SQL Subqueries
Task - 6

What is subquerys ?

A subquery (also called nested query) is a query inside another SQL query. It helps you filter, compute, or
transform data by using the result of one query as input to another.
This guide shows step-by-step how to write scalar subqueries, correlated subqueries, and subqueries inside

IN, *EXISTS*, *=*, and *FROM* clauses.

Step 1 — Create sample tables (Departments and Employees)

```
CREATE TABLE Departments (DeptID INT PRIMARY KEY, DeptName
VARCHAR(50));
CREATE TABLE Employees (EmployeeID INT PRIMARY KEY, Name
VARCHAR(50), DeptID INT,
Salary INT);
INSERT INTO Departments (DeptID, DeptName) VALUES (10, 'Sales'),
(20, 'HR'), (30,
'IT');
INSERT INTO Employees (EmployeeID, Name, DeptID, Salary) VALUES
(1, 'John', 10, 5000),
(2, 'Alice', 10, 7000),
(3, 'Bob', 20, 4500),
(4, 'Eve', 20, 6000),
(5, 'Carol', 30, 3000);
```

Step 2 — Scalar subquery (single value) in SELECT

A scalar subquery returns a single value (one row, one column). It can be used in the SELECT list or WHERE
clause.

Example: Show each employee's salary and the overall average salary.

```
SELECT Name, Salary,
(SELECT AVG(Salary) FROM Employees) AS AvgAllSalariesFROM
Employees;
```

Step-by-step explanation:

1. The subquery *(SELECT AVG(Salary) FROM Employees)* computes one value: the average salary across
all employees.
2. The outer query lists each employee together with that single computed value.

Step 3 — Correlated subquery (depends on outer query) in WHERE

A correlated subquery references columns from the outer query and is evaluated once per outer row.

Example: Find employees who earn more than the average salary of their own department.

```
SELECT e.Name, e.Salary, e.DeptID
FROM Employees e
WHERE e.Salary > (SELECT AVG(e2.Salary) FROM Employees e2 WHERE
e2.DeptID = e.DeptID);
```

Step-by-step explanation:

1. For each row in Employees (alias *e*), the subquery computes the average salary for that employee's DeptID.
2. The outer WHERE tests if the employee's salary is greater than that department average.
3. This returns employees above their department average.

Step 4 — Subquery with IN

Use IN when the subquery returns multiple rows and you want to test membership.

Example: List departments that have at least one employee with salary > 5000.

```
SELECT DeptName FROM Departments WHERE DeptID IN (SELECT DeptID
FROM Employees
WHERE Salary > 5000);
```

Explanation:

1. The inner query returns DeptIDs that have employees with Salary > 5000.
2. The outer query returns department names for those DeptIDs.

Step 5 — EXISTS (efficient for existence checks)

EXISTS checks whether the subquery returns any row. It stops at the first match (often more efficient).

Example**:** Get department names that have any employee earning > 5000.

```
SELECT DeptName FROM Departments d WHERE EXISTS (SELECT 1 FROM
Employees e WHERE
e.DeptID = d.DeptID AND e.Salary > 5000);
```

Note: Use SELECT 1 inside EXISTS — the returned columns are ignored, only row existence matters.

Step 6 — Subquery in FROM (Derived table / inline view)

A subquery in the FROM clause acts like a temporary table (derived table) that the outer query can use.

Example: Compute department averages first, then select departments with average > 4500.

```
SELECT sub.DeptID, sub.DeptAvg FROM (SELECT DeptID, AVG(Salary)
AS DeptAvg FROM
Employees GROUP BY DeptID) sub WHERE sub.DeptAvg > 4500;
```

Step-by-step:

1. Inner query groups salaries by DeptID and computes DeptAvg.
2. Outer query filters those dept averages.

Step 7 — When subqueries return multiple rows

If a subquery returns multiple rows, you cannot use '=' (equals) — use IN or EXISTS instead.

Bad: `WHERE DeptID = (SELECT DeptID FROM Employees WHERE Salary > 5000);` -- causes error
if multiple DeptIDs returned.

Good: `WHERE DeptID IN (SELECT DeptID FROM Employees WHERE Salary > 5000);`

.Interview Questions

1.what is subquery ?
A query nested inside another query. It provides intermediate results used by the outer query.

2.Difference between subquery and join ?

Subqueries compute values or sets used by the outer query; JOINs combine rows across tables directly.
Sometimes JOIN is clearer and faster; sometimes subquery is simpler

3.What is a correlated subquery ?
A subquery that references columns from the outer query. It runs once per outer row.

4.Can subqueries return multiple rows ?
Yes. Use IN, EXISTS or JOIN to handle multiple-row results. '=' only works for single-row results.

5.How does EXISTS work ?
EXISTS checks whether the subquery returns at least one row. It is boolean (true/false) and usually efficient
as it stops at the first match.

6.How is performance affected by subqueries ?
Correlated subqueries can be slow because they execute per outer row. Use indexes and consider JOINs or
derived tables for better performance.

7.What is scalar subquery?
A subquery that returns exactly one value (one row, one column).

8.Where can we use subqueries?
In SELECT, WHERE, FROM, HAVING clauses — and even inside INSERT/UPDATE to compute values.

9.Can a subquery be in FROM clause?
Yes — that creates a derived table (inline view) which the outer query can query from.

10. What is a derived table?
A derived table is a subquery used inside the FROM clause acting like a temporary table for the outer query.