MENTORNESS ARTICLE TASK 1

ADVANCE SQL TECHNIQUE – SUBQUERIES & CTE

~ By Faiqua Sadiq

ADVANCE SQL – SUBQUERY AND CTE

INTRODUCTION

A database is a well-ordered collection of data. A database is a system that permits users to easily manipulate, access, update the data. In simple words databases allows storing, managing, and retrieval of information. Some of the common databases are MySQL, PostgreSQL, Oracle, and SQL Server.

Relational Databases are those databases that are used to store the structured data in the form of a table. A table is a structure that is organizes structured data into rows and columns.

Relational databases are quite large and has multiple columns and rows. In most cases the data is organized into multiple tables spanning over different databases and these data from myriad sources needs to be co-related to retrieve meaningful data. Data scientists and data analysts do not pull the entire database for querying, instead only a part if the information stored table is needed to be extracted. This is where SQL comes into picture.

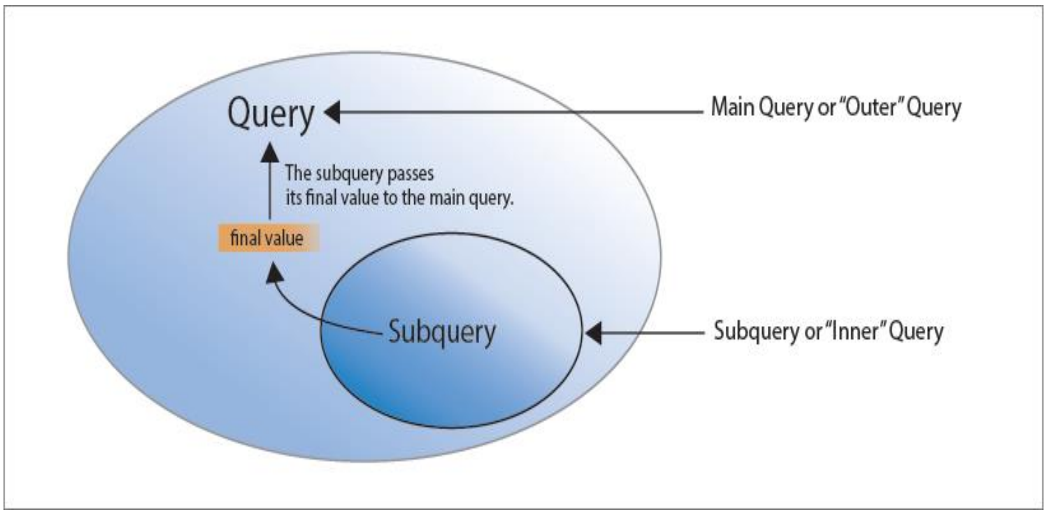
SQL is an acronym for Structural Query Language. SQL is a query language used for storing, manipulation, and retrieving data from the databases.

However, to address complex scenarios in data analytics, we need something beyond the basic SQL skills. Advanced SQL solves this problem. It enable a wider range of analytic abilities, to work more quickly and efficiently with structured databases. In this article we are going to discuss about sub queries and CTE.

ADVANCE SQL TYPES

1. SUBQUERIES

In SQL, a subquery can simply be referred to as a query within another query. A subquery is basically a SELECT statement that is nested inside another statement consisting of either of SELECT, DELETE, INSERT, UPDATE statements. The subquery is executed first and its result is then used as a parameter or condition for the outer query.


Query
Main Query or "Outer" Query
The subquery passes its final value to the main query.
final value
Subquery
Subquery or "Inner" Query

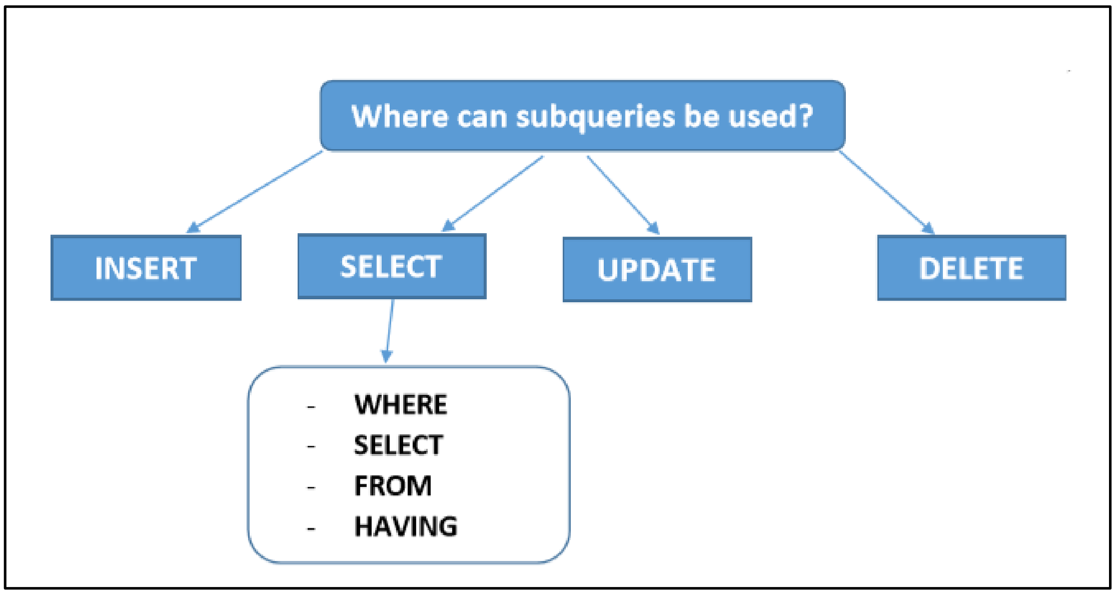
Example: **To find the details of the highest rating movie in the movies dataset.**

```
SELECT * FROM movies
WHERE rating = (SELECT max(rating) from movies)
```

OUTER QUERY

INNER SUB-QUERY

Where Can Subqueries be used



SUBQUERIES WITH SELECT STATEMENT

Considering an EMPLOYEE table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedab	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Now, considering the following example of subquery with a SELECT statement.

Example : To get the percentage salary of each employee

SELECT name,

(salary / (SELECT sum(salary) FROM EMPLOYEE))*100 as per_salary

FROM EMPLOYEE

Output

NAME	per_salary
Ramesh	5.71%
Khilan	4.29%
kaushik	5.71%
Chaitali	18.57%
Hardik	24.29%
Komal	12.86%
Muffy	28.57%

Usage with where

Example: To select details of employees whose salary is more than 5000

```
SELECT *   FROM employee
WHERE ID IN (SELECT ID
 FROM employee
 WHERE SALARY > 5000)
```

Output

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Usage with from

Considering the following Employee table

ID	Name	Department	Salary
1	John	HR	50000
2	Alice	IT	60000
3	Bob	Finance	55000
4	Sarah	Marketing	52000
5	Michael	HR	48000
6	Emily	IT	65000
7	David	Finance	58000

Example: To find average salary of each department

```
SELECT b.department, a.av_sal
FROM  (SELECT Department, AVG(Salary) AS av_sal
    FROM Employee
    GROUP BY Department) AS a
JOIN Employee AS b ON a.Department = b.Department
```

Output

Department	Average Salary
HR	49000
IT	62500
Finance	56500
Marketing	52000

Usage with having

Example: To find departments having average salary grater then average salar of all the departments.

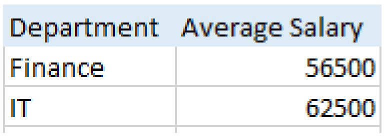
```
SELECT department, AVG(salary)

FROM employee

GROUP BY department

HAVING AVG(salary) > (SELECT AVG(salary) FROM employee)
```

Output



Department	Average Salary
Finance	56500
IT	62500

SUBQUERIES WITH INSERT STATEMENT

Subqueries can be used with INSERT statements. The INSERT statement is used to return the data from the subquery and is inserted into another table. The selected data in the subquery can be modified with any of the, date or character or number functions.

Example: Consider a table cust_table_backup with similar structure as “cust_table” table. Now to copy the complete “cust_table” table into the cust_table_backup table, we can use the following syntax.

```
INSERT INTO cust_table_backup

  SELECT * FROM cust_table

  WHERE ID IN (SELECT ID

  FROM cust_table)
```

SUBQUERIES WITH UPDATE STATEMENT

The subquery can be used along with an UPDATE statement. Single or multiple columns in a table can be updated by using a subquery in conjugation with an UPDATE statement.

Example: We have cust_table_backup table available which is backup of cust_table table. The following example updates SALARY by 10 times in the cust_table table for all the customers whose AGE is equal to 35.

```
UPDATE cust_table

SET salary = SALARY * 10

WHERE age IN (SELECT age FROM cust_table_backup WHERE age = 35)
```

SUBQUERIES WITH DELETE STATEMENT

The subquery can also be used in conjunction with the DELETE statement like any other statements mentioned previously.

Example: Assuming, we have a "cust_table_backup" table available which is a backup of the cust_table table. The following example deletes the records from the cust_table table for all the customers whose AGE is greater than or equal to 28.

```
DELETE FROM cust_table

WHERE AGE IN (SELECT AGE FROM cust_table_backup

WHERE AGE >= 28 )
```

2. CTE (Common Table Expressions)

A **common table expression** (CTE) is a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statements. Each SQL CTE is like a **named query**, the result of which is stored in a virtual table (a CTE) that is to be referenced in the main query.

Using the WITH statement, one can create temporary tables to store results, making complex queries more readable and maintainable. These temporary tables exist only for the duration of the main query, which is used to further streamline the analysis process.

with ■ as (■■■)
select *
from ■ where ■■■

BASIC CTE SYNTAX

```
WITH my_cte AS (
  SELECT col_name1, col_name2, col_name3
  FROM Table1
)
```

→ **CTE**

```
SELECT col_name1, col_name2
FROM my_cte
WHERE [condition]
```

→ **Query using CTE**

Types of CTE

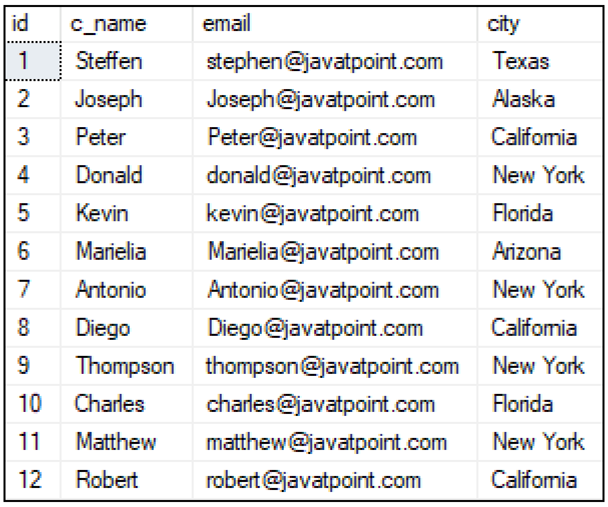
CTEs are divided into 2 Categories

a. **Recursive CTE**:

A recursive CTE is a query which references itself. Its concept is based on recursion. When recursive query is executed, it repeatedly iterates over a subset of the data. It is simply defined as a query that calls itself. There is an end condition or a base condition given, so that the recursive CTE does not call itself infinitely.

A recursive CTE must have a UNION ALL statement along with a second query definition that references the CTE itself in order to be recursive.

Considering an example to generate a series of 1st 5 odd numbers to demonstrate recursive CTE from the following dataset

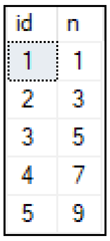


id	c_name	email	city
1	Steffen	stephen@javatpoint.com	Texas
2	Joseph	Joseph@javatpoint.com	Alaska
3	Peter	Peter@javatpoint.com	California
4	Donald	donald@javatpoint.com	New York
5	Kevin	kevin@javatpoint.com	Florida
6	Marielia	Marielia@javatpoint.com	Arizona
7	Antonio	Antonio@javatpoint.com	New York
8	Diego	Diego@javatpoint.com	California
9	Thompson	thompson@javatpoint.com	New York
10	Charles	charles@javatpoint.com	Florida
11	Matthew	matthew@javatpoint.com	New York
12	Robert	robert@javatpoint.com	California

Recursive CTE Query

```
WITH
odd_num_cte (id, n) AS
(
SELECT 1, 1
UNION ALL
SELECT id+1, n+2 from odd_num_cte where id < 5
)
SELECT * FROM odd_num_cte;
```

Output of Recursive CTE



id	n
1	1
2	3
3	5
4	7
5	9

b. <u>Non-recursive CTE</u>

A common table expression (CTE) that doesn't reference itself is known as a non-recursive CTE. It does not use the concept of recursion. According to the CTE Syntax, each CTE query begins with

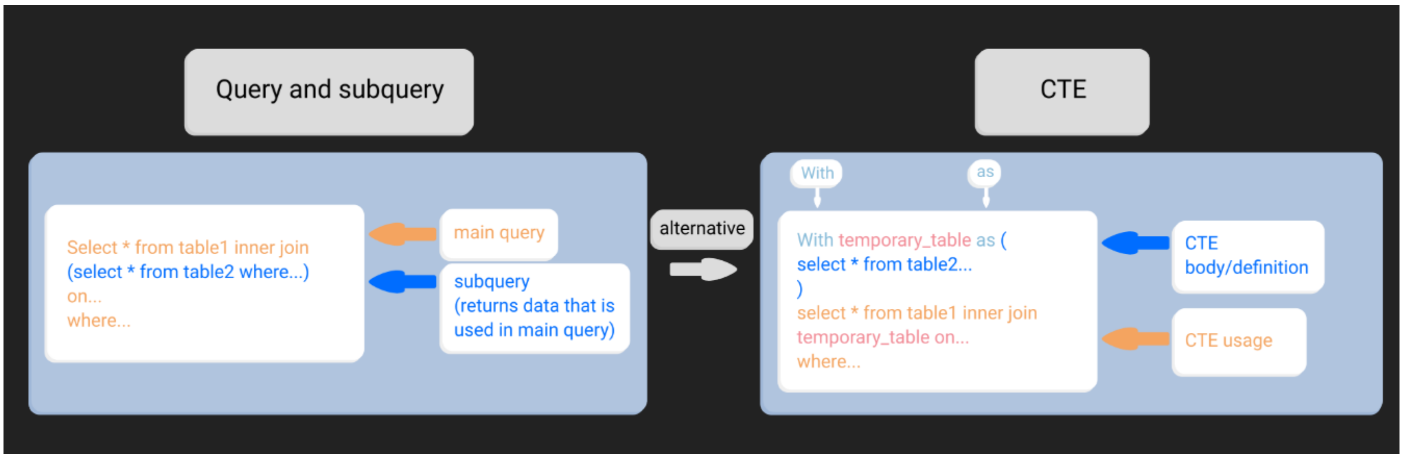
i. "**With**" clause
ii. CTE name
iii. column list
iv. AS with parenthesis.

CONCLUSION

In SQL, both Subqueries and Common Table Expressions (CTEs) are used to perform data manipulation on complex queries. Both these queries are however different in terms of syntax and usage.

CTEs are defined using WITH keyword and these CTEs make it easier for users to be referenced multiple times in a query. On the other hand, Subqueries are nested queries, i.e, it is a query within another query. Both these techniques are helpful in writing complex SQL queries efficiently.

In the actual world, several tables are needed to be combined to form a single data model where performed calculations are referred to several times. Subqueries and CTEs both aid in efficient writing of queries.


Query and subquery
CTE
Select * from table1 inner join
(select * from table2 where...)
on...
where...
main query
subquery
(returns data that is used in main query)
alternative
With
as
With temporary_table as (
select * from table2...
)
select * from table1 inner join
temporary_table on...
where...
CTE
body/definition
CTE usage