<u>SUB QUERY</u>

A Subquery or Inner query or Nested query is a query within SQL query and embedded within the WHERE clause. A Subquery is a SELECT statement that is embedded in a clause of another SQL statement. They can be very useful to select rows from a table with a condition that depends on the data in the same or another table.

A Subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. The subquery can be placed in the following SQL clauses they are WHERE clause, HAVING clause, FROM clause.

<u>Advantages of Subquery:</u>

- Subqueries divide the complex query into isolated parts so that a complex query can be broken down into a series of logical steps.
- It is easy to understand, and code maintenance is also at ease.
- Subqueries allow you to use the results of another query in the outer query.
- In some cases, subqueries can replace complex joins and unions.

<u>Disadvantages of Subquery:</u>

- The optimizer is more mature for MYSQL for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as join.
- We cannot modify a table and select from the same table within a subquery in the same SQL statement.

<u>Rules for Sub-query:</u>

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

<u>Nested Subqueries:</u>

In Nested query, a query is written inside another query and the result of inner query is used in execution of outer query.

- Nested subqueries are subqueries that don't rely on an outer query. In other words, both queries in a nested subquery may be run as separate queries. In Nested Query,
- Inner query runs first, and only once.
- Bottom Up approach I.e. Outer query is executed with result from Inner query. Hence, Inner query is used in execution of Outer query.
- Inner query execution is not dependent on Outer query.
- Performs better than Correlated Query but is slower than Join Operation.

Example:

```
SELECT AVG(number_of_students) FROM classes
WHERE teacher_id IN (
        SELECT id
        FROM teachers
        WHERE subject = 'English' OR subject = 'History');
```

<u>Subquery with ALL Operator:</u>

The ALL operator compares a value to every value FROM the result table.

<u>Subquery with ANY Operator</u>**:**

The ANY operator compares a value to each value in a table and evaluates whether the result of an inner query contains at least one row.

<u>Correlated Subqueries:</u>

In Correlated query, a query is nested inside another query and inner query uses values from outer query.

- Subqueries are correlated when the inner and outer queries are interdependent, that is, when the outer query is a query that contains a subquery and the subquery itself is an inner query.
- It is like nested loops in programming concepts.
- In Correlated Query, outer query executes first and for every Outer query row Inner query is executed. Hence, Inner query uses values from Outer query.
- Top to Down Approach i.e. Outer query executes first and for every Outer query row Inner query is executed.
- Inner query is dependent on Outer query.
- Performs slower than both Nested Query and Join operations as for every outer query inner query is executed.

Example:

```
SELECT last_name, first_name
FROM    employee e1
WHERE NOT EXISTS (SELECT ph.last_name
                  FROM payment_history ph
                  WHERE ph.employee_id = e1.employee_id
                  AND ph.payment_type = 'award')
```

Subquery with EXISTS:

The EXISTS operator checks if the row FROM the subquery matches any row in the outer query. If there's no data matched, the EXISTS operator returns FALSE.

- When a subquery is used, the query optimizer performs additional steps before the results FROM the subquery are used. If a query that contains a subquery can be written using a join, it should be done this way. Joins usually allow the query optimizer to retrieve the data in a more efficient way.

Temporary table

Temporary Tables are most likely as Permanent Tables. Temporary Tables are Created in TempDB and are automatically deleted as soon as the last connection is terminated.

- Temporary Tables helps us to store and process intermediate results. Temporary tables are very useful when we need to store temporary data.

The Syntax to create a Temporary Table is given below:

- To Create Temporary Table:
  ```
  CREATE TABLE #EmpDetails (id INT, name VARCHAR(25))
  ```

- To Insert Values Into Temporary Table:
  ```
  INSERT INTO #EmpDetails VALUES (01, 'Lalit'), (02, 'Atharva')
  ```

- To Select Values from Temporary Table:
  ```
  SELECT * FROM #EmpDetails
  ```

There are 2 types of Temporary Tables:

- Local Temporary Table,
- Global Temporary Table.

Local Temporary Table:

A Local Temp Table is available only for the session that has created it. It is automatically dropped (deleted) when the connection that has created it, is closed.

- To create Local Temporary Table Single "#" is used as the prefix of a table name.

- Also, the user can drop this temporary table by using the “DROP TABLE #EmpDetails” query. There will be Random Numbers are appended to the Name of Table Name.
- If the Temporary Table is created inside the stored procedure, it gets dropped automatically upon the completion of stored procedure execution.
- Example:

```
CREATE PROCEDURE ProcTemp
AS
BEGIN
CREATE TABLE #EmpDetails
INSERT INTO #EmpDetails VALUES ( 01, 'Lalit'), ( 02, 'Atharva')
SELECT * FROM #EmpDetails
END
EXECUTE ProcTemp
```

Global Temporary Table:

Global Temporary Tables are visible to all connections and Dropped when the last connection referencing the table is closed.

- To create a Global Temporary Table, add the “##” symbol before the table name.
- Global Table Name must have a Unique Table Name. There will be no random Numbers suffixed at the end of the Table Name.

Syntax : CREATE TABLE ##EmpDetails (id INT, name VARCHAR(25))

By default, all the temporary tables are deleted by MySQL when your database connection gets terminated. Still if you want to delete them in between, then you can do so by issuing a DROP TABLE command.

SELF-JOIN

The self-join, as its name implies, joins a table to itself. To use a self-join, the table must contain a column (call it X) that acts as the primary key and a different column (call it Y) that stores values that can be matched up with the values in Column X. The values of Columns X and Y do not have to be the same for any given row, and the value in Column Y may even be null.

Example:

```
SELECT employee.Id, employee.FullName, employee.ManagerId,
FROM Employees employee
JOIN Employees manager
ON employee.ManagerId = manager.Id
```

The **JOIN** keyword connects two tables and is usually followed by an ON or USING clause that specifies the common columns used for linking the two tables.

Self joins are commonly used in the following areas:

- Hierarchical relationships
- Sequential relationships
- Graph data

<u>Sequential Data</u>

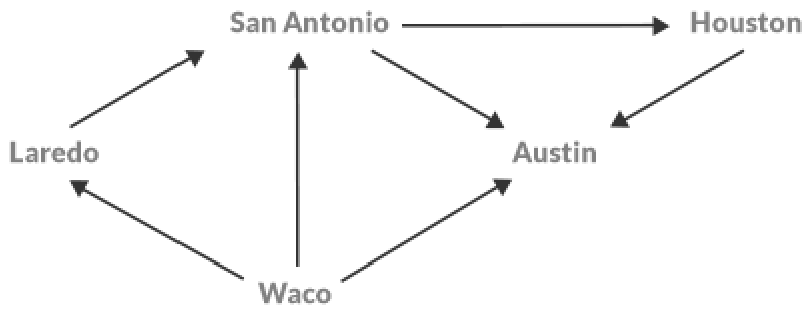
Sequential data can also benefit from using a SQL self join. For example, suppose you have records that describe the consecutive steps required to prepare a dish. All of the steps can be placed in a single table. Their order is determined based on the columns that point to the IDs of the previous and next records in the same table.

<u>Hierarchical Data</u>

Processing hierarchical data is one of the most frequent applications of the SQL self join. This occurs when there is an additional column pointing to an identifier in the same table, such as in our employee table. In our case, the manager_id column refers to (has the same value as) the id column.

<u>Graphs</u>

Selfjoin SQL can also be used to show the relationships needed for graphs. A GRAPH is a structure consisting of nodes connected to each other with edges (relations). One example of a graph is the road network between multiple cities. Take below example



Example:

```
SELECT c1.name AS from_city, c2.name AS to_city
FROM city c1
JOIN route r ON c1.id = r.from_city_id
JOIN city c2  ON c2.id = r.to_city_id ;
```

The city table stores the ID number and the name of each city. The route table contains the route ID number, the starting city (the from_city_id column) and the target city (the to_city_id column).