IQRA UNIVERSITY IU

DATABASE MANAGEMENT SYSTEM LAB

Week # 10

INTRODUCTION

The Purpose of this Lab is to get familiar with nested queries and subqueries in SQL without being tied to a specific schema, making them adaptable to various database structures.

SQL Subqueries

An SQL Subquery, is a SELECT query within another query. It is also known as **Inner query** or **Nested query** and the query containing it is the **outer query.**

The outer query can contain the **SELECT**, **INSERT**, **UPDATE**, and **DELETE** statements. We can use the subquery as a column expression, as a condition in SQL clauses, and with operators like =, >, <, >=, <=, IN, BETWEEN, etc.

Types of Subqueries

1. **Single-Row Subqueries:** These return only one value (e.g., one department ID).
2. **Multi-Row Subqueries:** These return multiple values (e.g., IDs of multiple departments).
3. **Correlated Subqueries:** These depend on a column from the main query.
4. **Subqueries in Different Clauses:** You'll see how subqueries can be used in different parts of an SQL query, like:

   1. **WHERE** (to filter results),

   2. **SELECT** (to add a calculated column),

   3. **FROM** (to create temporary tables).

Employees Table

EmployeeID	EmployeeName	DepartmentID	Salary
101	John Smith	2	50000
102	Jane Doe	2	60000
103	Alice Brown	3	62000
104	Bob Martin	2	70000
105	Charlie Hill	3	65000

Departments Table

DepartmentID	DepartmentName
1	IT
2	Sales
3	HR

Subquery in the WHERE Clause

A **subquery in the WHERE clause** is used to filter the result set by checking if a condition is true based on the result of another query. The subquery returns a value (or a set of values) that is compared to the main query’s columns.

Find employees whose DepartmentID matches the department of "Sales“.

```
SELECT EmployeeID, EmployeeName, DepartmentID, Salary
FROM Employees
WHERE DepartmentID = (
    SELECT DepartmentID
    FROM Departments
    WHERE DepartmentName = 'Sales’
);
```

OUTPUT:

EmployeeID	EmployeeName	DepartmentID	Salary
101	John Smith	2	50000
102	Jane Doe	2	60000
104	Bob Martin	2	70000

Explanation:
The subquery finds DepartmentID for "Sales" (e.g., 2), and the outer query retrieves employees in that department.

Subquery in the SELECT Clause

A **subquery in the SELECT clause** allows you to include a calculated value or aggregate result from another query as a new column in the output.

Find employees and their department names along with the average salary in their department.

```
SELECT E.EmployeeID, E.EmployeeName, D.DepartmentName,
       (SELECT AVG(Salary)
        FROM Employees
        WHERE DepartmentID = E.DepartmentID) AS AvgSalary
FROM Employees E
JOIN Departments D ON E.DepartmentID = D.DepartmentID;
```

E is an alias for the Employees table.

It is used to refer to columns from the Employees table concisely.

D is an alias for the Departments table.

It is used to refer to columns from the Departments table concisely.

How Aliases Work in SQL:

Aliases are shortcuts that allow you to reference tables (or columns) with shorter names in your query. They make the query easier to write and read, especially when dealing with multiple tables or subqueries.

Query without Aliases:

```
SELECT Employees.EmployeeID, Employees.EmployeeName, Departments.Departm
entName,
       (SELECT AVG(Salary)
        FROM Employees
        WHERE Employees.DepartmentID = Employees.DepartmentID) AS AvgSalary
FROM Employees
JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

Output:

EmployeeID	EmployeeName	DepartmentName	AvgSalary
101	John Smith	Sales	60000.00
102	Jane Doe	Sales	60000.00
103	Alice Brown	HR	63500.00
104	Bob Martin	Sales	60000.00
105	Charlie Hill	HR	63500.00

Explanation:

- **The subquery calculates the average salary for each department by referencing the DepartmentID of each employee.**
- **The result is displayed as a new column AvgSalary for each employee in the result set.**

Subquery in the FROM Clause:

A **subquery in the FROM clause** is treated as a derived table or a temporary result set that can be used by the outer query. This type of subquery is helpful for performing intermediate calculations or grouping data.

Find departments with an average salary greater than 60,000.

```
SELECT DepartmentName, AvgSalary
FROM (
    SELECT D.DepartmentName, AVG(E.Salary) AS AvgSalary
    FROM Employees E
    JOIN Departments D ON E.DepartmentID = D.DepartmentID
    GROUP BY D.DepartmentName
) AS DepartmentAvgSalary
WHERE AvgSalary > 60000;
```

Output:

DepartmentName	AvgSalary
HR	63500.00

Explanation:

- **The subquery inside the FROM clause calculates the average salary for each department.**
- **The outer query filters departments where the average salary is above 60,000.**

Correlated Subquery:

A **correlated subquery** is a type of subquery that references a column from the outer query. It is evaluated once for each row processed by the outer query, and is often used when the result of the subquery depends on values from the outer query.

Find employees whose salary is greater than the average salary in their department.

```
SELECT EmployeeID, EmployeeName, Salary, DepartmentID
FROM Employees E
WHERE Salary > (
    SELECT AVG(Salary)
    FROM Employees
    WHERE DepartmentID = E.DepartmentID
);
```

Output:

EmployeeID	EmployeeName	Salary	DepartmentID
104	Bob Martin	70000	2
105	Charlie Hill	65000	3

Explanation:

- **The subquery calculates the average salary for each department, using the DepartmentID of the current employee from the outer query.**
- **The outer query returns employees whose salary is higher than the average salary in their respective department.**

Nested Subqueries Versus Correlated Subqueries

With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query.

A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

In this example, the subquery calculates the average salary for employees in the same department as the current employee in the outer query.

The E.DepartmentID in the subquery references the DepartmentID of the current row being evaluated in the outer query. This makes it a correlated subquery because it depends on the outer query for its execution.

Using IN with Subqueries:

The IN operator can be used to match a value against a list of values returned by a subquery. This allows you to check if a value exists within a set of results generated by another query.

Find employees who work in either the "HR" or "Sales" department:

```
SELECT EmployeeID, EmployeeName, DepartmentID, Salary
FROM Employees
WHERE DepartmentID IN (
    SELECT DepartmentID
    FROM Departments
    WHERE DepartmentName IN ('HR', 'Sales')
);
```

Output:

EmployeeID	EmployeeName	DepartmentID	Salary
101	John Smith	2	50000
102	Jane Doe	2	60000
103	Alice Brown	3	62000
104	Bob Martin	2	70000
105	Charlie Hill	3	65000

Explanation:

- **The subquery returns the DepartmentID for both the "HR" and "Sales" departments.**
- **The outer query retrieves employees working in these departments.**

Subqueries with EXISTS:

The EXISTS operator checks whether a subquery returns any rows. If the sub query returns one or more rows, the condition is true. This is useful for testing the existence of certain data without actually returning values.

Find departments that have at least one employee:

```
SELECT DepartmentName
FROM Departments D
WHERE EXISTS (
    SELECT 1
    FROM Employees E
    WHERE E.DepartmentID = D.DepartmentID
);
```

Output:

DepartmentName
Sales
HR

Explanation:

- **The subquery checks if there is at least one employee in each department by testing if any rows exist for the DepartmentID.**
- **The outer query lists the departments where such employees exist.**

The EXISTS operator checks whether the subquery returns any rows. If the subquery finds at least one matching row, the condition is true, and the outer query will return the DepartmentName

Conclusion:

- Subqueries provide a powerful way to perform complex filtering, aggregation, and data manipulation in SQL. By using subqueries in the WHERE, FROM, and SELECT clauses, you can create more efficient and readable queries, enabling you to tackle a wide range of problems in database management. Through the examples provided, you can see how subqueries help solve real-world scenarios and enhance your ability to retrieve and process data dynamically.

Student Table

StudentID	StudentName	CourseID	Marks
1	John	101	85
2	Jane	102	90
3	Alice	101	95
4	Bob	103	80
5	Charlie	102	88

Course Table

CourseID	CourseName
101	Math
102	English
103	Science

Class Task #1: Subquery to Filter Records by a Single Value

Objective:

Write a query to retrieve all records from a table where a certain column value matches a value returned by another query. The subquery should return a single value that is used to filter the main query's results.

```
SELECT StudentID, StudentName, CourseID, Marks
FROM Student
WHERE CourseID = (
    SELECT CourseID
    FROM Course
    WHERE CourseName = 'Math’
);
```

StudentID	StudentName	CourseID	Marks
1	John	101	85
3	Alice	101	95

Class Task #2: Correlated Subquery to Compare Records with Group Average.

Objective:

Write a query to find records from a table where a certain column has values g
reater than the average of that column for a group of records. Use a **correlated subquery** to compare the value of each record with the average of the column for its group.

```
SELECT StudentID, StudentName, Marks, CourseID
FROM Student S
WHERE Marks > (
    SELECT AVG(Marks)
    FROM Student
    WHERE CourseID = S.CourseID
);
```

StudentID	StudentName	Marks	CourseID
3	Alice	95	101
2	Jane	90	102

Class Task #3: Subquery in HAVING Clause to Filter Groups Based on Member Count

Objective:

Write a query to retrieve records where the associated group (based on a speci fic column) has more than a specified number of members. Use a subquery in the **HAVING** clause to count the number of records in each group and return only those groups that have more than the specified number.

```
SELECT CourseID, COUNT(StudentID) AS StudentCount
FROM Student
GROUP BY CourseID
HAVING COUNT(StudentID) > 1;
```

CourseID	StudentCount
101	2
102	2