SQL Sub-queries

A subquery, also known as a nested query, is a powerful feature in MySQL that allows you to embed a complete SELECT statement within another SQL statement (typically another SELECT, INSERT, UPDATE, or DELETE). The subquery acts as a mini-query that executes first, and its results are used by the outer query to filter, compare, or manipulate data.

Types of Subqueries:

MySQL supports various subquery types to achieve diverse functionalities:

A. Scalar Subqueries:

- Return a single value (one row and one column).
- Used for comparisons, calculations, or as conditions.

Example:

```
SELECT * FROM customers
WHERE credit_limit > (SELECT AVG(credit_limit) FROM
customers);
```

This query finds customers whose credit limit exceeds the average credit limit for all customers.

B. Row Subqueries:

- Return a single row, but can have multiple columns.
- Used for comparisons with multiple values.

Example:

```
SELECT product_name, price
FROM products
WHERE (category_id, brand_id) = (SELECT category_id,
brand_id FROM preferred_products WHERE customer_id = 123);
```

This query retrieves products with the same category and brand as the preferred products for customer ID 123.

C. Comparison Subqueries:

- Use comparison operators (=, <, >, <=, >=, etc.) to compare the results of the subquery with values in the outer query.

Example (combines scalar and comparison):

```
SELECT order_id, customer_name
FROM orders
WHERE total_amount = (SELECT MAX(total_amount) FROM orders);
```

This query finds orders with the highest total amount among all orders.

D. List Subqueries:

- Use the IN operator to check if a value in the outer query exists within a list returned by the subquery.

Example:

```
SELECT * FROM employees
WHERE department_id IN (SELECT department_id FROM
departments WHERE location = 'New York');
```

This query selects employees who work in departments located in New York.

E. Correlated Subqueries:

- Reference a column value from the outer query within the subquery.
- Evaluated once for each row in the outer query.

Example:

```
SELECT employee_name, salary, department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE salary > (SELECT AVG(salary) FROM employees WHERE
department_id = e.department_id);
```

This query finds employees who earn more than the average salary in their respective departments.

Considerations and Best Practices:

- Efficiency: Subqueries can add complexity and potentially impact performance. Consider using joins or views for simpler alternatives when feasible.
- Readability: Break down complex subqueries into temporary variables or CTEs (Common Table Expressions) to enhance readability.
- Clarity: Use parentheses around subqueries to ensure proper evaluation order.
- Nesting Levels: While MySQL allows deep nesting, excessive levels can make queries difficult to maintain. Explore alternative approaches if necessary.