SQL Subqueries – Complete Notes

1. Introduction to Subqueries

A subquery is a SQL query embedded inside another query. It's also called a nested query or inner query. The main (outer) query uses the result of the subquery to perform its operation.

Subqueries allow you to:

Perform multiple steps in a single query.

Retrieve data based on values calculated dynamically.

Simplify complex logic by breaking it into steps.

2. Basic Syntax of a Subquery

SELECT column_name

FROM table_name

WHERE column_name operator (SELECT column_name FROM table_name WHERE condition);

The subquery:

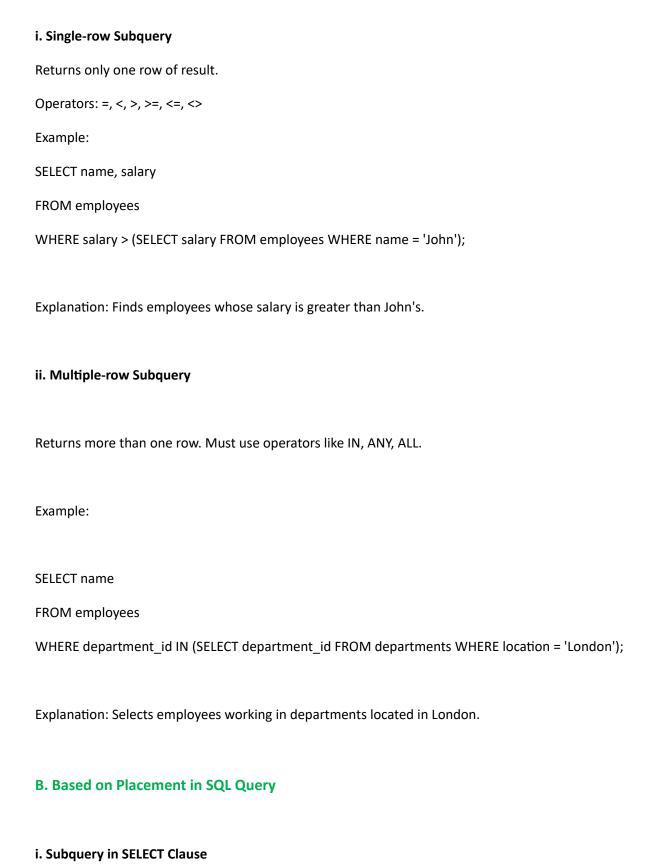
Must be enclosed in parentheses ()

Must be placed after a comparison operator (=, <, IN, EXISTS, etc.)

Is executed before the outer query

3. Types of Subqueries

A. Based on Rows Returned



Returns a scalar value (single value per row) as a column in the SELECT list.
Example:
SELECT name,
(SELECT MAX(salary) FROM employees) AS max_salary
FROM employees;
Explanation: Each row includes the highest salary in the company.
ii. Subquery in FROM Clause (Inline View)
Acts as a temporary table or view used by the outer query.
Example:
SELECT dept, AVG(salary)
FROM (SELECT department_id AS dept, salary FROM employees) AS sub
GROUP BY dept;
iii. Subquery in WHERE Clause
Filters rows based on a condition that uses the result of a subquery.
Example:
SELECT name
FROM employees

WHERE department_id = (SELECT id FROM departments WHERE name = 'IT'); C. Correlated vs Non-Correlated i. Non-Correlated Subquery Independent of outer query. Executes once and result is reused. Example: SELECT name FROM employees WHERE salary > (SELECT AVG(salary) FROM employees); ii. Correlated Subquery Depends on the outer query. Executes once per row in the outer query. Example: SELECT name, salary FROM employees e1 WHERE salary > (SELECT AVG(salary) FROM employees e2

```
WHERE e1.department_id = e2.department_id
);
Explanation: Compares each employee's salary to the average salary of their department.
4. Operators Used in Subqueries
i. IN
Checks if a value exists in a list returned by a subquery.
SELECT name FROM employees WHERE department_id IN (SELECT id FROM departments WHERE
location = 'London');
ii. EXISTS
Checks if subquery returns any row (used with correlated subqueries).
SELECT name
FROM employees e
WHERE EXISTS (
SELECT 1 FROM departments d WHERE e.department_id = d.id AND d.location = 'London'
);
iii. ANY / SOME
```

Returns true if any value from the subquery satisfies the condition.

SELECT name

```
FROM employees
WHERE salary > ANY (SELECT salary FROM employees WHERE department_id = 3);
iv. ALL
Returns true if all values satisfy the condition.
SELECT name
FROM employees
WHERE salary > ALL (SELECT salary FROM employees WHERE department_id = 3);
5. Nested Subqueries
You can nest a subquery inside another subquery.
Example:
SELECT name
FROM employees
WHERE department_id = (
SELECT id
FROM departments
WHERE location_id = (
 SELECT id FROM locations WHERE city = 'New York'
)
);
```

6. Subqueries in INSERT, UPDATE, DELETE

i. INSERT with Subquery

INSERT INTO employees_archive (id, name, salary)

SELECT id, name, salary FROM employees WHERE salary > 80000;

ii. UPDATE with Subquery

UPDATE employees

SET salary = salary * 1.10

WHERE department_id = (SELECT id FROM departments WHERE name = 'Sales');

iii. DELETE with Subquery

DELETE FROM employees

WHERE department_id IN (SELECT id FROM departments WHERE location = 'Berlin');

7. Performance Considerations

Non-correlated subqueries are more efficient than correlated subqueries.

In large datasets, JOINs might be faster than subqueries.

Indexes can improve subquery performance.

Real-Life Use Cases

Finding top performers (salary, score, sales)

Getting latest order for each customer

Fetching dependent data (like address of a customer)

Comparing with averages, totals, or minimums

Summary

Subqueries are a powerful way to break down complex queries.

They can be used in SELECT, FROM, WHERE, INSERT, UPDATE, and DELETE statements.

Understand the difference between correlated and non-correlated subqueries.

Always check for performance implications and consider JOINs when needed.