept_manager.dump
     inflating: employeesdb/sql_test.sh
     inflating: employeesdb/load_departments.dump
     inflating: employeesdb/load_salaries1.dump
     inflating: employeesdb/show_elapsed.sql
     inflating: employeesdb/load_employees.dump
     inflating: employeesdb/sakila/README.md
     inflating: employeesdb/sakila/sakila-mv-data.sql
     inflating: employeesdb/sakila/sakila-mv-schema.sql
     inflating: employeesdb/images/employees.jpg
     inflating: employeesdb/images/employees.png
     inflating: employeesdb/images/employees.gif
   theia@theiadocker-[redacted]:/home/project$
   ```

3. Now, let's change directories so that we're able to access the files in the newly created **employeesdb** folder.

   ```
   cd employeesdb
   ```

   Copied!

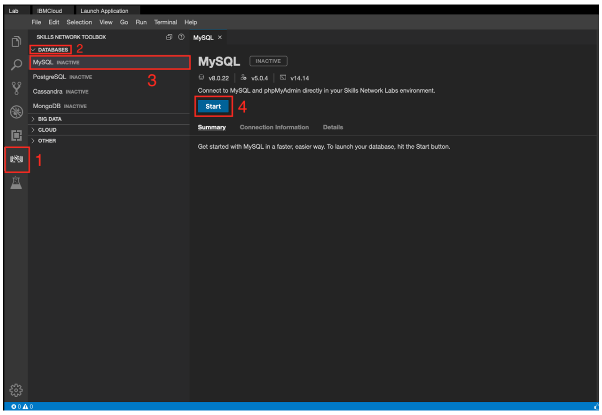
   Check the line next to **theia@theiadocker**. If it reads **/home/project/employeesdb**, then you have successfully changed directories!

   ```
   theia@theiadocker-[redacted]:/home/project$ unzip employeesdb.zip
   Archive:  employeesdb.zip
      creating: employeesdb/
      creating: employeesdb/sakila/
     inflating: employeesdb/load_salaries2.dump
     inflating: employeesdb/test_versions.sh
     inflating: employeesdb/objects.sql
     inflating: employeesdb/load_salaries3.dump
     inflating: employeesdb/load_dept_emp.dump
     inflating: employeesdb/test_employees_sha.sql
     inflating: employeesdb/Changelog
      creating: employeesdb/images/
     inflating: employeesdb/employees_partitioned_5.1.sql
     inflating: employeesdb/test_employees_md5.sql
     inflating: employeesdb/README.md
     inflating: employeesdb/employees.sql
     inflating: employeesdb/load_titles.dump
     inflating: employeesdb/employees_partitioned.sql
     inflating: employeesdb/load_dept_manager.dump
     inflating: employeesdb/sql_test.sh
     inflating: employeesdb/load_departments.dump
     inflating: employeesdb/load_salaries1.dump
     inflating: employeesdb/show_elapsed.sql
     inflating: employeesdb/load_employees.dump
     inflating: employeesdb/sakila/README.md
     inflating: employeesdb/sakila/sakila-mv-data.sql
     inflating: employeesdb/sakila/sakila-mv-schema.sql
     inflating: employeesdb/images/employees.jpg
     inflating: employeesdb/images/employees.png
     inflating: employeesdb/images/employees.gif
   theia@theiadocker-[redacted]:/home/project$ cd employeesdb
   theia@theiadocker-[redacted]:/home/project/employeesdb$
   ```

4. In order to import the data, we'll need to load the data through MySQL. We can do that by navigating to the **Skills Network Toolbox**, selecting **Databases** and then selecting **MySQL**.

   Press **Start**. This will start a session of MySQL in SN Labs.

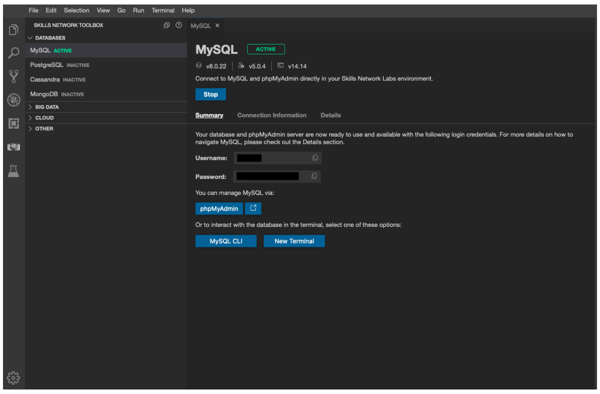


Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environment.

Get started with MySQL in a faster, easier way. To launch your database, hit the Start button.

The **Inactive** label will change to **Starting**. This may take a few moments.

When it changes to **Active**, it means your session has started.



Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environment.

Your database and phpMyAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate MySQL, please check out the Details section.

Username: [redacted]

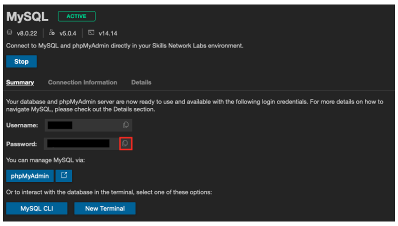
Password: [redacted]

You can manage MySQL via:

Or to interact with the database in the terminal, select one of these options:

Take note of your password. You will need this to start MySQL.

You can copy your password by clicking the button next to it, as shown in the screenshot below:



Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environment.

Your database and phpMyAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate MySQL, please check out the Details section.

Username: [redacted]

Password: [redacted]

You can manage MySQL via:

Or to interact with the database in the terminal, select one of these options:

5. With your password handy, we can now import the data. You can do this by entering the following into the Terminal:

```
mysql --host=127.0.0.1 --port=3306 --user=root --password -t < employees.sql
```

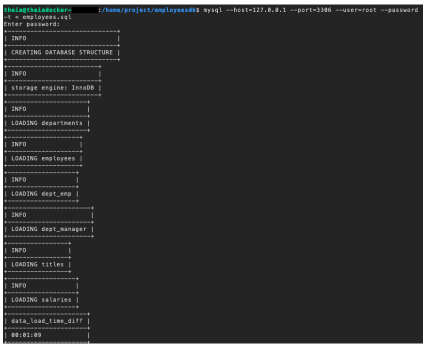
Copied!

When prompted for your password, paste the password that you copied earlier into the Terminal and press **Enter**.

Please note, you won't be able to see your password when typing it in. Not to worry, this is expected!

6. Your data will now load. This may take a minute or so.

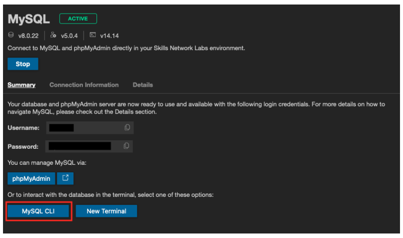
When you've finished loading the data, you'll see the following:



```
theia@theiadocker-[redacted]:/home/project/employeesdb$ mysql --host=127.0.0.1 --port=3306 --user=root --password
-t < employees.sql
Enter password:
+-----------------------------+
| INFO                        |
+-----------------------------+
| CREATING DATABASE STRUCTURE |
+-----------------------------+
+------------------------+
| INFO                   |
+------------------------+
| storage engine: InnoDB |
+------------------------+
+---------------------+
| INFO                |
+---------------------+
| LOADING departments |
+---------------------+
+-------------------+
| INFO              |
+-------------------+
| LOADING employees |
+-------------------+
+------------------+
| INFO             |
+------------------+
| LOADING dept_emp |
+------------------+
+----------------------+
| INFO                 |
+----------------------+
| LOADING dept_manager |
+----------------------+
+----------------+
| INFO           |
+----------------+
| LOADING titles |
+----------------+
+------------------+
| INFO             |
+------------------+
| LOADING salaries |
+------------------+
+---------------------+
| data_load_time_diff |
+---------------------+
| 00:01:09            |
+---------------------+
```

This means that your data has been imported.

7. To enter the MySQL command-line interface, return to your MySQL tab and select **MySQL CLI**.



Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environment.

Your database and phpMyAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate MySQL, please check out the Details section.

Username: [redacted]

Password: [redacted]

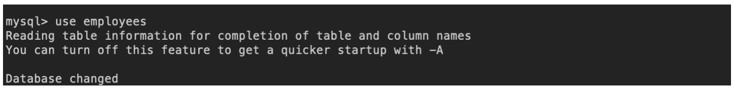
You can manage MySQL via:

Or to interact with the database in the terminal, select one of these options:

8. Recall that the name of the database that we're using is **Employees**. To access it, we can use this command:

```
use employees
```

Copied!

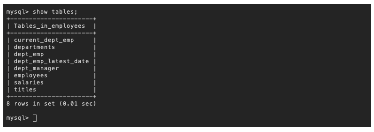


```
mysql> use employees
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

9. Let's see which tables are available in this database:

```
show tables;
```

Copied!



```
mysql> show tables;
+----------------------+
| Tables_in_employees  |
+----------------------+
| current_dept_emp     |
| departments          |
| dept_emp             |
| dept_emp_latest_date |
| dept_manager         |
| employees            |
| salaries             |
| titles               |
+----------------------+
8 rows in set (0.01 sec)

mysql> 
```

In this database, there are 8 tables, which we can confirm with the database’s ERD.

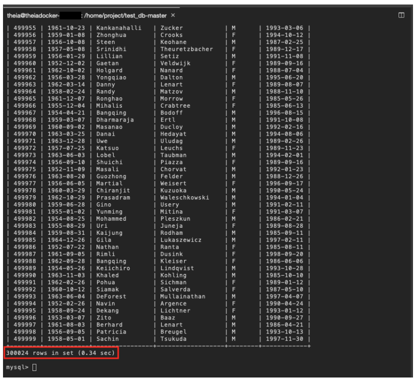
Now that your database is all set up, let’s take a look at how we can check a query’s performance!

Exercise 2: Check Your Query's Performance with EXPLAIN

The EXPLAIN statement, which provides information about how MySQL executes your statement, will offer you insight about the number of rows your query is planning on looking through. This statement can be helpful when your query is running slow. For example, is it running slow because it’s scanning the entire table each time?

1. Let’s start with selecting all the data from the **employees** table:

   ```
   1. 1
   1. SELECT * FROM employees;
   ```
   Copied!

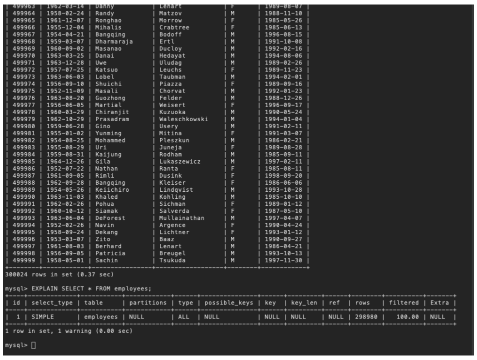
   theia@theiadocker-[redacted]: /home/project/test_db-master ×

   ```
   | 499955 | 1961-10-23 | Kankanahalli   | Zucker           | M      | 1993-03-06 |
   | 499956 | 1959-01-08 | Zhonghua       | Crooks           | F      | 1994-10-12 |
   | 499957 | 1956-10-08 | Steen          | Keohane          | M      | 1987-02-25 |
   | 499958 | 1957-05-08 | Srinidhi       | Theuretzbacher   | F      | 1989-12-17 |
   | 499959 | 1956-01-29 | Lillian        | Setiz            | M      | 1991-11-08 |
   | 499960 | 1952-12-02 | Gaetan         | Veldwijk         | F      | 1989-09-16 |
   | 499961 | 1962-10-02 | Holgard        | Nanard           | F      | 1988-07-04 |
   | 499962 | 1956-03-28 | Yongqiao       | Dalton           | M      | 1995-06-20 |
   | 499963 | 1962-03-14 | Danny          | Lenart           | F      | 1989-08-07 |
   | 499964 | 1958-02-24 | Randy          | Matzov           | M      | 1988-11-10 |
   | 499965 | 1961-12-07 | Ronghao        | Morrow           | F      | 1985-05-26 |
   | 499966 | 1955-12-04 | Mihalis        | Crabtree         | F      | 1985-06-13 |
   | 499967 | 1954-04-21 | Bangqing       | Bodoff           | M      | 1996-08-15 |
   | 499968 | 1959-03-07 | Dharmaraja     | Ertl             | M      | 1991-10-08 |
   | 499969 | 1960-09-02 | Masanao        | Ducloy           | M      | 1992-02-16 |
   | 499970 | 1963-03-25 | Danai          | Hedayat          | M      | 1994-08-06 |
   | 499971 | 1963-12-28 | Uwe            | Uludag           | M      | 1989-02-26 |
   | 499972 | 1957-07-25 | Katsuo         | Leuchs           | F      | 1989-11-23 |
   | 499973 | 1963-06-03 | Lobel          | Taubman          | M      | 1994-02-01 |
   | 499974 | 1956-09-10 | Shuichi        | Piazza           | F      | 1989-09-16 |
   | 499975 | 1952-11-09 | Masali         | Chorvat          | M      | 1992-01-23 |
   | 499976 | 1963-08-20 | Guozhong       | Felder           | M      | 1988-12-26 |
   | 499977 | 1956-06-05 | Martial        | Weisert          | F      | 1996-09-17 |
   | 499978 | 1960-03-29 | Chiranjit      | Kuzuoka          | M      | 1990-05-24 |
   | 499979 | 1962-10-29 | Prasadram      | Waleschkowski    | M      | 1994-01-04 |
   | 499980 | 1959-06-28 | Gino           | Usery            | M      | 1991-02-11 |
   | 499981 | 1955-01-02 | Yunming        | Mitina           | F      | 1991-03-07 |
   | 499982 | 1954-08-25 | Mohammed       | Pleszkun         | M      | 1986-02-21 |
   | 499983 | 1955-08-29 | Uri            | Juneja           | F      | 1989-08-28 |
   | 499984 | 1959-08-31 | Kaijung        | Rodham           | M      | 1985-09-11 |
   | 499985 | 1964-12-26 | Gila           | Lukaszewicz      | M      | 1997-02-11 |
   | 499986 | 1952-07-22 | Nathan         | Ranta            | F      | 1985-08-11 |
   | 499987 | 1961-09-05 | Rimli          | Dusink           | F      | 1998-09-20 |
   | 499988 | 1962-09-28 | Bangqing       | Kleiser          | F      | 1986-06-06 |
   | 499989 | 1954-05-26 | Keiichiro      | Lindqvist        | M      | 1993-10-28 |
   | 499990 | 1963-11-03 | Khaled         | Kohling          | M      | 1985-10-10 |
   | 499991 | 1962-02-26 | Pohua          | Sichman          | F      | 1989-01-12 |
   | 499992 | 1960-10-12 | Siamak         | Salverda         | F      | 1987-05-10 |
   | 499993 | 1963-06-04 | DeForest       | Mullainathan     | M      | 1997-04-07 |
   | 499994 | 1952-02-26 | Navin          | Argence          | F      | 1990-04-24 |
   | 499995 | 1958-09-24 | Dekang         | Lichtner         | F      | 1993-01-12 |
   | 499996 | 1953-03-07 | Zito           | Baaz             | M      | 1990-09-27 |
   | 499997 | 1961-08-03 | Berhard        | Lenart           | M      | 1986-04-21 |
   | 499998 | 1956-09-05 | Patricia       | Breugel          | M      | 1993-10-13 |
   | 499999 | 1958-05-01 | Sachin         | Tsukuda          | M      | 1997-11-30 |
   +--------+------------+----------------+------------------+--------+------------+
   300024 rows in set (0.34 sec)

   mysql> 
   ```

   As you can see, all 300,024 rows were loaded, taking about 0.34 seconds.

2. We can use EXPLAIN to see how many rows were scanned:

   ```
   1. 1
   1. EXPLAIN SELECT * FROM employees;
   ```
   Copied!

   ```
   | 499963 | 1962-03-14 | Danny          | Lenart           | F      | 1989-08-07 |
   | 499964 | 1958-02-24 | Randy          | Matzov           | M      | 1988-11-10 |
   | 499965 | 1961-12-07 | Ronghao        | Morrow           | F      | 1985-05-26 |
   | 499966 | 1955-12-04 | Mihalis        | Crabtree         | F      | 1985-06-13 |
   | 499967 | 1954-04-21 | Bangqing       | Bodoff           | M      | 1996-08-15 |
   | 499968 | 1959-03-07 | Dharmaraja     | Ertl             | M      | 1991-10-08 |
   | 499969 | 1960-09-02 | Masanao        | Ducloy           | M      | 1992-02-16 |
   | 499970 | 1963-03-25 | Danai          | Hedayat          | M      | 1994-08-06 |
   | 499971 | 1963-12-28 | Uwe            | Uludag           | M      | 1989-02-26 |
   | 499972 | 1957-07-25 | Katsuo         | Leuchs           | F      | 1989-11-23 |
   | 499973 | 1963-06-03 | Lobel          | Taubman          | M      | 1994-02-01 |
   | 499974 | 1956-09-10 | Shuichi        | Piazza           | F      | 1989-09-16 |
   | 499975 | 1952-11-09 | Masali         | Chorvat          | M      | 1992-01-23 |
   | 499976 | 1963-08-20 | Guozhong       | Felder           | M      | 1988-12-26 |
   | 499977 | 1956-06-05 | Martial        | Weisert          | F      | 1996-09-17 |
   | 499978 | 1960-03-29 | Chiranjit      | Kuzuoka          | M      | 1990-05-24 |
   | 499979 | 1962-10-29 | Prasadram      | Waleschkowski    | M      | 1994-01-04 |
   | 499980 | 1959-06-28 | Gino           | Usery            | M      | 1991-02-11 |
   | 499981 | 1955-01-02 | Yunming        | Mitina           | F      | 1991-03-07 |
   | 499982 | 1954-08-25 | Mohammed       | Pleszkun         | M      | 1986-02-21 |
   | 499983 | 1955-08-29 | Uri            | Juneja           | F      | 1989-08-28 |
   | 499984 | 1959-08-31 | Kaijung        | Rodham           | M      | 1985-09-11 |
   | 499985 | 1964-12-26 | Gila           | Lukaszewicz      | M      | 1997-02-11 |
   | 499986 | 1952-07-22 | Nathan         | Ranta            | F      | 1985-08-11 |
   | 499987 | 1961-09-05 | Rimli          | Dusink           | F      | 1998-09-20 |
   | 499988 | 1962-09-28 | Bangqing       | Kleiser          | F      | 1986-06-06 |
   | 499989 | 1954-05-26 | Keiichiro      | Lindqvist        | M      | 1993-10-28 |
   | 499990 | 1963-11-03 | Khaled         | Kohling          | M      | 1985-10-10 |
   | 499991 | 1962-02-26 | Pohua          | Sichman          | F      | 1989-01-12 |
   | 499992 | 1960-10-12 | Siamak         | Salverda         | F      | 1987-05-10 |
   | 499993 | 1963-06-04 | DeForest       | Mullainathan     | M      | 1997-04-07 |
   | 499994 | 1952-02-26 | Navin          | Argence          | F      | 1990-04-24 |
   | 499995 | 1958-09-24 | Dekang         | Lichtner         | F      | 1993-01-12 |
   | 499996 | 1953-03-07 | Zito           | Baaz             | M      | 1990-09-27 |
   | 499997 | 1961-08-03 | Berhard        | Lenart           | M      | 1986-04-21 |
   | 499998 | 1956-09-05 | Patricia       | Breugel          | M      | 1993-10-13 |
   | 499999 | 1958-05-01 | Sachin         | Tsukuda          | M      | 1997-11-30 |
   +--------+------------+----------------+------------------+--------+------------+
   300024 rows in set (0.37 sec)

   mysql> EXPLAIN SELECT * FROM employees;
   +----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
   | id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra |
   +----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
   |  1 | SIMPLE      | employees | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 298980 |   100.00 | NULL  |
   +----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
   1 row in set, 1 warning (0.00 sec)

   mysql> 
   ```

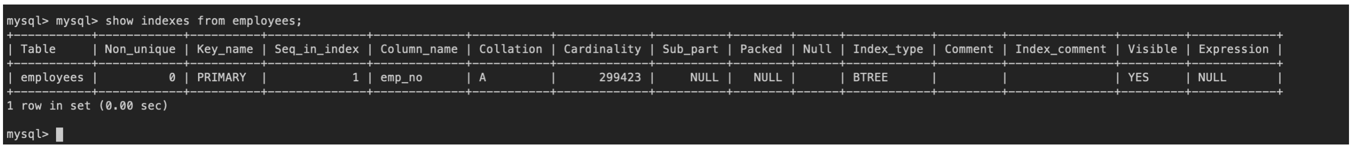
   Notice how EXPLAIN shows that it is examining 298,980 rows, almost the entire table! With a larger table, this could result in the query running slowly.

So, how can we make this query faster? That’s where indexes come in!

Exercise 3: Add an Index to Your Table

1. To begin, let’s take at the existing indexes. We can do that by entering the following command:

   ```
   1. 1
   1. SHOW INDEX FROM employees;
   ```
   Copied!

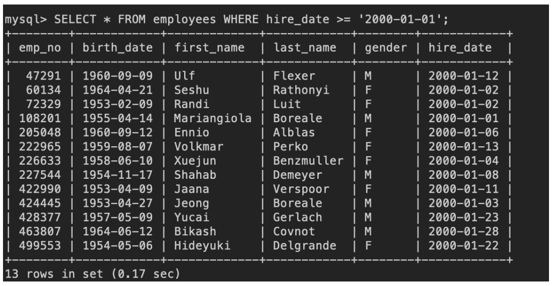
   

   ```
   mysql> mysql> show indexes from employees;
   +-----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
   | Table     | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
   +-----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
   | employees |          0 | PRIMARY  |            1 | emp_no      | A         |      299423 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
   +-----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
   1 row in set (0.00 sec)

   mysql> 
   ```

   Remember that indexes for primary keys are created automatically, as we can see above. An index has already been created for the primary key, **emp_no**. If we think about this, this makes sense because each employee number is unique to the employee, with no NULL values.

2. Now, let’s say we wanted to see all the information about employees who were hired on or after January 1, 2000. We can do that with the query:

   ```
   1. 1
   1. SELECT * FROM employees WHERE hire_date >= '2000-01-01';
   ```
   Copied!

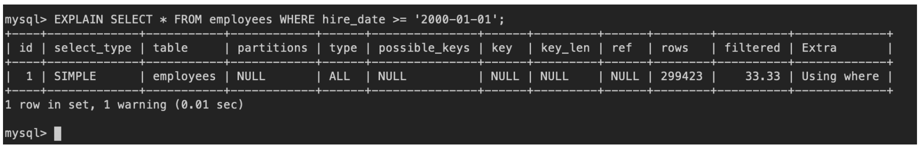
   

   ```
   mysql> SELECT * FROM employees WHERE hire_date >= '2000-01-01';
   +--------+------------+-------------+------------+--------+------------+
   | emp_no | birth_date | first_name  | last_name  | gender | hire_date  |
   +--------+------------+-------------+------------+--------+------------+
   |  47291 | 1960-09-09 | Ulf         | Flexer     | M      | 2000-01-12 |
   |  60134 | 1964-04-21 | Seshu       | Rathonyi   | F      | 2000-01-02 |
   |  72329 | 1953-02-09 | Randi       | Luit       | F      | 2000-01-02 |
   | 108201 | 1955-04-14 | Mariangiola | Boreale    | M      | 2000-01-01 |
   | 205048 | 1960-09-12 | Ennio       | Alblas     | F      | 2000-01-06 |
   | 222965 | 1959-08-07 | Volkmar     | Perko      | F      | 2000-01-13 |
   | 226633 | 1958-06-10 | Xuejun      | Benzmuller | F      | 2000-01-04 |
   | 227544 | 1954-11-17 | Shahab      | Demeyer    | M      | 2000-01-08 |
   | 422990 | 1953-04-09 | Jaana       | Verspoor   | F      | 2000-01-11 |
   | 424445 | 1953-04-27 | Jeong       | Boreale    | M      | 2000-01-03 |
   | 428377 | 1957-05-09 | Yucai       | Gerlach    | M      | 2000-01-23 |
   | 463807 | 1964-06-12 | Bikash      | Covnot     | M      | 2000-01-28 |
   | 499553 | 1954-05-06 | Hideyuki    | Delgrande  | F      | 2000-01-22 |
   +--------+------------+-------------+------------+--------+------------+
   13 rows in set (0.17 sec)
   ```

   As we can see, the 13 rows returned took about 0.17 seconds to execute. That may not seem like a long time with this table, but keep in mind that with larger tables, this time can vary greatly.

3. With the EXPLAIN statement, we can check how many rows this query is scanning:

   ```
   1. 1
   1. EXPLAIN SELECT * FROM employees WHERE hire_date >= '2000-01-01';
   ```
   Copied!

   ```
   mysql> EXPLAIN SELECT * FROM employees WHERE hire_date >= '2000-01-01';
   +----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
   | id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra       |
   +----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
   |  1 | SIMPLE      | employees | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 299423 |    33.33 | Using where |
   +----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
   1 row in set, 1 warning (0.01 sec)

   mysql> 
   ```

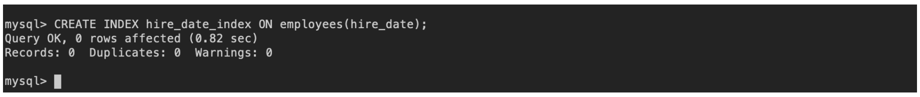
   This query results in a scan of 299,423 rows, which is nearly the entire table!

   By adding an index to the **hire_date** column, we’ll be able to reduce the query’s need to search through every entry of the table, instead only searching through what it needs.

4. You can add an index with the following:

   ```
   1. 1
   1. CREATE INDEX hire_date_index ON employees(hire_date);
   ```
   Copied!

   The CREATE INDEX command creates an index called **hire_date_index** on the table **employees** on column **hire_date**.



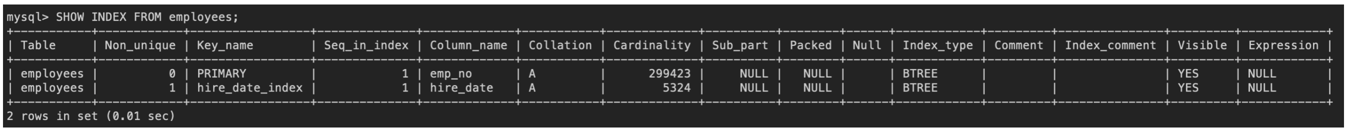
```
mysql> CREATE INDEX hire_date_index ON employees(hire_date);
Query OK, 0 rows affected (0.82 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> 
```

5. To check your index, you can use the SHOW INDEX command:

```
1. 1
1. SHOW INDEX FROM employees;
```

Copied!

Now you can see that we have both the **emp_no** index and **hire_date** index.



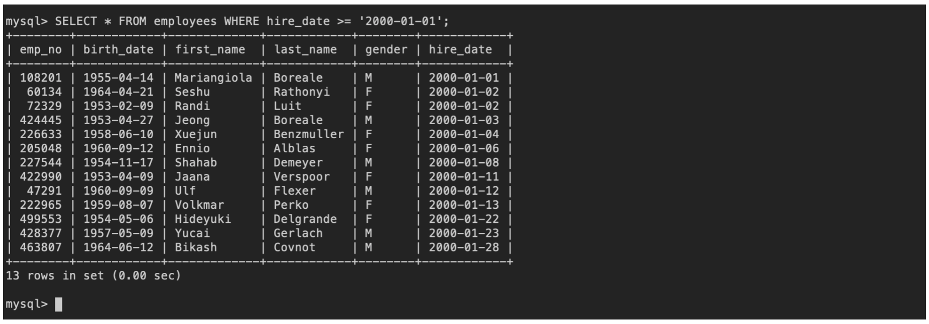
```
mysql> SHOW INDEX FROM employees;
+-----------+------------+-----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table     | Non_unique | Key_name        | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-----------+------------+-----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| employees |          0 | PRIMARY         |            1 | emp_no      | A         |      299423 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| employees |          1 | hire_date_index |            1 | hire_date   | A         |        5324 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-----------+------------+-----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
2 rows in set (0.01 sec)
```

With the index added,

6. Once more, let's select all the employees who were hired on or after January 1, 2000.

```
1. 1
1. SELECT * FROM employees WHERE hire_date >= '2000-01-01';
```

Copied!



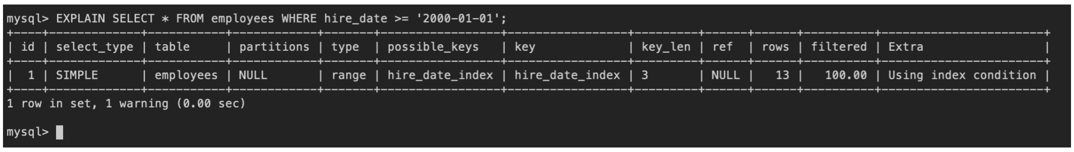
```
mysql> SELECT * FROM employees WHERE hire_date >= '2000-01-01';
+--------+------------+-------------+------------+--------+------------+
| emp_no | birth_date | first_name  | last_name  | gender | hire_date  |
+--------+------------+-------------+------------+--------+------------+
| 108201 | 1955-04-14 | Mariangiola | Boreale    | M      | 2000-01-01 |
|  60134 | 1964-04-21 | Seshu       | Rathonyi   | F      | 2000-01-02 |
|  72329 | 1953-02-09 | Randi       | Luit       | F      | 2000-01-02 |
| 424445 | 1953-04-27 | Jeong       | Boreale    | M      | 2000-01-03 |
| 226633 | 1958-06-10 | Xuejun      | Benzmuller | F      | 2000-01-04 |
| 205048 | 1960-09-12 | Ennio       | Alblas     | F      | 2000-01-06 |
| 227544 | 1954-11-17 | Shahab      | Demeyer    | M      | 2000-01-08 |
| 422990 | 1953-04-09 | Jaana       | Verspoor   | F      | 2000-01-11 |
|  47291 | 1960-09-09 | Ulf         | Flexer     | M      | 2000-01-12 |
| 222965 | 1959-08-07 | Volkmar     | Perko      | F      | 2000-01-13 |
| 499553 | 1954-05-06 | Hideyuki    | Delgrande  | F      | 2000-01-22 |
| 428377 | 1957-05-09 | Yucai       | Gerlach    | M      | 2000-01-23 |
| 463807 | 1964-06-12 | Bikash      | Covnot     | M      | 2000-01-28 |
+--------+------------+-------------+------------+--------+------------+
13 rows in set (0.00 sec)

mysql> 
```

The difference is quite evident! Rather than taking about 0.17 seconds to execute the query, it takes 0.00 seconds—almost no time at all.

7. We can use the EXPLAIN statement to see how many rows were scanned:

```
1. 1
1. EXPLAIN SELECT * FROM employees WHERE hire_date >= '2000-01-01';
```

Copied!



```
mysql> EXPLAIN SELECT * FROM employees WHERE hire_date >= '2000-01-01';
+----+-------------+-----------+------------+-------+-----------------+-----------------+---------+------+------+----------+-----------------------+
| id | select_type | table     | partitions | type  | possible_keys   | key             | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-----------+------------+-------+-----------------+-----------------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | employees | NULL       | range | hire_date_index | hire_date_index | 3       | NULL |   13 |   100.00 | Using index condition |
+----+-------------+-----------+------------+-------+-----------------+-----------------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)

mysql> 
```

Under **rows**, we can see that only the necessary 13 columns were scanned, leading to the improved performance.

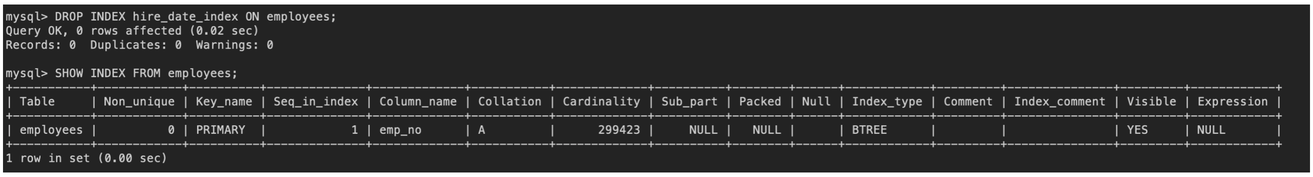
Under **Extra**, you can also see that it has been explicitly stated that the index was used, that index being **hire_date_index** based on the **possible_keys** column.

Now, if you want to remove the index, enter the following into the Terminal:

```
1. 1
1. DROP INDEX hire_date_index ON employees;
```

Copied!

This will remove the **hire_date_index** on the **employees** table. You can check with the SHOW INDEX command to confirm:



```
mysql> DROP INDEX hire_date_index ON employees;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> 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 | Visible | Expression |
+-----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| employees |          0 | PRIMARY  |            1 | emp_no      | A         |      299423 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-----------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
1 row in set (0.00 sec)
```

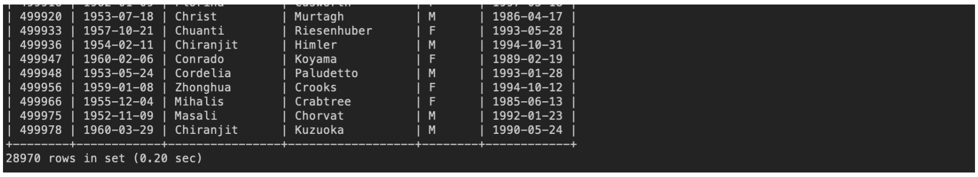
Exercise 4: Use an UNION ALL Clause

Sometimes, you might want to run a query using the OR operator with LIKE statements. In this case, using a UNION ALL clause can improve the speed of your query, particularly if the columns on both sides of the OR operator are indexed.

1. To start, let's run this query:

```
1. 1
1. SELECT * FROM employees WHERE first_name LIKE 'C%' OR last_name LIKE 'C%';
```

Copied!



```
| 499920 | 1953-07-18 | Christ          | Murtagh          | M      | 1986-04-17 |
| 499933 | 1957-10-21 | Chuanti         | Riesenhuber      | F      | 1993-05-28 |
| 499936 | 1954-02-11 | Chiranjit       | Himler           | M      | 1994-10-31 |
| 499947 | 1960-02-06 | Conrado         | Koyama           | F      | 1989-02-19 |
| 499948 | 1953-05-24 | Cordelia        | Paludetto        | M      | 1993-01-28 |
| 499956 | 1959-01-08 | Zhonghua        | Crooks           | F      | 1994-10-12 |
| 499966 | 1955-12-04 | Mihalis         | Crabtree         | F      | 1985-06-13 |
| 499975 | 1952-11-09 | Masali          | Chorvat          | M      | 1992-01-23 |
| 499978 | 1960-03-29 | Chiranjit       | Kuzuoka          | M      | 1990-05-24 |
+--------+------------+-----------------+------------------+--------+------------+
28970 rows in set (0.20 sec)
```

This query searches for first names or last names that start with "C". It returned 28,970 rows, taking about 0.20 seconds.

2. Check using the EXPLAIN command to see how many rows are being scanned!

▸ Hint (Click Here)
▸ Solution (Click Here)

Once more, we can see that almost all the rows are being scanned, so let's add indexes to both the **first_name** and **last_name** columns.

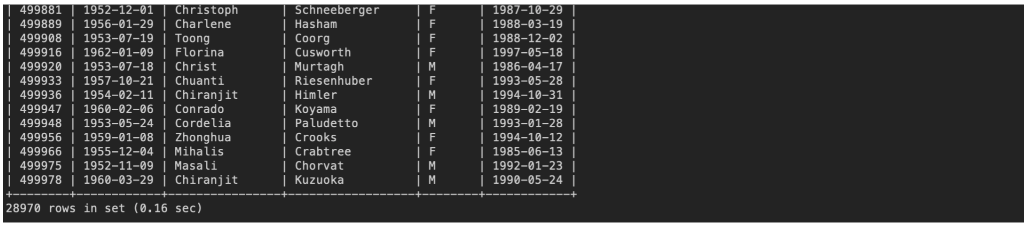
3. Try adding an index to both the **first_name** and **last_name** columns.

▸ Hint (Click Here)
▸ Solution (Click Here)

4. Great! With your indexes now in place, we can re-run the query:

```
1. 1
1. SELECT * FROM employees WHERE first_name LIKE 'C%' OR last_name LIKE 'C%';
```

Copied!

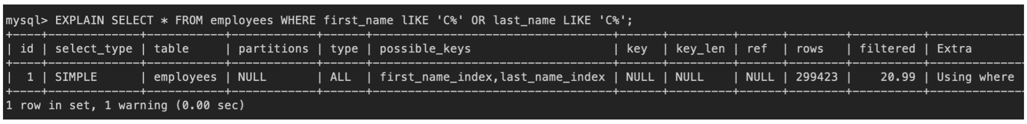


```
| 499881 | 1952-12-01 | Christoph       | Schneeberger     | F      | 1987-10-29 |
| 499889 | 1956-01-29 | Charlene        | Hasham           | F      | 1988-03-19 |
| 499908 | 1953-07-19 | Toong           | Coorg            | F      | 1988-12-02 |
| 499916 | 1962-01-09 | Florina         | Cusworth         | F      | 1997-05-18 |
| 499920 | 1953-07-18 | Christ          | Murtagh          | M      | 1986-04-17 |
| 499933 | 1957-10-21 | Chuanti         | Riesenhuber      | F      | 1993-05-28 |
| 499936 | 1954-02-11 | Chiranjit       | Himler           | M      | 1994-10-31 |
| 499947 | 1960-02-06 | Conrado         | Koyama           | F      | 1989-02-19 |
| 499948 | 1953-05-24 | Cordelia        | Paludetto        | M      | 1993-01-28 |
| 499956 | 1959-01-08 | Zhonghua        | Crooks           | F      | 1994-10-12 |
| 499966 | 1955-12-04 | Mihalis         | Crabtree         | F      | 1985-06-13 |
| 499975 | 1952-11-09 | Masali          | Chorvat          | M      | 1992-01-23 |
| 499978 | 1960-03-29 | Chiranjit       | Kuzuoka          | M      | 1990-05-24 |
+--------+------------+-----------------+------------------+--------+------------+
28970 rows in set (0.16 sec)
```

Let's also see how many rows are being scanned:

```
1. 1
1. EXPLAIN SELECT * FROM employees WHERE first_name LIKE 'C%' OR last_name LIKE 'C%';
```

Copied!



```
mysql> EXPLAIN SELECT * FROM employees WHERE first_name lIKE 'C%' OR last_name LIKE 'C%';
+----+-------------+-----------+------------+------+----------------------------------+------+---------+------+--------+----------+------------
| id | select_type | table     | partitions | type | possible_keys                    | key  | key_len | ref  | rows   | filtered | Extra
+----+-------------+-----------+------------+------+----------------------------------+------+---------+------+--------+----------+------------
|  1 | SIMPLE      | employees | NULL       | ALL  | first_name_index,last_name_index | NULL | NULL    | NULL | 299423 |    20.99 | Using where
+----+-------------+-----------+------------+------+----------------------------------+------+---------+------+--------+----------+------------
1 row in set, 1 warning (0.00 sec)
```

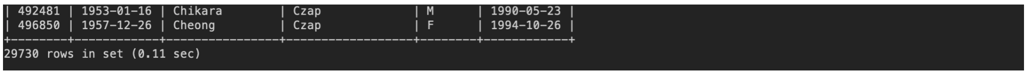
With indexes, the query still scans all the rows.

5. Let's use the UNION ALL clause to improve the performance of this query.

We can do this with the following:

1. 1
1. `SELECT * FROM employees WHERE first_name LIKE 'C%' UNION ALL SELECT * FROM employees WHERE last_name LIKE 'C%';`

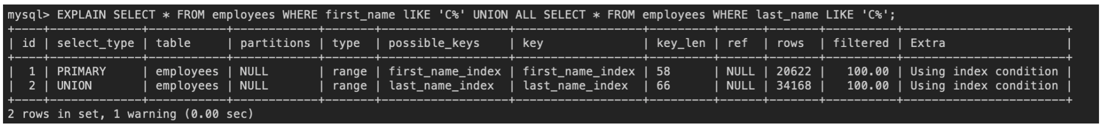
Copied!



```
| 492481 | 1953-01-16 | Chikara        | Czap             | M      | 1990-05-23 |
| 496850 | 1957-12-26 | Cheong         | Czap             | F      | 1994-10-26 |
+--------+------------+----------------+------------------+--------+------------+
29730 rows in set (0.11 sec)
```

As we can see, this query only takes 0.11 seconds to execute, running faster than when we used the OR operator.

Using the EXPLAIN statement, we can see why that might be:



```
mysql> EXPLAIN SELECT * FROM employees WHERE first_name lIKE 'C%' UNION ALL SELECT * FROM employees WHERE last_name LIKE 'C%';
+----+-------------+-----------+------------+-------+------------------+------------------+---------+------+-------+----------+-----------------------+
| id | select_type | table     | partitions | type  | possible_keys    | key              | key_len | ref  | rows  | filtered | Extra                 |
+----+-------------+-----------+------------+-------+------------------+------------------+---------+------+-------+----------+-----------------------+
|  1 | PRIMARY     | employees | NULL       | range | first_name_index | first_name_index | 58      | NULL | 20622 |   100.00 | Using index condition |
|  2 | UNION       | employees | NULL       | range | last_name_index  | last_name_index  | 66      | NULL | 34168 |   100.00 | Using index condition |
+----+-------------+-----------+------------+-------+------------------+------------------+---------+------+-------+----------+-----------------------+
2 rows in set, 1 warning (0.00 sec)
```

As the EXPLAIN statement reveals, there were two SELECT operations performed, with the total number of rows scanned sitting at 54,790. This is less than the original query that scanned the entire table and, as a result, the query performs faster.

Please note, if you choose to perform a leading wildcard search with an index, the entire table will still be scanned. You can see this yourself with the following query:

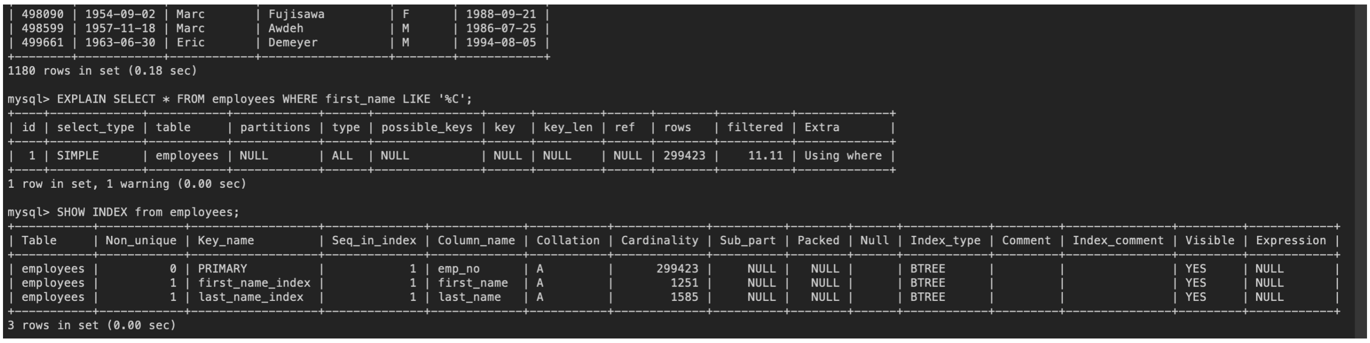
1. 1
1. `SELECT * FROM employees WHERE first_name LIKE '%C';`

Copied!

With this query, we want to find all the employees whose first names end with "C".

When checking with the EXPLAIN and SHOW INDEX statements, we can see that although we have an index on **first_name**, the index is not used and results in a search of the entire table.

Under the EXPLAIN statement's **possible_keys** column, we can see that this index has not been used as the entry is NULL.

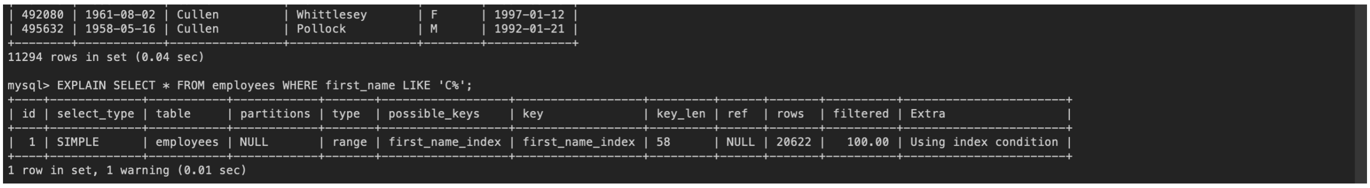


```
| 498090 | 1954-09-02 | Marc       | Fujisawa         | F      | 1988-09-21 |
| 498599 | 1957-11-18 | Marc       | Awdeh            | M      | 1986-07-25 |
| 499661 | 1963-06-30 | Eric       | Demeyer          | M      | 1994-08-05 |
+--------+------------+------------+------------------+--------+------------+
1180 rows in set (0.18 sec)

mysql> EXPLAIN SELECT * FROM employees WHERE first_name LIKE '%C';
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
|  1 | SIMPLE      | employees | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 299423 |    11.11 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> 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 | Visible | Expression |
+-----------+------------+------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| employees |          0 | PRIMARY          |            1 | emp_no      | A         |      299423 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| employees |          1 | first_name_index |            1 | first_name  | A         |        1251 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
| employees |          1 | last_name_index  |            1 | last_name   | A         |        1585 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL       |
+-----------+------------+------------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
3 rows in set (0.00 sec)
```

On the other hand, indexes do work with trailing wildcards, as seen with the following query that finds all employees whose first names begin with "C":

1. 1
1. `SELECT * FROM employees WHERE first_name LIKE 'C%';`

Copied!



```
| 492080 | 1961-08-02 | Cullen         | Whittlesey       | F      | 1997-01-12 |
| 495632 | 1958-05-16 | Cullen         | Pollock          | M      | 1992-01-21 |
+--------+------------+----------------+------------------+--------+------------+
11294 rows in set (0.04 sec)

mysql> EXPLAIN SELECT * FROM employees WHERE first_name LIKE 'C%';
+----+-------------+-----------+------------+-------+------------------+------------------+---------+------+-------+----------+-----------------------+
| id | select_type | table     | partitions | type  | possible_keys    | key              | key_len | ref  | rows  | filtered | Extra                 |
+----+-------------+-----------+------------+-------+------------------+------------------+---------+------+-------+----------+-----------------------+
|  1 | SIMPLE      | employees | NULL       | range | first_name_index | first_name_index | 58      | NULL | 20622 |   100.00 | Using index condition |
+----+-------------+-----------+------------+-------+------------------+------------------+---------+------+-------+----------+-----------------------+
1 row in set, 1 warning (0.01 sec)
```

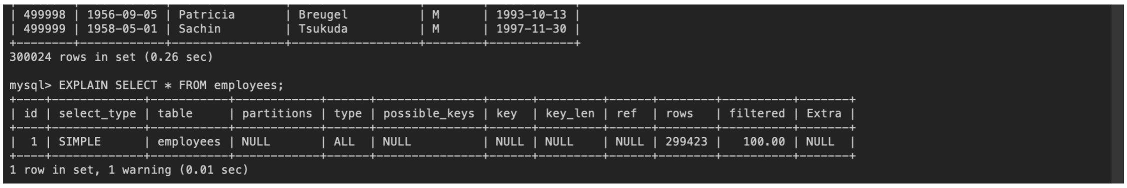
Under the EXPLAIN statement's **possible_keys** and **Extra** columns, we can see that the **first_name_index** is used. With only 20,622 rows scanned, the query performs better.

Exercise 5: Be SELECTive

In general, it's best practice to only select the columns that you need. For example, if you wanted to see the names and hire dates of the various employees, you could show that with the following query:

1. 1
1. `SELECT * FROM employees;`

Copied!



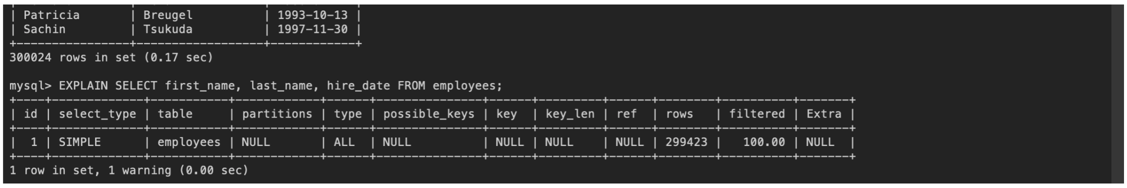
```
| 499998 | 1956-09-05 | Patricia       | Breugel          | M      | 1993-10-13 |
| 499999 | 1958-05-01 | Sachin         | Tsukuda          | M      | 1997-11-30 |
+--------+------------+----------------+------------------+--------+------------+
300024 rows in set (0.26 sec)

mysql> EXPLAIN SELECT * FROM employees;
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
|  1 | SIMPLE      | employees | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 299423 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
1 row in set, 1 warning (0.01 sec)
```

Notice how the query loads 300,024 rows in about 0.26 seconds. With the EXPLAIN statement, we can see that the entire table is being scanned, which makes sense because we are looking at all the entries.

If we, however, only wanted to see the names and hire dates, then we should select those columns:

1. 1
1. `SELECT first_name, last_name, hire_date FROM employees;`

Copied!



```
| Patricia       | Breugel          | 1993-10-13 |
| Sachin         | Tsukuda          | 1997-11-30 |
+----------------+------------------+------------+
300024 rows in set (0.17 sec)

mysql> EXPLAIN SELECT first_name, last_name, hire_date FROM employees;
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
|  1 | SIMPLE      | employees | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 299423 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

As you can see, this query was executed a little faster despite scanning the entire table as well.

Give this a try!

Practice Exercise 1

Let's take a look at the **salaries** table. What if we wanted to see how much each employee earns?

When running the query, keep in mind how long it takes the query to run and how many rows are scanned each time.

1. First, let's select all the rows and columns from this table.
   - ▸ Hint (Click Here)
   - ▸ Solution (Click Here)
2. Now, let's see if there's a way to optimize this query. Since we only want to see how much each employee earns, then we can just select a few columns instead of all of them. Which ones would you select?
   - ▸ Hint (Click Here)
   - ▸ Solution (Click Here)

Practice Exercise 2

Let's take a look at the **department** table. What if we wanted to see the employee and their corresponding title?

Practice by selecting only the necessary columns and run the query!

▸ Hint (Click Here)
▸ Solution (Click Here)

Conclusion

Congratulations! Now, not only can you now identify common causes to slow queries, but you can resolve them by applying the knowledge that you have gained in this lab. Equipped with this problem-solving skill, you will be able to improve your queries performance, even in large databases.

Author(s)

Kathy An

Other Contributor(s)

Rav Ahuja

Changelog

Date	Version	Changed by	Change Description
2021-10-05	1.0	Kathy An	Created initial version