DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 1.3

Student Name: S S Sneha
UID: 23BAI70259
Branch: BE-AIT-CSE
Section/Group: 23AML-1 (B)
Semester: 5th
Date of Performance: 19 August 2025
Subject Name: ADBMS
Subject Code: 23CSP-333

1. **Experiment Name:**
   To understand and apply SQL concepts such as keys, joins, subqueries, and set operations for effective data retrieval and analysis.

2. **Objective:**

Medium-Level Problem

Problem Title: Top Earners in Each Department Using Joins and Aggregates

Procedure (Step-by-Step):

1. Create two tables:
   - Departments(DeptID, DeptName)
   - Employees(EmpID, EmpName, Salary, DeptID [foreign key referencing Departments]).
2. Insert at least 10–12 records into the Employees table, ensuring:
   - Multiple employees belong to the same department.
   - Some employees share the same highest salary in a department.
3. Write a query using JOIN to connect employees with their department names.
4. Use a subquery or window function to determine the maximum salary within each department.
5. Select the department name, employee name, and salary of only those employees whose salary matches the maximum salary of their department.
6. Order the result set by department name for clarity.

Hard-Level Problem

Problem Title: Merging Legacy HR Systems and Finding Lowest Salary per Employee

Procedure (Step-by-Step):

1. Create two tables to represent the legacy systems:
   - System A (EmpID, Ename, Salary)
   - System B (EmpID, Ename, Salary)
2. Insert at least 6–8 employee records into both tables, ensuring:
   - Some employees appear in both systems (overlap).
   - Some employees appear only in one system.
   - Salaries may differ for the same employee across systems.
3. Use UNION (or UNION ALL) to merge records from both tables into a single combined dataset.
4. For each EmpID, find the minimum salary across the merged dataset.
5. Select and display the EmpID, Employee Name, and Lowest Salary.
6. Order the results by EmpID for clarity.

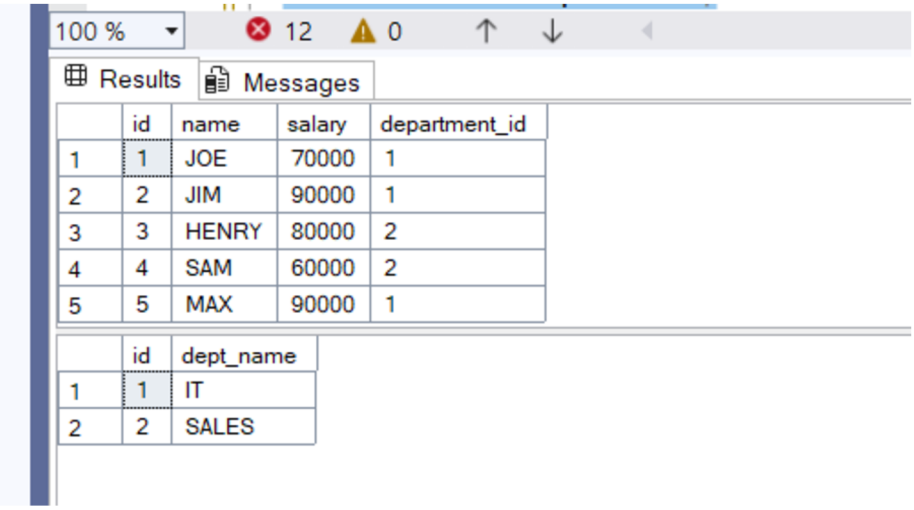
Code:

```
CREATE DATABASE EXPERIMENT3;
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 * FROM employee;
SELECT * FROM department;

SELECT D.DEPT_NAME, E.NAME, E.salary
FROM employee AS E
INNER JOIN department AS D
  ON D.ID = E.department_id
WHERE salary IN (
  SELECT MAX(salary)
  FROM employee AS E2
  WHERE E2.department_id = E.department_id
)
ORDER BY D.DEPT_NAME;



CREATE TABLE A (
  EMPID INT PRIMARY KEY,
  ENAME VARCHAR(MAX),
  SALARY INT
);
CREATE TABLE B (
  EMPID INT PRIMARY KEY,
  ENAME VARCHAR(MAX),
  SALARY INT
);
INSERT INTO A VALUES (1, 'AA', 5000), (2, 'BB', 3000);
INSERT INTO B VALUES (2, 'BB', 7000), (3, 'CC', 4000);
SELECT * FROM A;
SELECT * FROM B;
SELECT EMPID,
    MIN(ENAME) AS ENAME,
    MIN(SALARY) AS SALARY
```

```
FROM (
 SELECT * FROM A
 UNION ALL
 SELECT * FROM B
) AS INTERMEDIATE_RESULT
GROUP BY EMPID;
```

OUTPUTS:

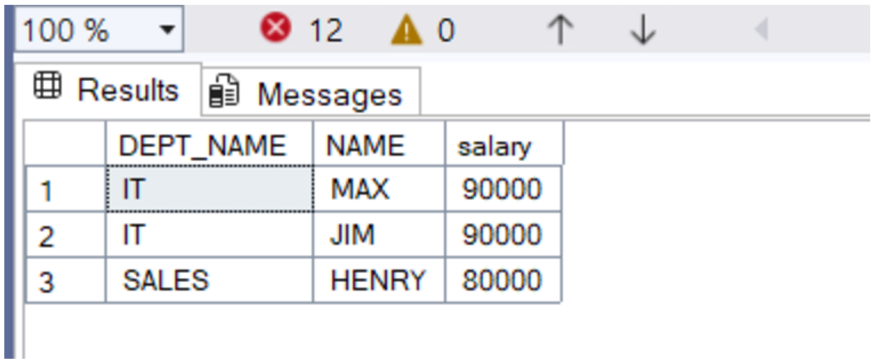


100 % 12 0

Results Messages

	id	name	salary	department_id
1	1	JOE	70000	1
2	2	JIM	90000	1
3	3	HENRY	80000	2
4	4	SAM	60000	2
5	5	MAX	90000	1

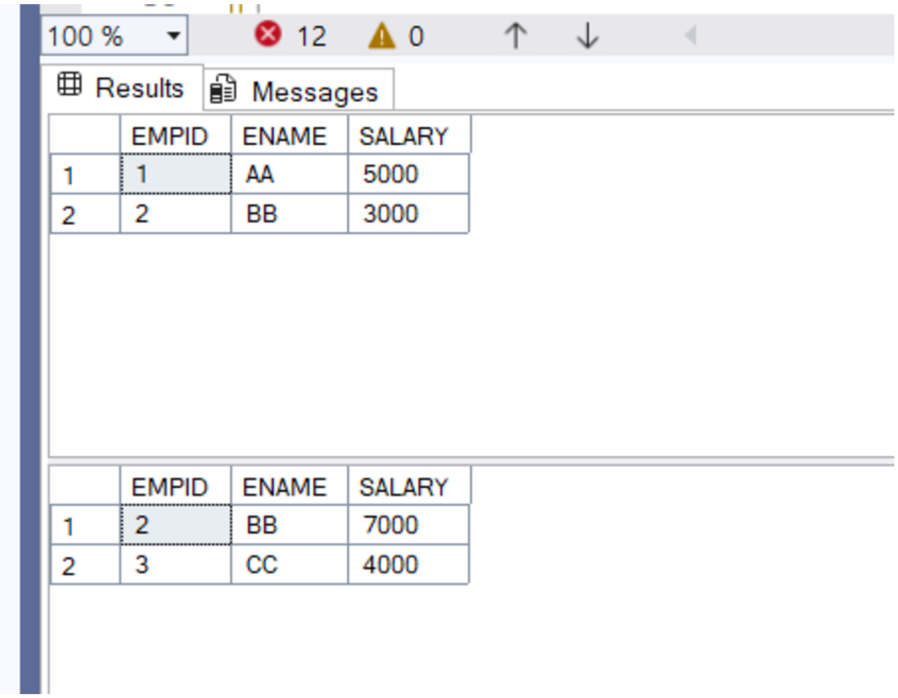
	id	dept_name
1	1	IT
2	2	SALES



100 % 12 0

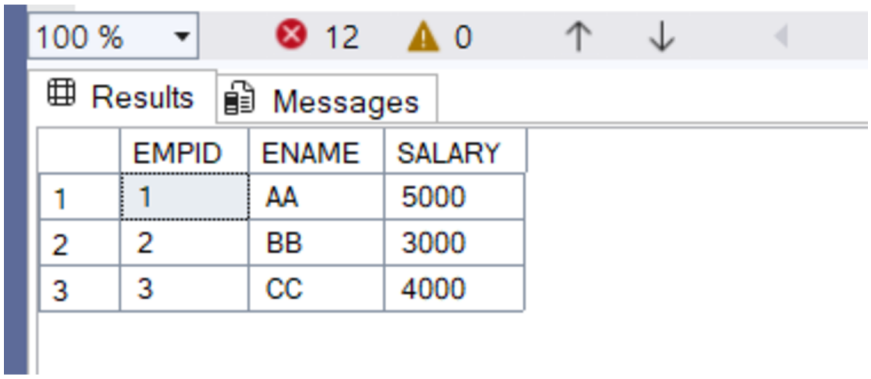
Results Messages

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



	EMPID	ENAME	SALARY
1	1	AA	5000
2	2	BB	3000

	EMPID	ENAME	SALARY
1	2	BB	7000
2	3	CC	4000



	EMPID	ENAME	SALARY
1	1	AA	5000
2	2	BB	3000
3	3	CC	4000

4. Learning Outcomes:

- Understand and implement **self-joins** and **foreign key relationships** for hierarchical data within the same table.
- Practiced **aggregate functions & subqueries** (MAX, SUM, COUNT).
- Applied **joins** to combine data across tables.
- Used UNION ALL and GROUP BY for data merging and summarisation.
- Improved **problem-solving** from easy (subqueries) → medium (joins) → hard (set operations).