**ScienceQtech Employee Performance Mapping**

**SQL PROJECT**

Table of Contents

[1. Create a database named employee 5](#_Toc145264561)

[1.1. Syntax: 5](#_Toc145264562)

[1.2. Result: 6](#_Toc145264563)

[1.3. Output: 6](#_Toc145264564)

[1.4. Import data\_science\_team.csv into the employee database from the given resources 6](#_Toc145264565)

[1.4.1. Steps: 7](#_Toc145264566)

[1.4.2. Result: 11](#_Toc145264567)

[1.4.3. Output: 11](#_Toc145264568)

[1.5. Import proj\_table.csv into the employee database from the given resources 11](#_Toc145264569)

[1.5.1. Steps: 11](#_Toc145264570)

[1.5.2. Result: 12](#_Toc145264571)

[1.5.3. Output: 12](#_Toc145264572)

[1.6. Import emp\_record\_table.csv into the employee database from the given resources 12](#_Toc145264573)

[1.6.1. Steps: 12](#_Toc145264574)

[1.6.2. Result: 12](#_Toc145264575)

[1.6.3. Output: 13](#_Toc145264576)

[2. Create an ER diagram for the given employee database 13](#_Toc145264577)

[2.1. ER Diagram: 13](#_Toc145264578)

[3. Write a query to fetch EMP\_ID, FIRST\_NAME, LAST\_NAME, GENDER, and DEPARTMENT from the employee record table, and make a list of employees and details of their department 13](#_Toc145264579)

[3.1. Syntax: 13](#_Toc145264580)

[3.2. Result: 14](#_Toc145264581)

[3.3. Output: 14](#_Toc145264582)

[4. Write a query to fetch EMP\_ID, FIRST\_NAME, LAST\_NAME, GENDER, DEPARTMENT, and EMP\_RATING if the EMP\_RATING is 14](#_Toc145264583)

[4.1. If Emp\_rating is Less than two 14](#_Toc145264584)

[4.1.1. Syntax: 14](#_Toc145264585)

[4.1.2. Result: 15](#_Toc145264586)

[4.1.3. Output: 15](#_Toc145264587)

[4.2. If Emp\_rating is Greater than four 15](#_Toc145264588)

[4.2.1. Syntax: 15](#_Toc145264589)

[4.2.2. Result: 16](#_Toc145264590)

[4.2.3. Output: 16](#_Toc145264591)

[4.3. If Emp\_rating is Between two and four 16](#_Toc145264592)

[4.3.1. Syntax: 16](#_Toc145264593)

[4.3.2. Result: 17](#_Toc145264594)

[4.3.3. Output: 17](#_Toc145264595)

[5. Write a query to concatenate the FIRST\_NAME and the LAST\_NAME of employees in the Finance department from the employee table and then give the resultant column alias as NAME 17](#_Toc145264596)

[5.1. Syntax: 17](#_Toc145264597)

[5.2. Result: 18](#_Toc145264598)

[5.3. Output: 18](#_Toc145264599)

[6. Write a query to list only those employees who have someone reporting to them. Also, show the number of reporters (including the President) 18](#_Toc145264600)

[6.1. Syntax: 18](#_Toc145264601)

[6.1.1. Error Handling: 18](#_Toc145264602)

[6.2. Result: 21](#_Toc145264603)

[6.3. Output: 21](#_Toc145264604)

[7. Write a query to list down all the employees from the healthcare and finance departments using union. 21](#_Toc145264605)

[7.1. Syntax: 21](#_Toc145264606)

[7.2. Result: 21](#_Toc145264607)

[7.3. Output: 22](#_Toc145264608)

[8. Write a query to list down employee details such as EMP\_ID, FIRST\_NAME, LAST\_NAME, ROLE, DEPARTMENT, and EMP\_RATING grouped by dept. Also include the respective employee rating along with the max emp rating for the department 22](#_Toc145264609)

[8.1. Syntax: 22](#_Toc145264610)

[8.2. Result: 23](#_Toc145264611)

[8.3. Output: 23](#_Toc145264612)

[9. Write a query to calculate the minimum and the maximum salary of the employees in each role. 23](#_Toc145264613)

[9.1. Syntax: 23](#_Toc145264614)

[9.2. Result: 23](#_Toc145264615)

[9.3. Output: 24](#_Toc145264616)

[10. Write a query to assign ranks to each employee based on their experience. 24](#_Toc145264617)

[10.1. Syntax: 24](#_Toc145264618)

[10.2. Result: 25](#_Toc145264619)

[10.3. Output: 25](#_Toc145264620)

[11. Write a query to create a view that displays employees in various countries whose salary is more than six thousand. 25](#_Toc145264621)

[11.1. Syntax: 25](#_Toc145264622)

[11.2. Result: 25](#_Toc145264623)

[11.3. Output: 26](#_Toc145264624)

[12. Write a nested query to find employees with experience of more than ten years. 26](#_Toc145264625)

[12.1. Syntax: 26](#_Toc145264626)

[12.2. Result: 26](#_Toc145264627)

[12.3. Output: 27](#_Toc145264628)

[13. Write a query to create a stored procedure to retrieve the details of the employees whose experience is more than three years. 27](#_Toc145264629)

[13.1. Syntax: 27](#_Toc145264630)

[13.2. Result: 28](#_Toc145264631)

[13.3. Output: 28](#_Toc145264632)

[14. Write a query using stored functions in the project table to check whether the job profile assigned to each employee in the data science team matches the organization’s set standard 28](#_Toc145264633)

[14.1. Syntax: 29](#_Toc145264634)

[14.2. Result: 31](#_Toc145264635)

[14.2.1. Result for emp\_id available in the database: 31](#_Toc145264636)

[14.2.2. Result for emp\_id not available in the database: 31](#_Toc145264637)

[14.3. Output: 31](#_Toc145264638)

[14.3.1. Output for emp\_id available in the database:: 31](#_Toc145264639)

[14.3.2. Output for emp\_id not available in the database:: 32](#_Toc145264640)

[15. Create an index to improve the cost and performance of the query to find the employee whose FIRST\_NAME is ‘Eric’ in the employee table after checking the execution plan 32](#_Toc145264641)

[15.1. Syntax: 32](#_Toc145264642)

[15.1.1. Before creating index: 32](#_Toc145264643)

[15.1.2. Creating index: 32](#_Toc145264644)

[15.1.3. After creating index: 33](#_Toc145264645)

[15.2. Result: 33](#_Toc145264646)

[15.2.1. Result before creating Index: 33](#_Toc145264647)

[15.2.2. Result after creating Index: 33](#_Toc145264648)

[15.3. Output: 34](#_Toc145264649)

[16. Write a query to calculate the bonus for all the employees, based on their ratings and salaries (Use the formula: 5% of salary \* employee rating) 34](#_Toc145264650)

[16.1. Syntax: 34](#_Toc145264651)

[16.2. Result: 34](#_Toc145264652)

[16.3. Output: 35](#_Toc145264653)

[17. Write a query to calculate the average salary distribution based on the continent and country. Take data from the employee record table 35](#_Toc145264654)

[17.1. Syntax: 35](#_Toc145264655)

[17.2. Result: 36](#_Toc145264656)

[17.3. Output: 36](#_Toc145264657)

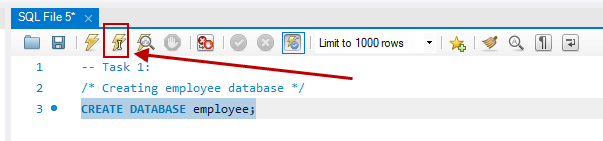
# Create a database named employee

In order to help HR department to facilitate better understanding and finalize the employee performance mapping using the ratings provided by the managers, we need to create a Database “Employee in MySQL

## Syntax:

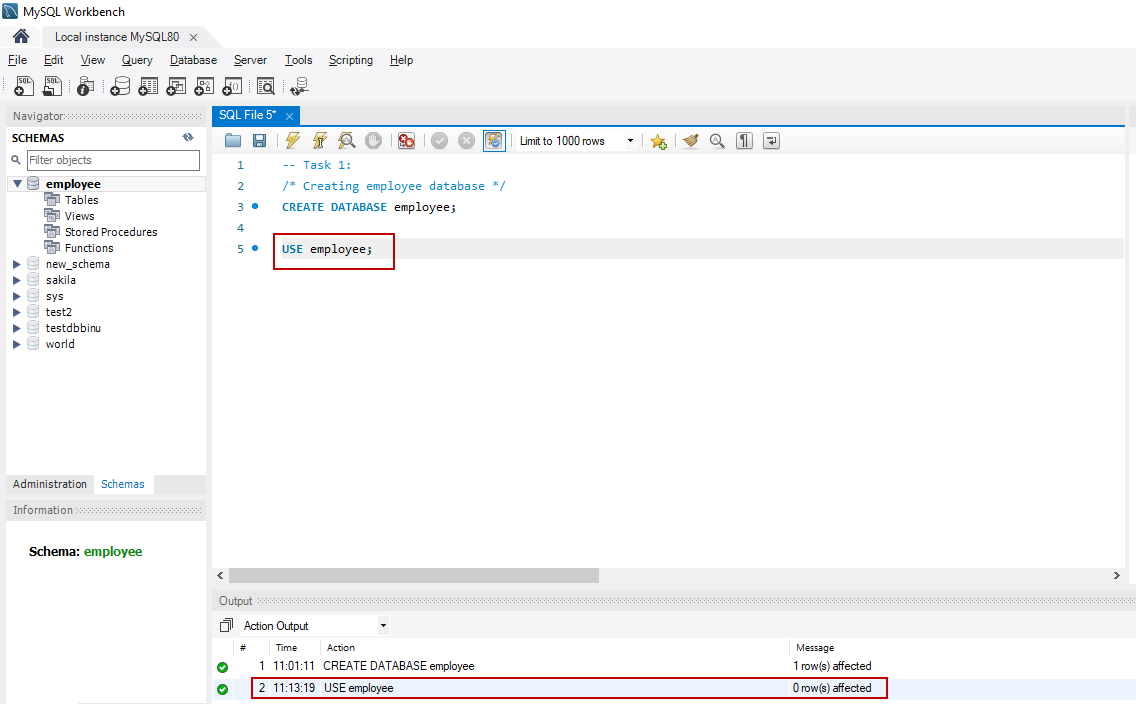
Use the below syntax to create the database “Employee” and click on “Execute the statement under keyboard cursor”

**CREATE DATABASE employee;**

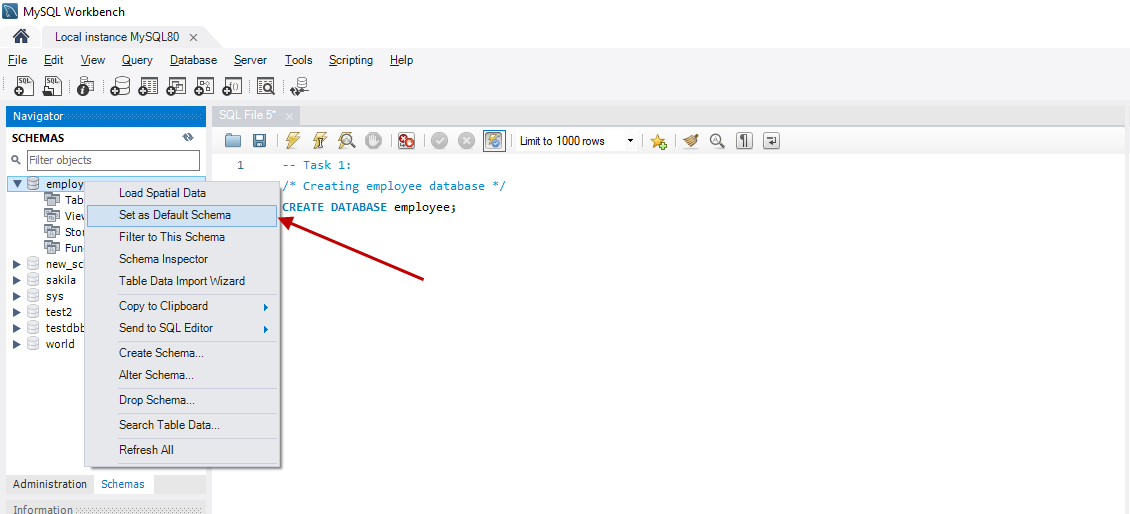


Enter the below syntax and execute or right click on the “employee” database and select “Set as Default Schema” for SQL to choose “employee” database by default for all future execution

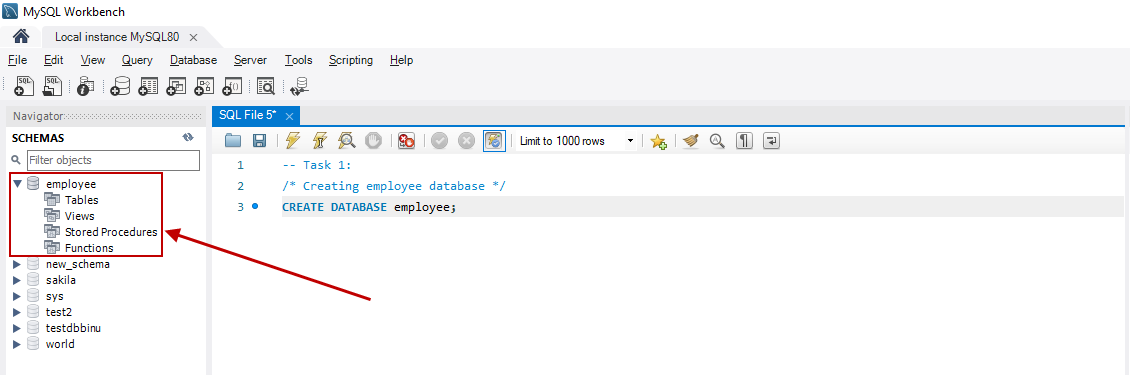
**USE employee;**



**(OR)**

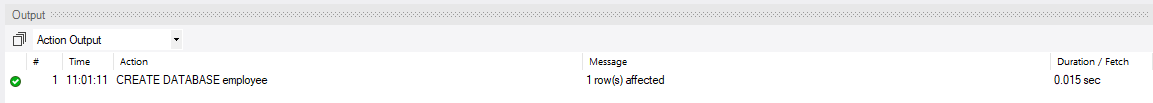


## Result:



## Output:

“employee” database successfully created and will be used as default database



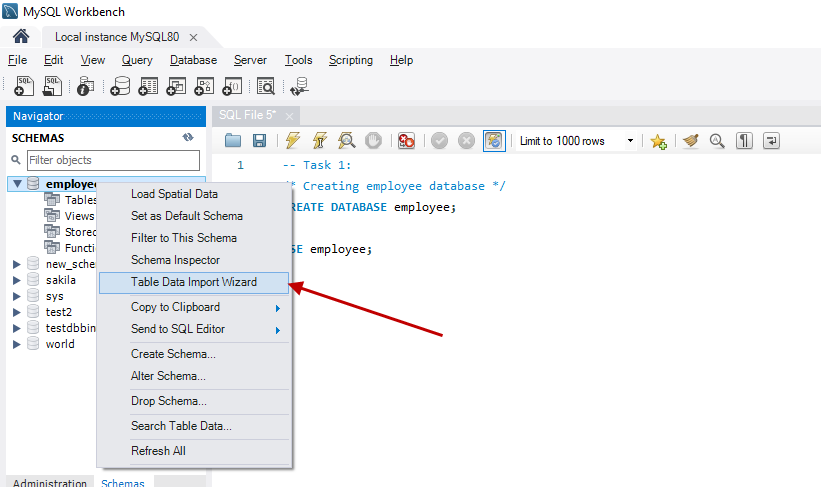
**Then**

## Import data\_science\_team.csv into the employee database from the given resources

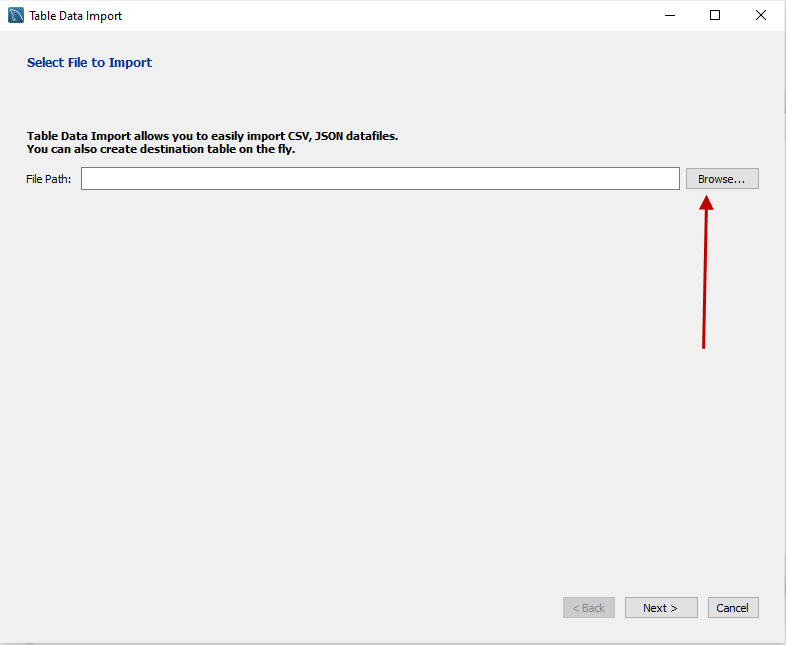
The below process is to import Data\_Science\_Team. Csv file into the “employee” database created to further use.

### Steps:

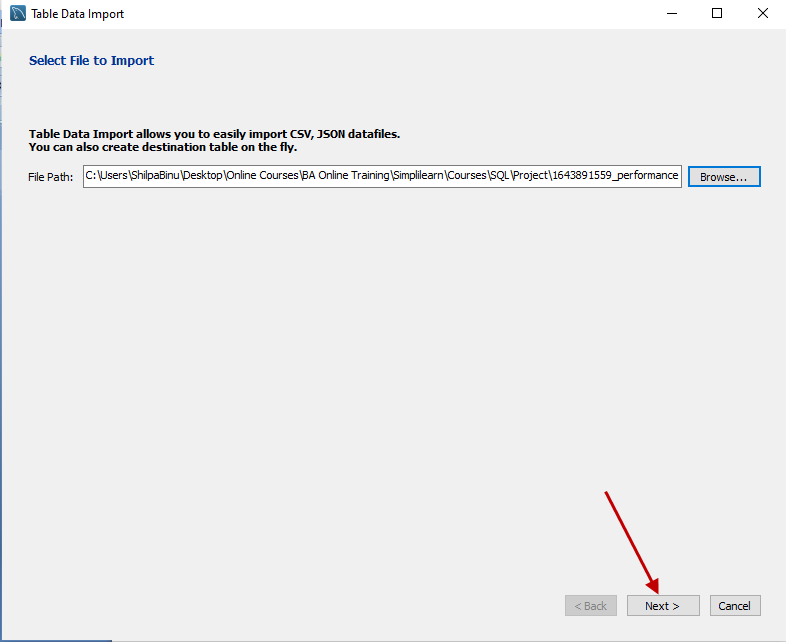
Right click on the “employee” database and choose “Table Data Import Wizard” as shown below



The below pop-up window “Table Data Import” will appear, then click on “Browse” to select the file data\_science\_team.csv from the right location where the .csv file is saved.

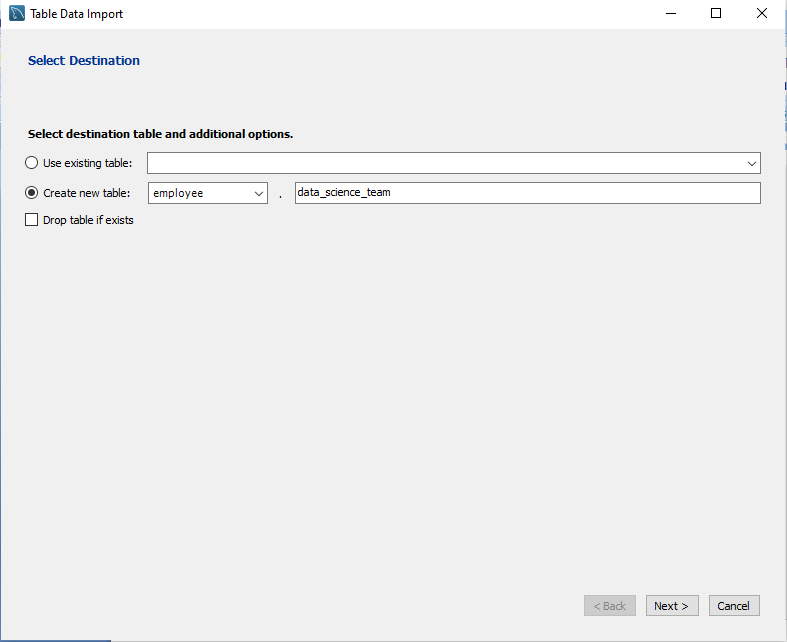


Select the file and click on “Next”

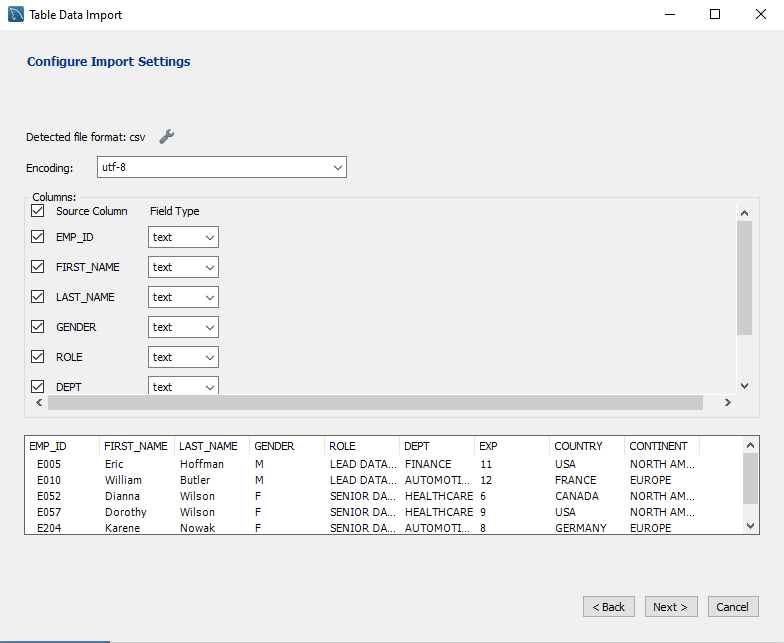


Select “Create new table” and click “Next”

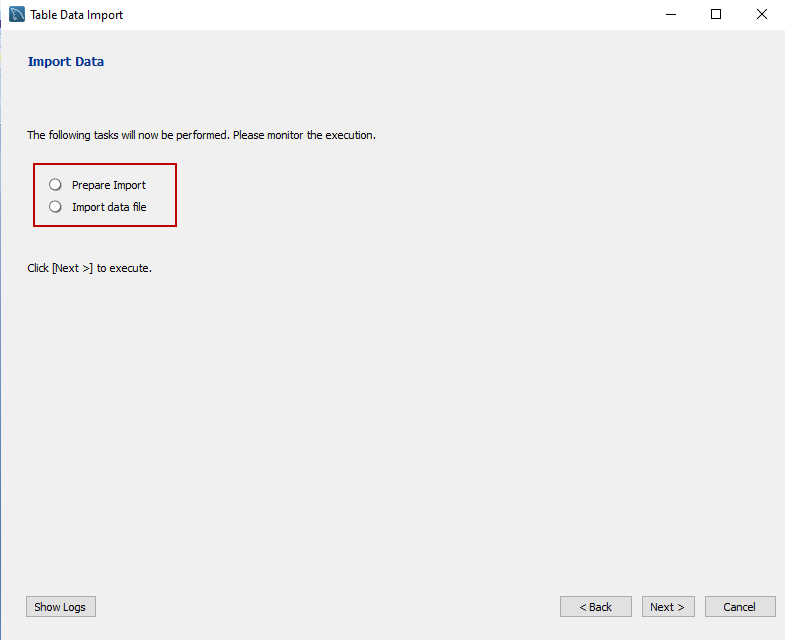
Note: Please make sure the database is “employee”.



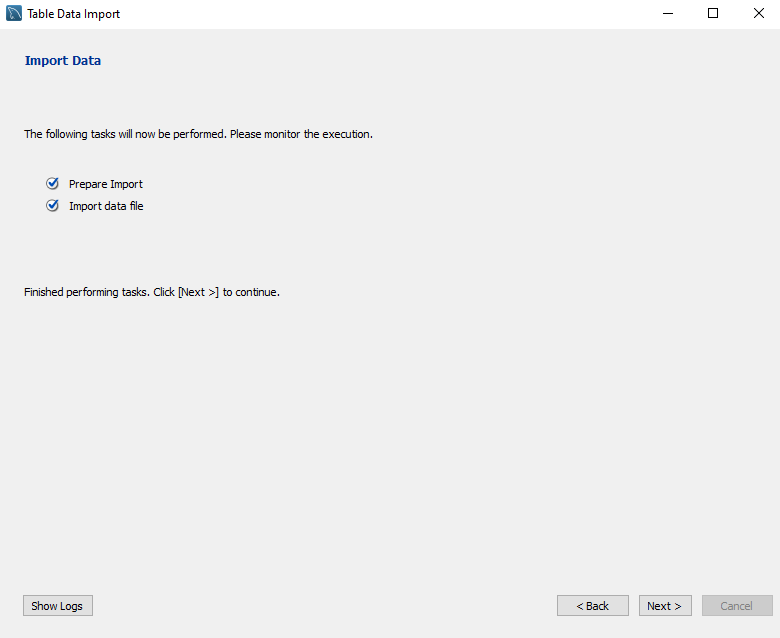
In the “Configure Import Settings” window verify and make note of all the datatypes from the table which will be imported and click on “Next”



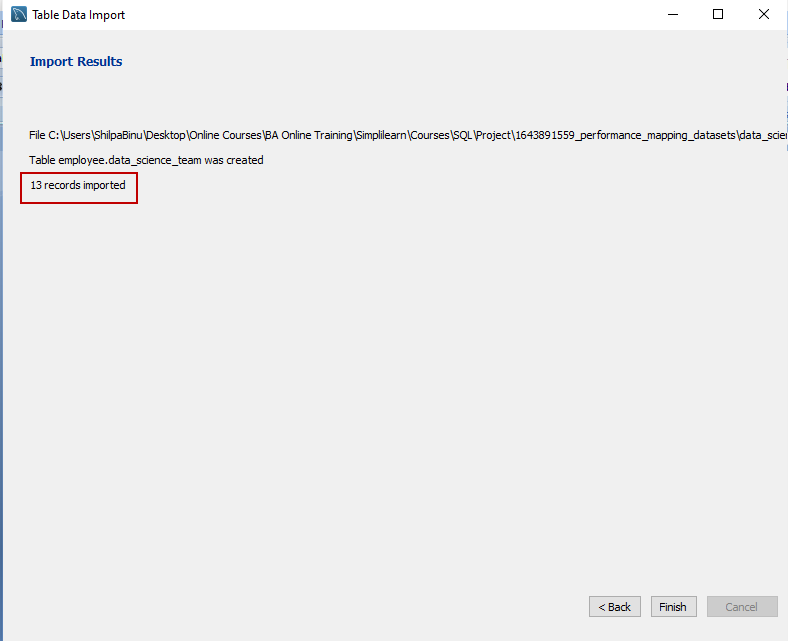
In the “Import Data” window we can see the steps which will be performed while importing the table and click “Next” to start importing the table



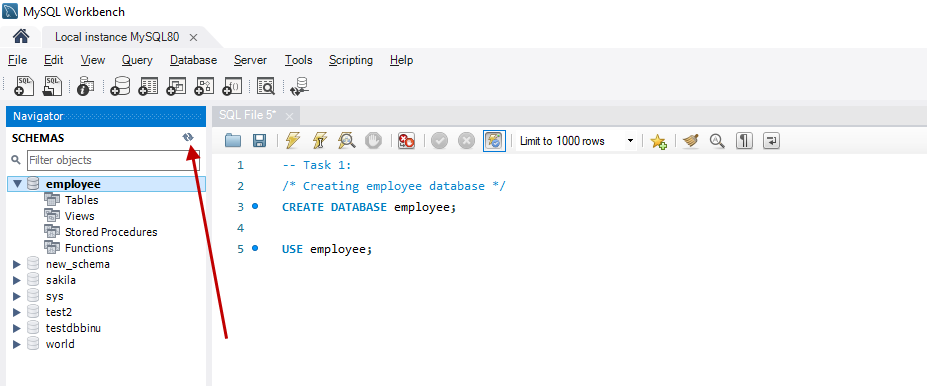
Click “Next”



Click “Finish”

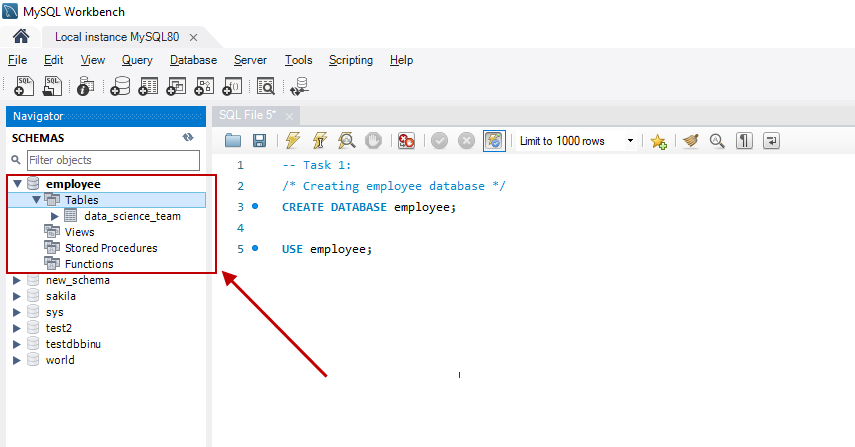


Click the “Refresh” button to see the result

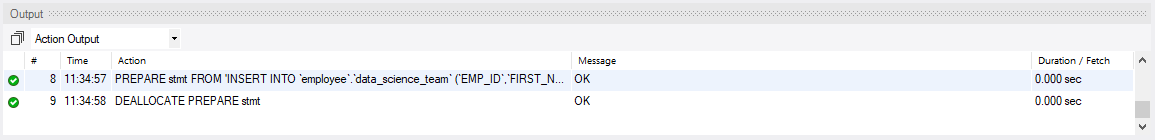


### Result:

Data\_science\_team.csv table with 13 records is imported successfully



### Output:



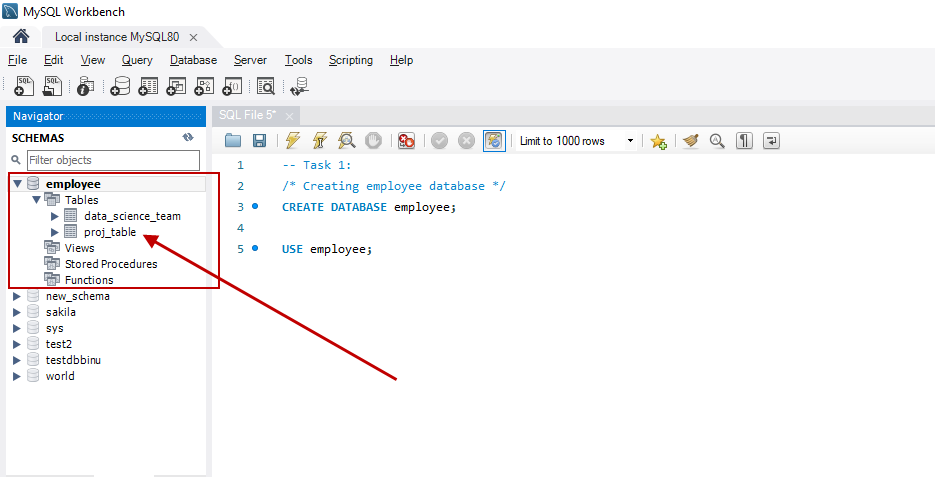
## Import proj\_table.csv into the employee database from the given resources

### Steps:

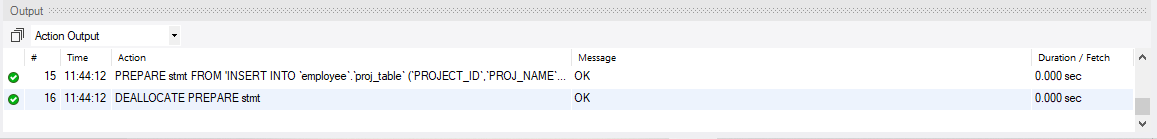
Please follow the steps mentioned in 1.4.1 section above to import the table proj\_table\_.csv

### Result:

Proj\_table.csv table with 6 records is imported successfully



### Output:



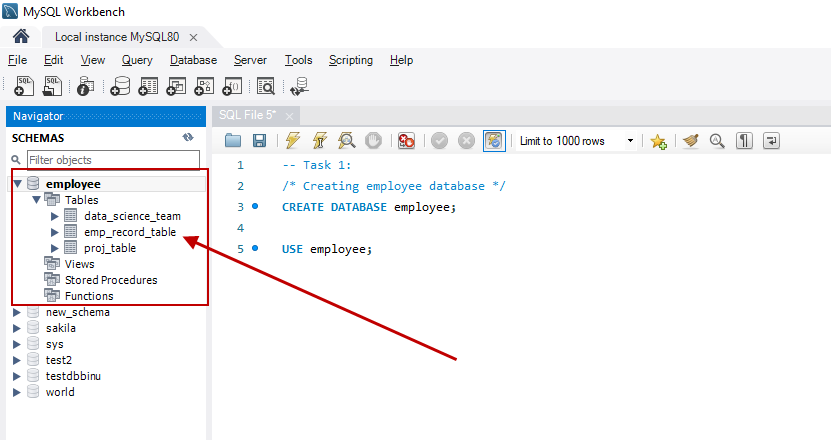
## Import emp\_record\_table.csv into the employee database from the given resources

### Steps:

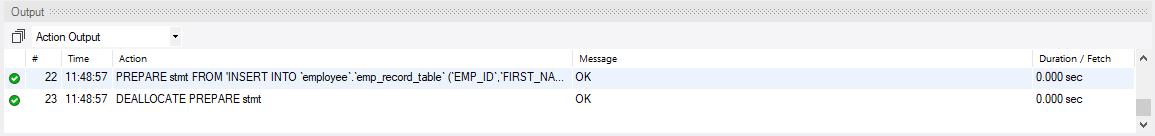
Please follow the steps mentioned in 1.4.1 section above to import the table emp\_record\_table\_.csv

### Result:

Emp\_record\_table.csv table with 19 records is imported successfully

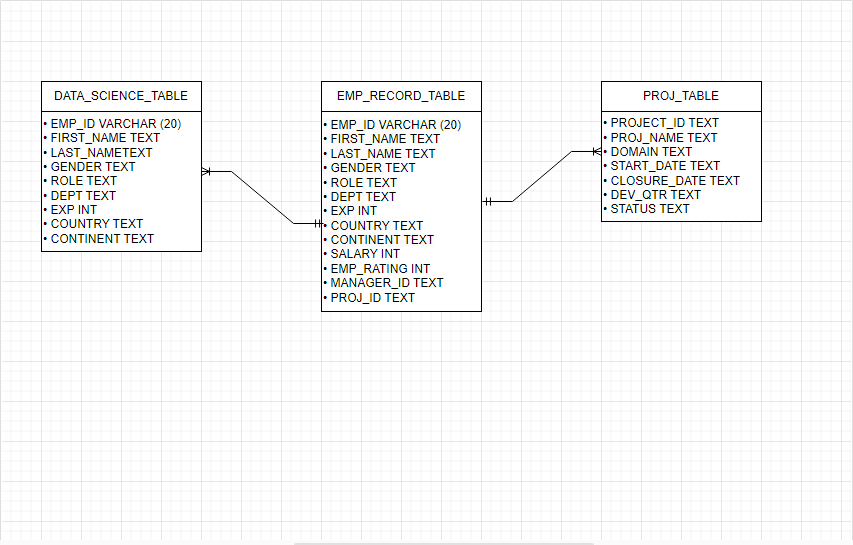


### Output:



# Create an ER diagram for the given employee database

## ER Diagram:



# Write a query to fetch EMP\_ID, FIRST\_NAME, LAST\_NAME, GENDER, and DEPARTMENT from the employee record table, and make a list of employees and details of their department

In this task, we will be able to fetch emp\_id, first\_name, last\_name, gender, and department from the employee\_record\_table and also group the search result by department by using “ORDER BY” clause

## Syntax:

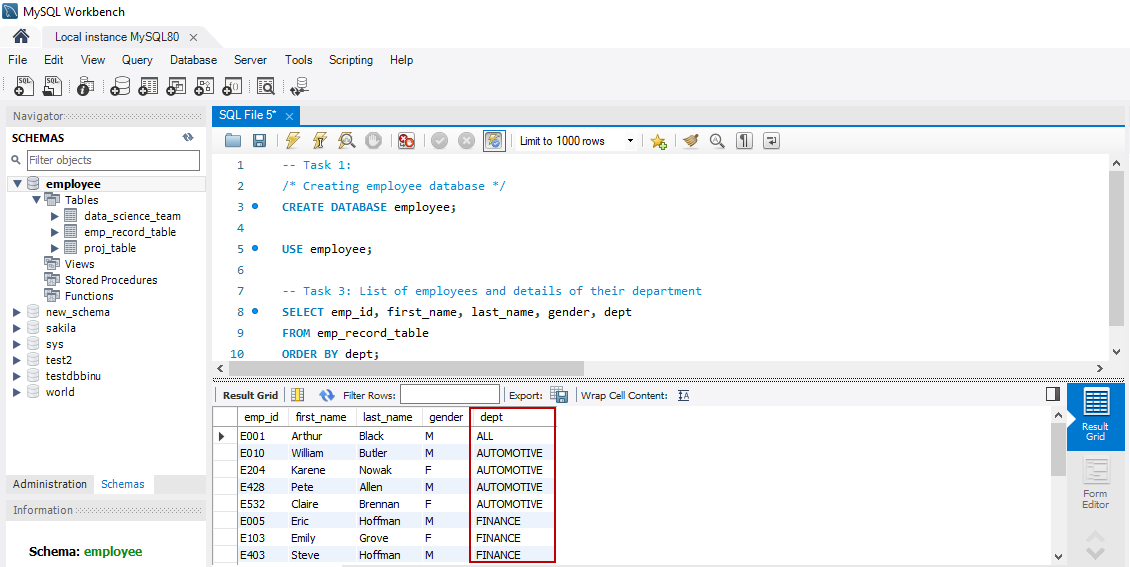
Enter the syntax as shown below and click “Execute the statement under keyboard cursor”

**SELECT EMP\_ID, FIRST\_NAME, LAST\_NAME, GENDER, DEPT**

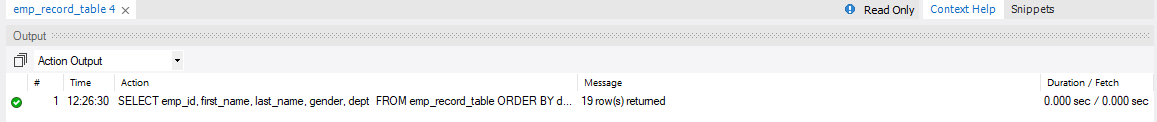
**FROM emp\_record\_table**

**ORDER BY dept;**

## Result:



## Output:



# Write a query to fetch EMP\_ID, FIRST\_NAME, LAST\_NAME, GENDER, DEPARTMENT, and EMP\_RATING if the EMP\_RATING is

In this step, we will be able to fetch emp\_id, first\_name, Last\_name, gender, department based on the condition provided below for emp\_rating from the emp\_record\_table

## If Emp\_rating is Less than two

### Syntax:

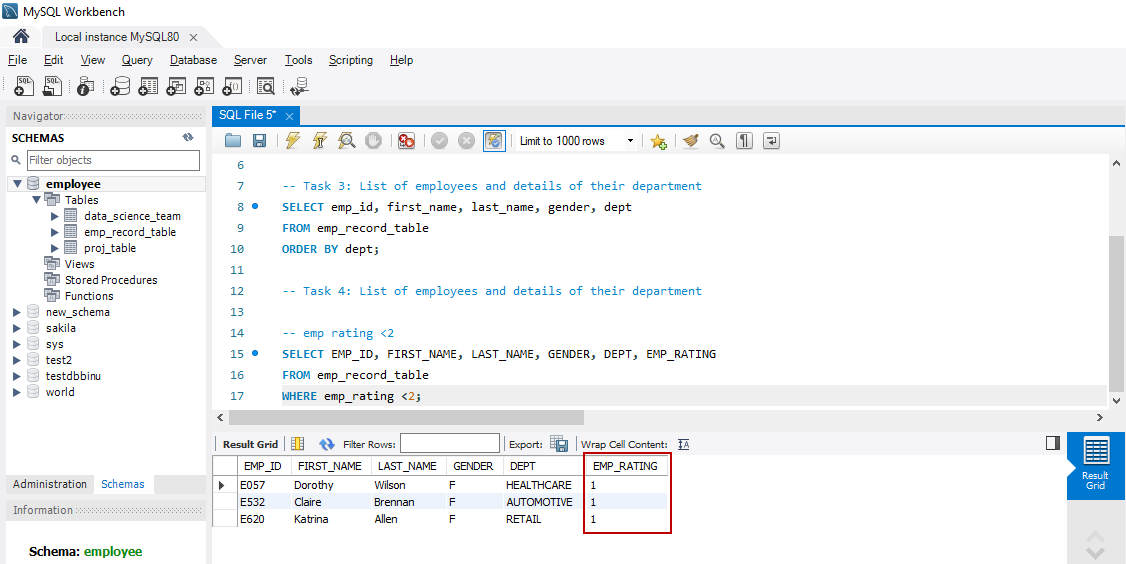
Use the below syntax to obtain the list of employees whose emp\_rating is less than 2 from the emp\_record\_table and click “Execute the statement under keyboard cursor”

**SELECT EMP\_ID, FIRST\_NAME, LAST\_NAME, GENDER, DEPT, EMP\_RATING**

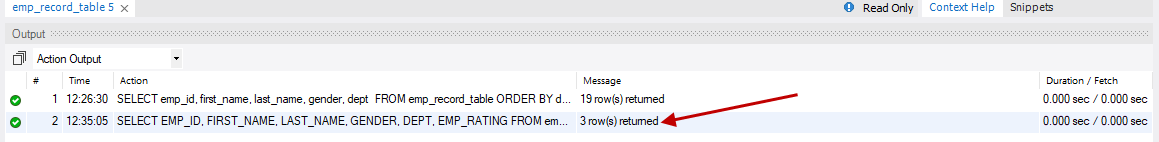
**FROM emp\_record\_table**

**WHERE emp\_rating <2;**

### Result:



### Output:



## If Emp\_rating is Greater than four

### Syntax:

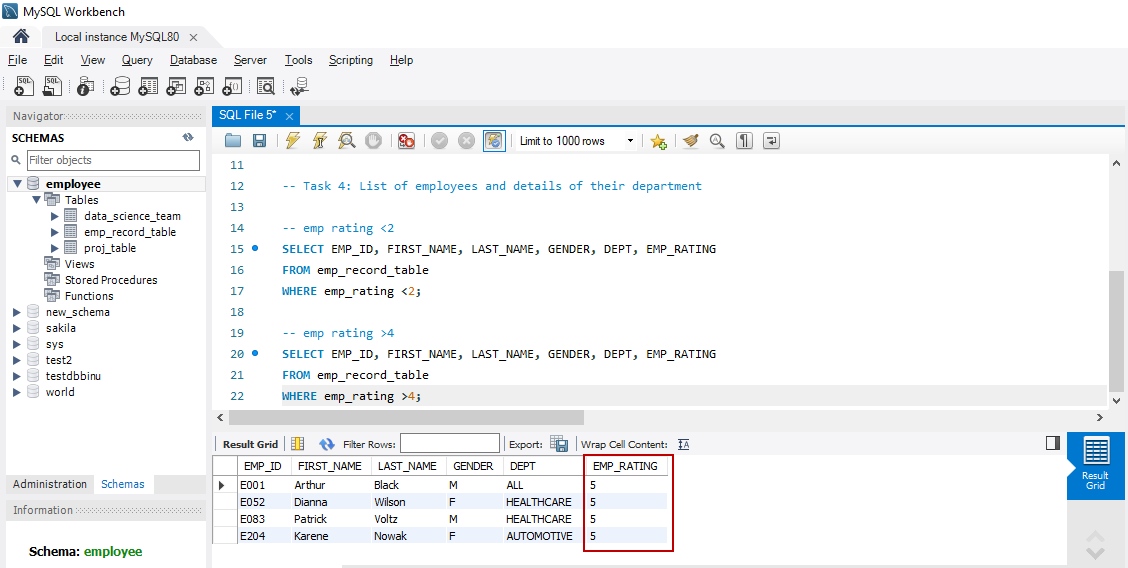
Use the below syntax to obtain the list of employees whose emp\_rating is greater than 4 from the emp\_record\_table and click “Execute the statement under keyboard cursor”

**SELECT EMP\_ID, FIRST\_NAME, LAST\_NAME, GENDER, DEPT, EMP\_RATING**

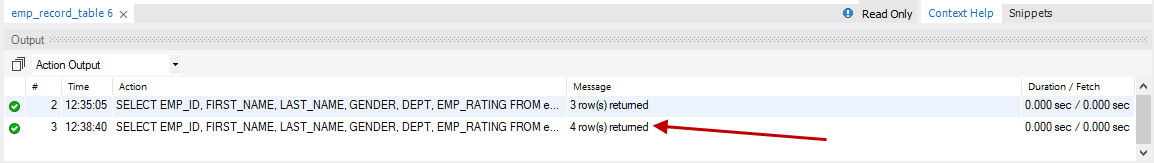
**FROM emp\_record\_table**

**WHERE emp\_rating >4;**

### Result:



### Output:



## If Emp\_rating is Between two and four

### Syntax:

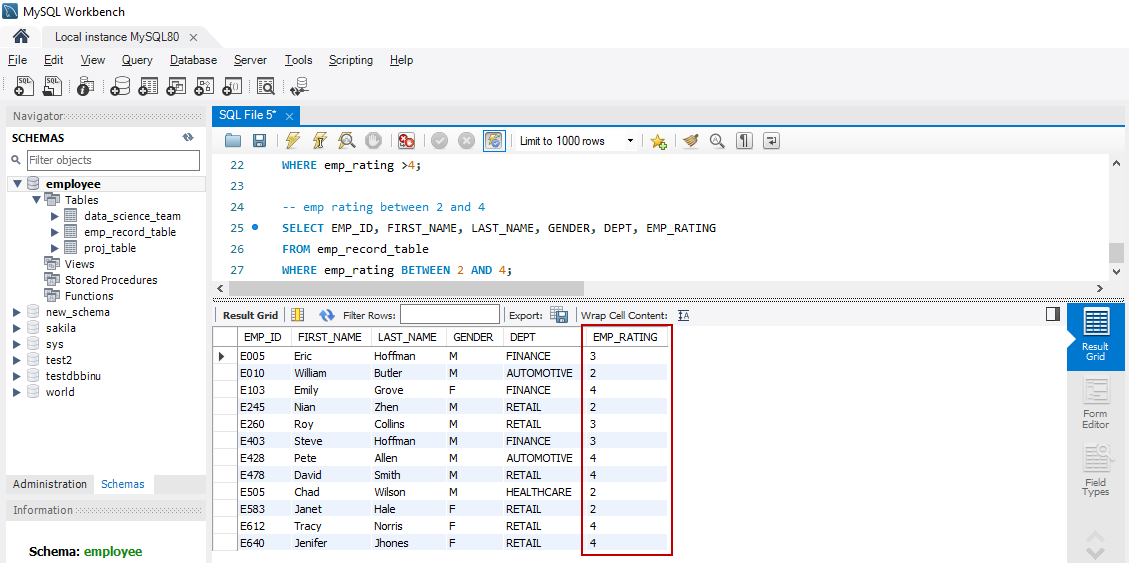
Use the below syntax to obtain the list of employees whose emp\_rating is between 2 and 4 from the emp\_record\_table and click “Execute the statement under keyboard cursor”

**SELECT EMP\_ID, FIRST\_NAME, LAST\_NAME, GENDER, DEPT, EMP\_RATING**

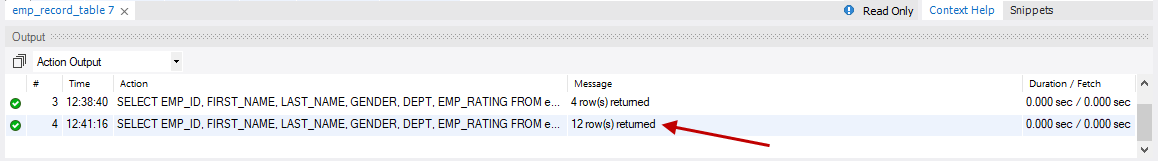
**FROM emp\_record\_table**

**WHERE emp\_rating BETWEEN 2 AND 4;**

### Result:



### Output:



# Write a query to concatenate the FIRST\_NAME and the LAST\_NAME of employees in the Finance department from the employee table and then give the resultant column alias as NAME

In this task, we will be able to concatenate first\_name and last\_name of employees in the finance department only from the emp\_record\_table and name the resultant column as “name”

## Syntax:

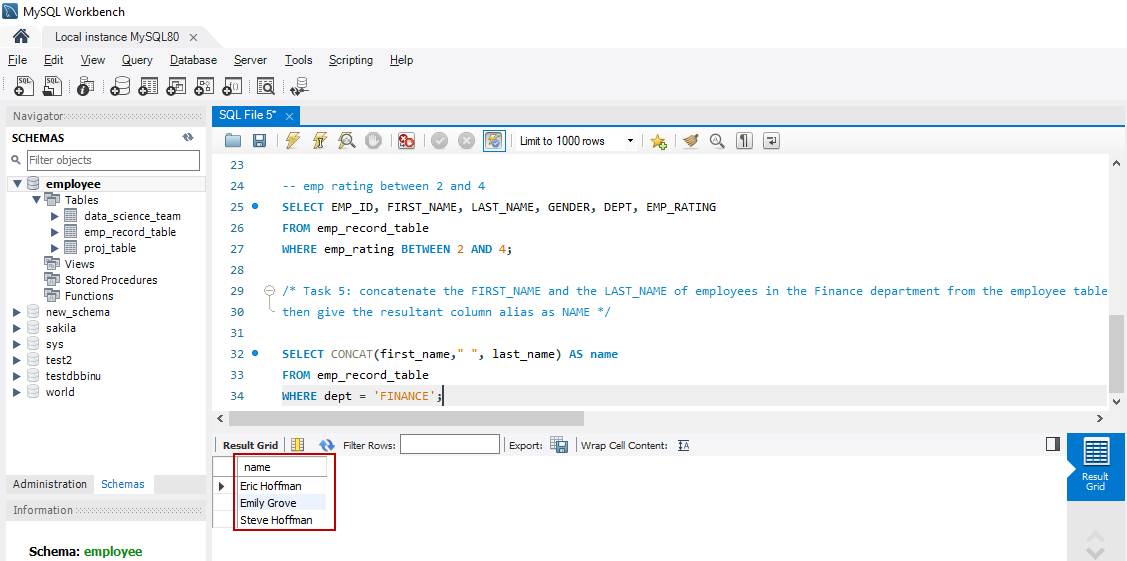
Use the below syntax to concatenate the first\_name and last\_name columns in emp\_record\_table

**SELECT CONCAT(first\_name," ", last\_name) AS name**

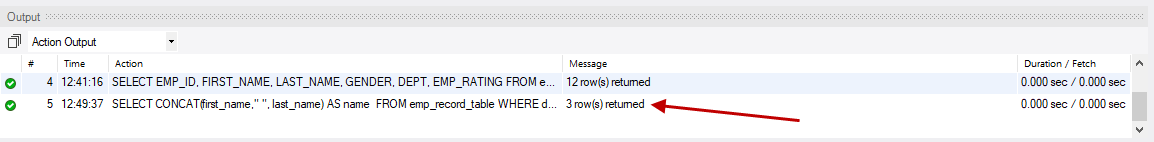
**FROM emp\_record\_table**

**WHERE dept = 'FINANCE';**

## Result:



## Output:



# Write a query to list only those employees who have someone reporting to them. Also, show the number of reporters (including the President)

In this task, we will be able to list only those employees who have someone reporting to them and also showing the number of reportees (including the President)

## Syntax:

Use the below syntax to obtain the list of employees who have someone reporting to them including the President

**SELECT m.emp\_id, m.first\_name MGR\_NAME, COUNT(e.first\_name) AS reportees**

**FROM emp\_record\_table e, emp\_record\_table m**

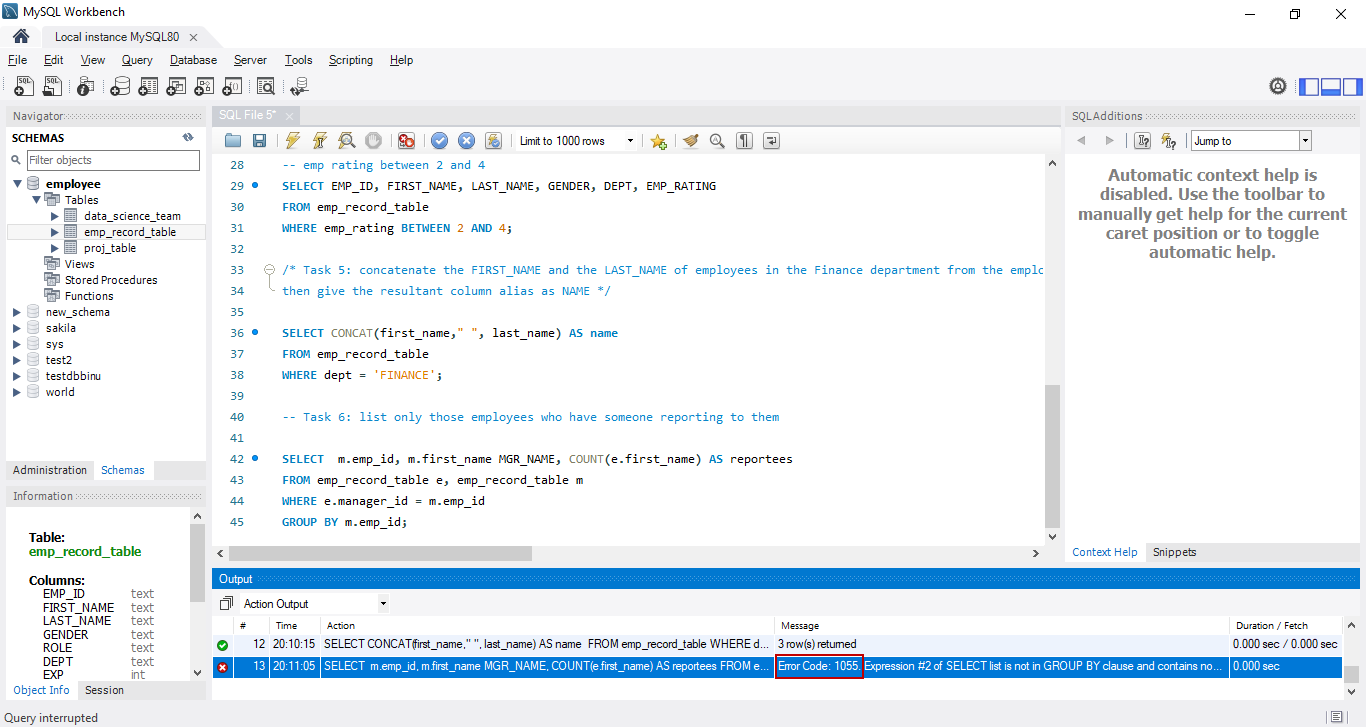
**WHERE e.manager\_id = m.emp\_id**

**GROUP BY m.emp\_id;**

### Error Handling:

When the above syntax is executed we get the below error

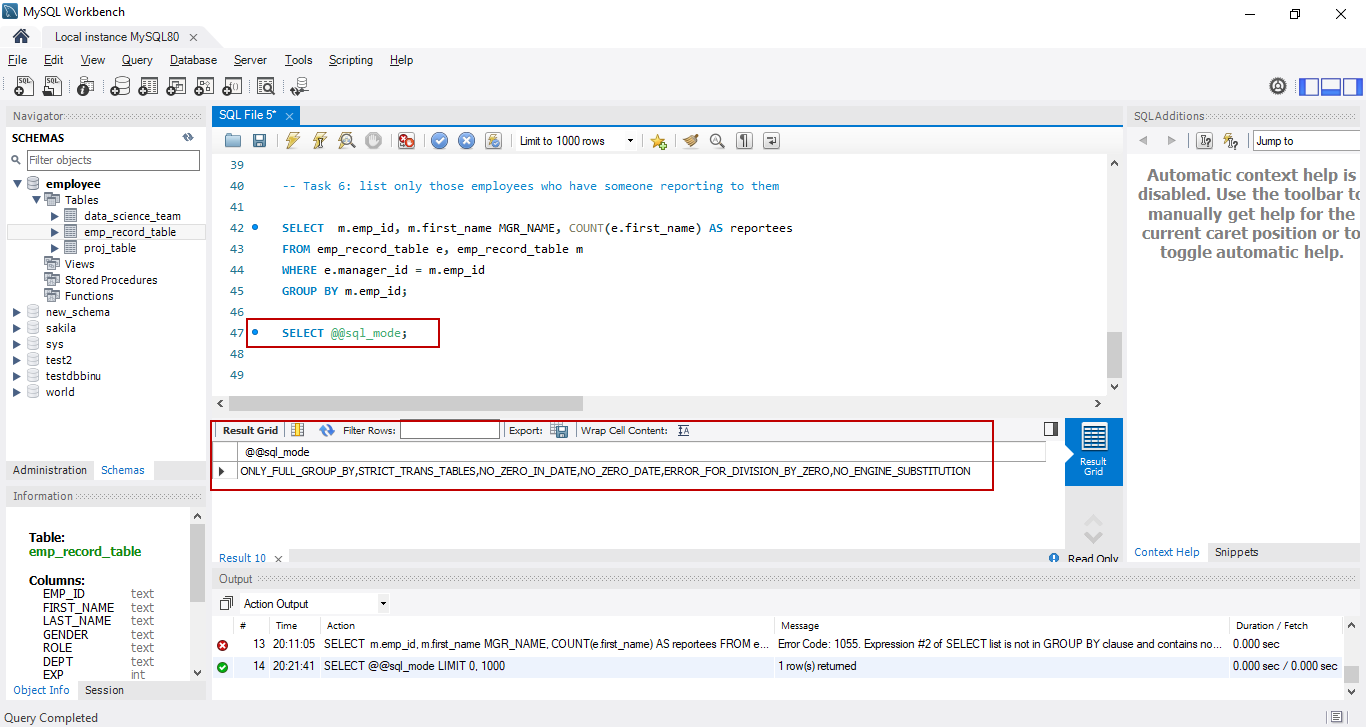
“Error Code: 1055. Expression #2 of SELECT list is not in GROUP BY clause and contains nonaggregated column 'employee.m.FIRST\_NAME' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql\_mode=only\_full\_group\_by 0.000 sec”



Error code 1055 occurs when we include a column in the *SELECT* list and we omit it from the *GROUP BY* clause. This is because our sql\_mode contains ONLY\_FULL\_GROUP\_BY enabled.

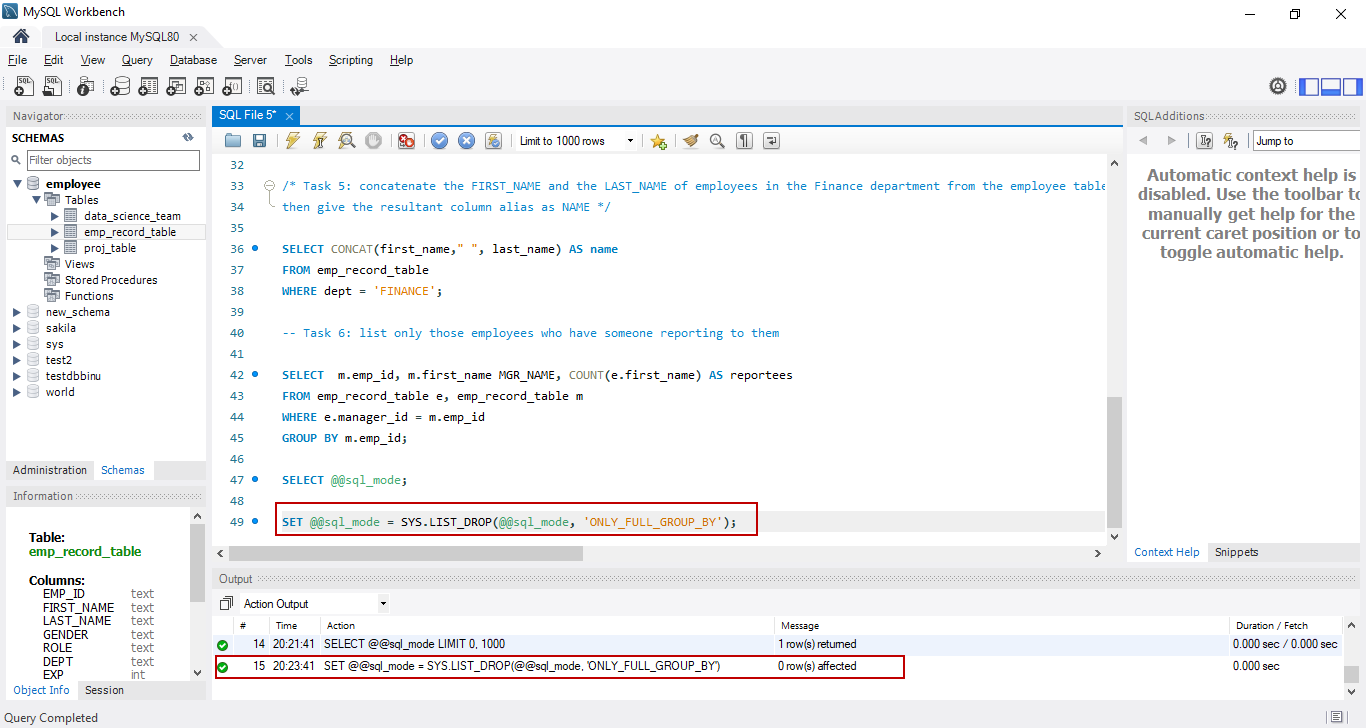
Execute the below syntax to check the sql\_mode

**SELECT @@sql\_mode;**



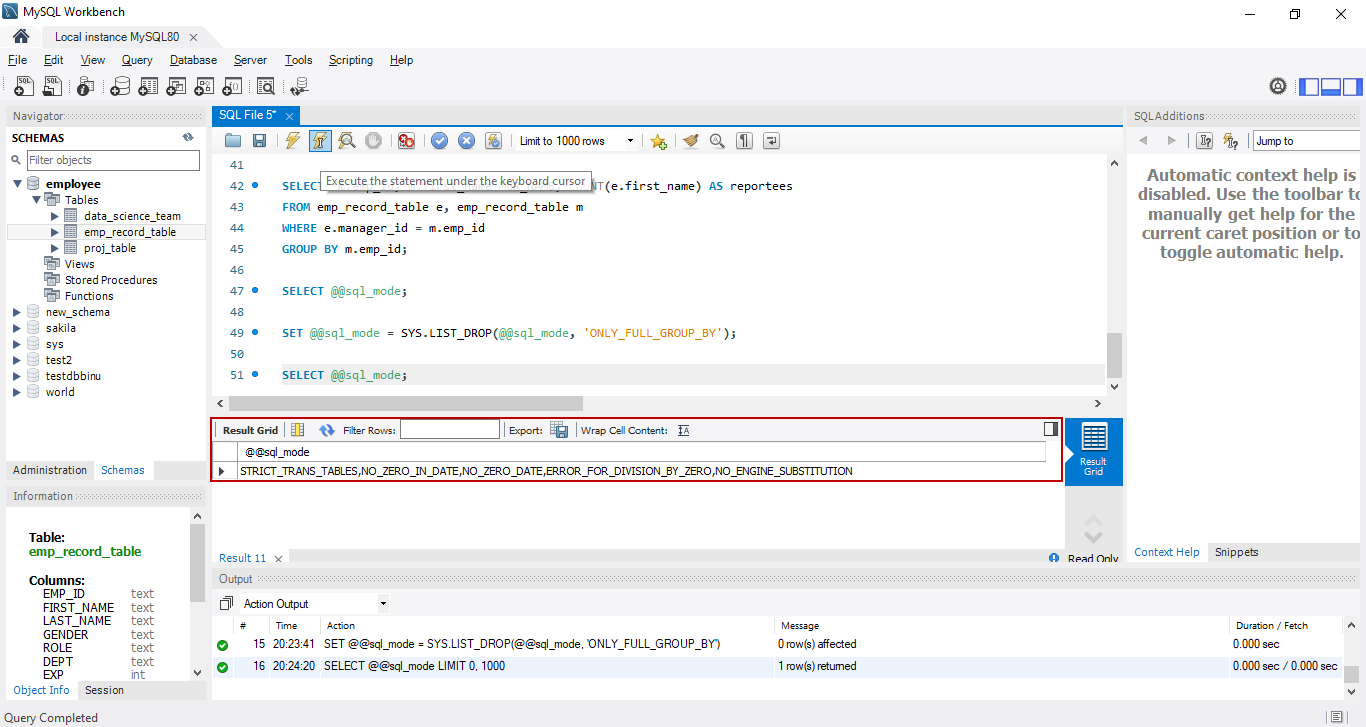
In order to disable the ONLY\_FULL\_GROUP\_BY mode execute the below syntax

**SET @@sql\_mode = SYS.LIST\_DROP(@@sql\_mode, 'ONLY\_FULL\_GROUP\_BY');**



Then execute the below syntax to verify if ONLY\_FULL\_GROUP\_BY mode is disabled. In the below screenshot we notice that ONLY\_FULL\_GROUP\_BY is not showing anymore

**SELECT @@sql\_mode;**



Now when we execute the below syntax, we will be able to see the result as shown in section 6.2

**SELECT m.emp\_id, m.first\_name MGR\_NAME, COUNT(e.first\_name) AS reportees**

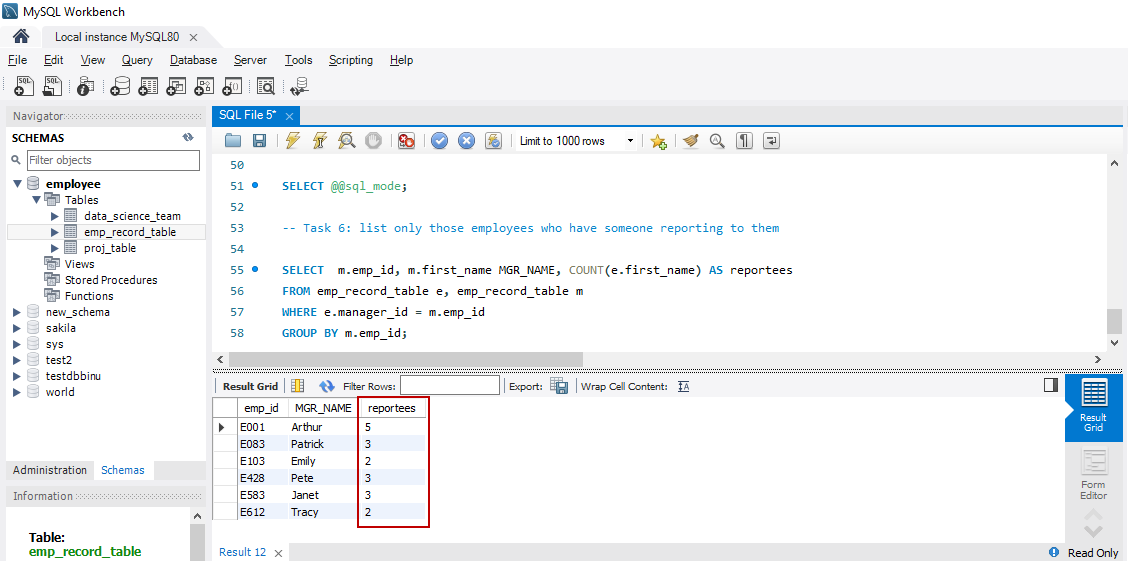
**FROM emp\_record\_table e, emp\_record\_table m**

**WHERE e.manager\_id = m.emp\_id**

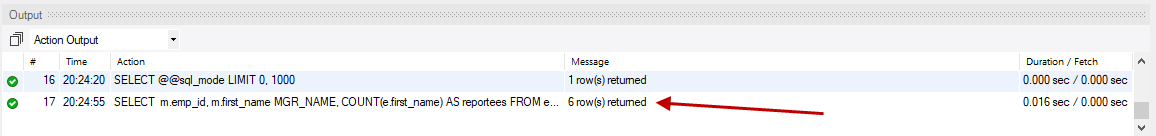
**GROUP BY m.emp\_id;**

## Result:

Below is the list of only those employees who have someone reporting to them and also showing the number of reportees (including the President)



## Output:



# Write a query to list down all the employees from the healthcare and finance departments using union.

In this task, we will able to list all the employees from both healthcare and finance departments using *UNION* clause

## Syntax:

Use the below syntax to fetch all the employees from healthcare and finance departments

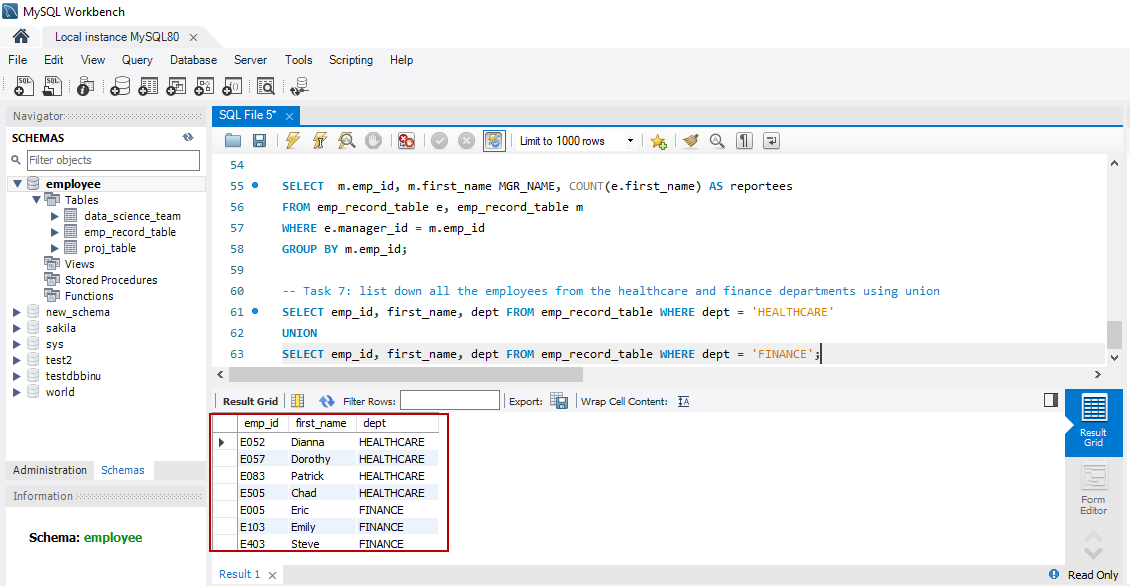
**SELECT emp\_id, first\_name, dept FROM emp\_record\_table WHERE dept = 'HEALTHCARE'**

**UNION**

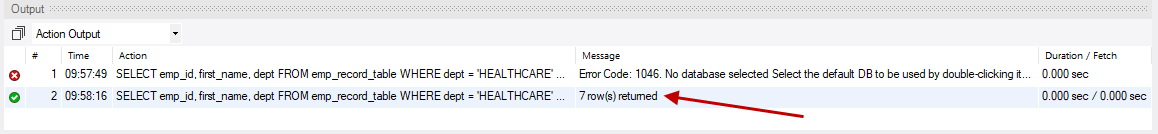
**SELECT emp\_id, first\_name, dept FROM emp\_record\_table WHERE dept = 'FINANCE';**

## Result:

Below is the list of all employees from healthcare and finance departments using *UNION* clause



## Output:



# Write a query to list down employee details such as EMP\_ID, FIRST\_NAME, LAST\_NAME, ROLE, DEPARTMENT, and EMP\_RATING grouped by dept. Also include the respective employee rating along with the max emp rating for the department

In this task, we are able to fetch the data such as emp\_id, first\_name, last\_name, role, department and emp\_rating from the emp\_record\_table with the below requirements

* Grouped by department
* Including the respective employee rating and
* The maximum emp\_rating for the department

## Syntax:

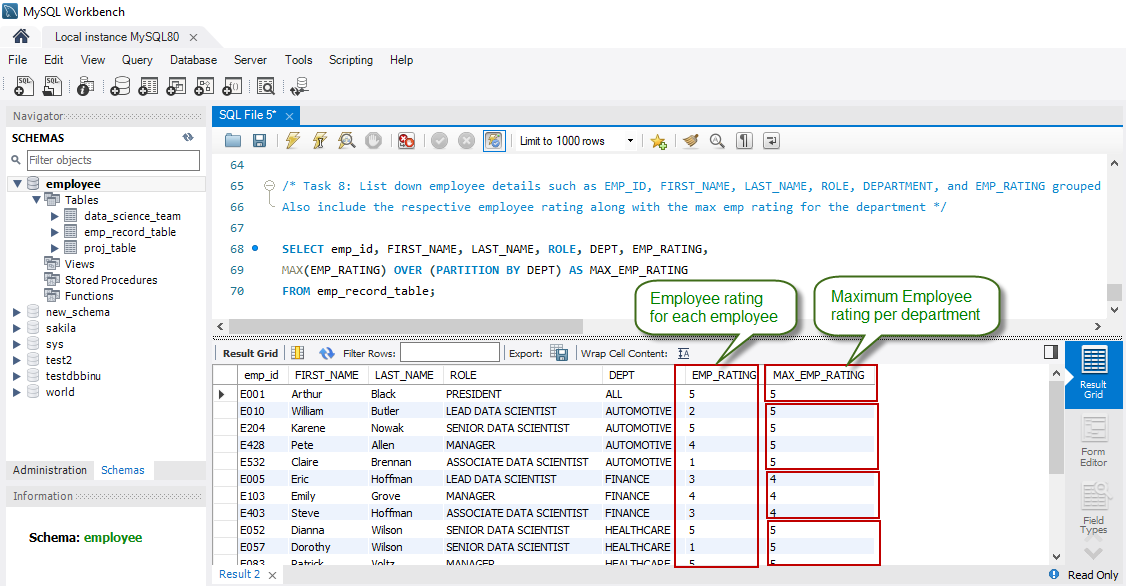
Use the below syntax

**SELECT emp\_id, FIRST\_NAME, LAST\_NAME, ROLE, DEPT, EMP\_RATING,**

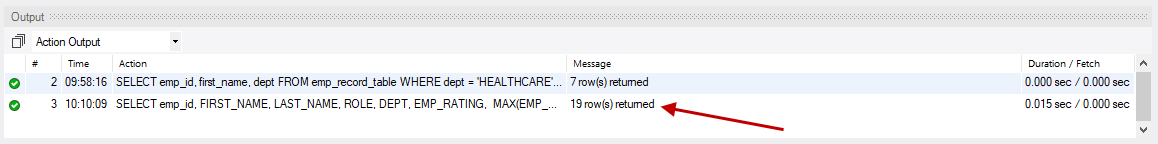
**MAX(EMP\_RATING) OVER (PARTITION BY DEPT) AS MAX\_EMP\_RATING**

**FROM emp\_record\_table;**

## Result:



## Output:



# Write a query to calculate the minimum and the maximum salary of the employees in each role.

In this task, we will be able fetch the emp\_id, first\_name, last\_name, role, department and salary of each employee together with the minimum and maximum salary for that particular role in the company

## Syntax:

Use the below syntax

**SELECT emp\_id, FIRST\_NAME, LAST\_NAME, ROLE, DEPT, salary,**

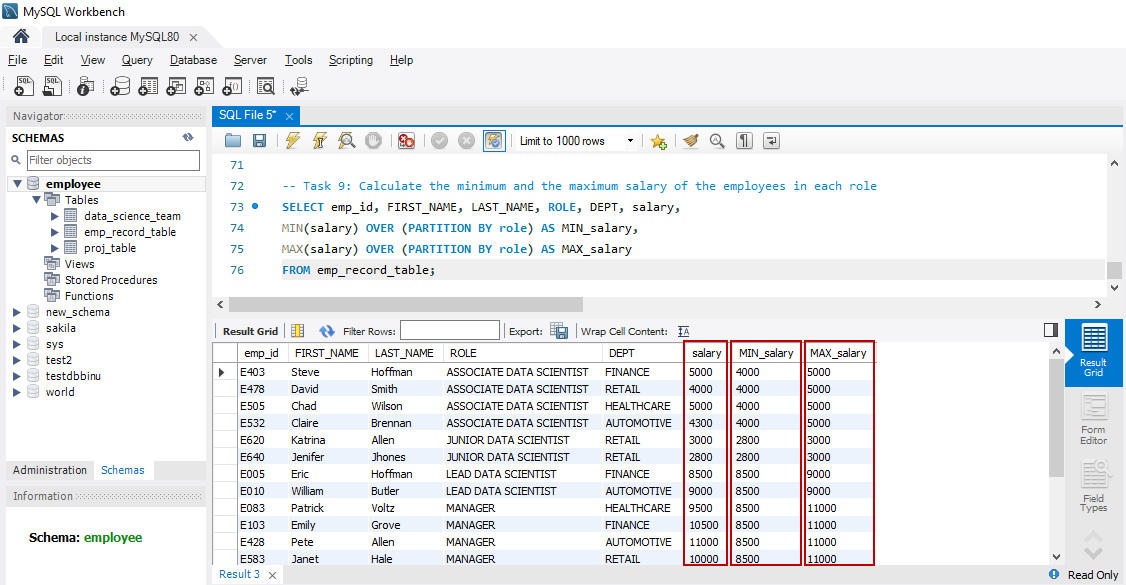
**MIN(salary) OVER (PARTITION BY role) AS MIN\_salary,**

**MAX(salary) OVER (PARTITION BY role) AS MAX\_salary**

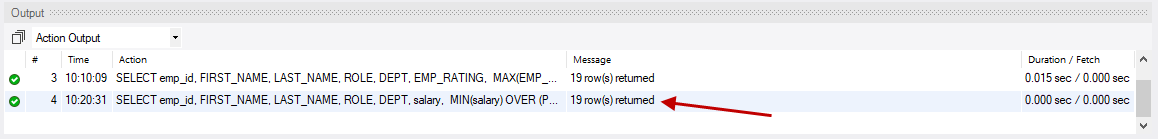
**FROM emp\_record\_table;**

## Result:

The below result shows the salary of each employee along with the minimum and maximum salary for that particular role in the company



## Output:



# Write a query to assign ranks to each employee based on their experience.

In this task, we will be assigning ranks to each employee based on their experience and also assigning dense rank

## Syntax:

Use the below syntax to assign ranks to each employee based on their experience

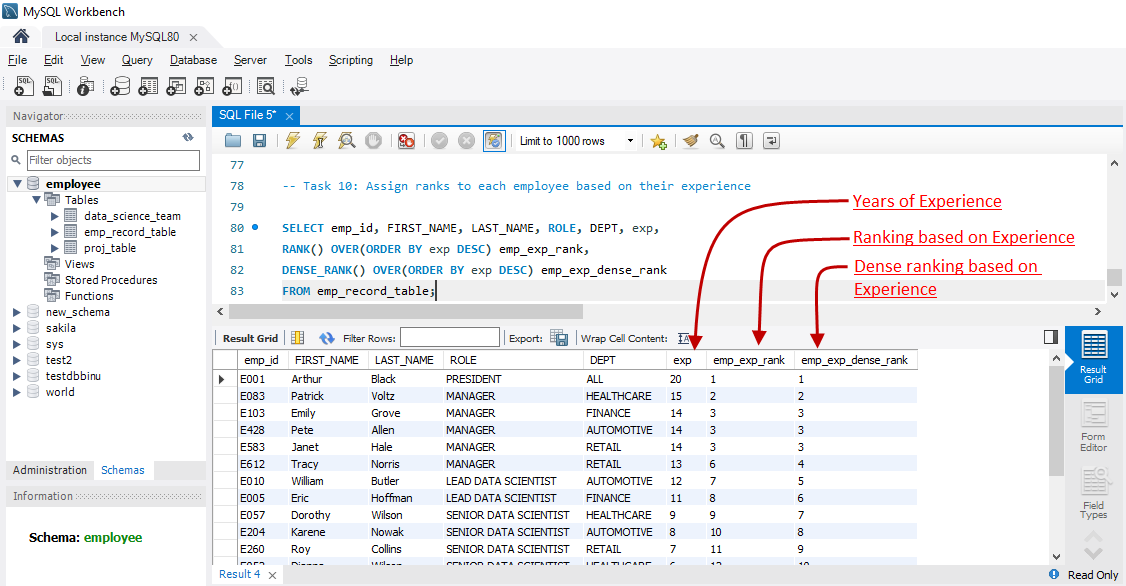
**SELECT emp\_id, FIRST\_NAME, LAST\_NAME, ROLE, DEPT, exp,**

**RANK() OVER(ORDER BY exp DESC) emp\_exp\_rank,**

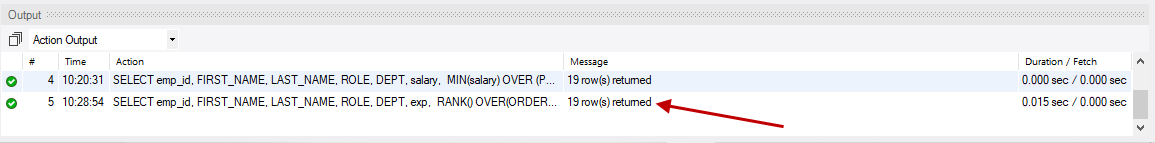
**DENSE\_RANK() OVER(ORDER BY exp DESC) emp\_exp\_dense\_rank**

**FROM emp\_record\_table;**

## Result:



## Output:



# Write a query to create a view that displays employees in various countries whose salary is more than six thousand.

In this task, we will be creating a View that displays employees in various countries whose salary is more than six thousand

## Syntax:

Use the below syntax

**CREATE VIEW v\_emp AS**

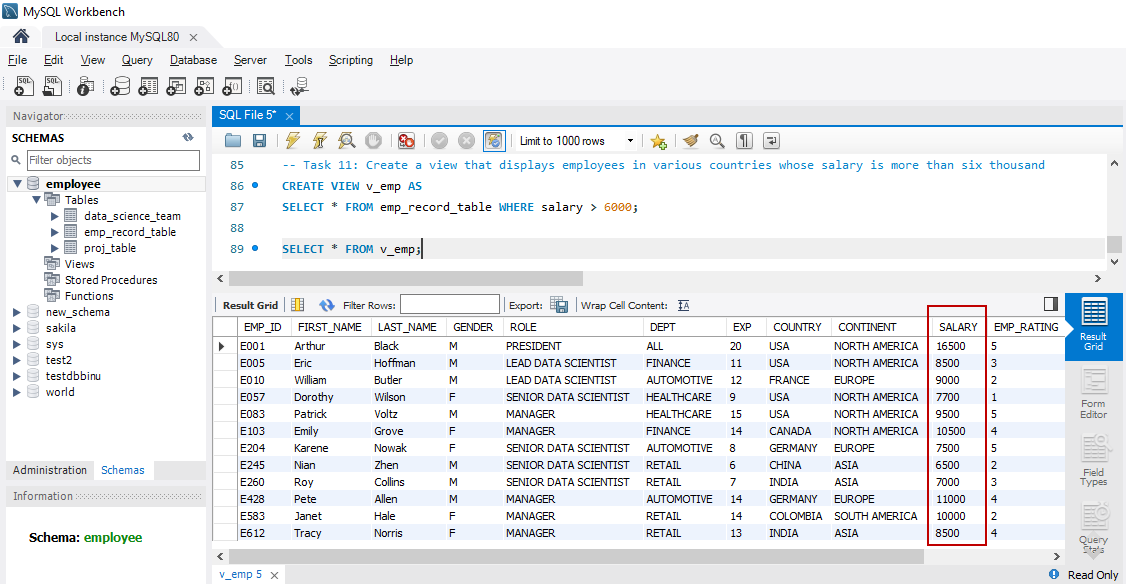
**SELECT \* FROM emp\_record\_table WHERE salary > 6000;**

Then execute the below syntax

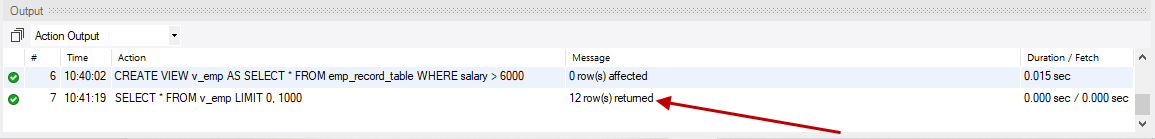
**SELECT \* FROM v\_emp;**

## Result:

Below is the list of all employees from various countries whose salary is more than six thousand



## Output:



# Write a nested query to find employees with experience of more than ten years.

In this task, we will able to fetch the employees with experience of more than ten years

## Syntax:

Use the below syntax

**SELECT e.emp\_id, e.first\_name, e.exp**

**FROM emp\_record\_table e**

**WHERE e.emp\_id IN**

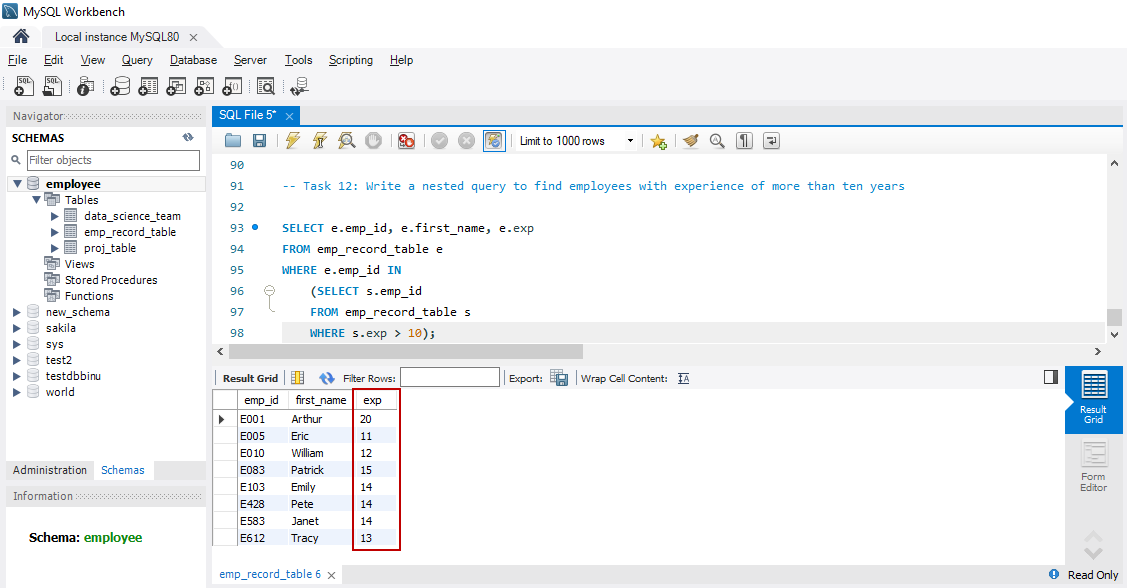
**(SELECT s.emp\_id**

**FROM emp\_record\_table s**

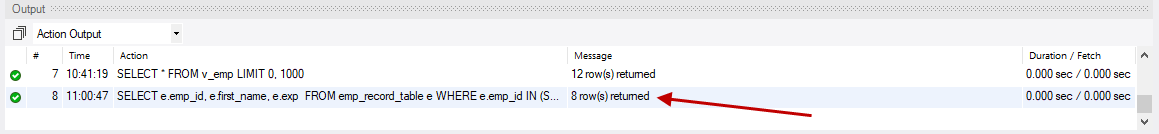
**WHERE s.exp > 10);**

## Result:

Below is the list of all employees with more than ten years of experience using Subquery or nested query



## Output:



# Write a query to create a stored procedure to retrieve the details of the employees whose experience is more than three years.

In this task, we will create a stored procedure to retrieve the details of the employees whose experience is more than three years

## Syntax:

Use the below syntax

Note:

* The standard delimiter “;” must be converted to “//” before creating a stored procedure
* Once the stored procedure is created, we have to call the procedure

**DELIMITER //**

**CREATE PROCEDURE sp\_emp()**

**BEGIN**

**SELECT \***

**FROM emp\_record\_table**

**WHERE exp > 3;**

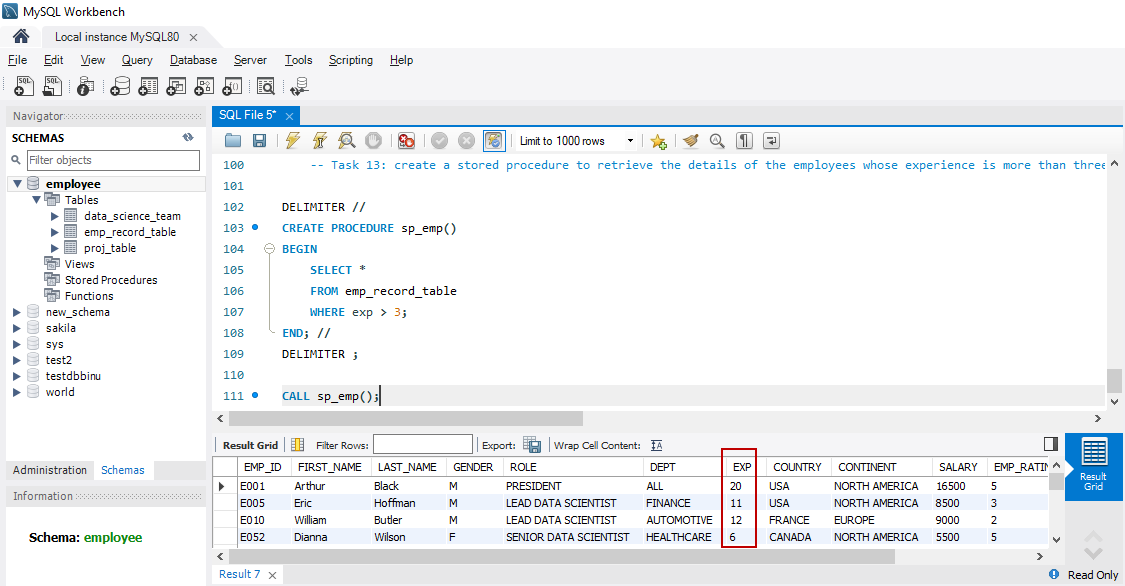
**END; //**

**DELIMITER ;**

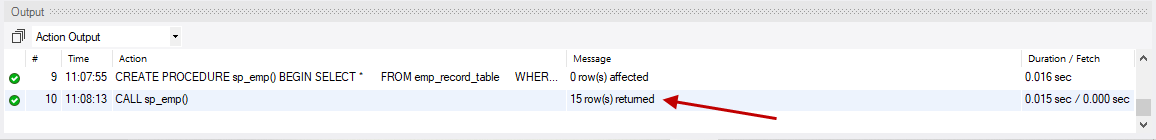
**CALL sp\_emp();**

## Result:

Below is the list of 15 employees whose experience is more than three years



## Output:



# Write a query using stored functions in the project table to check whether the job profile assigned to each employee in the data science team matches the organization’s set standard

**The standard being:**

**For an employee with experience less than or equal to 2 years assign 'JUNIOR DATA SCIENTIST',**

**For an employee with the experience of 2 to 5 years assign 'ASSOCIATE DATA SCIENTIST',**

**For an employee with the experience of 5 to 10 years assign 'SENIOR DATA SCIENTIST',**

**For an employee with the experience of 10 to 12 years assign 'LEAD DATA SCIENTIST',**

**For an employee with the experience of 12 to 16 years assign 'MANAGER'.**

In this task, we will be able to check whether the job profile assigned to each employee in the data\_science\_team matches the organization’s set standard y sing stored functions in the project\_table

In order to check the job profiles, the following steps must be performed

* Declaration of variables such as exist\_exp, exist\_role, and set\_role
* Selecting role and experience from data\_science\_tablefor the given employee ID
* Evaluating the existing experience and assigning role as per organization’s set standard
* Compare set role to existing role and return the result based on profile matching

## Syntax:

Use the below syntax

**DELIMITER //**

**CREATE FUNCTION fn\_prof\_match(e\_id VARCHAR(4)) RETURNS VARCHAR(50) DETERMINISTIC**

**BEGIN**

-- Declaration of variables

**DECLARE exist\_exp INT DEFAULT NULL;**

**DECLARE exist\_role VARCHAR(24) DEFAULT NULL;**

**DECLARE set\_role VARCHAR(24) DEFAULT NULL;**

-- Selecting experience and role from data\_science\_team table for the given employee id.

**SELECT exp, role INTO exist\_exp, exist\_role**

**FROM data\_science\_team**

**WHERE emp\_id = e\_id;**

-- Evaluating the existing experience and assigning role as per organization's set standard

**IF exist\_exp <=2 THEN**

**SET set\_role = "JUNIOR DATA SCIENTIST";**

**ELSEIF exist\_exp > 2 AND exist\_exp <= 5 THEN**

**SET set\_role = "ASSOCIATE DATA SCIENTIST";**

**ELSEIF exist\_exp > 5 AND exist\_exp <= 10 THEN**

**SET set\_role = "SENIOR DATA SCIENTIST";**

**ELSEIF exist\_exp > 10 AND exist\_exp <= 12 THEN**

**SET set\_role = "LEAD DATA SCIENTIST";**

**ELSEIF exist\_exp > 12 AND exist\_exp <= 16 THEN**

**SET set\_role = "MANAGER";**

**END IF;**

-- Compare set role to existing role and return the result based on profile matching

**IF exist\_role = set\_role THEN**

**RETURN "Congratulations! Profile matches set standard.";**

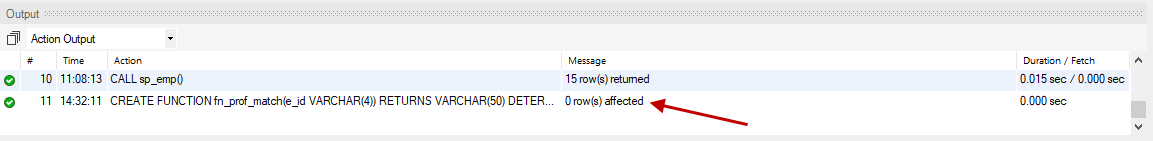
**ELSE**

**RETURN "Sorry! Profile doesn't match set standard.";**

**END IF;**

**END; //**

**DELIMITER ;**

****

Once the stored function is created as shown in the picture above, use the below syntax

**SHOW FUNCTION STATUS WHERE db = 'employee';**

To verify the roles and also to obtain the predefined messages as outcome, execute the below syntax

Mention the emp\_id available in the database

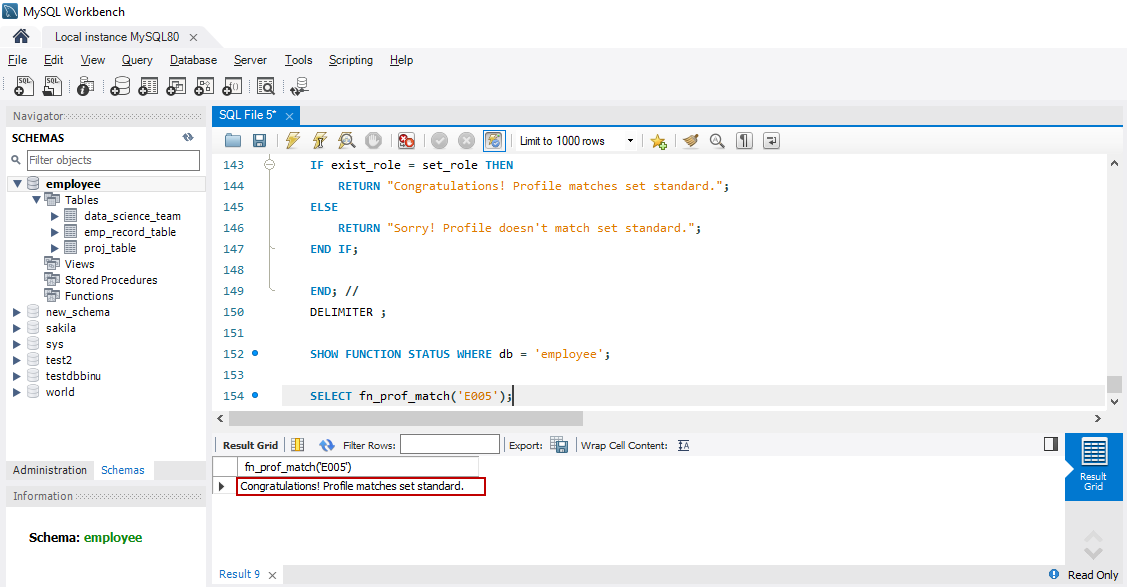
**SELECT fn\_prof\_match('E005');**

Mention the emp\_id not available in the database

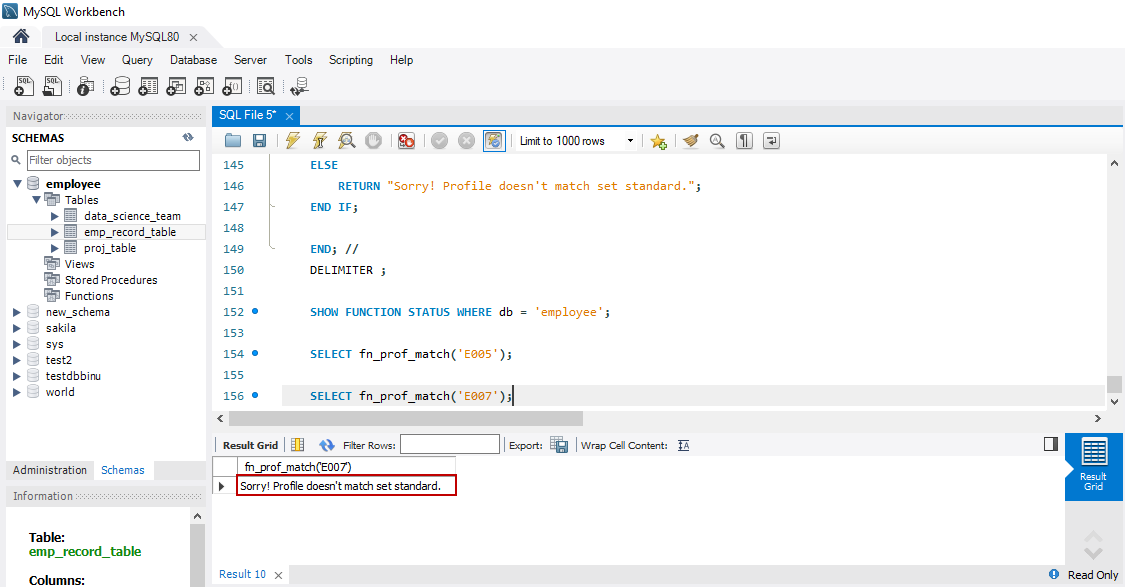
**SELECT fn\_prof\_match('E007');**

## Result:

## Result for emp\_id available in the database:

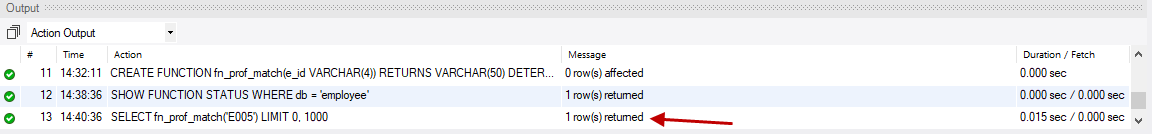


## Result for emp\_id not available in the database:

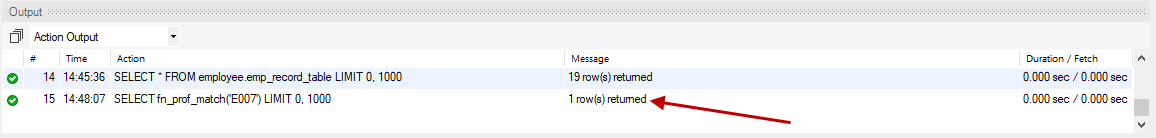


## Output:

## Output for emp\_id available in the database::



## Output for emp\_id not available in the database::



# Create an index to improve the cost and performance of the query to find the employee whose FIRST\_NAME is ‘Eric’ in the employee table after checking the execution plan

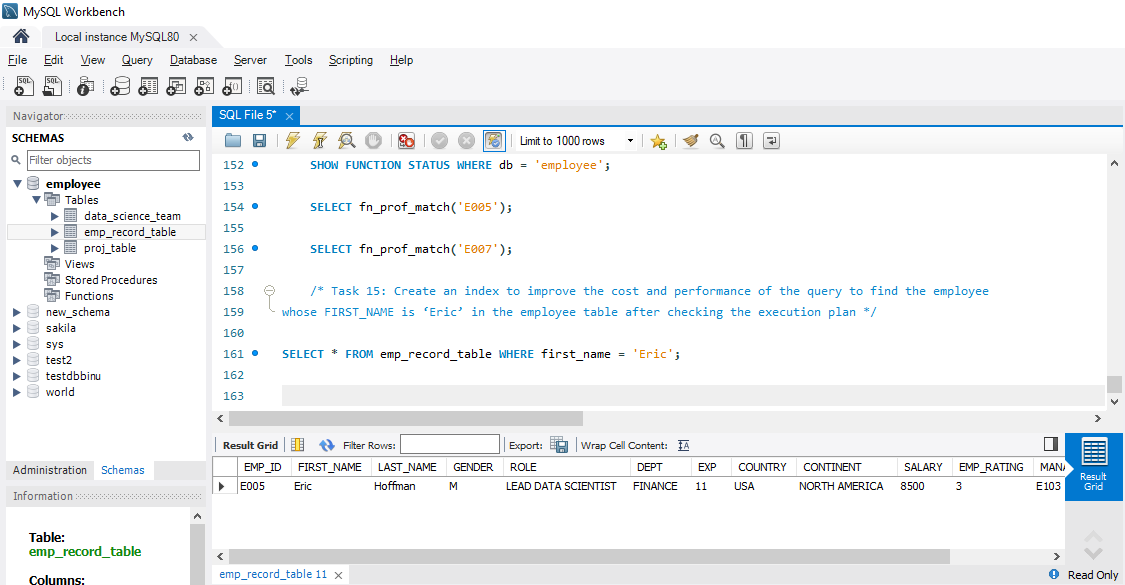
In this task, we will create an Index to improve the cost and performance of the query to find the employee whose first\_name is ‘ERIC’ in the employee table by comparing the execution plan before and after creating the index

## Syntax:

Use the below syntax

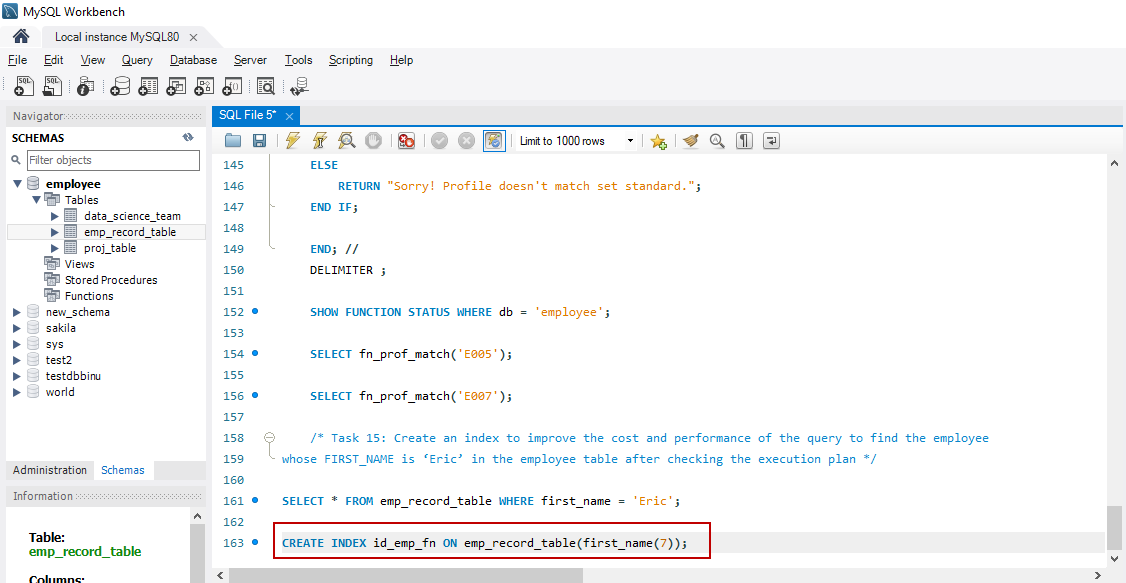
## Before creating index:

**SELECT \* FROM emp\_record\_table WHERE first\_name = 'Eric';**

****

## Creating index:

**CREATE INDEX id\_emp\_fn ON emp\_record\_table(first\_name(7));**

****

## After creating index:

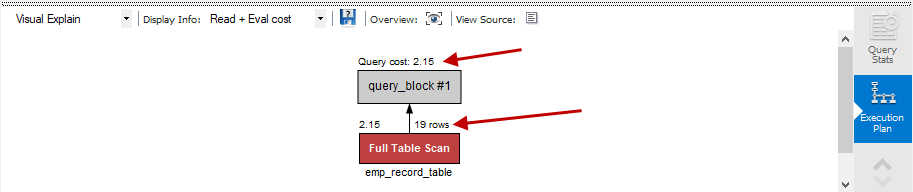
**SELECT \* FROM emp\_record\_table WHERE first\_name = 'Eric';**

## Result:

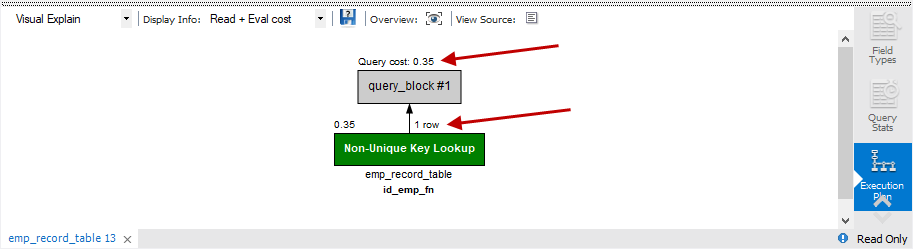
Below is the comparison of query cost between before and after creating index.

|  |  |
| --- | --- |
|  | Query Cost |
| **Before creating Index** | **2.15** |
| **After creating Index** | **0.35** |

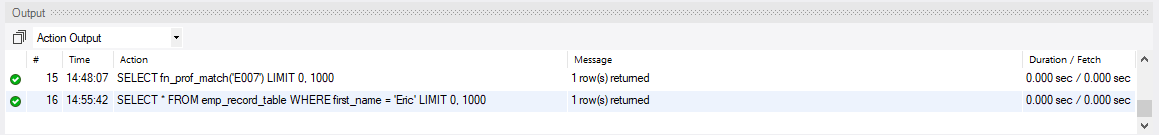
## Result before creating Index:

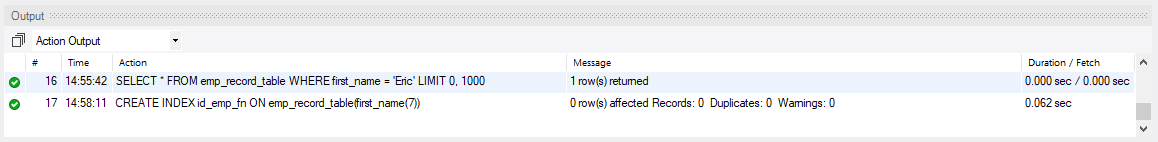


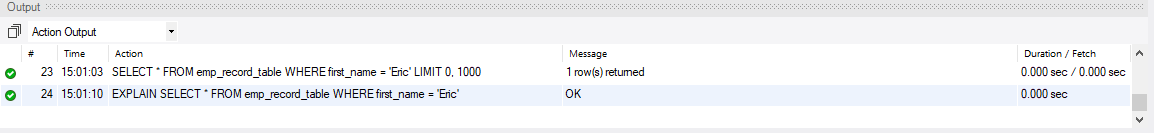
## Result after creating Index:



## Output:







# Write a query to calculate the bonus for all the employees, based on their ratings and salaries (Use the formula: 5% of salary \* employee rating)

In this task, we will be calculating the bonus for al the employees based on their ratings and salaries

## Syntax:

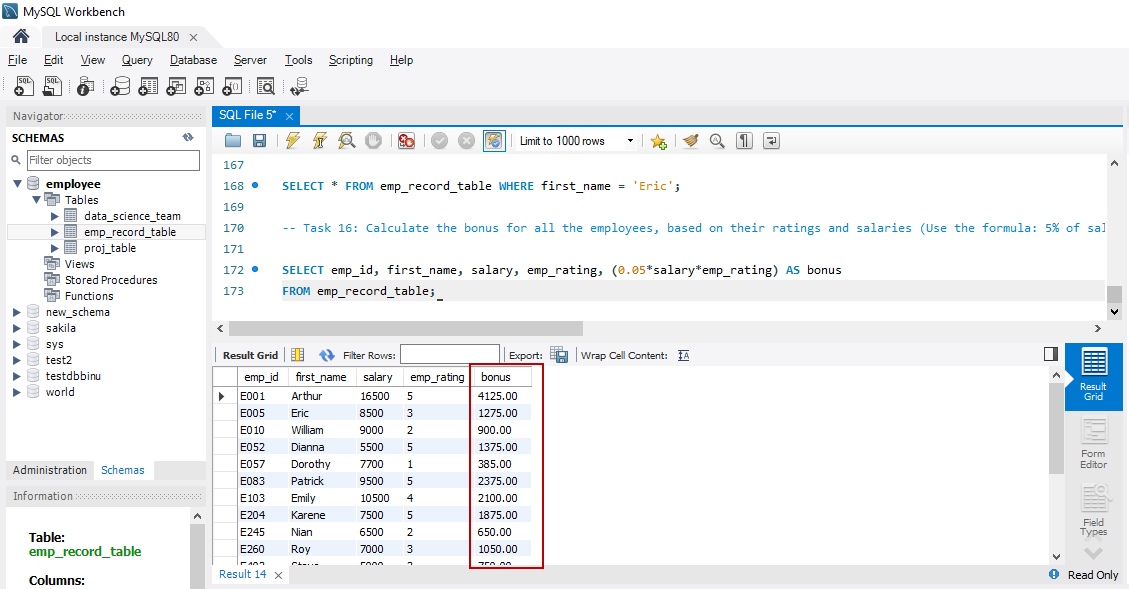
Use the below syntax

**SELECT emp\_id, first\_name, salary, emp\_rating, (0.05\*salary\*emp\_rating) AS bonus**

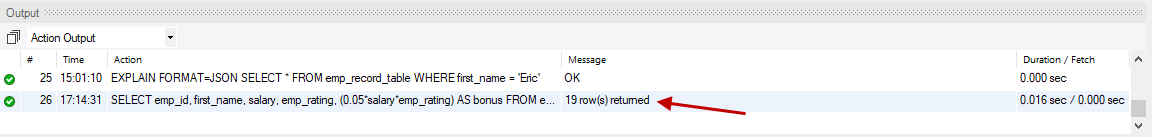
**FROM emp\_record\_table;**

## Result:

Below is the bonus calculation for each employee based on 5% of their salary and employee rating



## Output:



# Write a query to calculate the average salary distribution based on the continent and country. Take data from the employee record table

In this task, we will calculate the average salary distribution for each employee based on the continent and country

## Syntax:

Use the below syntax

**SELECT emp\_id, FIRST\_NAME, salary, country, continent,**

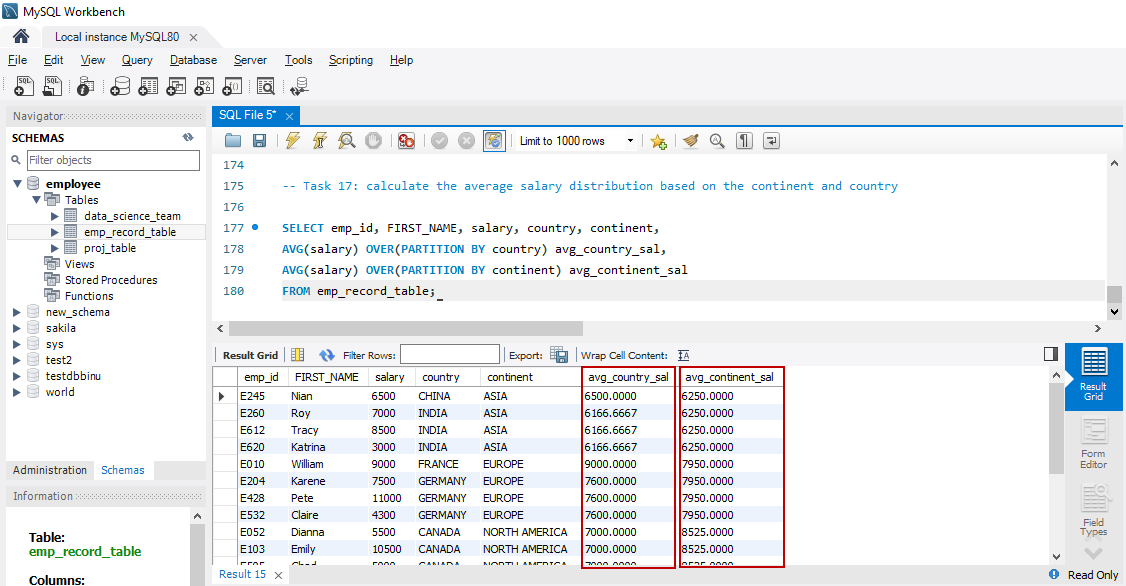
**AVG(salary) OVER(PARTITION BY country) avg\_country\_sal,**

**AVG(salary) OVER(PARTITION BY continent) avg\_continent\_sal**

**FROM emp\_record\_table;**

## Result:

Below is the average salary distribution for each employee based on the continent and country



## Output:

