**SQL Subqueries**

A **subquery** in SQL is a query nested inside another query. It is used to perform operations and return data that the outer query can use. Subqueries can exist in various parts of an SQL statement, such as the SELECT, FROM, or WHERE clauses.

**Types of Subqueries**

1. **Single-row Subqueries**:
   * Returns one row with one or more columns.
   * Commonly used with comparison operators like =, <, >, etc.
2. SELECT name
3. FROM employees
4. WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Sales');
5. **Multi-row Subqueries**:
   * Returns multiple rows with one column.
   * Typically used with operators like IN, ANY, or ALL.
6. SELECT name
7. FROM employees
8. WHERE department\_id IN (SELECT department\_id FROM departments WHERE location\_id = 1700);
9. **Correlated Subqueries**:
   * Refers to columns in the outer query and evaluates for each row processed by the outer query.
   * Often used with EXISTS or NOT EXISTS.
10. SELECT name
11. FROM employees e
12. WHERE EXISTS (
13. SELECT 1
14. FROM projects p
15. WHERE p.employee\_id = e.employee\_id AND p.status = 'Active'
16. );
17. **Scalar Subqueries**:
    * Returns a single value (one row, one column) and can be used where a single value is expected.
18. SELECT name,
19. (SELECT department\_name
20. FROM departments
21. WHERE department\_id = employees.department\_id) AS department\_name
22. FROM employees;

**Placement of Subqueries**

1. **In the SELECT Clause**:
   * Used to calculate derived columns.
2. SELECT name,
3. (SELECT COUNT(\*) FROM projects WHERE projects.employee\_id = employees.employee\_id) AS project\_count
4. FROM employees;
5. **In the FROM Clause**:
   * Acts as a temporary table for the outer query.
6. SELECT avg\_salary
7. FROM (SELECT AVG(salary) AS avg\_salary FROM employees WHERE department\_id = 10) AS avg\_data;
8. **In the WHERE Clause**:
   * Used to filter rows based on conditions.
9. SELECT name
10. FROM employees
11. WHERE salary > (SELECT AVG(salary) FROM employees);

**Benefits of Subqueries**

* **Modularity**: Breaks complex queries into manageable components.
* **Reusability**: Allows for dynamic querying within a single SQL statement.
* **Clarity**: Makes queries easier to understand in certain contexts.

**Limitations**

* **Performance**: Subqueries can be slower than equivalent joins, especially correlated subqueries.
* **Complexity**: Excessive use can make queries harder to maintain and debug.

By using subqueries effectively, you can streamline data analysis and manipulation tasks in SQL.