What is a Subquery?

A subquery, also known as an inner query or nested query, is a query that is embedded within another SQL query. The result of the subquery is used by the outer query to perform its operations. Subqueries can be placed in various parts of an SQL statement, such as in the `SELECT`, `FROM`, `WHERE`, or `HAVING` clauses.

Types of Subqueries

1. Single-Row Subquery:

- Returns a single value (one row and one column).
- Used with comparison operators such as `=`, `<`, `>`, etc.

2. Multiple-Row Subquery:

- Returns multiple rows but one column.
- Used with operators like `IN`, `ANY`, `ALL`, etc.

3. Correlated Subquery:

- Refers to columns from the outer query, meaning the subquery depends on the outer query for its values.
- It is executed once for each row processed by the outer query.

4. Nested Subquery:

- A subquery within another subquery.

Practical Uses of Subqueries in Different Industries:

1. Retail Industry: Finding Top-Selling Products

Scenario: In the retail industry, you may want to find the products that have generated the highest revenue in a particular month.

```
SELECT ProductName, TotalSales
FROM Products
WHERE ProductID IN (
    SELECT ProductID
    FROM Sales
    WHERE Month = '2024-08'
    GROUP BY ProductID
    HAVING SUM(SalesAmount) = (
        SELECT MAX(SumSales)
        FROM (
            SELECT SUM(SalesAmount) AS SumSales
```

```
            FROM Sales
            WHERE Month = '2024-08'
            GROUP BY ProductID
        ) AS MonthlySales
    )
);
```

- Explanation: The subquery identifies the `ProductID` of the product with the highest sales in August. The outer query returns the product details.

2. Banking Industry: Identifying Customers with High Average Balances

Scenario: A bank might want to identify customers whose average account balance is higher than the overall average balance for all customers.

```
SELECT CustomerID, AVG(Balance) AS AvgBalance
FROM Accounts
GROUP BY CustomerID
HAVING AVG(Balance) > (
    SELECT AVG(Balance)
    FROM Accounts
);
```

- Explanation: The inner subquery calculates the average balance across all accounts, and the outer query retrieves customers whose average balance exceeds this value.

3. Healthcare Industry: Finding Hospitals with Above-Average Bed Occupancy

Scenario: In healthcare, administrators may want to identify hospitals that have higher bed occupancy rates than the national average.

```
SELECT HospitalName, OccupancyRate
FROM Hospitals
WHERE OccupancyRate > (
    SELECT AVG(OccupancyRate)
    FROM Hospitals
);
```

- Explanation: The subquery calculates the average occupancy rate across all hospitals, and the outer query retrieves hospitals with a higher-than-average occupancy rate.

4. E-commerce Industry: Finding Customers Who Have Placed More Than 5 Orders

Scenario: In e-commerce, you might want to find customers who have placed more than 5 orders.

```
SELECT CustomerID, CustomerName
FROM Customers
WHERE CustomerID IN (
    SELECT CustomerID
    FROM Orders
    GROUP BY CustomerID
    HAVING COUNT(OrderID) > 5
);
```

- Explanation: The inner subquery identifies customers who have placed more than 5 orders, and the outer query retrieves their details.

5. Telecommunications Industry: Retrieving Plans with Above-Average Revenue

Scenario: In a telecommunications company, you want to identify mobile plans that generate more revenue than the average across all plans.

```
SELECT PlanName, Revenue
FROM MobilePlans
WHERE Revenue > (
    SELECT AVG(Revenue)
    FROM MobilePlans
);
```

- Explanation: The subquery calculates the average revenue across all mobile plans, and the outer query returns the plans that generate more than this average.

6. Education Industry: Students Scoring Above Average

Scenario: In the education sector, you might want to find students who scored higher than the average score on an exam.

```
SELECT StudentID, StudentName, Score
FROM Students
WHERE Score > (
    SELECT AVG(Score)
    FROM Students
);
```

- Explanation: The subquery calculates the average score for the exam, and the outer query retrieves students with scores above the average.

7. Logistics Industry: Finding Delivery Times Above the Average

Scenario: In logistics, you might want to identify deliveries that took longer than the average delivery time across all shipments.

```
SELECT DeliveryID, DeliveryTime

FROM Deliveries

WHERE DeliveryTime > (

    SELECT AVG(DeliveryTime)

    FROM Deliveries

);
```

- Explanation: The inner subquery calculates the average delivery time, and the outer query retrieves the deliveries that took longer than the average.

Common Use Cases for Subqueries

1. Filtering with Calculated Results:

Subqueries can be used to filter results based on aggregate functions like `AVG`, `SUM`, `MAX`, or `MIN`, without needing to store these calculations in separate tables.

2. Comparing with Grouped Data:

Subqueries are frequently used to compare individual records with grouped data (e.g., finding employees with salaries higher than the department average).

3. Self-Referencing Data:

Subqueries allow you to perform operations on data that references itself, such as finding employees who earn more than their department's average salary.

4. Dynamic Filtering:

A subquery can generate a dynamic set of values (e.g., `IN` or `NOT IN` queries) based on the conditions of other data, allowing for flexible and dynamic filtering of results.

Why Data Analysts Use Subqueries

Data analysts use subqueries because they provide:

1. Flexibility: Subqueries allow analysts to break down complex queries into smaller, manageable parts.
2. Reusability: A subquery can be reused within the same SQL query, avoiding redundancy.
3. Efficiency: Subqueries help in filtering and aggregating data without creating new tables or temporary structures.

Subqueries are vital tools in SQL that enable data analysts to perform advanced data manipulation and filtering, making their queries more powerful and efficient across various industries.