Certainly! Subqueries in SQL are queries that are nested inside another query. They can be categorized into three main types: single-row subqueries, multiple-row subqueries, and correlated subqueries. Let's explore each type with two code examples for each.

1. Single-row subqueries:

A single-row subquery returns only one row to the outer SQL statement. Here's an example:

Example 1: Using a single-row subquery in the WHERE clause:

```sql
-- Retrieve employees whose salary is greater than the average salary
SELECT employee_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

In this example, the single-row subquery `(SELECT AVG(salary) FROM employees)` calculates the average salary, and the main query retrieves employees whose salary is greater than the calculated average.

Example 2: Using a single-row subquery in the SELECT clause:

```sql
-- Retrieve employees along with the department name where they work
SELECT employee_name, department_id, (SELECT department_name FROM departments
WHERE department_id = employees.department_id) AS department_name
FROM employees;
```

Here, the single-row subquery is used in the SELECT clause to fetch the department name for each employee based on their `department_id`.

2. Multiple-row subqueries:

A multiple-row subquery returns more than one row to the outer SQL statement. Here are two examples:

Example 3: Using a multiple-row subquery with the IN operator:

```sql
-- Retrieve employees in departments located in 'New York' or 'San Francisco'
SELECT employee_name, department_id
FROM employees
```

```
WHERE department_id IN (SELECT department_id FROM departments WHERE location IN
('New York', 'San Francisco'));
```

In this example, the multiple-row subquery returns department IDs for departments located in 'New York' or 'San Francisco', and the main query retrieves employees in those departments.

Example 4: Using a multiple-row subquery with the EXISTS operator:

```sql
-- Retrieve employees who have at least one project assigned
SELECT employee_name
FROM employees
WHERE EXISTS (SELECT 1 FROM projects WHERE projects.employee_id =
employees.employee_id);
```

This example uses the EXISTS operator to check whether there is at least one project assigned to each employee.

3. Correlated subqueries:

A correlated subquery refers to a subquery that depends on the outer query. It executes once for each row processed by the outer query. Here are two examples:

Example 5: Correlated subquery in the WHERE clause:

```sql
-- Retrieve employees whose salary is greater than the average salary in their department
SELECT employee_name, salary
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department_id =
e.department_id);
```

In this example, the correlated subquery compares the salary of each employee with the average salary in their respective department.

Example 6: Correlated subquery in the SELECT clause:

```sql
-- Retrieve employees along with the total salary expenditure in their department
SELECT employee_name, department_id, salary,
```

```
    (SELECT SUM(salary) FROM employees WHERE department_id = e.department_id) AS
total_department_salary
FROM employees e;
```

Here, the correlated subquery calculates the total salary expenditure for each employee's department, and the result is included in the main query's result set.

Certainly! Let's continue with more examples:

4. Nested Subqueries:

Nested subqueries involve multiple levels of nesting. Each level can be a single-row or multiple-row subquery. Here are two examples:

Example 7: Nested Single-row Subqueries:

```sql
-- Retrieve employees whose salary is greater than the average salary of their department,
-- and the department is located in 'New York'
SELECT employee_name, salary
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department_id =
e.department_id)
  AND e.department_id IN (SELECT department_id FROM departments WHERE location =
'New York');
```

In this example, there's a nested single-row subquery for calculating the average salary in each department, and another level for checking the department's location.

Example 8: Nested Multiple-row Subqueries:

```sql
-- Retrieve departments where the maximum salary is higher than the average maximum salary
in all departments
SELECT department_id, department_name
FROM departments d
WHERE (SELECT MAX(salary) FROM employees WHERE department_id = d.department_id)
>
    (SELECT AVG(MAX(salary)) FROM employees GROUP BY department_id);
```

Here, the nested multiple-row subquery calculates the maximum salary for each department, and the outer query compares it with the average maximum salary across all departments.

5. Subqueries in UPDATE and DELETE Statements:

Subqueries can also be used within UPDATE and DELETE statements. Here are two examples:

Example 9: Using a Subquery in an UPDATE Statement:

```sql
-- Increase the salary of employees in the 'IT' department by 10%
UPDATE employees
SET salary = salary * 1.1
WHERE department_id = (SELECT department_id FROM departments WHERE
department_name = 'IT');
```

In this example, the subquery is used to identify the 'IT' department's ID, and the UPDATE statement increases the salary for employees in that department.

Example 10: Using a Subquery in a DELETE Statement:

```sql
-- Delete employees who have no assigned projects
DELETE FROM employees
WHERE NOT EXISTS (SELECT 1 FROM projects WHERE projects.employee_id =
employees.employee_id);
```

Here, the subquery in the DELETE statement checks if an employee has no assigned projects using the NOT EXISTS operator, and if true, the employee is deleted.

These examples should give you a comprehensive understanding of different types of subqueries and how they can be used in various scenarios in SQL.