9-SQL Subquery

Author: Vincent Lau

Learning Objectives

- Understand the purpose of implementing subqueries
- Common Table Expression (The WITH Clause)
- Exists and Not Exists
- Understand the drawbacks of using subqueries

Introduciton

A subquery, also known as a nested query or inner query, is a query that is embedded within another query. Subqueries allow you to retrieve data from one table based on the result of another query. They are used to break down complex tasks into simpler steps or to perform calculations on subsets of data. Subqueries can be used in various parts of SQL statements, including SELECT, FROM, WHERE, and more.

Types of Subquery

Here are the different scenarios where subqueries can be used, along with code examples:

1. Scalar Subquery

A scalar subquery returns a single value, typically used within expressions or comparisons.

```
SELECT column1, (SELECT MAX(column2) FROM table2) AS max_value
FROM table1;
```

2. Single-Row Subquery

A single-row subquery returns a single row and is used for comparisons with a single value.

```
SELECT column1
FROM table1
WHERE column2 = (SELECT column2 FROM table2 WHERE condition);
```

3. Multi-Row Subquery

A multi-row subquery returns multiple rows, often used with `IN`, `ANY`, or `ALL` operators.

```
SELECT column1
FROM table1
WHERE column2 IN (SELECT column2 FROM table2 WHERE condition);
```

4. Correlated Subquery

A correlated subquery references columns from the outer query and is evaluated for each row in the outer query's result set.

```
SELECT product_name
FROM products p
WHERE p.price > (SELECT AVG(price) FROM products WHERE category_id =
p.category_id);
```

5. Subquery with Aggregates

Subqueries can involve aggregate functions for calculations on subsets of data.

```sql
SELECT department, MAX(salary) AS max_salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department = 'Sales')
GROUP BY department;
```

6. EXISTS Subquery

The `EXISTS` keyword is used to check if a subquery returns any rows.

```sql
SELECT p.product_name
FROM products p
WHERE EXISTS (SELECT * FROM orders WHERE orders.product_id = p.id);
```

8. NOT EXISTS Subquery

The `NOT EXISTS` keyword is used to check if a subquery returns no rows.

```sql
SELECT customer_name
FROM customers c
WHERE NOT EXISTS (SELECT * FROM orders WHERE orders.customer_id = c.id);
```

These examples illustrate various scenarios where subqueries can be employed to retrieve and manipulate data based on the result of another query. Subqueries are powerful tools that allow you to perform complex operations in a step-by-step manner.

Common Table Expression (The With Clause)

The `WITH` clause, also known as a Common Table Expression (CTE), is a feature in SQL that allows you to define a temporary result set within a query. CTEs provide a way to improve the readability and maintainability of complex queries by breaking them into smaller, more manageable parts.

A CTE is defined using the `WITH` keyword followed by a name for the CTE and a query that defines the result set. This CTE can then be referenced in the main query, making it easier to build complex queries step by step.

Example

Here's the basic syntax of using the `WITH` clause:

```sql
WITH cte_name AS (
    SELECT ...
    FROM ...
    WHERE ...
)
SELECT ...
FROM cte_name
;
```

And here's an example to help illustrate how the `WITH` clause works:

```sql
WITH top_customers AS (
    SELECT customer_id, SUM(order_total) AS total_spent
    FROM orders
    GROUP BY customer_id
    ORDER BY total_spent DESC
    LIMIT 5
)
SELECT c.customer_name, tc.total_spent
FROM customers c INNER JOIN top_customers tc ON c.customer_id = tc.customer_id
;
```

In this example, the `top_customers` CTE calculates the total amount each customer has spent on orders and selects the top 5 customers based on their total spending. Then, the main query retrieves the customer names and their corresponding total spending by joining the `top_customers` CTE with the `customers` table.

Advantages of CTE

1. **Readability**: CTEs improve query readability by breaking complex logic into smaller parts.
2. **Code Reusability**: CTEs can be referenced multiple times within a query, reducing redundancy.
3. **Simpler Debugging**: Isolating complex logic in CTEs makes debugging easier.
4. **Optimization**: The query optimizer can sometimes optimize CTEs for better performance.
5. **Recursive Queries**: CTEs can be used to create recursive queries for hierarchical data.

Keep in mind that not all database systems support CTEs or may have slightly different syntax rules. Always refer to your specific database documentation for accurate information on using the `WITH` clause.

Limitation of Subquery

- Different database management systems have certain **limitations** on the number of subquery levels (e.g. up to **32** levels in SQL Server). However, in practice, you will rarely have more than 2-3 levels of nested queries.
- Subqueries are often **computationally inefficient**. Thus, it is recommended to avoid nested queries when other options are available (e.g. JOINs). The optimizer is more mature in most database systems for JOINs than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as JOIN.