SUBQUERY

Query inside another Query.

~~Step Step4~~

Outer

Step4 → Aggregation → Main Query

Step3 → Transformation → Subquery

Step2 → Filtering → Sub query → Inner.

Step1 → Join Table → Sub query

Dependency

- Non-corelated Sub-Query
- Co-related Query.

Result Types

- Scalar Subquery
- ROW Sub Query
- Table Sub Query.

Location | Clauses

- SELECT
- FROM
- JOIN
- WHERE
  - comparison operator
    <, >, =, != , >=, <=
  - logical operator
    IN, ANY, ALL, EXIST.

Result types :

Scalar subquery : single value - e AVG(sales)

Row subquery : Multiple Rows & columns.

1
2
3
4

```
SELECT
  custID
  FROM sales.order
```

Table subquery : Multiple Rows & columns.

1	4
2	b
c	c

Location / clause :

~~[illegible]~~

FROM clauses : uses a temporary table for the main query.

Syntax. SELECT col1 , col2, --- → Main query.
FROM (SELECT col FROM Table WHERE condn) As alias
↓ Subquery

Ex:-
```
SELECT
  * FROM(
        SELECT
            PID,
            price,
            AVG(price) OVER avgp
            FROM sales. orders
       ) t
       where price > avgp
```

Ex:-
```
SELECT
   *,
  RANK() OVER(ORDER BY TS DESC) CR
   FROM(
        SELECT
            custID,
        SUM(sales) Totalsales
            FROM sales. order
            GROUP BY custID) t.
```

SELECT clause : used to aggregate data side by side with main Query's data, allowing for direct comparison.

Syntax: SELECT ⟶ Main Query
```
SELECT
    Column1,
  (SELECT column FROM table1 WHERE condi) As alias
  FROM Table1
```
(SELECT column FROM table1 WHERE condi) — Subquery

Rule : only scalar subqueries are allowed to be used. (only single value).

Ex:
```
SELECT
    PID,
    product,
    price,
    (SELECT
        COUNT(*) ~~[illegible]~~
      FROM sales.orders) AS Totalorder
  FROM sales.products.
```

JOIN clause : used to prepare the data (filtering or aggregation) before joining it with other Tables.

Ex:-
```
SELECT
    c.*,
    o.TO.
    FROM sales.customer c
    LEFT JOIN (
            SELECT
                custID,
                COUNT(*) Totalorder
                FROM sales.orders
                GROUP BY custID) o
      ON c.custoID = o.custID.
```

WHERE Subquery : used for complex logic and makes Query more flexible and dynamic.

Comparison operators: use to filter data by comparing two values.

Syntax

```
SELECT col1, col2          → Main Query.
  FROM Table1
  WHERE column = (SELECT col FROM Table2
                  WHERE condn)
                  ____________
                    Subquery
```

Rule: only scalar subqueries are allowed.
only single value.

Eg:

```
SELECT
  *
  FROM sales.products
  WHERE price > (
                SELECT
                AVG(price) avgprice
                FROM sales.products
               )
```

IN OPERATOR: ~~check~~ check whether a value matches any value from a list.

Syntax

```
SELECT col1, col2     → Main Query.
  FROM Table 1
  WHERE Col IN(SELECT col FROM table2 WHERE conn)
                 ________________
                     Subquery
```

Eg:

```
SELECT *
  FROM Sales.orders
  WHERE customerID IN (
                SELECT
                customerID
                FROM sales.customers
                WHERE country = 'Germany')
```

use NOT IN to get all order who are not from germany.

ANY/ALL :: Check If a values matches all values within a list.

~~Check~~ check If a value matches ANY value within a list. used to check If a value is true for ATLEAST one of values in a list.

Syntax.
```
SELECT col1 , col2 ---   → Main Query.
  FROM table 1
WHERE col < ANY/ALL(SELECT col FROM table1 WHERE
                                    condn)
                                 Sub Query
```

Ex: ~~SELECT~~ (saved in laptop)
~~EID~~,   file.
~~FN~~.

* NON-~~CORR~~ CORRELATED SUBQUERY: subquery that can run independently from Main Query.

* CORRELATED SUBQUERY: subquery that relays on values from main query.

Ex:- ~~saved in laptop~~: SELECT
```
*,
(SELECT COUNT(*) FROM sales.orders o WHERE o.custID = c.custID) TS
FROM sales.customers c.
```

CORRELATED SUBQUERY (EXIST). : checks If a subquery returns any rows.

Syntax
```
SELECT col1 , col2 --
  FROM table2
  WHERE EXISTS ( SELECT 1
                 FROM table 1
                WHERE Table1.ID = table2.ID)
```

HOW EXIST WORK

$$\begin{tikzcd}
 & \boxed{\text{For each ROW in Main Query}} \arrow[d] & \\
 & \boxed{\text{Run Sub Query}} \arrow[dl] \arrow[dr] & \\
\text{NO Result?} \arrow[d] & & \text{return value?} \arrow[d] \\
\text{Row of Main Query excluded} & & \text{Row of Main Query Included.}
\end{tikzcd}$$

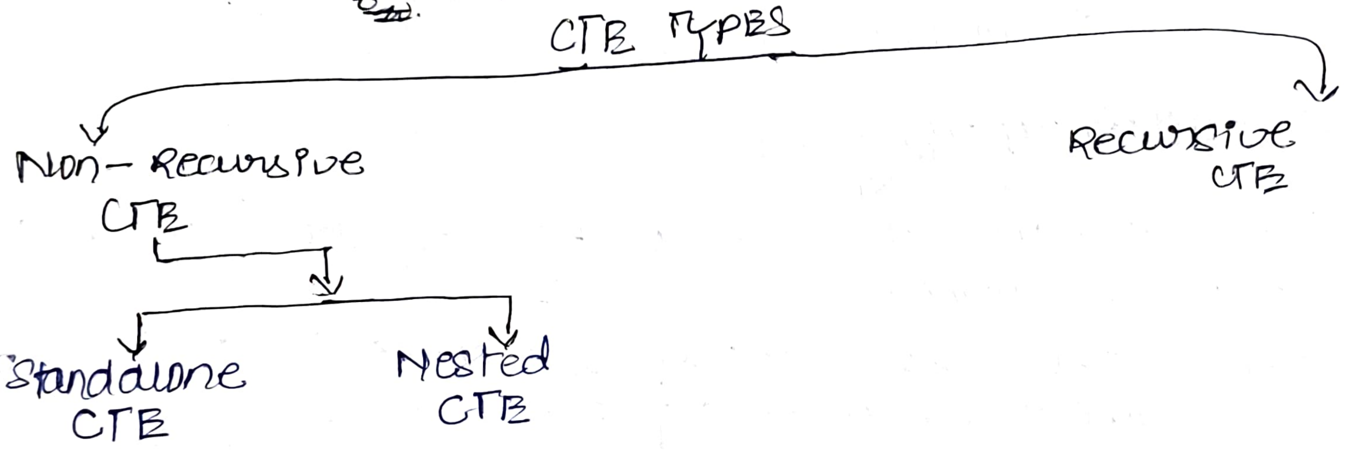
```
Ex: SELECT
    *
    FROM sales.orders o
    WHERE EXISTS (SELECT
                        ~~customerid~~ 1
                        FROM sales.customers c
                        WHERE country = 'Germany'
                        AND o.customerID = c.customerID)
```

CTE

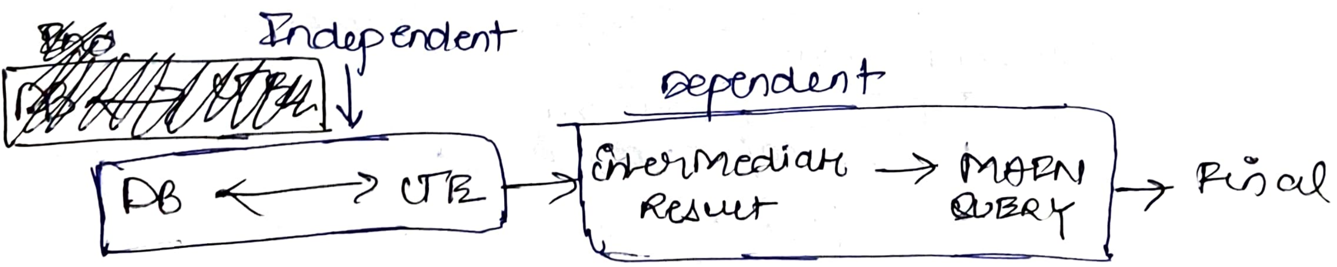
Temporary, named result set (virtual Table) that can be multiple times within your query to simplify and organize complex query.

~~[illegible]~~

~~why CTE~~



Standalone CTE. Define and used independently. Runs independently as it's self-contained and doesn't rely on other CTEs or queries.



Syntax

```
WITH CTE-NAME AS
(
  SELECT --
  FROM --                 ← CTE Query.
  WHERE --
)
SELECT ---                → Main Query.
FROM CTE-Name
WHERE --
```

Ex:- step1

```
SELECT
    custID
    SUM(sales) Totalsales
    FROM Sales.O
    Group BY custID
```

Ex:- Example is saved in Lap-top.

#Multiple Standalone CTE

Syntax

```
WITH CTE_name1 AS                 SELECT
 ( SELECT                         FROM CTE_name 1
    FROM                          JOIN CTE_name2
    WHERE                         WHERE
 )
 , CTE_name2 AS
 ( SELECT
    FROM
    WHERE
 )
```

Ex:- Saved in laptop.

Nested CTE

CTE inside another CTE.
A nested CTE uses the result of another CTE, so it can't run independently.

Syntax

```
WITH CTE_name 1 AS
(
  SELECT --          ]  Standalone
  FROM --            ]  CTE
  WHERE --
)
```

```
, CTE_name2 AS
(
  SELECT
  FROM CTE_name 1
  WHERE
)
SELECT
FROM CTE_name 2
WHERE --
```

Nested CTE

Ex:- Saved in laptop.

Recursive CTE :- self-referencing query that repeatedly processes data until a specific condition is met.

Syntax :

```
WITH CTE_Name AS
(
  SELECT ---
  FROM ---
  WHERE --
  -UNION ALL
    SELECT
    FROM CTE_Name
    WHERE (Break condition)
)
```

Anchor Query.

~~base~~ Recursive Query.

#. use union All bcoz in one query there is no 2 select.

```
-- Main Query
   SELECT
   FROM CTE_Name
   WHERE _
```

Main

MAX RECURSION: increase the recursion.

Ex:- save in laptop.

Views

- Databases server : stores, manages and provides access to databases for users or application.
- Database : collection of information that is stored in a structured way.
- Schema : logical layer that groups related objects together.
- Table : place where data is stored and organized into rows and columns.
- VIEW : Visual table that shows data without storing it physically.
- DDL : set of commands that allow to define and manage the structure of Database.

3 tier architecture.

1) External (view layer): Top layer, where end user or applications interact with database.
   - Defines how users view the data.
   - Different users has different views.

2) Conceptual (Logical layer) : Defines what data is stored in the database and relationships b/w them.
   - Hides physical storage details.
   - Maintains logical structure - tables, columns, data types, R/sn (primary/foreign key).

3) Internal level (physical) : Defines how data is stored in Memory or disk.
   - Deal with indexes, File organization, data blocks, compression etc.
   - Managed by DBMS engine, not visible to users.

#View : Virtual Table based on result set of a Query, without storing the data in database.

• Views are persisted SQL Queries in the database.

Table
physical

Q. | VIEW
Virtual
Table

Query.

Real Data ⟷ abstraction layer ⟷ You.

~~[illegible]~~ #1 use case

<u>Central Query logic</u>: Store central, complex Query logic in the database for access by multiple Queries, reducing project complexity.

VIEW	CTE
• Reduce Redundancy in Multi-Queries.	• Reduce Redundancy in 1 Query.
• Improve Reusability in Multi- Queries.	• Improve Reusability in 1 Query.
• persisted logic	• Temporary logic
• Need to Maintain -create/ drop	• No maintenance - auto cleanup.

<u>VIEW</u> (syntax)

DDL Command ← CREATE VIEW VIEW_Name AS

```
(
    SELECT ---
    FROM ---
    WHERE --
)
```

Query →

NOTE:- If a table or view is created without specifying a schema, it defaults to the DBD.

Delete View

DROP VIEW VIEW_NAME

UPDATE (In SQL server)

1) Drop the view.
2) Now create a view.

T-SQL : Transact SQL is an extension of SQL that adds programming features.

HOW DB Execute Views

2 USE CASE: HIDE COMPLEXITY: Views can be use to hide the complexity of a database tables and offers users more friendly and easy to -consume objects.

use CASE(3): Data Security: use views to enforce security and protect sensitive data, by hiding columns and/or rows from tables.

use case (4): Flexibility and dynamic:

use case(5): Multiple Languages:

use case (6): Virtual Data Marts in DWH: Views can be used as Data Marts in Data warehouse system. Because they provide a flexible and efficient way to present data.