Learning Objectives

- Understanding of Subqueries in MySQL.
- Common Subquery Formats.
- Types of Subqueries.

LO1: Understanding of Subqueries in MySQL.

Subquery

A subquery in MySQL is a query nested inside another query. It can be used with SELECT, INSERT, UPDATE, or DELETE statements. Subqueries can appear in various clauses, such as WHERE, FROM, or SELECT.

Subquery Rules in MySQL

- Subqueries must be enclosed in parentheses ().
- A subquery used with comparison operators must return a single column.
- Subqueries in the SELECT clause must return a scalar value.

Example

Retrieve employees who have never handled an order.

Table: employees

EmployeeID	FirstName	LastName
1	Davolio	Nancy
2	Fuller	Andrew
3	Leverling	Janet
4	Peacock	Margret
5	Buchanan	Steven

Table: orders

OrderID	CustomerID	EmployeeID
10248	90	5
10250	34	4
10251	84	3
10252	76	4

Requirements: Retrieve employees who have never handled an order

```
SELECT FirstName, LastName
FROM employees
WHERE EmployeeID NOT IN
                (SELECT EmployeeID FROM orders);
```

Results:

FirstName	LastName
Davolio	Nancy
Fuller	Andrew

Components of a Subquery

- **SELECT Statement**
- **FROM Clause**
- **Optional WHERE, GROUP BY, HAVING Clauses**

LO2: Common Subquery Formats

Understanding IN, ALL, ANY

The IN operator is used to check if a column’s value exists within a specified list of values. It is a more concise way of writing multiple OR conditions. For example, if we want to find all customers who belong from Sweden or Canada, we can write:

```
select CustomerName
from customers
where Country IN ('Sweden', 'Canada');
```

This is equivalent to:

```
select CustomerName
from customers
where Country = 'Sweden' OR Country = 'Canada';
```

Using IN makes queries shorter and easier to read, especially when working with long lists of values.

The ALL operator is used with comparison operators (>, <, >=, <=, !=) to check if a value satisfies a condition against all values in a given set. For example, if we want to find products that appear in all orders where the quantity is exactly 10, we can use:

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

This means that the product must be included in every single order where the quantity is 10. If it appears in only some of them but not all, it won't be included in the result.

The ANY operator is similar to ALL, but instead of checking against all values, it checks if the condition is true for at least one value in the given set. For example, if we want to find products that appear in at least one order where the quantity is exactly 10, we write:

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

This means a student will be included in the result if their marks are greater than either 60, 70, or 80. Even if they don’t beat all three values, if they beat at least one, they qualify.

WHERE expression [NOT] IN (subquery)

Example Get orders placed by customers that are not from 'Germany'

```
SELECT OrderID
FROM orders
WHERE CustomerID NOT IN (
    SELECT CustomerID
    FROM customers
    Country = 'Germany'
);
```

WHERE expression comparison operator [ANY | ALL] (subquery)

Example: Returns products with a price higher than the most expensive product in Category 2**.**

```
SELECT ProductName, Price
FROM products
WHERE Price > ALL (
SELECT Price
FROM products
WHERE categoryID=2);
```

WHERE IN (subquery)

Example: Get employees who have managed more than 10 orders

```
SELECT FirstName, LastName
FROM employees
WHERE EmployeeID IN (
    SELECT EmployeeID
    FROM orders
    GROUP BY EmployeeID
    HAVING COUNT(OrderID) > 10
);
```

LO3: Types of Subqueries.

Scalar Subquery

Subquery will return a single value (1 row, 1 column)

Example: Find products priced higher than average

```
SELECT ProductName
FROM products
WHERE Price > (
  SELECT AVG(Price)
  FROM products
);
```

Row Subquery

Subquery will return a single row (multiple columns)

Example: Find customer matching specific order details

```
SELECT CustomerName
FROM customers
WHERE (City, Country) = (
  SELECT City, Country
  FROM orders
  JOIN customers USING(CustomerID)
  WHERE OrderID = 10248
);
```

Column Subquery

Subquery will return a single column (multiple rows)

Example: Customers who placed orders

```
SELECT CustomerName
FROM customers
WHERE CustomerID IN (
  SELECT DISTINCT CustomerID
  FROM orders
);
```

Table Subquery

Subquery will return a full table (multiple rows/columns)

Example: Orders with total quantity > 100

```
SELECT *
FROM (
  SELECT OrderID, SUM(Quantity) AS TotalQty
  FROM orderdetails
  GROUP BY OrderID
) AS OrderTotals
WHERE TotalQty > 100;
```

Correlated Subquery

References outer query values

Example: Employees who handled orders for each customer

```
SELECT CustomerName,
  (SELECT CONCAT(FirstName, ' ', LastName)
   FROM employees
   WHERE employees.EmployeeID = orders.EmployeeID) AS Handler
    FROM customers
    JOIN orders USING(CustomerID);
```

Exists Subquery

Checks for existence of rows

Example: Suppliers with products in stock

```
SELECT SupplierName
FROM suppliers s
WHERE EXISTS (
  SELECT 1
  FROM products
  WHERE SupplierID = s.SupplierID
);
```

Nested Subquery

Subquery within another subquery

Example: Find employees who handled orders containing the most expensive product

```
SELECT FirstName, LastName
FROM employees
WHERE EmployeeID IN (
    SELECT EmployeeID
     FROM orders
     WHERE OrderID IN (
        SELECT OrderID
        FROM orderdetails
        WHERE ProductID = (
            SELECT ProductID
            FROM products
            WHERE Price = (SELECT MAX(Price) FROM products)
        )
    )
);
```

Inline View (FROM Clause Subquery)

Subquery acting as temporary table

Example: Average order quantities

```
SELECT od.OrderID, od.Quantity
FROM orderdetails od
JOIN (
  SELECT AVG(Quantity) AS AvgQty
  FROM orderdetails
) AS avg_table
WHERE od.Quantity > avg_table.AvgQty;
```

Derived Table Subquery

Complex subquery in FROM clause

Example: Product list with category names

```
SELECT p.ProductName, c.CategoryName
FROM products p
JOIN (
  SELECT CategoryID, CategoryName
  FROM categories
) AS c ON p.CategoryID = c.CategoryID;
```

Quantified Comparison (ALL/ANY/SOME)

Example: Products more expensive than ALL Beverages

```
SELECT p.ProductName, c.CategoryName
FROM products p
JOIN (
  SELECT CategoryID, CategoryName
  FROM categories
) AS c ON p.CategoryID = c.CategoryID;SELECT ProductName
FROM products
WHERE Price > ALL (
  SELECT Price
  FROM products
```

```
   WHERE CategoryID = (
     SELECT CategoryID
     FROM categories
     WHERE CategoryName = 'Beverages'
   )
);
```

Tasks:

1. Find products that have never been ordered. (ProductName)
2. List customers with no orders. (CustomerName)
3. Find suppliers providing 'Seafood' products. (SupplierName)
4. Find customers from same city as their suppliers. (CustomerName)
5. Find most expensive product in each category. (ProductName, CategoryID)
6. Find orders with total quantity > 100. (OrderID)
7. Find suppliers from same country as customers. (SupplierName)
8. Find employees handling >5 customers. (FirstName, LastName)
9. Find the most expensive product price. (ProductName, Price)
10. Find employees who have handled orders placed in 1997. (FirstName, LastName)
11. Retrieve orders that contain products costing more than 50. (OrderID)
12. Find suppliers who supply only one product. (SupplierName)
13. Get employees who handle orders containing 'Chai'. (FirstName, LastName)

HackerRank Tasks

- The-Pads
- The-Company
- Weather-Observation-Station-18
- Weather-Observation-Station-19

- The-Report
- Full-Score
- Challenges

Submission on Eduko

Submit the following file in Zip on Eduko:

- 2024-CS-X.txt (Containing solution of all lab tasks)
- 2024-CS-X.docx (Execute all the example queries and add result in tables in a word file as added in the first example – add max 10 records of each query if number of rows exceed 10).