Sub Queries

In SQL, a subquery is a query embedded within another SQL query. You can alternately call it a nested or an inner query. The containing query is often referred to as the outer query. Subqueries are utilized to retrieve data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used in various parts of a query, including:

- SELECT statement
- FROM clause
- WHERE clause
- GROUP BY clause
- HAVING clause

Syntax

In general, the syntax can be written as:

```sql
SELECT column_name [, column_name]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
   (SELECT column_name [, column_name]
   FROM table1 [, table2 ]
   [WHERE])
```

Types of Subqueries

Scalar

In SQL, a scalar type is a type that holds a single value as opposed to composite types that hold multiple values. In simpler terms, scalar types represent a single unit of data.

Some common examples of scalar types in SQL include:

- Integers (`INT`)
- Floating-point numbers (`FLOAT`)
- Strings (`VARCHAR` , `CHAR`)
- Date and Time (`DATE` , `TIME`)
- Boolean (`BOOL`)

Scalar Subquery: It returns single value.

```sql
SELECT name
FROM student
WHERE roll_id = (SELECT roll_id FROM student WHERE name='John');
```

Row

In SQL, a "row" refers to a record in a table. Each row in a table represents a set of related data, and every row in the table has the same structure.

For instance, in a table named "customers", a row may represent one customer, with columns containing information like ID, name, address, email, etc.

Here is a conceptual SQL table:

ID	NAME	ADDRESS	EMAIL
1	John	NY	john@example.com
2	Jane	LA	jane@example.com
3	Jim	Chicago	jim@example.com

Each of these line of data is referred to as a 'row' in the SQL table.

Row subquery: It returns a single row or multiple rows of two or more values.

```
SELECT * FROM student
WHERE (roll_id, age)=(SELECT MIN(roll_id),MIN(age) FROM student);
```

Column

In SQL, columns are used to categorize the data in a table. A column serves as a structure that stores a specific type of data (ints, str, bool, etc.) in a table. Each column in a table is designed with a type, which configures the data that it can hold. Using the right column types and size can help to maintain data integrity and optimize performance.

Common SQL Column Types

1. **CHAR(n)** - It is a fixed-length character string that holds `n` characters. The size can be 1 to 255. For example,

```
CREATE TABLE Employee(ID CHAR(25));
```

2. **VARCHAR(n)** - A variable-length character string up to `n` characters where `n` can be from 1 to 255. For example,

```
CREATE TABLE Employee(ID VARCHAR(100));
```

3. **INT** - This type is used for integers. For example,

```
CREATE TABLE Customers(Age INT);
```

4. **DECIMAL(p,s)** - This is a decimal type used for precision and scale. `p` represents the total number of digits and `s` for numbers after the decimal. For example,

   ```
   CREATE TABLE Products(Price DECIMAL(5,2));
   ```

5. **DATE** - This type is used for date format 'YYYY-MM-DD'. For example,

   ```
   CREATE TABLE Orders(OrderedDate DATE);
   ```

6. **BOOL** - It stores Boolean data types. It can only take `True` or `False` values. For example,

   ```
   CREATE TABLE Users(IsActive BOOL);
   ```

In SQL, the column type helps in interpreting what kind of data to store in which column, whether it's number, text, date, or logical data. Remember, a table contains multiple columns and each column should have its unique name.

When creating a table, you should specify the column names, types, and maximum length of the type [if required].

Column subquery: It returns single column value with multiple rows and one column.

```
SELECT name, age FROM student
WHERE name in (SELECT name FROM student);
```

Table

In SQL, a table is a collection of related data held in a structured format within a database. It consists of rows (records) and columns (fields).

A table is defined by its name and the nature of data it will hold, i.e., each field has a name and a specific data type.

Table subquery: It returns more than one row and more than one column.

```
SELECT name, age
FROM student
WHERE (name, age) IN (SELECT name, age FROM student);
```

General Note

Subqueries can be either correlated or uncorrelated. A correlated subquery is a subquery that uses values from the outer query. Conversely, an uncorrelated subquery is a subquery that can be run independently of the outer query.

Nested Subqueries

In SQL, a subquery is a query that is nested inside a main query. If a subquery is nested inside another subquery, it is called a nested subquery. They can be used in SELECT, INSERT, UPDATE, or DELETE statements or inside another subquery.

Nested subqueries can get complicated quickly, but they are essential for performing complex database tasks.

Basic Syntax:

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
   (SELECT column_name [, column_name ]
   FROM table1 [, table2 ]
   [WHERE])
```

Example:

Here's an example where we want to find the customer names who made orders above the average order amount.

```
SELECT CustomerName,Country
FROM Customers
WHERE CustomerID IN
    (SELECT CustomerID
     FROM Orders
     WHERE Amount>(SELECT AVG(Amount)
                   FROM Orders))
```

In the above code:

- The innermost query calculates the average order amount.
- The middle subquery finds the `CustomerID`s from the `Orders` table where the order `Amount` is greater than the average.
- The outer query then gets the `CustomerName` from the `Customers` table where the `CustomerID` is in the list of `CustomerID`s fetched from the middle subquery.

These are the basic aspects of nested subqueries in SQL. They can go as deep as the task requires, but keep in mind that too many nested subqueries can cause performance issues.

Correlated Subqueries

In SQL, a correlated subquery is a subquery that uses values from the outer query in its WHERE clause. The correlated subquery is evaluated once for each row processed by the outer query. It exists because it depends on the outer query and it cannot execute independently of the outer query because the subquery is correlated with the outer query as it uses its column in its WHERE clause.

Syntax:

```
SELECT column_name [, column_name...]
FROM   table1 [, table2...]
WHERE  column_name OPERATOR
  (SELECT column_name [, column_name...]
   FROM table_name
   WHERE condition [table1.column_name = table2.column_name...]);
```

Code Example

For instance, if you want to get the employees whose salaries are above their department's average salaries, it can be queried with a correlated subquery as follows:

```
SELECT e1.employee_name, e1.salary
FROM employee e1
WHERE salary >
   (SELECT AVG(salary)
   FROM employee e2
   WHERE e1.department = e2.department);
```

In the example above, the correlated subquery (the inner query) calculates the average salary for each department. The outer query then compares the salary of each employee to the average salary of their respective department. It returns the employees whose salaries are above their department's average. The correlated subquery is executed once for each row selected by the outer query.

Also note that `e1` and `e2` are the aliases of the `employee` table so that we can use it in both the inner query and outer query. Here, `e2.department` in the inner query comes from the outer query's `e1.department`.

Thus, a correlated subquery is a subquery that depends on the outer SQL query for its values. This means that the subquery is run once for every Row in the outer query, often resulting in quite a bit of processing, and thus slower results.