SQL Subqueries

LESSON

Generate a quiz about this unit

Summarize this unit

Have any doubt about this content? Ask!

Learning Objectives

By the end of this lesson, you will be able to:

- Apply subqueries within various sections of SQL statements effectively.

Subqueries

A subquery, also known as an inner query or nested query, is a SQL query embedded within a larger query. It allows you to utilize the results of a query as a condition in another query. Subqueries allow you to perform complex operations by breaking them down into smaller, more manageable parts.

Subqueries can be used with various SQL clauses including `SELECT`, `FROM`, and `WHERE`.

Basic Syntax

```
SELECT column1, column2, ...
FROM tablename
WHERE column_name OPERATOR (SELECT column_name FROM tablename WHERE condi
```

Copy

Explain this code

Example

employee_id	employee_name	salary
1	John Doe	60000
2	Jane Smith	55000
3	Alice Jones	48000
4	Bob Brown	52000

To find employees who earn more than the average salary:

```sql
SELECT employee_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Copy

Explain this code

Resulting Set:

employee_name	salary
John Doe	60000
Jane Smith	55000

Key Points

- **Single vs Multiple Values**: Subqueries can return a single value (scalar), a single list of values, or a table.
- **Dependent vs Independent**: A subquery can be independent (it can run by itself) or correlated (it relies on the main query for its values).

Types of Subqueries

1. Scalar Subqueries

These return a single value. They can be used where single values are acceptable, like in the SELECT, WHERE, or HAVING clauses.

```
FROM tablename
WHERE column3 = (SELECT MAX(column3) FROM tablename);
```

Explain this code

Example

Using the same employee table. Find the employee with the maximum salary:

```
SELECT employee_name, salary
FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);
```

Copy

Explain this code

Resulting Set:

employee_name	salary
John Doe	60000

2. Inline Views

An inline view is a subquery in the FROM clause. It's used when the result set needs to be treated as a table.

```
SELECT a.column1, a.column2
FROM (SELECT column1, column2 FROM tablename WHERE condition) AS a
WHERE a.column1 = condition;
```

Copy

Explain this code

Example

Using the same employee table. Consider that we only want the names of the employees who have a salary greater than 50000, and then, from that result, we want to select those whose names start with 'J':

```
SELECT a.employee_name
FROM (SELECT employee_name FROM employees WHERE salary > 50000) AS a
WHERE a.employee_name LIKE 'J%';
```

Copy

Resulting Set:

employee_name
John Doe
Jane Smith

Correlated vs. Non-Correlated Subqueries

Correlated Subqueries

A correlated subquery is evaluated once for each row processed by the outer query. It relies on inputs from the main query, making them interdependent.

```
SELECT column1, column2
FROM tablename a
WHERE column3 = (SELECT column3 FROM tablename b WHERE b.column1 = a.colu
```

Copy

Explain this code

Example

Assume we have two tables, `employees` and `employee_bonus`.

Table `employee_bonus`:

employee_id	bonus_amount
1	5000
3	3000

We want to find all employees who received a bonus.

```
SELECT e.employee_name, e.salary
FROM employees e
WHERE EXISTS (SELECT 1 FROM employee_bonus b WHERE b.employee_id = e.empl
```

Copy

Explain this code

employee_name	salary
John Doe	60000
Alice Jones	48000

Non-Correlated Subqueries

A non-correlated subquery can be run independently of the main query. It returns data that the main query uses.

```
SELECT column1, column2
FROM tablename
WHERE column3 IN (SELECT column3 FROM another_tablename);
```

Copy

Explain this code

Example

From our `employees` table, we want to find those whose IDs are in the `employee_bonus` table:

```
SELECT employee_name, salary
FROM employees
WHERE employee_id IN (SELECT employee_id FROM employee_bonus);
```

Copy

Explain this code

Resulting Set:

employee_name	salary
John Doe	60000
Alice Jones	48000

Common Pitfalls and Best Practices

1. **Performance**: Correlated subqueries can be slow, especially on large datasets, because the subquery runs once for each row in the outer query.

3. **Alternatives**: Sometimes JOINs or window functions can be used instead of subqueries for better performance and clarity.

4. **Testing**: Always test subqueries independently before integrating them into the main query.

Summary

Subqueries and set operations are advanced SQL techniques that allow for more complex data retrieval and manipulation. Understanding these concepts enhances your ability to write efficient and flexible SQL queries, especially when dealing with multiple datasets or conditions.

Lesson Completed

Mark as not completed

How would you rate the content of this unit?