Queries and sub queries

Objective: The objective of Subqueries in SQL is to enable nesting of queries within other queries to retrieve data dynamically based on the results of another query for more complex and efficient data analysis.

Queries

A **query** is an SQL statement used to retrieve, update, delete, or insert data from a database. A simple SQL query typically follows this structure:

Syntax: -

SELECT column1, column2, ...

FROM table_name

WHERE conditions;

Subqueries

A **subquery** is a query nested inside another query. It allows you to perform operations in multiple steps, where the result of one query (the subquery) is used as input for another query (the outer query). Subqueries are commonly used in the WHERE, FROM, or SELECT clauses of an SQL query.

Types of Subqueries

1. **Single-Row Subqueries:** Returns a single row and is used with operators like =, <, or >.
2. **Multi-Row Subqueries:** Returns multiple rows and uses operators like IN, ANY, or ALL.
3. **Nested Subqueries:** Subqueries inside other subqueries.
4. **Correlated Subquery:** Refers to columns in the outer query.

1. Single-row Subquery

A **Single-Row Subquery** is a type of SQL subquery that returns at most one row of results (zero or one) to the outer SQL statement. Because the result is a single value, the outer query uses standard single-row comparison operators.

Used with operators like =, <, >, etc.

Syntax

SELECT column1, column2

```
FROM table_name
WHERE column_to_compare [SINGLE-ROW OPERATOR] (
   SELECT single_value_column
   FROM another_table
   [WHERE subquery_condition]
);
SELECT emp_name
FROM Employees
WHERE dept_id = (SELECT dept_id FROM Departments WHERE dept_name = 'HR');
```

(This query retrieves the names of employees in the "HR" department.)

Example (Using = Operator)

To find employees who work in the 'Sales' department:

```
SELECT employee_name, department_id
FROM employees
WHERE department_id = (
   SELECT department_id
   FROM departments
   WHERE department_name = 'Sales'
);
```

Example (Using > Operator)

To find employees who earn more than the overall average salary:

```
SELECT employee_name, salary
FROM employees
WHERE salary > (
   SELECT AVG(salary)
   FROM employees
```

);

2. Multi-row Subquery

A **Multi-Row Subquery** is a type of SQL subquery that returns **more than one row** of results to the outer (or main) SQL statement. Because the outer query needs to compare a single value (or set of values) against a list of values, a multi-row subquery must use multi-row comparison operators.

The following operators are mandatory when working with a multi-row subquery:

Operator	Meaning	Example Use Case
IN	Equal to any member in the list.	Find employees in a list of department IDs.
NOT IN	Not equal to any member in the list.	Find employees not belonging to a list of department IDs.
ANY	Compares a value to each value returned by the subquery. Requires a preceding comparison operator (>, <, =).	Find employees earning more than at least one employee in a list.
ALL	Compares a value to every value returned by the subquery. Requires a preceding comparison operator (>, <, =).	Find employees earning more than every employee in a list.

Syntax

```
SELECT column1, column2
FROM table_name
WHERE column_to_compare [MULTI-ROW OPERATOR] (
   SELECT column_to_return_list
   FROM another_table
   [WHERE subquery_condition]
);
```

Example (Using IN Operator)

To find all employees who work in departments located in 'New York' or 'London':

```
SELECT employee_name, department_id
FROM employees
```

```
WHERE department_id IN (
   SELECT department_id
   FROM departments
   WHERE location IN ('New York', 'London')
);
```

Example (Using > ANY Operator)

To find employees whose salary is greater than the minimum salary of the Marketing department (i.e., greater than at least one person in Marketing):

```
SELECT employee_name, salary
FROM employees
WHERE salary > ANY (
   SELECT salary
   FROM employees
   WHERE department_id = 20
);
```

3. Nested Subquery

A Nested Subquery involves three or more layers of queries. The innermost query executes first, returning a result set that the middle query processes. The middle query then returns a result set or a single value for the outermost query (the main query) to use.

A nested subquery can be either a Single-Row or a Multi-Row subquery, depending on what the middle query expects.

Syntax:

The subquery layers are executed from the innermost to the outermost.

```
SELECT column1
FROM table_outer
WHERE column_A [OPERATOR_1] (
   SELECT column_B
   FROM table_middle
```

```
    WHERE column_C [OPERATOR_2] (
        SELECT column_D
        FROM table_inner
        WHERE inner_condition
    )
);
```

Example: Nested Multi-Row Subquery (Finding Employees in Departments with the Minimum Employee Count)

```
SELECT employee_name
FROM employees
WHERE department_id IN (
    -- Middle Subquery (Multi-Row): Finds all Department IDs matching the minimum count
    SELECT department_id
    FROM employees
    GROUP BY department_id
    HAVING COUNT(employee_id) = (
        -- Innermost Query (Single-Row): Finds the overall minimum employee count
        SELECT MIN(COUNT(employee_id))
        FROM employees
        GROUP BY department_id
    )
);
```

4. Correlated Subquery

A Correlated Subquery is a nested query where the inner query references a column from the table in the outer query. This reference is often called a correlation variable or alias. The subquery uses the value from the current row of the outer query to calculate a result, which it then passes back to the outer query for filtering.

Key Requirements and Operators

1. Correlation Variable (Alias): The outer table must be assigned an alias (e.g., e) so the inner subquery can reference its columns (e.g., e.department_id).
2. Execution: It conceptually executes once for each row of the outer query.
3. Operators: Correlated subqueries often use single-row operators (=, >, <) or the multi-row EXISTS/NOT EXISTS operators.

Syntax

```
SELECT column1, column2
FROM table_outer outer_alias -- Alias is mandatory for correlation
WHERE outer_alias.column_to_compare [OPERATOR] (
   SELECT [AGGREGATE_FUNCTION] (column_B)
   FROM table_inner inner_alias
   WHERE inner_alias.join_column = outer_alias.join_column -- This is the correlation link
);
```

Example (Using $>$ Operator)

To find the names of employees who earn more than the **average salary of their own department**:

```
SELECT employee_name, salary, department_id
FROM employees e -- Outer table alias 'e'
WHERE e.salary > (
   SELECT AVG(salary)
   FROM employees e2 -- Inner table alias 'e2'
   WHERE e2.department_id = e.department_id -- CORRELATION LINK
);
```

Example (Using EXISTS Operator)

To find departments that have **at least one employee** earning over $100,000:

```
SELECT department_name
FROM departments d -- Outer table alias 'd'
WHERE EXISTS (
    SELECT 1 -- Any value will suffice
    FROM employees e
    WHERE e.department_id = d.department_id -- CORRELATION LINK
      AND e.salary > 100000
);
```

Term Work – 4

Create the following table "**departments**"

Column Name (Constraint)	Datatype	Size
Department_ID (PK)	NUMBER	10
Department_Name	VARCHAR2	20
Location	VARCHAR2	10

Insert the following data in the **Departments** table

department_id	department_name	Location
10	Sales	New York
20	Marketing	London
30	Executive	New York
40	Operations	Chicago

Create the following table "**Employees**" with constraint FOREIGN KEY using department table.

Column Name (Constraint)	Datatype	Size
emp_id (PRIMARY KEY)	NUMBER	10
emp_name (NOT NULL)	VARCHAR2	20
salary CHECK(salary>=30000)	NUMBER	10,2
dept_id	NUMBER	10

Insert the following data in the **Employees** table

emp_id	emp_name	salary	dept_id
101	Alice Johnson	50000	10
102	Bob Smith	60000	20
103	Charlie Brown	70000	10
104	David Williams	55000	30
105	Eve Carter	65000	20
106	Allen	75000	40

Q. Based on previous tables answer the following Questionaries: -

1. Which employees earn the same salary as the employee with Employee_ID = 101? (Excluding the employee with ID 101 itself).
2. List the names of all employees who work in a department located in 'New York'.
3. List the employee whose salary is the second highest overall.
4. List all employees working in departments located in 'New York' or 'London'.
5. Find employees whose salary is less than the salary of at least one employee in Department 10.
6. Find employees whose salary is greater than the salary of all employees in Department 40.
7. List the names of all employees who work in departments located in a city that starts with the letter 'L'.
8. Find the department IDs that have an average salary greater than the minimum salary of the 'Marketing' department.
9. List all employees whose salary is greater than the maximum salary of any department.
10. Find all departments whose maximum salary is less than the overall average salary of the entire company.
11. Find departments where ALL employee salaries are greater than the lowest salary in the 'Operations' department.
12. Identify employees whose salary is less than the lowest salary of any department located in 'New York'.