**Project Write-Up: Employee Performance Mapping Using MySQL**

**1. Introduction**

The project focuses on analyzing employee performance data for **ScienceQtech**, a data science startup. The HR team requires reports on **employee details, performance ratings, and project assignments** to finalize performance mapping and ensure job profile alignment. As a **Junior Database Administrator (DBA)**, I performed **basic SQL operations** to extract insights from the given datasets.

**2. Approach Taken ( SQL Commands Used)**  
**> Creating a database and importing data**  
**>Fetching records using SELECT queries**  
**> Using WHERE, GROUP BY, and ORDER BY for data filtering**  
**>**  **Using JOINs to merge multiple tables**  
 **> Applying UNION, aggregate functions (MAX, MIN, AVG), and ranking**  
 **> Using stored procedures and functions for structured querying**  
**>**  **Creating indexes for performance optimization**

**3. Steps & Queries Executed**

**Step 1: Database Setup**

* Created a **database** named employee\_management.
* Imported CSV files (data\_science\_team.csv, proj\_table.csv, emp\_record\_table.csv).

CREATE DATABASE employee\_management;

USE employee\_management;

**Step 2: Fetching Employee Details**

* Retrieved basic employee details from emp\_record\_table.

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

FROM emp\_record\_table;

**Step 3: Filtering Employees Based on Performance**

* Listed employees based on different performance rating criteria.

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

FROM emp\_record\_table

WHERE EMP\_RATING < 2

OR EMP\_RATING > 4

OR EMP\_RATING BETWEEN 2 AND 4;

**Step 4: Concatenating Employee Names in Finance Department**

* Merged first and last names for employees in the **Finance** department.

select concat(FIRST\_NAME, LAST\_NAME) as NAME

from emp\_record\_table

where DEPT = ('Finance');

**Step 5: Listing Employees with Direct Reports**

* Identified managers and the number of employees reporting to them.

select manager\_id,

count(emp\_id) as no\_of\_reporters

from emp\_record\_table

group by manager\_id

order by manager\_id;

**Step 6: Union Query for Healthcare & Finance Departments**

* Combined employees from **Healthcare** and **Finance** using UNION.

SELECT first\_name, last\_name, dept

FROM emp\_record\_table

WHERE dept = 'HEALTHCARE'

UNION

SELECT first\_name, last\_name, dept

FROM emp\_record\_table

WHERE dept = 'FINANCE';

**Step 7: Grouping Employees by Department & Ratings**

* Grouped employee ratings with department-wise max ratings.

select EMP\_ID, FIRST\_NAME, LAST\_NAME, ROLE,DEPT, EMP\_RATING, EMP\_RATING as max\_rating

from emp\_record\_table

where (dept,emp\_rating)

IN (SELECT dept, MAX(emp\_rating) FROM emp\_record\_table GROUP By dept)

ORDER BY dept ASC;

**Step 8: Finding Minimum & Maximum Salaries for Each Role**

SELECT role, MIN(salary) AS minSalary, MAX(salary) AS maxSalary

FROM emp\_record\_table

GROUP BY role;

**Step 9: Ranking Employees Based on Experience**

SELECT FIRST\_NAME, LAST\_NAME, EXP as experience,

dense\_rank() over (order by exp desc)AS emp\_rank

FROM emp\_record\_table ;

**Step 10: Creating a View for High Salary Employees**

create view emp\_by\_contries as

select COUNTRY, EMP\_ID, FIRST\_NAME, LAST\_NAME,SALARY

from emp\_record\_table

where salary > 6000;

**Step 11: Finding Employees with More Than 10 Years of Experience (Nested Query)**

select emp\_id,first\_name,last\_name,exp from emp\_record\_table

WHERE exp IN (

SELECT exp

FROM emp\_record\_table

WHERE exp > 10

);

**Step 12: Stored Procedure for Employees with >3 Years Experience**

USE `employee\_management`;

DROP procedure IF EXISTS `emp\_exp`;

USE `employee\_management`;

DROP procedure IF EXISTS `employee\_management`.`emp\_exp`;

;

DELIMITER $$

USE `employee\_management`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `emp\_exp`()

BEGIN

select \* from emp\_record\_table

where exp>3;

END$$

DELIMITER ;

;

call emp\_exp();

**Step 13: Stored Function for Job Profile Validation**

USE `employee\_management`;

DROP procedure IF EXISTS `check\_role`;

USE `employee\_management`;

DROP procedure IF EXISTS `employee\_management`.`check\_role`;

;

DELIMITER $$

USE `employee\_management`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `check\_role`()

BEGIN

SELECT emp\_id, first\_name, exp,

CASE

WHEN exp <= 2 THEN 'JUNIOR DATA SCIENTIST'

WHEN exp BETWEEN 3 AND 5 THEN 'ASSOCIATE DATA SCIENTIST'

WHEN exp BETWEEN 6 AND 10 THEN 'SENIOR DATA SCIENTIST'

WHEN exp BETWEEN 11 AND 12 THEN 'LEAD DATA SCIENTIST'

WHEN exp BETWEEN 13 AND 16 THEN 'MANAGER'

ELSE 'ALL GOOD'

END AS assigned\_role

FROM emp\_record\_table;

END$$

DELIMITER ;

;

call check\_role();

**Step 14: Creating an Index for Fast Search on Employee Names**

ALTER TABLE emp\_record\_table ADD INDEX fname\_index (first\_name);

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

**Step 15: Calculating Bonus for Employees**

select first\_name,last\_name,salary, floor(salary\*5/100\*emp\_rating) as bonus from emp\_record\_table;

**Step 16: Average Salary Distribution by Continent & Country**

select continent, floor(avg(salary))as avg\_sal

from emp\_record\_table group by continent

order by continent asc;

select country, avg(salary)as avg\_sal

from emp\_record\_table group by country

order by country asc;

**4. Conclusion & Learnings**

**Basic SQL Queries Helped Solve Key Problems:**

* Extracted employee details efficiently
* Grouped and filtered employees based on **experience, ratings, and salaries**
* Ensured **job profile alignment** through **stored functions**
* Improved **query performance** using **indexes**
* Used **views, stored procedures, and nested queries** to simplify complex operations