Reading: Examples for Subqueries and Nested Selects

Estimated Time: 15 minutes

Objectives

At the end of this reading, you will be able to:

- Explain the concept of subqueries and nested selects in SQL
- Identify scenarios where subqueries and nested selects can be useful
- Use subqueries and nested selects to perform advanced data retrieval and manipulation in SQL

Sample data

The E-Commerce database consists of information related to customer orders and customer details. The database has two tables: **orders** and **customers**.

Orders

This table contains information about orders made by customers. The information includes the order number `(order_id)`, customer identification number `(customer_id)`, and the order value `(order_price)`.

order_id	customer_id	order_price
1	101	50.00
2	101	75.00
3	101	100.00
4	101	30.00
5	101	120.00
6	102	40.00
7	102	90.00
8	103	60.00
9	106	65.00
10	104	95.00
11	105	45.00

Customers

This table contains information about customers. The information includes the customer identification number `(customer_id)` and the customer's country `(country)`.

customer_id	country
101	USA
102	USA
103	Canada
104	Canada
105	USA
106	Mexico

Subqueries and nested selects

Subqueries and nested selects are advanced SQL techniques used to perform complex data manipulations and retrievals. They involve embedding one query within another, allowing you to perform operations that would be challenging or impossible with a single query alone.

Subqueries

A subquery is a query nested within another query, typically enclosed within parentheses. It can be used within a `WHERE` clause, `SELECT` statement, or `FROM` clause to filter or manipulate data.

Syntax statement for subqueries

The general syntax for subqueries:

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name OPERATOR (SELECT column_name FROM table_name WHERE condition);
```

Explanation:

- The subquery (`SELECT column_name FROM table_name WHERE condition`) is enclosed in parentheses and returns a single value or set of values.
- The main query then uses these values to filter or manipulate data based on the condition specified.

Example of subqueries for filtering data using WHERE clause

Consider the following example where we want to find all orders made by customers from a specific country ('USA').

```
SELECT *
FROM orders
WHERE customer_id IN (
    SELECT customer_id
    FROM customers
    WHERE country = 'USA'
);
```

Explanation:

- The subquery (`SELECT customer_id FROM customers WHERE country = 'USA'`) retrieves the IDs of customers from the USA. It filters the customers table based on the condition `country = 'USA'`.
- The `customer_id` retrieved from the subquery is used to filter the orders table, effectively selecting all orders made by customers from the USA.
- The outer query filters the orders table based on the result of the subquery, selecting only those rows where the `customer_id` matches any of the IDs returned by the subquery.

Response:
The resulting output displays all orders where the `customer_id` matches any of the IDs retrieved from the subquery. These orders are placed by customers from the USA, as specified in the subquery's condition.

order_id	customer_id	order_price
1	101	50.00
2	101	75.00
3	101	100.00
4	101	30.00
5	101	120.00
6	102	40.00
7	102	90.00
11	105	45.00

Nested selects

Nested selects involve using a SELECT statement within another SELECT statement. This enables the retrieval of data from multiple tables or the use of aggregated functions on subquery results.

Syntax statement for nested selects

The general syntax for nested selects:

```
SELECT column1, column2, ...
FROM (
    SELECT column1, column2, ...
    FROM table1
    JOIN table2 ON table1.column_name = table2.column_name
    WHERE condition
) AS alias_name;
```

Explanation:

- The inner query (nested select) retrieves data from one or more tables, applies filtering or joins, and can perform aggregation or other operations.
- The outer query then operates on the result set produced by the nested select, allowing further filtering, grouping, or manipulation.
- The result set produced by the nested select is treated as a temporary table and referenced by the alias specified in the outer query.

Example of nested queries for aggregation with JOIN and GROUP BY clause

Consider the following example where we want to find the total number of orders placed by customers from each country.

```
SELECT country, COUNT(*) AS num_orders
FROM (
    SELECT country
    FROM customers
    JOIN orders ON customers.customer_id = orders.customer_id
) AS customer_orders
GROUP BY country;
```

Explanation:

- The nested query joins the customers and orders tables using the JOIN keyword and the common column `customer_id`. It retrieves the country of each customer who placed an order.
- The outer query groups the results by country using the GROUP BY clause and calculates the count of orders for each country using the COUNT(*) function. This aggregation function counts the number of rows for each group.

Response
The result is a summary showing the number of orders (`num_orders`) for each country. In this case, there are five orders from the USA, two orders from Canada, and one order from Mexico.

country	num_orders
USA	5
Canada	2
Mexico	1

Example of nested queries for aggregation with GROUP BY and filtering with HAVING clause

Consider the following example where we want to find the average order value for customers who have placed more than four orders.

```
SELECT AVG(order_value) AS avg_order_value
FROM (
    SELECT customer_id, SUM(order_price) AS order_value
    FROM orders
    GROUP BY customer_id
    HAVING COUNT(*) > 4
) AS frequent_customers;
```

Explanation:

- The inner query groups the orders by `customer_id` using the `GROUP BY` clause. This groups together all orders placed by the same customer.
- Within each group of orders for a specific customer, the `SUM(order_price)` function calculates the total order value. This aggregates the prices of all orders placed by each customer.
- The `HAVING` clause filters the grouped data, ensuring that only customers who have placed more than four orders are included in the result set. This condition is applied after the grouping and aggregation.
- The outer query calculates the average order value `(AVG(order_value))` for the selected customers. This provides the average amount spent per order by customers who have placed more than five orders.

Response:
The resulting average order value is approximately 73.3333. This means that the average order value for customers who have placed more than four orders is approximately $73.33.

avg_order_value
73.33333

These examples highlight the flexibility and effectiveness of subqueries and nested queries in SQL, enabling a diverse array of data manipulations and analyses to be conducted with efficiency.

Summary:

In this reading, you explored the advanced SQL techniques of subqueries and nested selects, gaining an understanding of their concepts, applications, and practical usage. You learned that:

- A subquery is a query nested within another query, typically enclosed within parentheses.
- It can be used within a WHERE clause, SELECT statement, or FROM clause to filter or manipulate data.
- Nested selects involve using a SELECT statement within another SELECT statement.
- This enables the retrieval of data from multiple tables or the use of aggregated functions on subquery results.

In the lab that follows later in the module, you will apply these concepts and practice subqueries and nested queries hands-on.

Good luck!

Author(s)

Pratiksha Verma

Other Contributors

Malika Singla, Lakshmi Holla

Changelog

Date	Version	Changed by	Change Description
2024-8-05	0.1	Pratiksha Verma	Initial version created