**Industrial Internship at Elite Tech Intern.**

This report showcases advanced SQL queries written as part of Task 2 of the Elite Tech Intern program. The task focuses on analysing database trends and patterns using:

* Window Functions (e.g., ROW\_NUMBER, RANK, NTILE, LAG, etc.)
* Common Table Expressions (CTEs) for modular, readable queries
* Subqueries for filtering, comparison, and aggregation

Each query has been executed on the EmployeeDB database, and corresponding outputs have been captured and included below as screenshots. This demonstrates both query logic and practical results from real sample data.

**Task 2- Data Analysis & Complex Queries**

* **Window Functions**

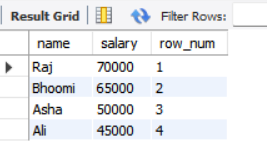
-- 1. ROW\_NUMBER: Assign row number based on salary.

SELECT e.name, s.salary,

ROW\_NUMBER() OVER (ORDER BY s.salary DESC) AS row\_num

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Assigns a unique row number to each employee in descending salary order.*

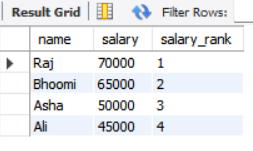
2. RANK: Rank employees based on salary.

SELECT e.name, s.salary,

RANK() OVER (ORDER BY s.salary DESC) AS salary\_rank

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Ranks employees by salary, allowing gaps for duplicate values.*

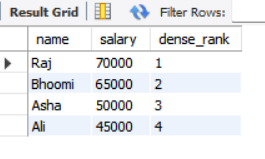
3. DENSE\_RANK – Dense rank without gaps.

SELECT e.name, s.salary,

DENSE\_RANK() OVER (ORDER BY s.salary DESC) AS dense\_rank

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Ranks employees by salary without skipping any ranks.*

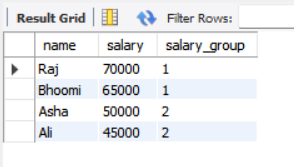
4. NTILE – Divide into salary groups.

SELECT e.name, s.salary,

NTILE(2) OVER (ORDER BY s.salary DESC) AS salary\_group

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Divides employees into two equal-sized salary groups.*

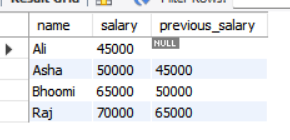
5. LAG – Previous salary.

SELECT e.name, s.salary,

LAG(s.salary, 1) OVER (ORDER BY s.salary) AS previous\_salary

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Shows each employee’s previous salary in sorted order.*

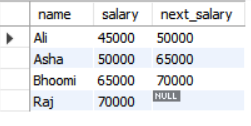
6. LEAD – Next salary.

SELECT e.name, s.salary,

LEAD(s.salary, 1) OVER (ORDER BY s.salary) AS next\_salary

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Shows each employee’s next salary in sorted order.*

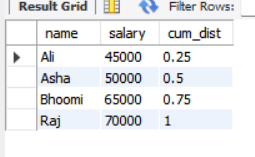
7. CUME\_DIST – Cumulative salary distribution.

SELECT e.name, s.salary,

CUME\_DIST() OVER (ORDER BY s.salary) AS cum\_dist

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Displays cumulative distribution of salaries.*

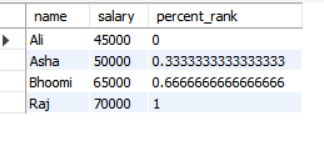
8. PERCENT\_RANK – Salary percentile.

SELECT e.name, s.salary,

PERCENT\_RANK() OVER (ORDER BY s.salary) AS percent\_rank

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Shows each employee’s salary percentile rank.*

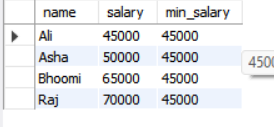
9. FIRST\_VALUE – First salary.

SELECT e.name, s.salary,

FIRST\_VALUE(s.salary) OVER (ORDER BY s.salary) AS min\_salary

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Returns the lowest salary from the result.*

10. LAST\_VALUE – Last salary.

SELECT e.name, s.salary,

LAST\_VALUE(s.salary) OVER (

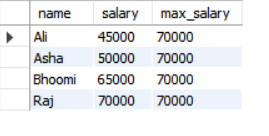
ORDER BY s.salary

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

) AS max\_salary

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Returns the highest salary from the result.*

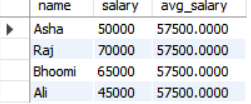
11. AVG OVER – Global average.

SELECT e.name, s.salary,

AVG(s.salary) OVER () AS avg\_salary

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id;



*Calculates the average salary across all employees.*

12. AVG PARTITION BY – Department average.

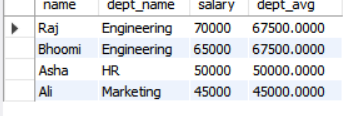
SELECT e.name, d.dept\_name, s.salary,

AVG(s.salary) OVER (PARTITION BY d.dept\_name) AS dept\_avg

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

JOIN Departments d ON e.dept\_id = d.dept\_id;



*Calculates the average salary per department.*

13. SUM PARTITION BY – Total department salary.

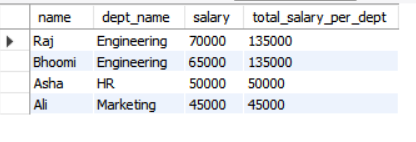
SELECT e.name, d.dept\_name, s.salary,

SUM(s.salary) OVER (PARTITION BY d.dept\_name) AS total\_salary\_per\_dept

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

JOIN Departments d ON e.dept\_id = d.dept\_id;



*Shows total salary for each department.*

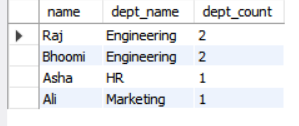
14. COUNT PARTITION BY – Department employee count.

SELECT e.name, d.dept\_name,

COUNT(\*) OVER (PARTITION BY d.dept\_name) AS dept\_count

FROM Employees e

JOIN Departments d ON e.dept\_id = d.dept\_id;



*Counts how many employees are in each department.*

15. MIN PARTITION BY – Department minimum salary.

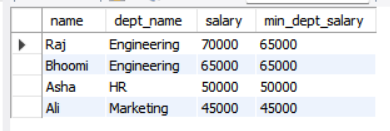
SELECT e.name, d.dept\_name, s.salary,

MIN(s.salary) OVER (PARTITION BY d.dept\_name) AS min\_dept\_salary

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

JOIN Departments d ON e.dept\_id = d.dept\_id;



*Finds the lowest salary in each department.*

16. MAX PARTITION BY – Department maximum salary.

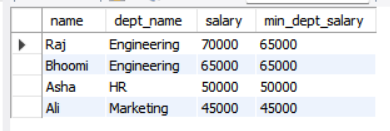
SELECT e.name, d.dept\_name, s.salary,

MAX(s.salary) OVER (PARTITION BY d.dept\_name) AS max\_dept\_salary

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

JOIN Departments d ON e.dept\_id = d.dept\_id;



*Finds the highest salary in each department.*

17. ROW\_NUMBER PARTITION BY – Row number within department.

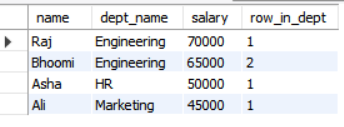
SELECT e.name, d.dept\_name, s.salary,

ROW\_NUMBER() OVER (PARTITION BY d.dept\_name ORDER BY s.salary DESC) AS row\_in\_dept

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

JOIN Departments d ON e.dept\_id = d.dept\_id;



*Assigns row numbers within each department.*

18. RANK PARTITION BY – Salary rank within department.

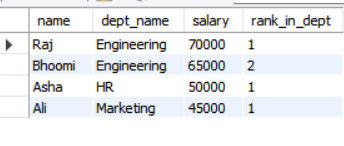
SELECT e.name, d.dept\_name, s.salary,

RANK() OVER (PARTITION BY d.dept\_name ORDER BY s.salary DESC) AS rank\_in\_dept

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

JOIN Departments d ON e.dept\_id = d.dept\_id;



*Ranks salaries within each department.*

19. DENSE\_RANK PARTITION BY – Dense rank within department.

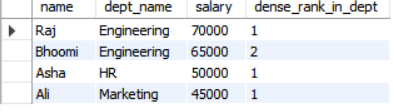
SELECT e.name, d.dept\_name, s.salary,

DENSE\_RANK() OVER (PARTITION BY d.dept\_name ORDER BY s.salary DESC) AS dense\_rank\_in\_dept

FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

JOIN Departments d ON e.dept\_id = d.dept\_id;



*Ranks salaries within each department without gaps.*

* **Common Table Expressions (CTEs)**

**1. CTE for employees with salary above average:**

WITH HighEarners AS (

SELECT e.emp\_id, e.name, s.salary

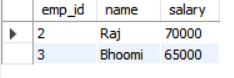
FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

WHERE s.salary > (SELECT AVG(salary) FROM Salaries)

)

SELECT \* FROM HighEarners;



*This query returns a list of employees whose salary is greater than the company’s average salary.*

**2. CTE for department-wise total salary:**

WITH DeptSalary AS (

SELECT d.dept\_name, SUM(s.salary) AS total\_salary

FROM Employees e

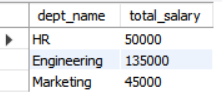
JOIN Salaries s ON e.emp\_id = s.emp\_id

JOIN Departments d ON e.dept\_id = d.dept\_id

GROUP BY d.dept\_name

)

SELECT \* FROM DeptSalary;



*It calculates the total salary paid by each department.*

**3. CTE for employee with latest hire date:**

WITH LatestHire AS (

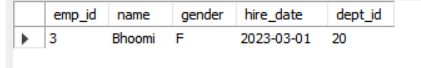
SELECT \*

FROM Employees

WHERE hire\_date = (SELECT MAX(hire\_date) FROM Employees)

)

SELECT \* FROM LatestHire;



*It retrieves the employee who was hired most recently.*

* **Subqueries**

1. **Employees earning more than Raj:**

SELECT name FROM Employees

WHERE emp\_id IN (

SELECT s.emp\_id FROM Salaries s

WHERE s.salary > (SELECT salary FROM Salaries WHERE emp\_id = 2)

);



*This query is designed to return the names of employees who earn more than Raj.  
In the current dataset, Raj has the highest salary, so no employee meets this condition — which is why the result is empty.*

1. **Departments with average salary above 60000:**

SELECT dept\_name FROM Departments

WHERE dept\_id IN (

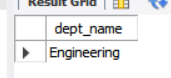
SELECT e.dept\_id FROM Employees e

JOIN Salaries s ON e.emp\_id = s.emp\_id

GROUP BY e.dept\_id

HAVING AVG(s.salary) > 60000

);



*Finds departments where the average salary of employees is more than 60,000.*

1. **Employees with the highest bonus:**

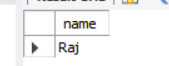
SELECT name FROM Employees

WHERE emp\_id IN (

SELECT emp\_id FROM Salaries

WHERE bonus = (SELECT MAX(bonus) FROM Salaries)

);

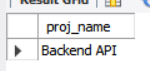


*Shows names of employees who received the maximum bonus.*

**4. Project with max hours:**

SELECT proj\_name FROM Projects

WHERE hours\_worked = (SELECT MAX(hours\_worked) FROM Projects);



*Gives the name of the project that has the highest number of hours worked*.

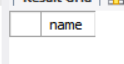
1. **Names of employees not working on any project:**

SELECT name FROM Employees

WHERE emp\_id NOT IN (

SELECT DISTINCT emp\_id FROM Projects

);



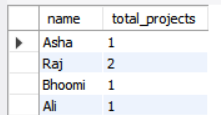
*This query checks for employees who are not assigned to any project.  
Since all employees in the dataset are assigned to at least one project, the query returns no output.*

**6. Total number of projects per employee (with subquery in SELECT):**

SELECT e.name,

(SELECT COUNT(\*) FROM Projects p WHERE p.emp\_id = e.emp\_id) AS total\_projects

FROM Employees e;



*Displays each employee’s name with the total number of projects they are working on.*

* *Note: Some queries like Subquery 1 and 5 may return no output due to the nature of the current dataset. This indicates that no rows met the specified condition — not an error in the query.*