DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment -3

Student Name: Daksh Pushpad **UID:** 23BAI70264

Branch: BE-AIT-CSE **Section/Group:** 23AIT_KRG-G1_A

Semester: 5th **Date of Performance:** 20 August, 2025

Subject Name: ADBMS **Subject Code:** 23CSP-333

1. Aim:

EASY LEVEL PROBLEM:

You are given a table named EMPLOYEE with a single column, EMP_ID. The table contains multiple entries, some of which are duplicates. Your task is to write a single query that finds the maximum value among only those EMP_IDs that appear exactly once. You must solve this problem using a subquery.

MEDIUM LEVEL PROBLEM:

In a bustling corporate organization, each department strives to retain the most talented (and well-compensated) employees. You have access to two key records: **one lists every employee along with their salary and department**, **while the other details the names of each department**. Your task is to identify the **top earners in every department**.

If multiple employees share the same highest salary within a department, all of them should be celebrated equally. The final result should present the **department name, employee name, and salary of these top-tier professionals** arranged by department.

HARD LEVEL PROBLEM:

Two legacy HR systems (A and B) have separate records of employee salaries. These records may overlap. Management wants to **merge these datasets** and identify **each unique employee (**by EmpID) along with their **lowest recorded salary** across both systems.

2. Objective:

The objective is to demonstrate a comprehensive command of SQL by solving a series of progressively complex data management and analysis challenges. This involves:

- **Advanced Filtering and Aggregation**: Using **subqueries** and GROUP BY to filter data based on counts and find aggregate values.
- **Data Analysis**: Employing window functions or nested queries to perform advanced analysis, such as identifying the highest values within specific groups.
- **Data Manipulation**: Using set operations like **UNION** to combine and de-duplicate records from multiple sources.

The ultimate goal is to showcase a strong understanding of SQL logic by applying these techniques to extract, analyze, and present accurate and meaningful insights from various datasets.

3. Theory:

Aggregation and Grouping:

Aggregation involves calculating a single value from a set of values. Functions like MAX(), MIN(), AVG(), SUM(), and COUNT() perform these calculations. When we want to apply these functions to subsets of data, we use the **GROUP BY** clause. It groups rows that have the same values in specified columns into summary rows. For instance, you could use GROUP BY department with MAX(salary) to find the highest salary for each individual department.

Subqueries:

A **subquery** (or inner query) is a query nested inside another query (the outer query). It executes first, and its result is then used by the outer query. Subqueries are often used to filter the outer query's results based on a value or set of values calculated from the inner query. For example, to find all employees with the highest salary in their department, you could use a subquery to first find the maximum salary for each department, and then use that result in the main query's WHERE clause.

Set Operations:

Set operations combine the results of two or more SELECT statements. The **UNION** operator merges the result sets, removing any duplicate rows. This is useful for combining data from multiple tables with the same column structure into a single, clean result. In contrast, UNION ALL performs the same merge but includes all rows,

keeping duplicates. UNION is perfect for scenarios where you need to get a single list of unique entries from different sources, like combining salary records from two separate systems to find each employee's lowest unique salary.

4. Procedure:

Easy Problem: Finding the Max Unique EMP_ID

- **Step 1:** The inner query (SELECT EMPID FROM EMP GROUP BY EMPID HAVING COUNT(EMPID) = 1) groups the EMP table by EMPID and uses the HAVING clause to filter for groups where the count is exactly one. This effectively isolates the unique EMPIDs (in this case, 2 and 7).
- **Step 2:** The outer query (SELECT MAX(EMPID) FROM EMP WHERE EMPID IN (...)) then takes the result of the subquery and finds the maximum value from that set, which is 7.

Medium Problem: Top Earners per Department

- **Step 1:** An INNER JOIN connects the employee and department tables on their common department_id, allowing the final output to include department names.
- **Step 2:** The WHERE clause contains a subquery (SELECT MAX(e2.salary) FROM employee AS e2 WHERE e2.department_id = e.department_id). This subquery is "correlated" because it references a column (e.department_id) from the outer query. For each employee in the outer query, the subquery calculates the maximum salary for that employee's specific department.
- **Step 3:** The outer query then filters the joined result set, keeping only those employees whose salary matches the MAX(salary) for their department, thus identifying the top earners.

Hard Problem: Lowest Salary per Unique Employee

- **Step 1:** The inner query (SELECT * FROM TABLEA UNION ALL SELECT * FROM TABLEB) combines all rows from both tables into a single, temporary result set. Using UNION ALL is crucial here because it includes all records, even if an employee appears in both tables, which is necessary to check for the lowest salary across both records.
- **Step 2:** The outer query (SELECT EMPID, MIN(ENAME) AS ENAME, MIN(SALARY) AS SALARY FROM ... GROUP BY EMPID) then performs the main analysis. It groups the combined data by EMPID.
- **Step 3:** Within each group, it applies the MIN() function to the SALARY column, effectively finding the lowest salary recorded for that employee across both tables. MIN(ENAME) is used to select a name for each EMPID since it's

required for the output, and in this case, the MIN function can arbitrarily select one of the potentially identical names.

5. Code:

```sql
-- Easy Problem

CREATE TABLE EMP (
     EMPID INT
);

INSERT INTO EMP VALUES
     (2),
     (4),
     (4),
     (6),
     (6),
     (7),
     (8),
     (8);

SELECT MAX(EMPID) FROM EMP
WHERE EMPID NOT IN
     (SELECT * FROM EMP
     GROUP BY EMPID
           HAVING COUNT(EMPID) > 1)
```

```sql
--Medium Level Problem

CREATE TABLE department (
    id INT PRIMARY KEY,
    dept_name VARCHAR(50)
);


CREATE TABLE employee (
    id INT,
    name VARCHAR(50),
    salary INT,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES department(id)
);



INSERT INTO department (id, dept_name) VALUES
(1, 'IT'),
(2, 'SALES');


INSERT INTO employee (id, name, salary, department_id) VALUES
(1, 'JOE', 70000, 1),
(2, 'JIM', 90000, 1),
(3, 'HENRY', 80000, 2),
(4, 'SAM', 60000, 2),
(5, 'MAX', 90000, 1);

SELECT D.DEPT_NAME , E.NAME , E.SALARY , E.DEPARTMENT_ID
```

```sql
FROM EMPLOYEE AS E
INNER JOIN
DEPARTMENT AS D
ON E.DEPARTMENT_ID = D.ID
WHERE E.SALARY IN
    (
        SELECT MAX(E2.SALARY)
        FROM EMPLOYEE AS E2
        WHERE E2.DEPARTMENT_ID = E.DEPARTMENT_ID
    )
ORDER BY D.DEPT_NAME


SELECT D.DEPT_NAME , E.NAME , E.SALARY , E.DEPARTMENT_ID
FROM EMPLOYEE AS E
INNER JOIN
DEPARTMENT AS D
ON E.DEPARTMENT_ID = D.ID
WHERE E.SALARY IN
    (
        SELECT MAX(E2.SALARY)
        FROM EMPLOYEE AS E2
        GROUP BY E2.DEPARTMENT_ID
    )

--Hard Level Problem

CREATE TABLE TABLEA
(
    EMPID INT PRIMARY KEY,
    ENAME VARCHAR(50),
    SALARY INT
);

CREATE TABLE TABLEB
(
    EMPID INT,
    ENAME VARCHAR(50),
    SALARY INT

);

INSERT INTO TABLEA (EMPID,ENAME,SALARY) VALUES
    (1,'AA',1000),
    (2,'BB',300);

INSERT INTO TABLEB (EMPID,ENAME,SALARY) VALUES
    (2,'BB',400),
    (3,'CC',100);

SELECT EMPID , MIN(ENAME) AS ENAME, MIN(SALARY) AS SALARY
FROM
(
    SELECT * FROM TABLEA
    UNION ALL
    SELECT * FROM TABLEB
) AS INTERMEDIATE_RESULT
GROUP BY EMPID

SELECT EMPID , ENAME, MIN(SALARY) AS SALARY
FROM
(
    SELECT * FROM TABLEA
    UNION ALL
```

```
    SELECT * FROM TABLEB
) AS INTERMEDIATE_RESULT
GROUP BY EMPID , ENAME
```

6. Output:

Results | Messages

	(No column name)
1	7

Results | Messages

	DEPT_NAME	NAME	SALARY	DEPARTMENT_ID
1	IT	MAX	90000	1
2	IT	JIM	90000	1
3	SALES	HENRY	80000	2

	DEPT_NAME	NAME	SALARY	DEPARTMENT_ID
1	IT	JIM	90000	1
2	IT	MAX	90000	1
3	SALES	HENRY	80000	2

Results | Messages

	EMPID	ENAME	SALARY
1	1	AA	1000
2	2	BB	300
3	3	CC	100

	EMPID	ENAME	SALARY
1	1	AA	1000
2	2	BB	300
3	3	CC	100

7. Learning Outcomes:

- **Translating Business Logic into SQL:** The ability to convert complex, real-world data problems into a structured series of SQL commands, demonstrating a practical understanding of how to retrieve and manipulate data to solve business challenges.
- **Developing a Multi-Step Approach:** Learning to break down a difficult problem into smaller, solvable parts by using techniques like subqueries and combining multiple operations within a single, elegant query.
- **Choosing the Right Tool:** Gaining a deeper understanding of when to use specific SQL clauses and functions—such as GROUP BY for aggregation, INNER JOIN for combining, UNION ALL for merging, and subqueries for advanced filtering—to achieve a desired result.