RDBMS PEER LEARNING DOCUMENT

Create a database named employee, then import data_science_team.csv proj_table.csv and emp_record_table.csv into the employee database from the given resources.

My code

```
create database employee;

use employee;
```

Explanation:

- "Create database employee" creates a new database named "employee" in the database management system.
- "Use employee" is used to select the database named "employee" for use.

Ratinder Pal Singh Solution's

create database employee;

use employee;

Difference:

- Same as me.

Mahesh Gupta Solution's

```
CREATE DATABASE RDBMS_ASSIGNMENT;

 USE RDBMS_ASSIGNMENT;
```

Explanation:

- Only database name is changed.

2. Create an ER diagram for the given employee database.

My code

```
ER diagram
```

Ratinder Pal Singh Solution’s

```
ER diagram
```

Mahesh Gupta Solution’s

```
ER diagram
```

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.

My code

```
SELECT EMP_ID, FIRST_NAME, LAST_NAME, GENDER, DEPT
FROM emp_record_table;
```

Explanation:

- The "SELECT" keyword is used to indicate that a query is being made to retrieve data from the table.
- After the SELECT keyword, we list the columns that we want to retrieve from the table, separated by commas. In this case, the columns are "EMP_ID", "FIRST_NAME", "LAST_NAME", "GENDER", and "DEPT".
- The "FROM" keyword is used to specify the table from which we want to retrieve data. In this case, the table is "emp_record_table".

Ratinder Pal Singh Solution’s

SELECT emp_id, first_name, last_name, gender, dept

FROM emp_record_table;

Difference:

- Same as me.

Mahesh Gupta Solution's

SELECT EMP_ID, FIRST_NAME,LAST_NAME,GENDER,DEPT FROM emp_record_table;

Explanation:

- Same as me.

4. Write a query to fetch EMP_ID, FIRST_NAME, LAST_NAME, GENDER, DEPARTMENT, and EMP_RATING if the EMP_RATING is:

● less than two

● greater than four

My code

```
SELECT EMP_ID, FIRST_NAME, LAST_NAME, GENDER, DEPT, EMP_RATING

FROM emp_record_table

WHERE EMP_RATING<2;

SELECT EMP_ID, FIRST_NAME, LAST_NAME, GENDER, DEPT, EMP_RATING

FROM emp_record_table

WHERE EMP_RATING BETWEEN 2 AND 4;

SELECT EMP_ID, FIRST_NAME, LAST_NAME, GENDER, DEPT, EMP_RATING

FROM emp_record_table

WHERE EMP_RATING>4;
```

Explanation:

- This query is returning a table that includes all the records from the "emp_record_table" table that meet the condition specified in the WHERE clause, and only with the columns specified in the SELECT statement. The order of the columns in the result set will match the order in which they are listed in the SELECT statement.

Ratinder Pal Singh Solution's

```
SELECT emp_id, first_name, last_name, gender, dept, emp_rating

FROM emp_record_table

WHERE emp_rating<2;
```

```
SELECT emp_id, first_name, last_name, gender, dept, emp_rating

FROM emp_record_table

WHERE emp_rating>4;
```

```
SELECT emp_id, first_name, last_name, gender, dept, emp_rating

FROM emp_record_table

WHERE emp_rating BETWEEN 2 AND 4;
```

Difference:

- Same as me.

Mahesh Gupta Solution's

```
SELECT EMP_ID, FIRST_NAME,LAST_NAME,GENDER,DEPT ,EMP_RATINGFROM emp_record_table WHERE EMP_RATING<2;
```

SELECT EMP_ID, FIRST_NAME,LAST_NAME,GENDER,DEPT ,EMP_RATING FROM emp_record_table WHERE EMP_RATING>4;

SELECT EMP_ID, FIRST_NAME,LAST_NAME,GENDER,DEPT ,EMP_RATING FROM emp_record_table WHERE EMP_RATING BETWEEN 2 AND 4;

Explanation:

- Same as me.

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.

My code

```
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) AS NAME
FROM emp_record_table
WHERE DEPT='FINANCE';
```

Explanation:

- CONCAT function to combine the "FIRST_NAME" and "LAST_NAME" columns, separated by a space.
- "AS" keyword to give the resulting concatenated column a name, which in this case is "NAME"
- This SQL statement will return the concatenated values of the "FIRST_NAME" and "LAST_NAME" columns as a new column named "NAME" from the "emp_record_table" table, but only for records where the department is equal to "FINANCE".

Ratinder Pal Singh Solution’s

```
SELECT CONCAT(first_name, ' ', last_name) as NAME
FROM emp_record_table WHERE dept='Finance';
```

Difference:

- Same as me.

Mahesh Gupta Solution’s

```
SELECT CONCAT(FIRST_NAME,' ',LAST_NAME) AS NAME
FROM emp_record_table
WHERE DEPT='FINANCE';
```

Explanation:

- Same as me.

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

My code

```
SELECT CONCAT(emp.FIRST_NAME, ' ', EMP.LAST_NAME) AS NAME FROM (SELECT
COUNT(W.MANAGER_ID) CNT, MANAGER_ID
FROM emp_record_table W
GROUP BY W.MANAGER_ID
HAVING CNT<>0) A
join emp_record_table emp on emp.emp_id=a.manager_id;
```

Explanation:

- Subquery in parentheses that counts the number of records in the "emp_record_table" table with a non-null "MANAGER_ID" column, grouped by the "MANAGER_ID" column.

- The "HAVING" keyword is used to filter out records where the count is not equal to zero.
- The result of this subquery is stored in a table named "A" with two columns: "CNT" and "MANAGER_ID".
- The main query then joins the "emp_record_table" table with the "A" table on the "MANAGER_ID" column. The "JOIN" keyword is used to specify the join operation, and the "ON" keyword is used to specify the column to join on.
- The result of this join is a new table that includes all the records from the "emp_record_table" table where the "MANAGER_ID" column matches a non-zero count in the "A" table.
- Finally, the CONCAT function is used to concatenate the "FIRST_NAME" and "LAST_NAME" columns of the resulting table, separated by a space. The resulting column is given the name "NAME" using the "AS" keyword.

Ratinder Pal Singh Solution's

```
SELECT mgr.emp_id, mgr.first_name, mgr.last_name, COUNT(e.emp_id)

FROM emp_record_table AS e INNER JOIN emp_record_table AS mgr

ON e.manager_id = mgr.emp_id

GROUP BY mgr.emp_id, mgr.first_name, mgr.last_name;
```

Difference:

- He has given emp_id, name, last name in different column.
- He has used self join on emp_record_table.

Mahesh Gupta Solution's

```
SELECT  M.EMP_ID , M.FIRST_NAME ,M.LAST_NAME , COUNT(*) AS CNT

FROM emp_record_table  E INNER JOIN emp_record_table M

ON E.MANAGER_ID=M.EMP_ID
```

```
GROUP BY M.EMP_ID;
```

Explanation:

- Same as Ratinder.

7. Write a query to list down all the employees from the healthcare and finance departments using union. Take data from the employee record table.

My code

```
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME)
FROM emp_record_table
WHERE DEPT='HEALTHCARE'
UNION
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME)
FROM emp_record_table
WHERE DEPT='FINANCE';
```

Explanation:

- This query will return a list of names of employees who work in either the "HEALTHCARE" or "FINANCE" department, with no duplicates.
- The UNION operator is used to combine the results of these two SELECT statements into a single result set.

Ratinder Pal Singh Solution's

```
SELECT emp_id, first_name, last_name, dept
FROM emp_record_table
WHERE dept='Healthcare'
```

```
UNION

SELECT emp_id, first_name, last_name, dept

FROM emp_record_table

WHERE dept='Finance';
```

Difference:

- He has given emp_id, name, last name in different column.
- Rest is same

Mahesh Gupta Solution's

```
SELECT *

FROM emp_record_table

WHERE DEPT='HEALTHCARE'

UNION

SELECT *

FROM emp_record_table

WHERE DEPT ='FINANCE';
```

Explanation:

- Same as Ratinder.

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.

My code

```
SELECT EMP_ID, FIRST_NAME, LAST_NAME, ROLE, DEPT, EMP_RATING, MAX(EMP_RATING)
OVER(PARTITION BY DEPT) AS MAX_RATING
FROM emp_record_table;
```

Explanation:

- The SELECT clause specifies which columns to retrieve from the table, including the employee ID, first name, last name, role, department, and employee rating.
- The MAX function is used in conjunction with the OVER clause to calculate the maximum rating for each department.
- The PARTITION BY clause divides the data into partitions based on the department column, and the MAX function then calculates the maximum rating within each partition.

Ratinder Pal Singh Solution's

```
SELECT emp_id, first_name, last_name, role, dept, emp_rating,
max(emp_rating) OVER(PARTITION BY dept) as max_rating_by_dept
FROM emp_record_table;
```

Difference:

- Same as me.

Mahesh Gupta Solution's

```
SELECT EMP_ID,FIRST_NAME,LAST_NAME,ROLE,DEPT,EMP_RATING,
MAX(EMP_RATING)  OVER(PARTITION BY DEPT) AS MAX_RATING_BY_DEPT
FROM emp_record_table;
```

Explanation:

- Same as me.

9. Write a query to calculate the minimum and the maximum salary of the employees in each role. Take data from the employee record table.

My code

```
SELECT DISTINCT ROLE, MIN(SALARY) OVER(PARTITION BY ROLE) AS MIN_SAL,
MIN(SALARY) OVER(PARTITION BY ROLE) AS MAX_SAL
FROM emp_record_table;
```

Explanation:

- The SELECT clause specifies which columns to retrieve from the table, including the distinct role, the minimum salary for each role, and the maximum salary for each role.
- The MIN function is used to calculate the minimum salary for each role. The MAX function is also used to calculate the maximum salary for each role.
- Both functions are used in conjunction with the OVER and PARTITION BY clauses to calculate these values within each partition.

Ratinder Pal Singh Solution's

```
SELECT MIN(salary) as min_sal_by_role, MAX(salary) as max_sal_by_role
FROM emp_record_table
GROUP BY role;
```

Difference:

- He has not provided the respective role.
- He has used Group By clause.

Mahesh Gupta Solution's

```
SELECT DISTINCT ROLE, MAX(SALARY) OVER(PARTITION BY ROLE) AS MAX_SALARY
,MIN(SALARY)    OVER(PARTITION BY ROLE)

AS MIN_SALARY  FROM emp_record_table;
```

Difference:

- Same as me.

10. Write a query to assign ranks to each employee based on their experience. Take data from the employee record table.

My code

```
SELECT *, row_NUMBER() OVER(ORDER BY EXP DESC) AS RANKING
FROM emp_record_table;
```

Explanation:

- The row_NUMBER() function is a window function that assigns a unique sequential number to each row within a result set, based on the specified ordering.
- The OVER clause is used to specify the window or partition of data to which the function will be applied.

- By using the wildcard character '*' in the SELECT clause, all columns from the "emp_record_table" table are selected, in addition to the new "RANKING" column.

Ratinder Pal Singh Solution's

```
SELECT DISTINCT emp_id, exp, DENSE_RANK() OVER(ORDER BY exp DESC) AS rank_by_exp

FROM emp_record_table;
```

Difference:

- He has used DENSE_RANK() function.
- Rest is same

Mahesh Gupta Solution's

```
SELECT EMP_ID ,EXP, DENSE_RANK() OVER(ORDER BY EXP DESC) AS RANK_BY_EXP

FROM emp_record_table ;
```

Explanation:

Same as Ratinder.

11. Write a query to create a view that displays employees in various countries whose salary is more than six thousand. Take data from the employee record table.

My code

```
CREATE VIEW EMP_RECORD AS
SELECT FIRST_NAME, COUNTRY
FROM emp_record_table
```

```
WHERE SALARY>6000;
```

Explanation:

- The view "EMP_RECORD" is created by selecting the "FIRST_NAME" and "COUNTRY" columns from the "emp_record_table" table, where the "SALARY" is greater than 6000.

Ratinder Pal Singh Solution's

```
CREATE OR REPLACE VIEW sal_greater_six_K

AS SELECT emp_id, first_name, last_name, country, salary

FROM emp_record_table

WHERE salary>6000;

SELECT * FROM sal_greater_six_K;
```

Difference:

- He has also used Replace clause for view.
- View name is different.
- Rest is same.

Mahesh Gupta Solution's

```
CREATE OR REPLACE VIEW SALAR_GREATER_THEN_SIX_THOUSAND AS

SELECT EMP_ID,COUNTRY,SALARY FROM emp_record_table

WHERE SALARY>6000;

SELECT * FROM SALAR_GREATER_THEN_SIX_THOUSAND;
```

Difference:

Same as Ratinder.

12. Write a nested query to find employees with experience of more than ten years. Take data from the employee record table.

My code

```
SELECT EMP_ID, FIRST_NAME, ROLE, EXP
FROM (SELECT *
FROM emp_record_table
where exp>10) A;
```

Explanation:

- In the subquery, "SELECT * FROM emp_record_table where exp>10", all columns from the "emp_record_table" table are selected where the "EXP" column value is greater than 10.
- The outer query is then executed on the result set returned by the subquery. The outer query selects only the "EMP_ID", "FIRST_NAME", "ROLE", and "EXP" columns from the result set.

Ratinder Pal Singh Solution's

```
SELECT *
FROM emp_record_table
WHERE emp_id IN
(SELECT emp_id
```

```
FROM emp_record_table

WHERE exp>10);
```

Difference:

- Did same things only data retrieved is different.

Mahesh Gupta Solution’s

```
SELECT *

FROM emp_record_table

WHERE EXP IN (SELECT EXP FROM emp_record_table WHERE EXP>10);
```

Difference:

- Did same things only data retrieved is different.

13. Write a query to create a stored procedure to retrieve the details of the employees whose experience is more than three years. Take data from the employee record table.

My code

```
DELIMITER $$
CREATE PROCEDURE EMP_DET()
BEGIN
      SELECT * FROM emp_record_table
```

```
    WHERE EXP>3;
END $$
DELIMITER ;

CALL EMP_DET();
```

Explanation:

- Created a store procedure using the syntex.
- This procedure is returning Records fro emp_record_table where experience is Greater than 3.

Ratinder Pal Singh Solution’s

```
delimiter $$
create procedure emp_details()
begin
select * from emp_record_table
where exp>3;
end$$
delimiter ;
call emp_details();
```

Difference:

- Same as me.

Mahesh Gupta Solution’s

```
DELIMITER $$

CREATE PROCEDURE Solve()

BEGIN

SELECT *

FROM emp_record_table

WHERE EXP>3;

END

$$ DELIMITER ;

CALL Solve();
```

Difference:

- Only procedure name is different.

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.

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'.

My code

```
DELIMITER $$
CREATE FUNCTION get_job_profile(experience INT, role VARCHAR(30))
RETURNS VARCHAR(50)
DETERMINISTIC

BEGIN
      DECLARE job_profile varchar(50);
    DECLARE flag varchar(30);
      IF experience<=2 THEN
            SET job_profile='JUNIOR DATA SCIENTIST';
      ELSEIF experience between 2 AND 5 THEN
            SET job_profile='ASSOCIATE DATA SCIENTIST';
      ELSEIF experience between 5 AND 10 THEN
            SET job_profile='SENIOR DATA SCIENTIST';
      ELSEIF experience between 10 AND 12 THEN
            SET job_profile='LEAD DATA SCIENTIST';
      ELSEIF experience between 12 AND 16 THEN
            SET job_profile='MANAGER';
      END IF;

    IF job_profile=role THEN
            SET flag='YES';
      ELSE
            SET flag='NO';
      END IF;

    RETURN flag;
END $$
DELIMITER ;
```

```
SELECT EMP_ID, FIRST_NAME, EXP, ROLE, get_job_profile(EXP, ROLE) AS
CHECK_PROFILE
FROM data_science_team;
```

Explanation:

- Code is started by defining a stored function named "get_job_profile" that takes in two arguments: an integer representing "experience" and a string representing "role". The function returns a string value representing whether the "role" matches the job profile determined by the "experience" parameter.
- The function first declares two variables, "job_profile" and "flag", both of type varchar. Then, it checks the value of "experience" against several ranges to determine the appropriate "job_profile". If none of the ranges apply, no job profile is assigned.
- After the job profile is assigned, the function checks if the job profile matches the "role" argument. If it does, it sets the "flag" variable to "YES". Otherwise, it sets the "flag" variable to "NO".
- The function ends by returning the value of the "flag" variable.

Ratinder Pal Singh Solution’s

```
delimiter $$
create function emp_details() returns tinyint(1) deterministic
begin
declare v_exp int default 0;
declare v_role varchar(50) default "";
```

```
declare finished int default 0;
declare dummy_cursor cursor for
select exp, role from emp_record_table;
declare continue handler for not found
set finished=1;
open dummy_cursor;
check_role: loop
fetch dummy_cursor into v_exp, v_role;
if finished = 1 then
leave check_role;
end if;
if (v_exp<=2 and v_role!='JUNIOR DATA SCIENTIST') then
return false;
elseif (v_exp>2 and v_exp<=5 and v_role!='ASSOCIATE DATA SCIENTIST') then
return false;
elseif (v_exp>5 and v_exp<=10 and v_role!='SENIOR DATA SCIENTIST') then
return false;
elseif (v_exp>10 and v_exp<=12 and v_role!='LEAD DATA SCIENTIST') then
return false;
elseif (v_exp>12 and v_exp<=16 and v_role!='MANAGER') then
return false;
end if;
end loop check_role;
```

```
close dummy_cursor;

return true;
```

```
end$$

delimiter ;

delimiter $$

create procedure helper_procedure()

begin

if emp_details() then

select 'The job profile assigned to each employee

in the data science team matches the organization’s set standard.' as message;

else

select 'The job profile assigned to each employee

in the data science team does not match the organization’s set standard.' as message;

end if;

end$$

delimiter ;

call helper_procedure();
```

Difference:

- He has used cursor
- Also used handler to handle error.
- Used a helper procedure.

Mahesh Gupta Solution’s

```
DELIMITER $$

CREATE FUNCTION SCIENTIST_EXP2()

RETURNS TINYINT(1)

DETERMINISTIC

BEGIN

DECLARE RES INT DEFAULT 1;

DECLARE RL VARCHAR(30);

DECLARE SETEND INTEGER DEFAULT 0;

DECLARE EP INT;

DECLARE CURNAME CURSOR FOR

SELECT ROLE,EXP FROM emp_record_table;

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET SETEND=1;


OPEN CURNAME;

MY_LOOP:LOOP

FETCH CURNAME INTO RL,EP;

IF SETEND=1 THEN

LEAVE MY_LOOP;

END IF;
```

```
IF(RL='JUNIOR DATA SCIENTIST' AND EP>2) THEN

SET RES=0;

LEAVE MY_LOOP;

ELSEIF(RL='ASSOCIATE DATA SCIENTIST' AND (EP<=2 OR EP>5)) THEN

SET RES=0;

LEAVE MY_LOOP;

ELSEIF(RL='SENIOR DATA SCIENTIST' AND (EP<=5 OR EP>10)) THEN

SET RES=0;

LEAVE MY_LOOP;

ELSEIF(RL='LEAD DATA SCIENTIST' AND (EP<=10 OR EP>12)) THEN

SET RES=0;

LEAVE MY_LOOP;

ELSEIF(RL='MANAGER' AND (EP<=12 OR EP>16)) THEN

SET RES=0;

LEAVE MY_LOOP;

END IF;

END LOOP MY_LOOP;

CLOSE CURNAME;

IF(RES=1) THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;
```

```
 END
 $$
 DELIMITER ;
```

```
 DELIMITER $$
 CREATE PROCEDURE Helper_Procedure()
 BEGIN
IF SCIENTIST_EXP2() THEN
SELECT 'THE PROFILE ASSIGNED TO EACH EMPLOYEE IN THE DATA SCIENCE
TEAM MATCHES THE ORGANIZATION SET STANDARD.' AS MESSAGE;
ELSE
SELECT 'THE PROFILE ASSIGNED TO EACH EMPLOYEE IN THE DATA SCIENCE
TEAM DOES NOT MATCH THE ORGANIZATION SET STANDARD.' AS MESSAGE;
END IF;
END $$
DELIMITER ;

CALL Helper_Procedure();
```

Difference:

- Only procedure name is different.

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.

My code

```
CREATE INDEX idx_first_name
ON emp_record_table(FIRST_NAME(20));
SELECT * FROM emp_record_table
WHERE FIRST_NAME='Eric';
```

Explanation:

- Created a store procedure using the syntex.
- This procedure is returning Records fro emp_record_table where experience is Greater than 3.

Ratinder Pal Singh Solution’s

```
create index ename_index

on emp_record_table(first_name);

select *

from emp_record_table

where first_name = 'Eric';
```

Difference:

- Same as me.

Mahesh Gupta Solution's

```
CREATE INDEX Eric_Index

ON emp_record_table(FIRST_NAME);

SELECT FIRST_NAME

FROM emp_record_table

WHERE FIRST_NAME='Eric';
```

Difference:

- Same as me.

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).

My code

```
SELECT EMP_ID, CONCAT(FIRST_NAME, ' ', LAST_NAME) AS NAME, SALARY,
ROUND(((SALARY*5)/100)*EMP_RATING) AS BONUS
FROM emp_record_table;
```

Explanation:

- Simple query same as explained above.

- Calculating the "BONUS" for each employee by multiplying the "SALARY" with 5% and then multiplying it with the employee's "EMP_RATING".
- The bonus amount is rounded to the nearest integer using the ROUND() function.

Ratinder Pal Singh Solution's

```
select emp_id, emp_rating, salary, (0.05*salary*emp_rating) as bonus
from emp_record_table;
```

Difference:

- Same as me.

Mahesh Gupta Solution's

```
SELECT EMP_ID,SALARY,EMP_RATING, (SALARY*0.05*EMP_RATING)
AS BONUS FROM emp_record_table ;
```

Difference:

- Same as me.

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

My code

```
SELECT  DISTINCT COUNTRY, AVG(SALARY) OVER(PARTITION BY COUNTRY) AS
COUNTRY_AVG, CONTINENT, AVG(SALARY) OVER(PARTITION BY CONTINENT) AS
CONTINENT_AVG
FROM emp_record_table;
```

Explanation:

- The DISTINCT keyword ensures that only unique country names are returned.
- The AVG function calculates the average salary for each group defined by the PARTITION BY clause.

Ratinder Pal Singh Solution's

```
select distinct continent, avg(salary) over(partition by continent) as
avg_sal_by_continent,

country, avg(salary) over(partition by country) as avg_sal_by_country

from emp_record_table;
```

Difference:

- Same as me.

Mahesh Gupta Solution's

```
SELECT DISTINCT CONTINENT,COUNTRY ,

AVG(SALARY) OVER(PARTITION BY CONTINENT) AS

AVG_SALARY_BY_CONTINENT,

AVG(SALARY) OVER(PARTITION BY COUNTRY) AS
```

```
AVG_SALARY_BY_COUNTRY

FROM emp_record_table;
```

Difference:

- Same as me.