Subqueries in SQL

Hia Al Saleh

November 11th, 2024

Contents

1 Introduction

The key component of databases is being able to optimize the data and reduce errors and redundancy. We create tables that hold information that is related, and the relationship between one table and another is represented by a **foreign key**, which is another table's primary key. This allows a column's value to provide a lookup in another table to retrieve additional information.

2 Subqueries Overview

With subqueries, we use the results of one query to filter another table's results. For example, if we want to know which supplier provided us with bread, we would query the `products` table to find suppliers providing bread.

```
SELECT supplierID
FROM products
WHERE productName LIKE '%bread%';
```

Important note: We asked for a single column of values, just the `supplierID`. Avoid creating a query that returns multiple columns.

```
SELECT supplierID, productName
FROM products
WHERE productName LIKE '%bread%';
```

3 Ensuring Unique Results

To ensure there are no duplicates in the results, use the `DISTINCT` keyword:

```
SELECT DISTINCT supplierID
FROM products
WHERE productName LIKE '%bread%';
```

We can now use these IDs to look up suppliers with specific companies.

```
SELECT *
FROM suppliers
WHERE id IN (8,2,6);
```

4 Simplifying Statements with Subqueries

Subqueries allow for simplification by combining multiple statements into one. For instance:

```
SELECT *
FROM suppliers
WHERE id IN (SELECT DISTINCT supplierID
FROM products
WHERE productName LIKE '%bread%');
```

The `IN` keyword checks if a value matches any value in a list, while `NOT IN` can be used to reverse the results.

4.1 Using Multiple Subqueries

Subqueries can be nested, following the same rules as a single subquery. Key differences include:

- Always enclose a subquery in parentheses.
- Do not end a subquery with a semicolon.
- Avoid using `ORDER BY` within a subquery.

Both queries must include the necessary tables to ensure columns appear in the final result, and subqueries may return an empty table.

5 Aggregate Functions and Subqueries

Subqueries are particularly useful with aggregate functions. For example, to find the cheapest product:

```
SELECT *
FROM products
WHERE price IN (SELECT MIN(price) FROM products);
```

5.1 Inner and Outer Subqueries

An *inner subquery* is the nested query, while an *outer subquery* is the statement containing the subquery. A simple subquery evaluates the inner subquery once and provides its result to the outer query.

```
SELECT *
FROM products
WHERE price IN (SELECT MAX(price) FROM products);
```

6 Excluding Certain Results

To find products that are not from Ontario:

```
SELECT *
FROM products
WHERE supplierID NOT IN (
SELECT id
FROM suppliers
WHERE province = 'Ontario');
```

7 Explicitly Declaring Column Names in Subqueries

Column names in a subquery can be explicitly declared, though implicit qualifications work when column names match across tables at the same nesting level.

```
SELECT pub_name
FROM publishers
WHERE publishers.pub_id IN
(SELECT titles.pub_id
FROM titles WHERE type='biography');
```

8 Scalar Subqueries as Column Expressions

Subqueries that are used as column expressions must return a single value. This can be achieved through aggregate functions or restrictive `WHERE` conditions.

For example, to list each product with its price and the average price:

```
SELECT productName, price, (SELECT AVG(price) FROM products)
FROM products;
```

9 Subqueries in Filtering Clauses

Subqueries can be used as filters in `WHERE` or `HAVING` clauses with comparison operators (e.g., =, ¡¿, ¡, etc.).

Rules:

- The `SELECT` clause should only include a single expression or column name.
- Compared values must have matching data types or be implicitly convertible.
- The subquery must return a single value, often through an aggregate.

9.1 Example: Matching Province

To display customers in the same province as "Thompson Inc":

```
SELECT * FROM customers
WHERE province =
(SELECT province
FROM suppliers WHERE companyName = 'Thompson Inc');
```

This query returns a single value, making it suitable for a subquery in a `WHERE` clause.

10 Conclusion

Subqueries are a powerful tool in SQL, allowing for complex queries that streamline data retrieval, especially when combined with aggregate functions and nested queries. Proper usage of subqueries enhances SQL's efficiency and readability, enabling advanced data manipulation and filtering.