



DATA WITH BARAA

ADVANCED SQL Techniques

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**



SOLUTIONS

Subquery

- CTE -

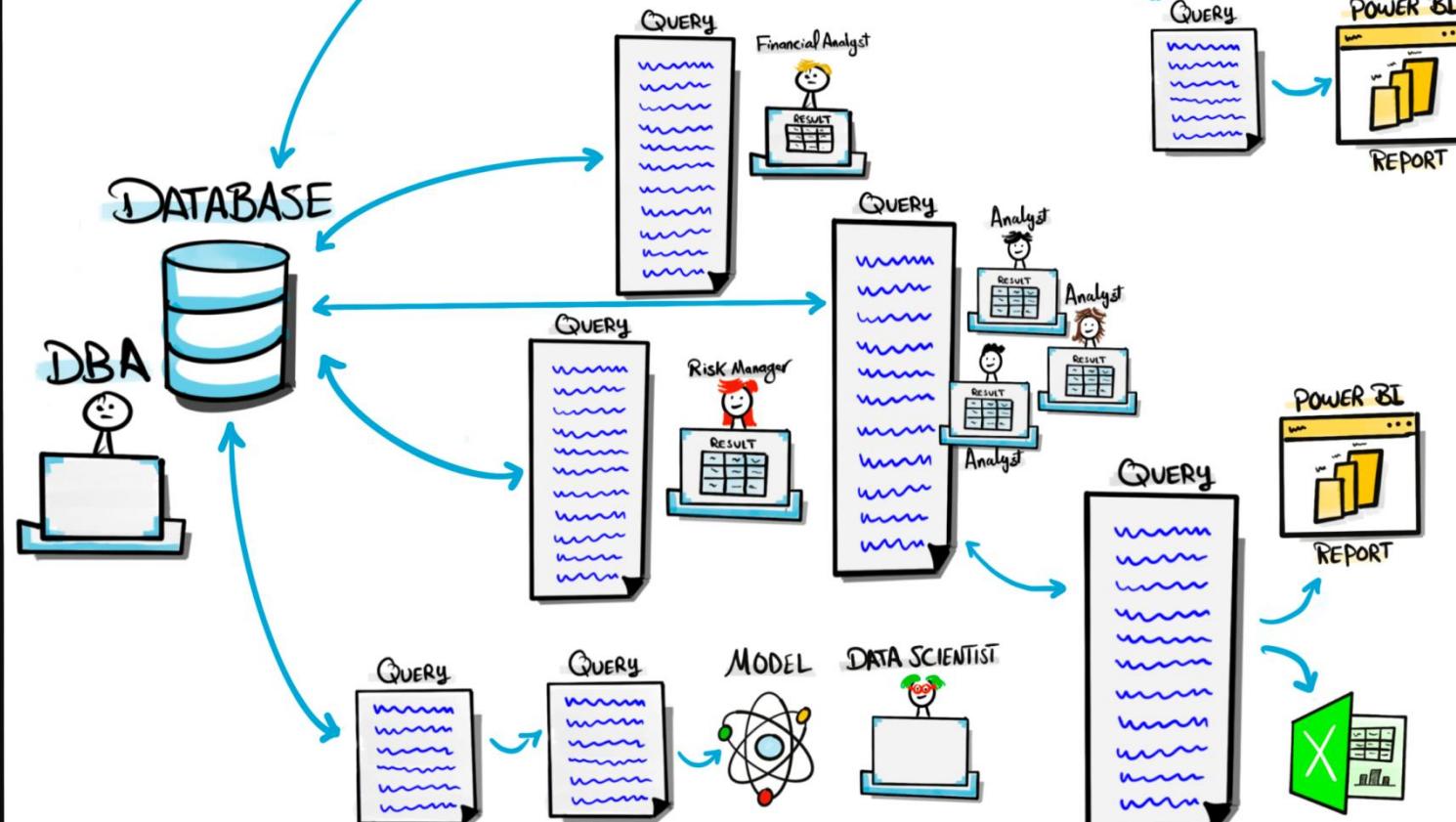
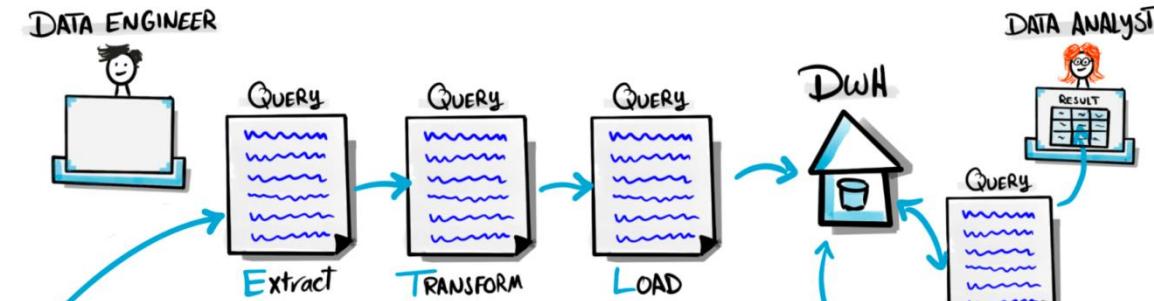
Common Table Expression

Views

Temp Tables

- CTAS -

Create Table As Select



CHALLENGES

Redundancy

Performance Issues

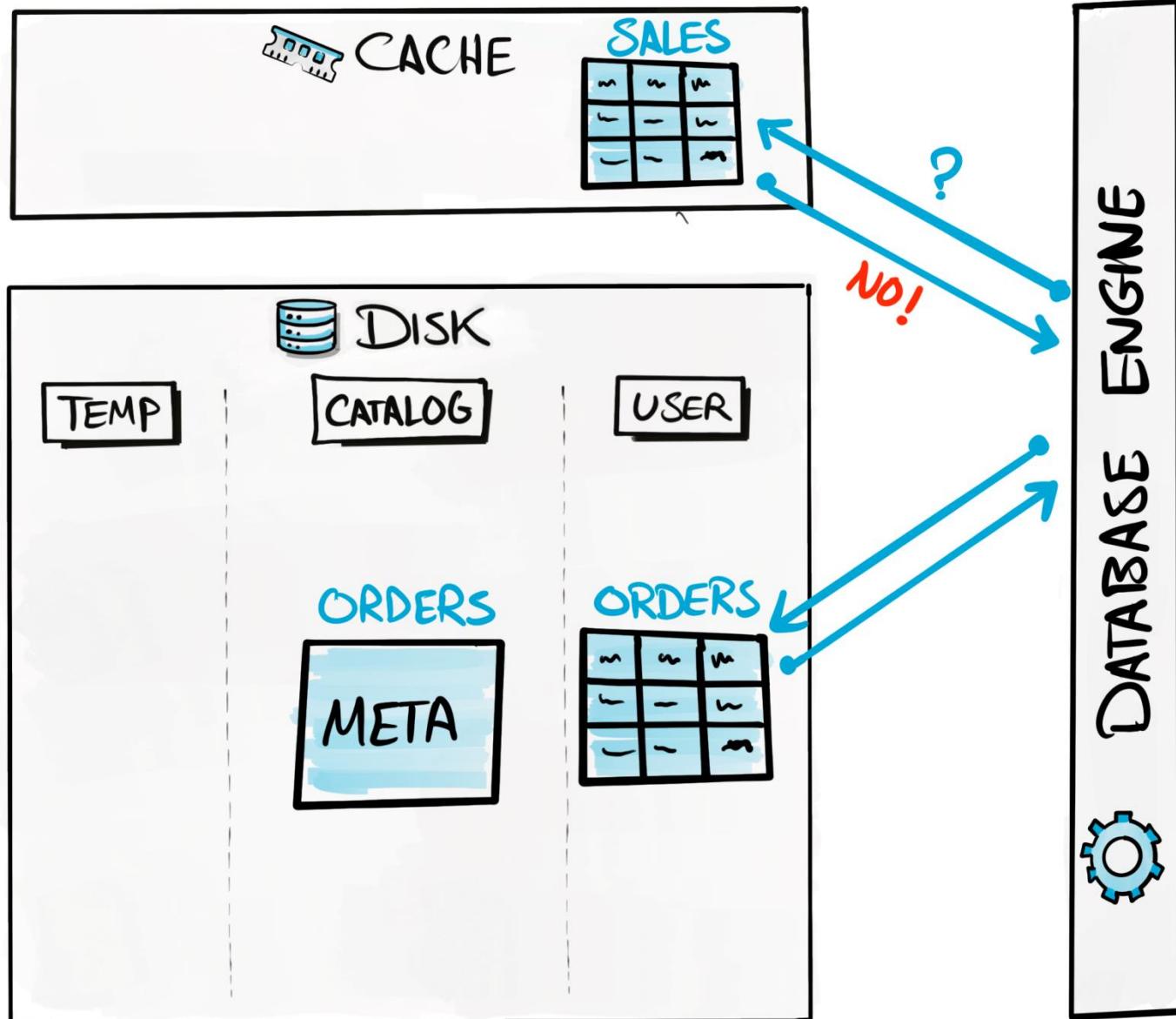
Complexity

Hard To Maintain

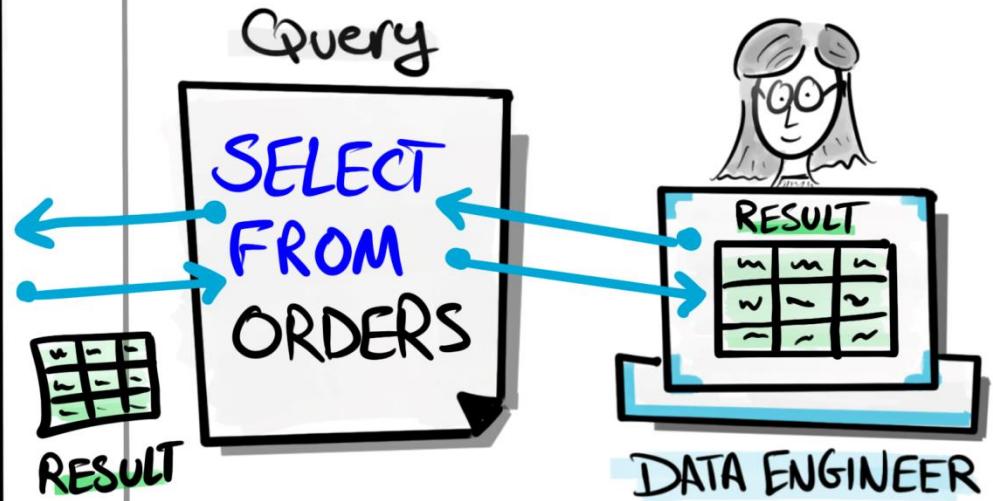
DB Stress

Security

SERVER



CLIENT



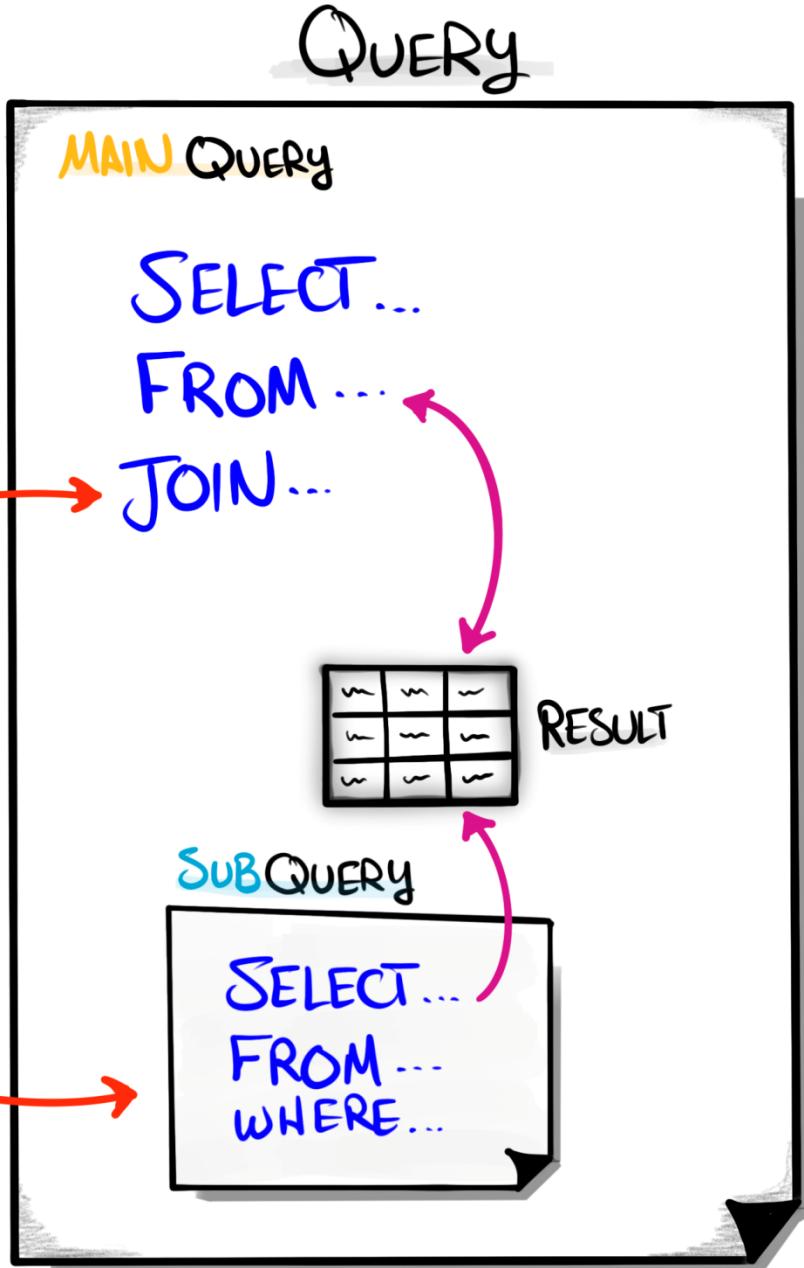
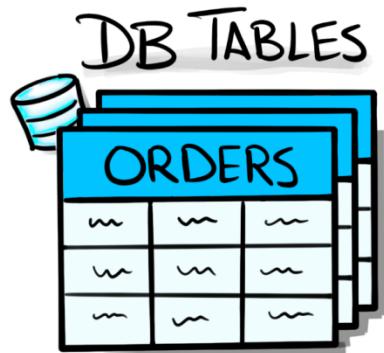


Subquery

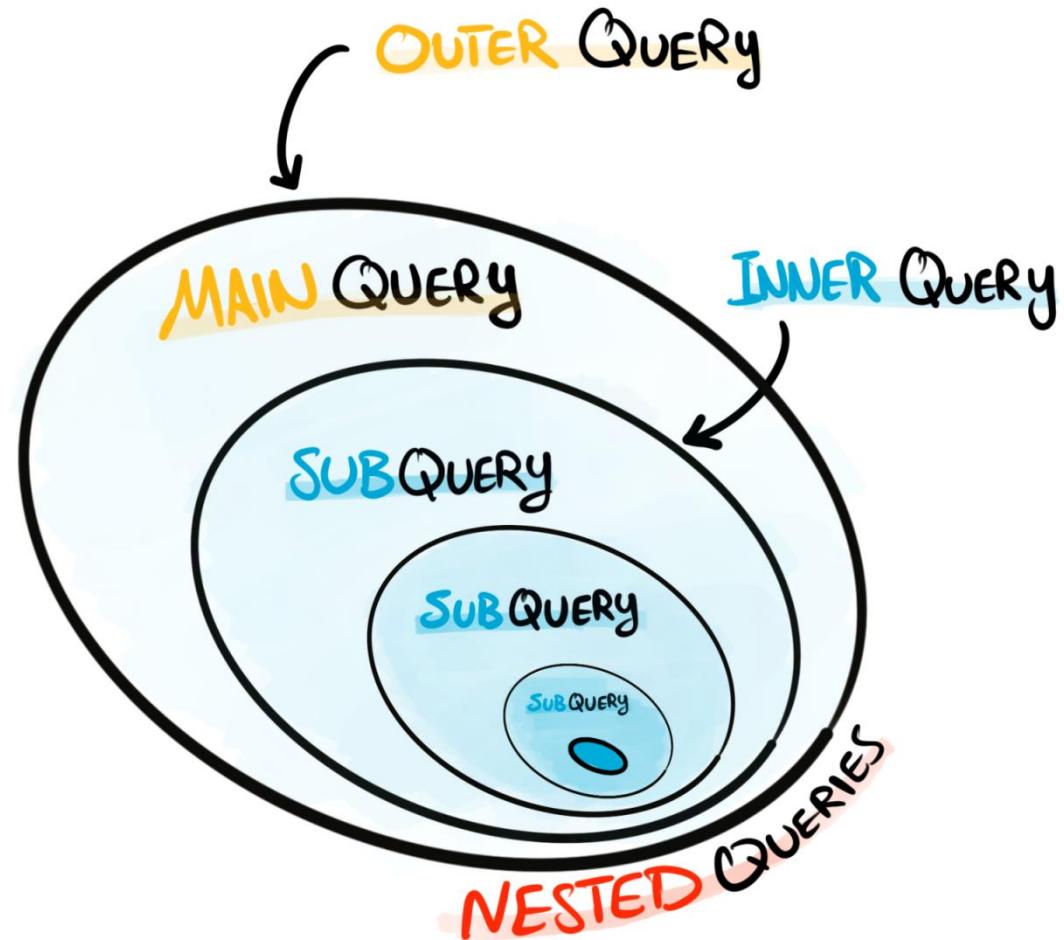
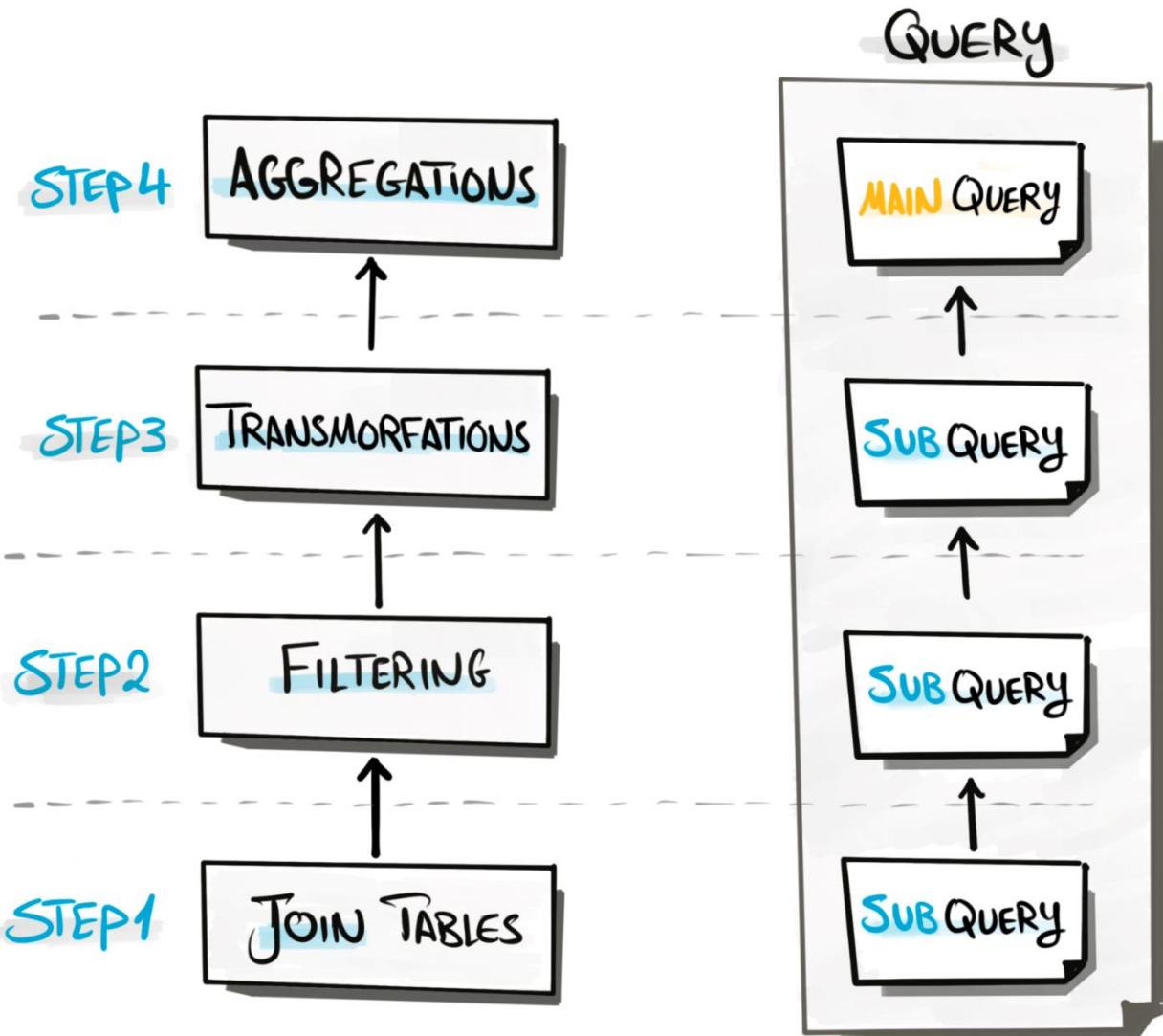
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Subquery



How Subquery Works



SUBQUERY



DATABASE

| ORDERS | | |
|--------|---|---|
| ~ | ~ | ~ |
| ~ | ~ | ~ |
| ~ | ~ | ~ |

| CUSTOMERS | | |
|-----------|---|---|
| ~ | ~ | ~ |
| ~ | ~ | ~ |
| ~ | ~ | ~ |

Subquery

SELECT...
FROM...
WHERE...

Result

| | | |
|---|---|---|
| ~ | ~ | ~ |
| ~ | ~ | ~ |
| ~ | ~ | ~ |

Subquery

SELECT...
FROM...
WHERE...

Result

| | | |
|---|---|---|
| ~ | ~ | ~ |
| ~ | ~ | ~ |
| ~ | ~ | ~ |

Subquery

SELECT...
FROM...
WHERE...

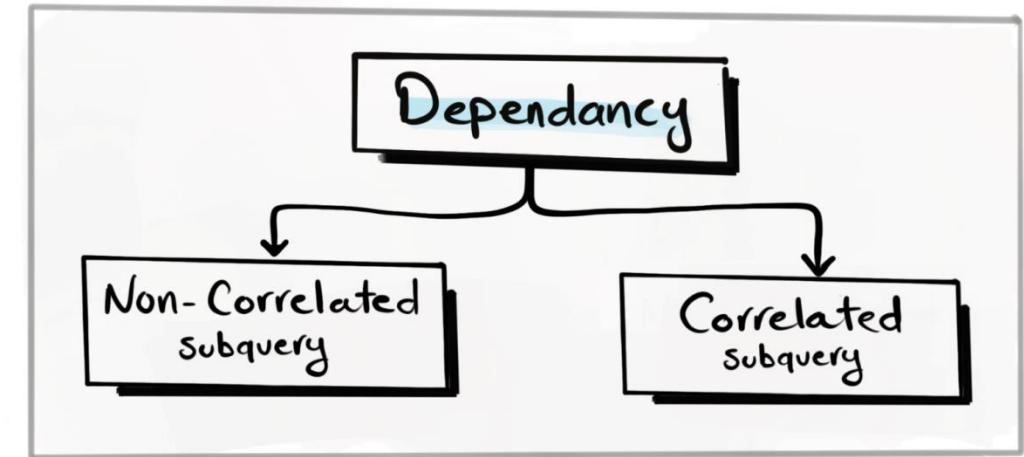
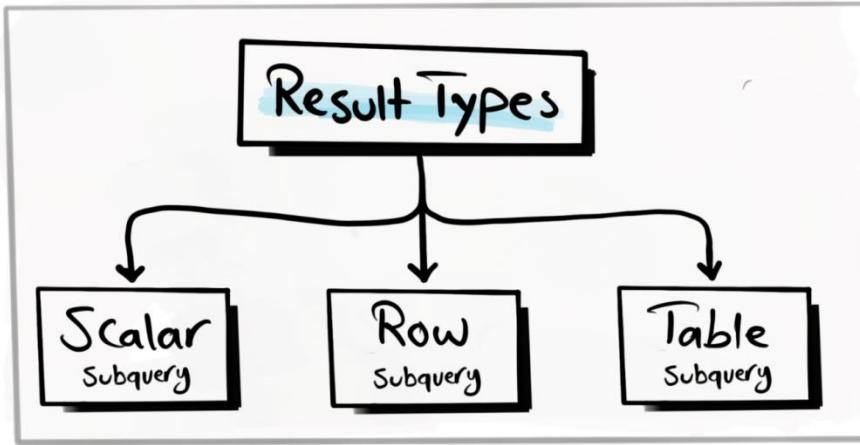
Result

| | | |
|---|---|---|
| ~ | ~ | ~ |
| ~ | ~ | ~ |
| ~ | ~ | ~ |

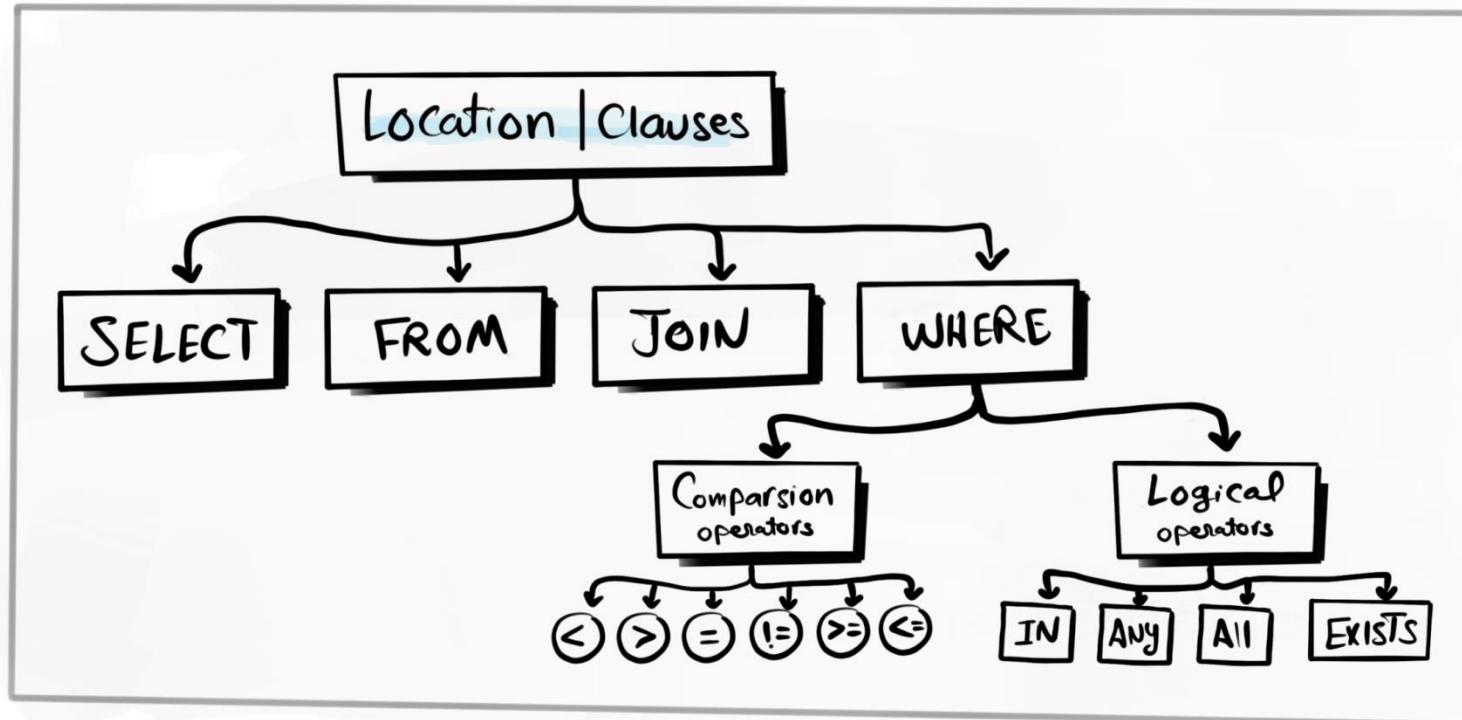
MAIN Query

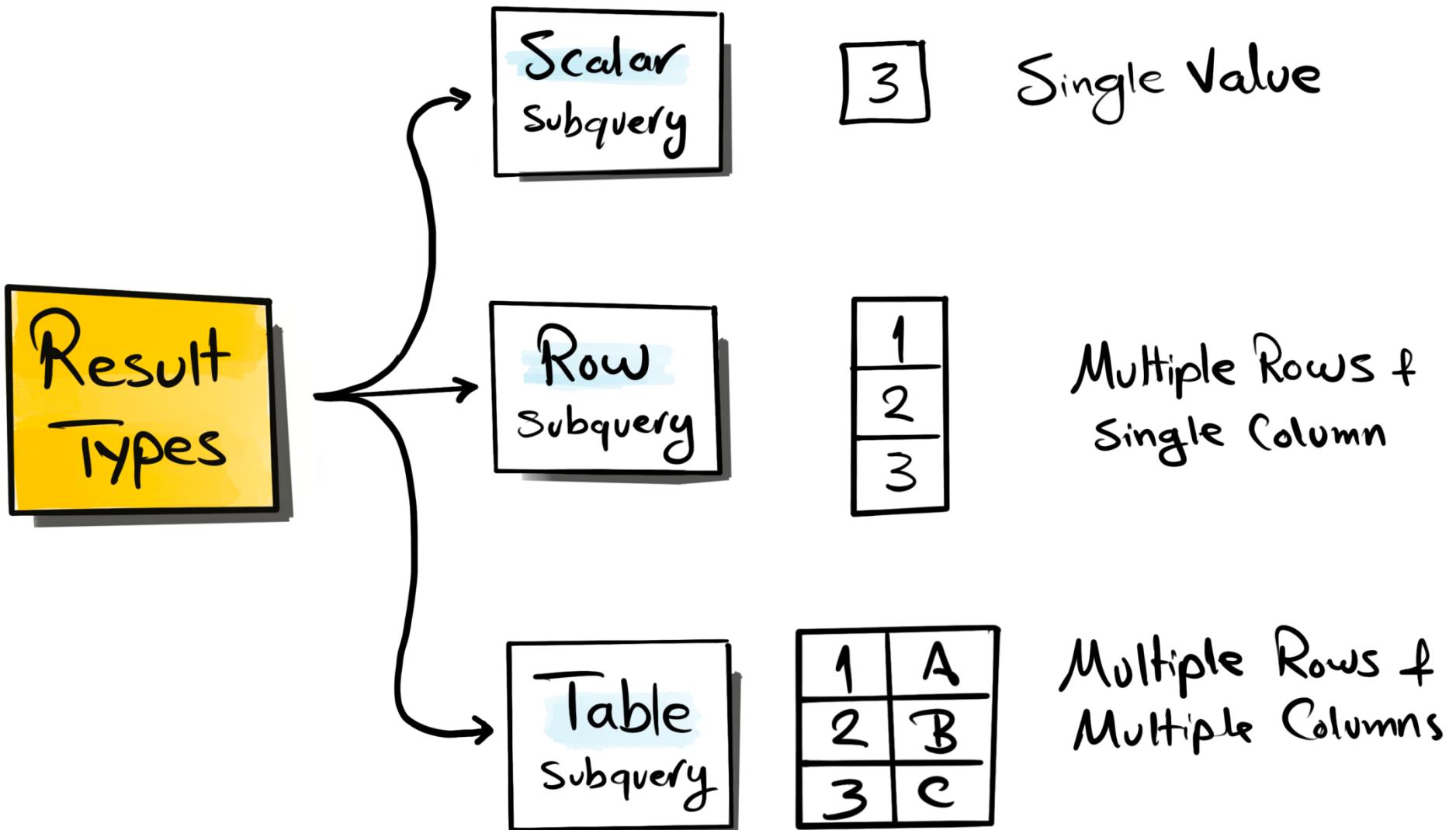
Final
Result

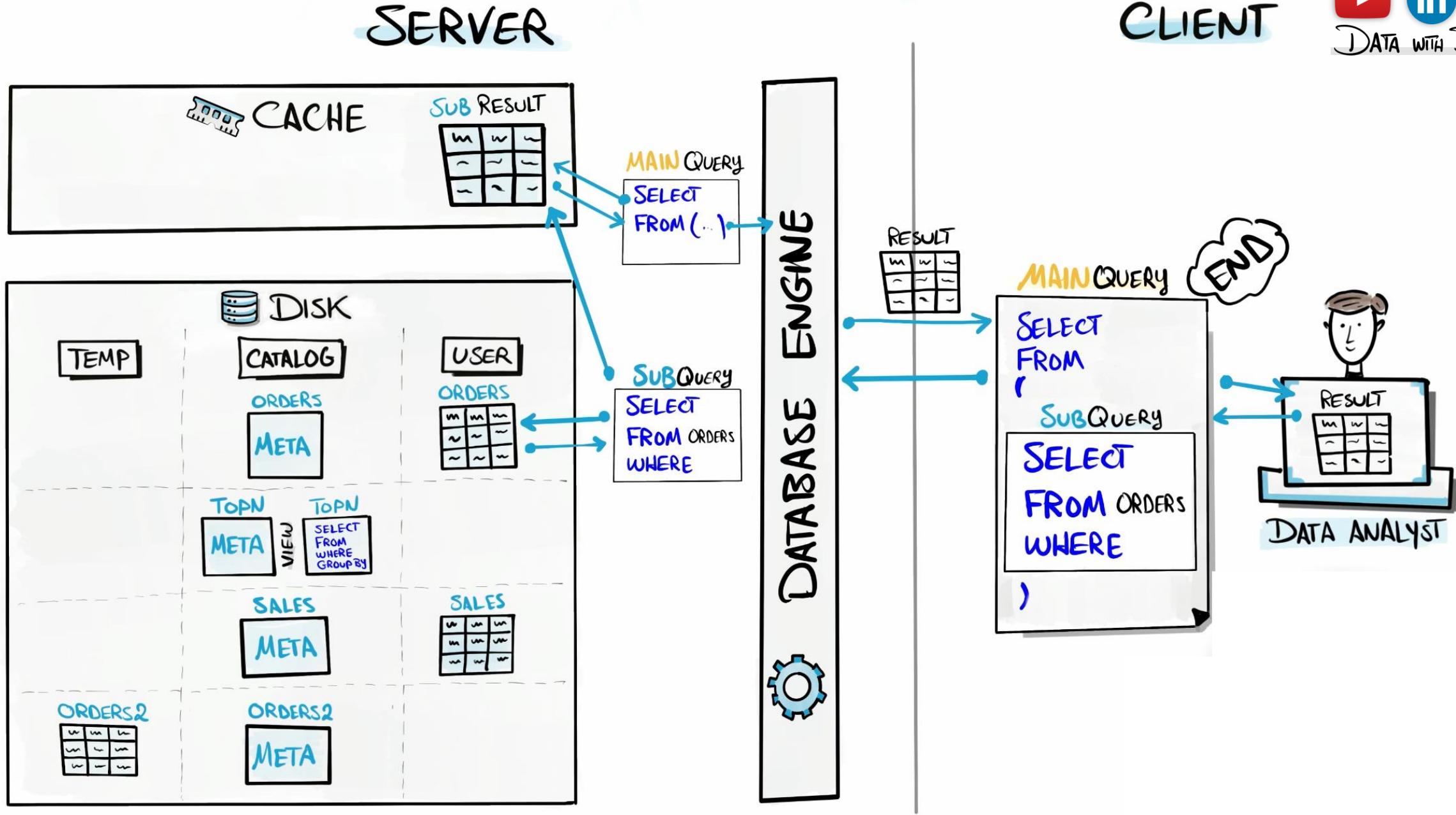
| | | |
|---|---|---|
| ~ | ~ | ~ |
| ~ | ~ | ~ |
| ~ | ~ | ~ |



SUBQUERY







Subquery in FROM Clause

```
SELECT column1, column2,...  
FROM ( SELECT column FROM table1 WHERE condition ) AS alias
```

Main Query

Subquery

Subquery in FROM Clause

| Orders | | |
|---------|------------|-------|
| OrderID | CustomerID | Sales |
| 1 | 2 | 10 |
| 2 | 3 | 15 |
| 3 | 1 | 20 |
| 4 | 1 | 60 |
| 5 | 2 | 25 |
| 6 | 3 | 50 |
| 7 | 1 | 60 |
| 8 | 4 | 90 |
| 9 | 2 | 20 |
| 10 | 3 | 60 |

Subquery

```
SELECT
    CustomerID,
    SUM(Sales) TotalSales
FROM Sales.Orders
GROUP BY CustomerID
```

| CustomerID | TotalSales |
|------------|------------|
| 1 | 110 |
| 2 | 55 |
| 3 | 125 |
| 4 | 90 |

Original Query

```
SELECT
    *,
    RANK() OVER (ORDER BY TotalSales DESC) CustomerRank
FROM
    (SELECT
        CustomerID,
        SUM(Sales) TotalSales
    FROM Sales.Orders
    GROUP BY CustomerID)t
```

Main Query

```
SELECT
    *,
    RANK() OVER (ORDER BY TotalSales DESC) R
FROM
```

| CustomerID | TotalSales | R |
|------------|------------|---|
| 3 | 125 | 1 |
| 1 | 110 | 2 |
| 4 | 90 | 3 |
| 2 | 55 | 4 |

FINAL
RESULT

Subquery in SELECT Clause

```
SELECT  
    Column1,  
    ( SELECT column FROM table1 WHERE condition ) AS alias  
FROM table1
```

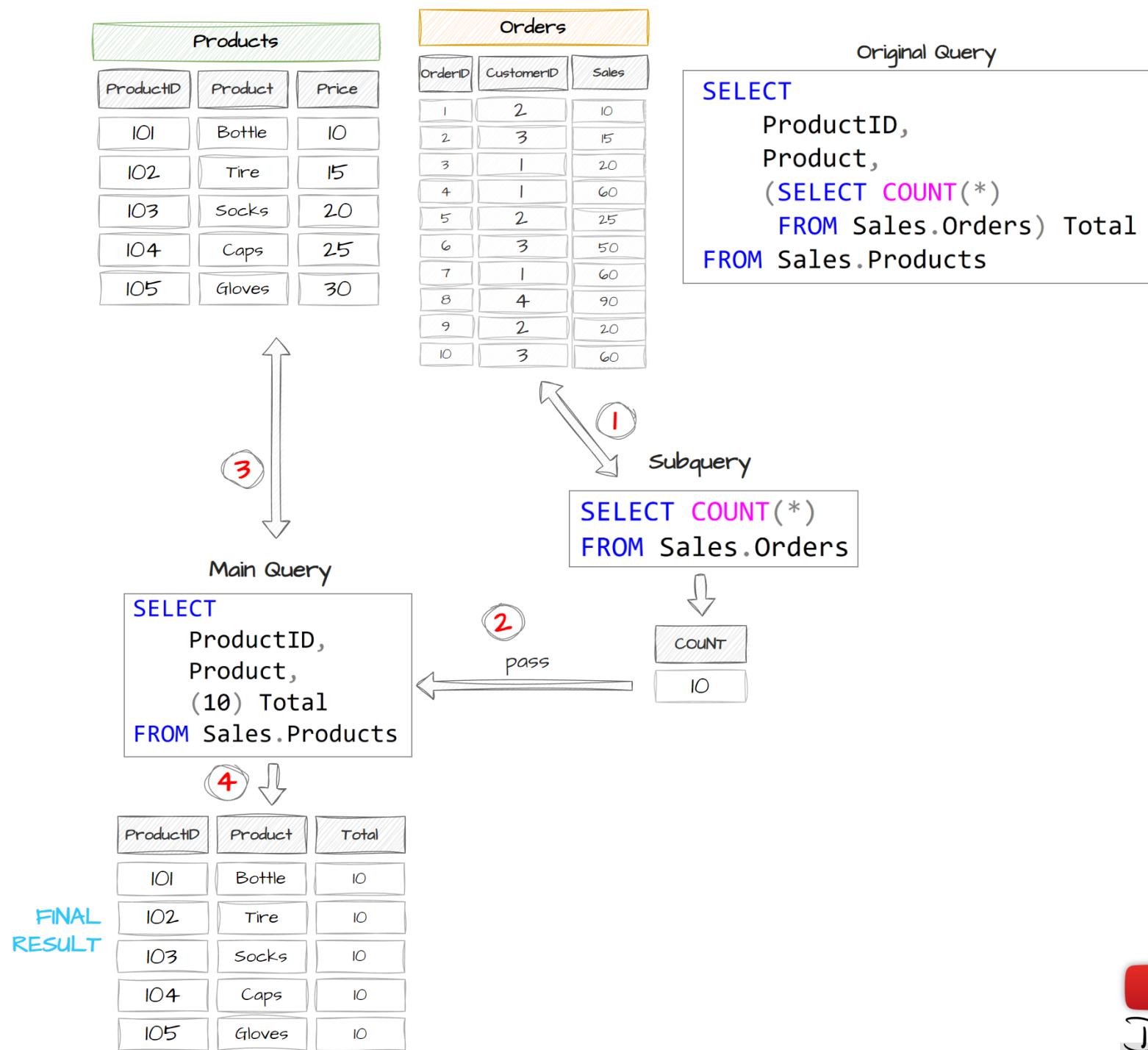
Main Query

Subquery

Rules

Only Scalar Subqueries are allowed to be used

Subquery in SELECT Clause



Subquery in WHERE Clause Comparison Operators

```
SELECT column1, column2, ...
FROM   table1
WHERE  column = ( SELECT column FROM table2 WHERE condition )
```

Main Query

Rules

Only Scalar Subqueries are allowed to be used

Subquery

Subquery in WHERE Clause

Comparison Operators

| | | |
|----|--------------------------|--|
| = | Equal | <code>WHERE Sales = (SELECT AVG(Sales) FROM ORDERS)</code> |
| != | Not Equal | <code>WHERE Sales != (SELECT AVG(Sales) FROM ORDERS)</code> |
| > | Greater than | <code>WHERE Sales > (SELECT AVG(Sales) FROM ORDERS)</code> |
| < | Less than | <code>WHERE Sales < (SELECT AVG(Sales) FROM ORDERS)</code> |
| >= | Greater than or equal to | <code>WHERE Sales >= (SELECT AVG(Sales) FROM ORDERS)</code> |
| <= | Less than or equal to | <code>WHERE Sales <= (SELECT AVG(Sales) FROM ORDERS)</code> |

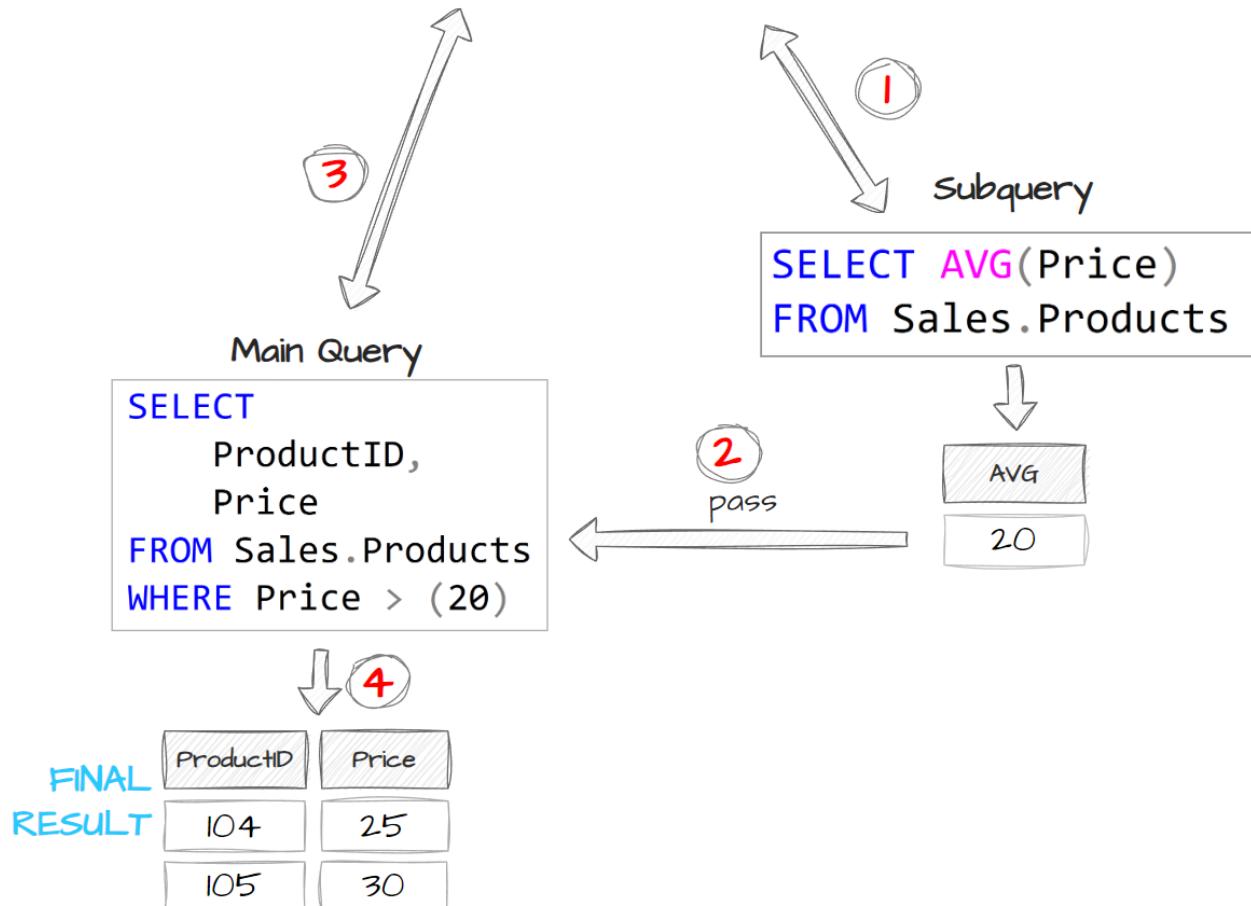


Subquery in
WHERE Clause
Comparison Operators

| Products | | | |
|-----------|---------|-------------|-------|
| ProductID | Product | Category | Price |
| 101 | Bottle | Accessories | 10 |
| 102 | Tire | Accessories | 15 |
| 103 | Socks | Clothing | 20 |
| 104 | Caps | Clothing | 25 |
| 105 | Gloves | Clothing | 30 |

Original Query

```
SELECT ProductID, Price
FROM Sales.Products
WHERE Price > (SELECT AVG(Price)
                FROM Sales.Products)
```



Subquery in WHERE Clause In Operator

Main Query

```
SELECT column1, column2, ...
FROM    table1
WHERE   column IN ( SELECT column FROM table2 WHERE condition )
```

Subquery

Logical Operators

IN

Checks if a value matches any value in a list

`WHERE Sales IN (SELECT ...)`

NOT IN

Checks if a value does not matches any value in a list

`WHERE Sales NOT IN (SELECT ...)`

EXISTS

Checks if subquery returns any rows

`WHERE EXISTS (SELECT ...)`

NOT EXISTS

Checks if subquery returns no rows

`WHERE NOT EXISTS (SELECT ...)`

ANY

Returns true if a value matches any value in a list.

`WHERE Sales < ANY (SELECT ...)`

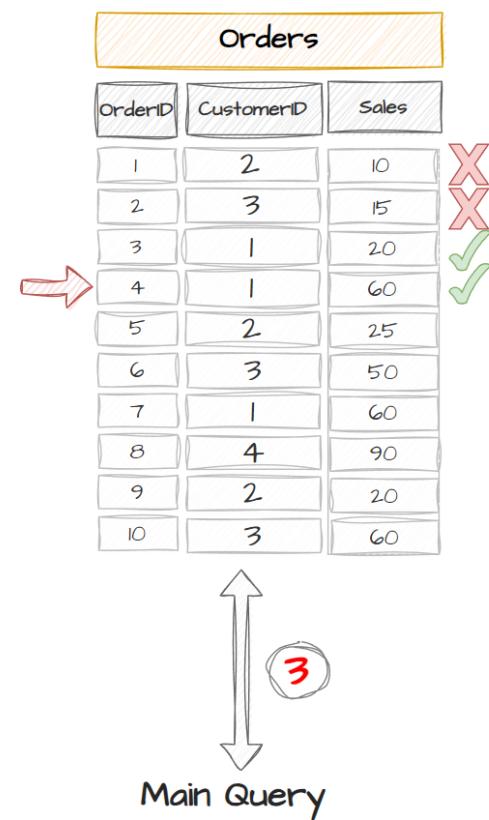
ALL

Returns true if a value matches all values in a list.

`WHERE Sales > ALL (SELECT ...)`



Subquery in WHERE Clause In Operator

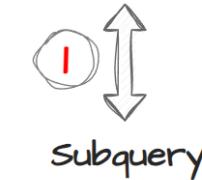


Customers

| CustomerID | FirstName | LastName | Country | Score |
|------------|-----------|----------|---------|-------|
| 1 | Jossef | Goldberg | Germany | 350 |
| 2 | Kevin | Brown | USA | 900 |
| 3 | Mary | NULL | USA | 750 |
| 4 | Mark | Schwarz | Germany | 500 |
| 5 | Anna | Adams | USA | NULL |

Original Query

```
SELECT
    OrderID,
    CustomerID,
    Sales
FROM Sales.Orders
WHERE CustomerID IN
    (SELECT CustomerID
    FROM Sales.Customers
    WHERE Country = 'Germany')
```



Subquery in
WHERE Clause
ALL Operators

Main Query

```
SELECT column1, column2,...
```

```
FROM    table1
```

```
WHERE   column < ALL( SELECT column FROM table1 WHERE condition )
```

Subquery

Subquery in
WHERE Clause
ANY Operator

Main Query

```
SELECT column1, column2,...
```

```
FROM    table1
```

```
WHERE   column < ANY( SELECT column FROM table1 WHERE condition )
```

Subquery

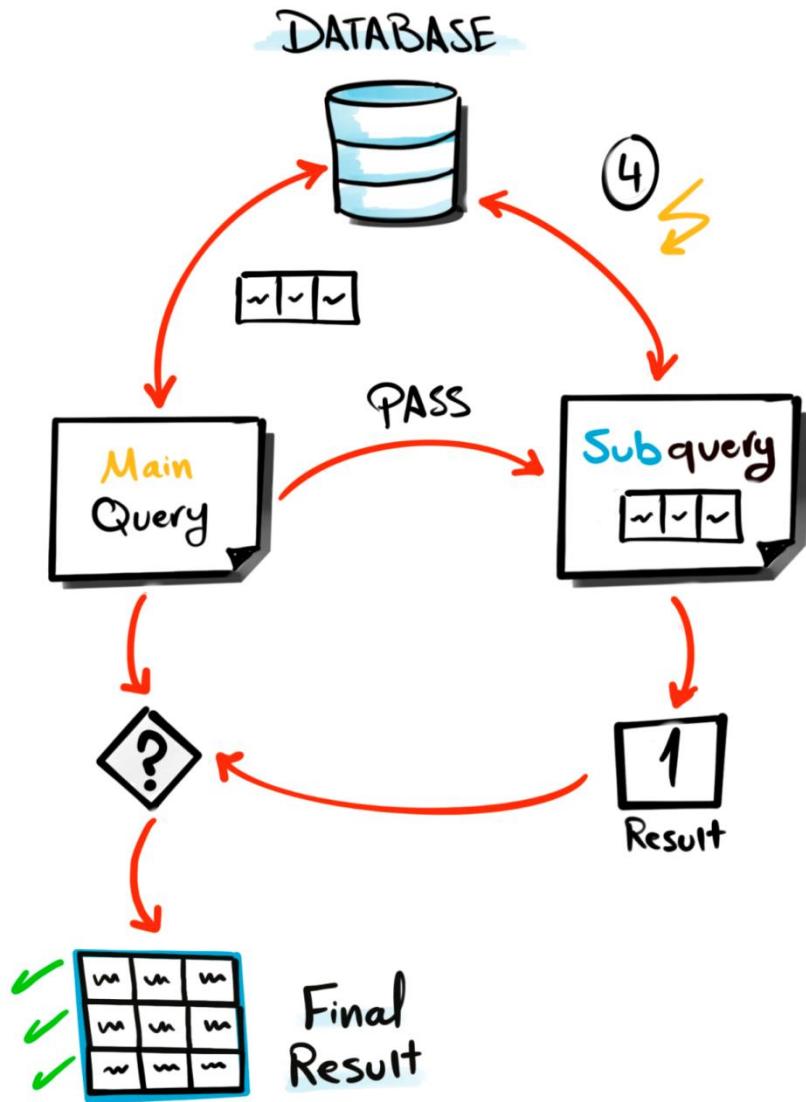
NON-CORRELATED SUBQUERY

A Subquery that can run **independently** from the Main Query

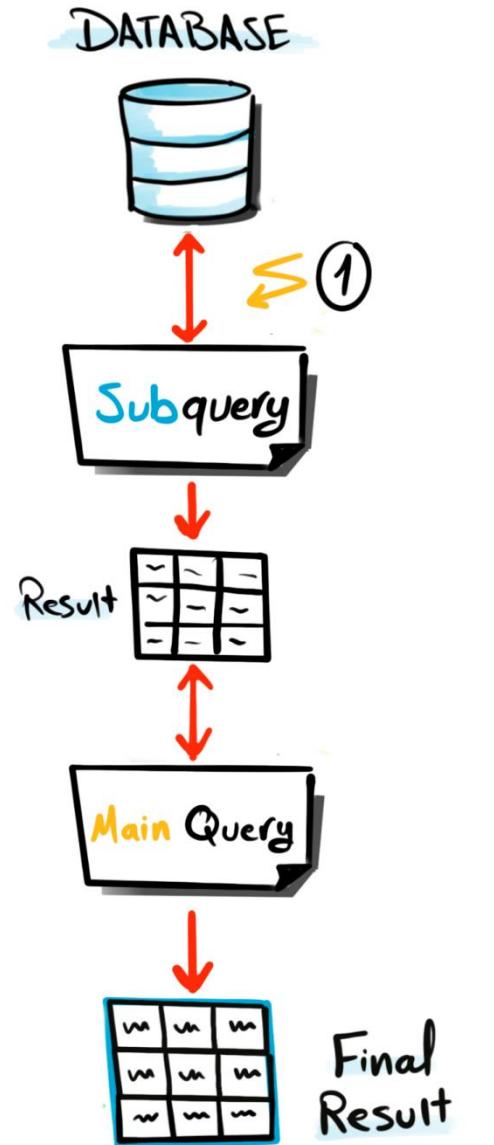
CORRELATED SUBQUERY

A Subquery that **relays** on values from the Main Query

Correlated Subquery



Non-Correlated Subquery



Non-Correlated Subquery

Definition

Subquery is **independent** of the main query

Execution

Executed **once** and its result is used by the main query

Can be executed on its Own

Easy to use

Easier to read

Executed **only once** leads to **better** Performance

Usage

Static Comparisons, Filtering with Constants

Correlated Subquery

Subquery is **dependent** of the main query

Executed for **each row** processed by the main query

Can't be executed on its Own.

Harder to read and more **complex**

Executed multiple times leads to **bad** Performance

Row-by-Row Comparisons, Dynamic Filtering

Correlated Subquery in
WHERE Clause
EXISTS Operator

Main Query

```
SELECT column1, column2,...
```

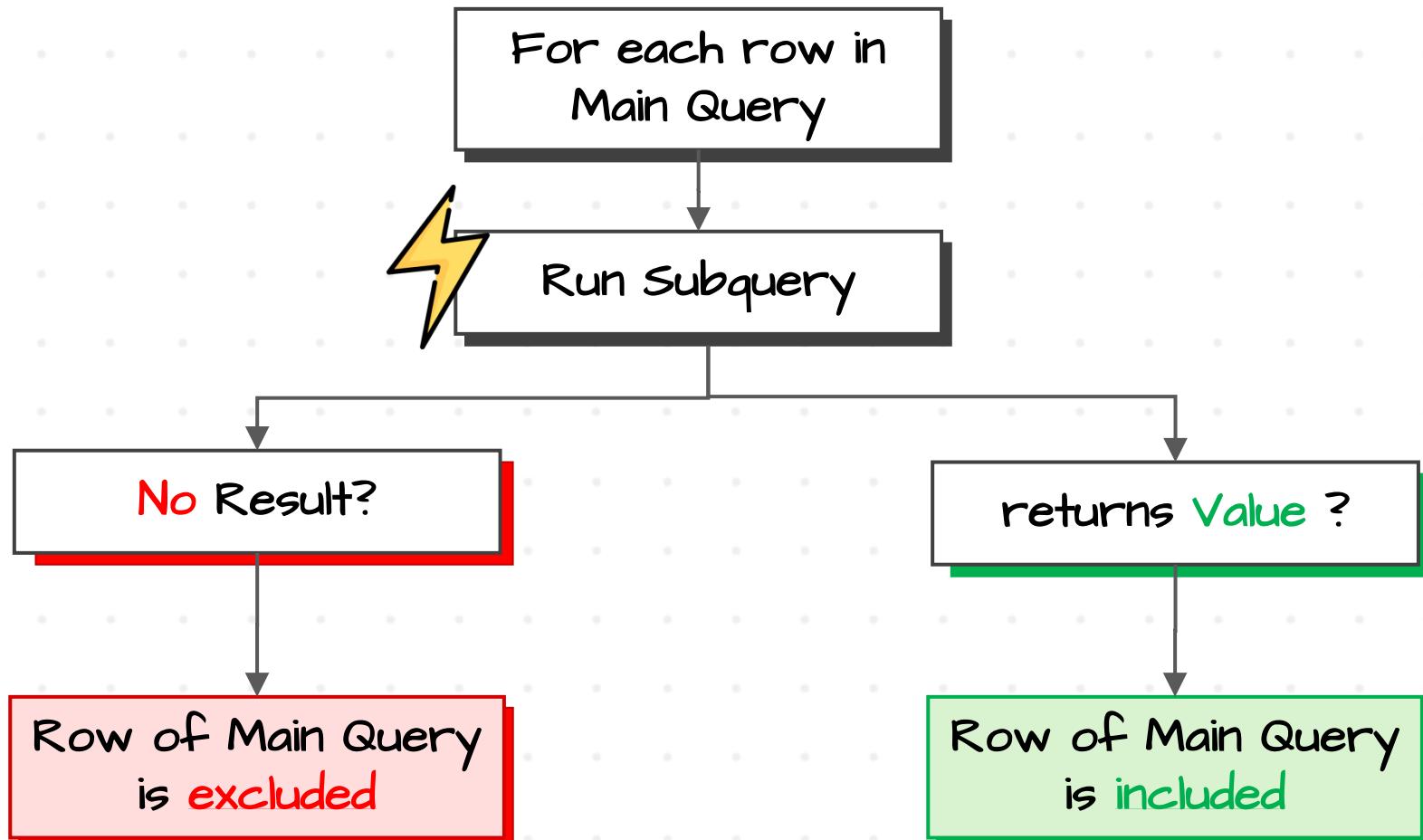
```
FROM Table2
```

```
WHERE EXISTS (
```

```
    SELECT 1  
  
    FROM Table1  
  
    WHERE Table1.ID = Table2.ID
```

Subquery

How EXISTS Works?



JOINS

Syntax

```
SELECT o.*  
FROM Orders o  
JOIN Customers c  
ON c.CustomerID = o.CustomerID  
AND c.Country = 'USA'
```

Readability

Not easy to read & maintain

Performance

Fast

Duplicate

May **lead to duplicate**

Best Practices

Useful with **Larg tables**

SUBQUERIES

```
SELECT *  
FROM Orders  
WHERE CustomerID IN  
(SELECT CustomerID  
FROM Customers  
WHERE Country = 'USA')
```

easy to read & maintain

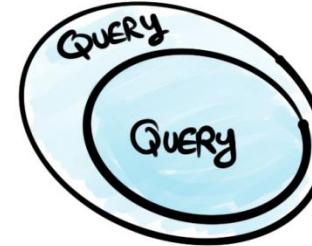
Slow

Safer
no risk to have duplicates

Useful with **small tables**

SUBQUERY

Query inside Another Query



Breaks Complex query into Smaller, manageable pieces.

USE CASES

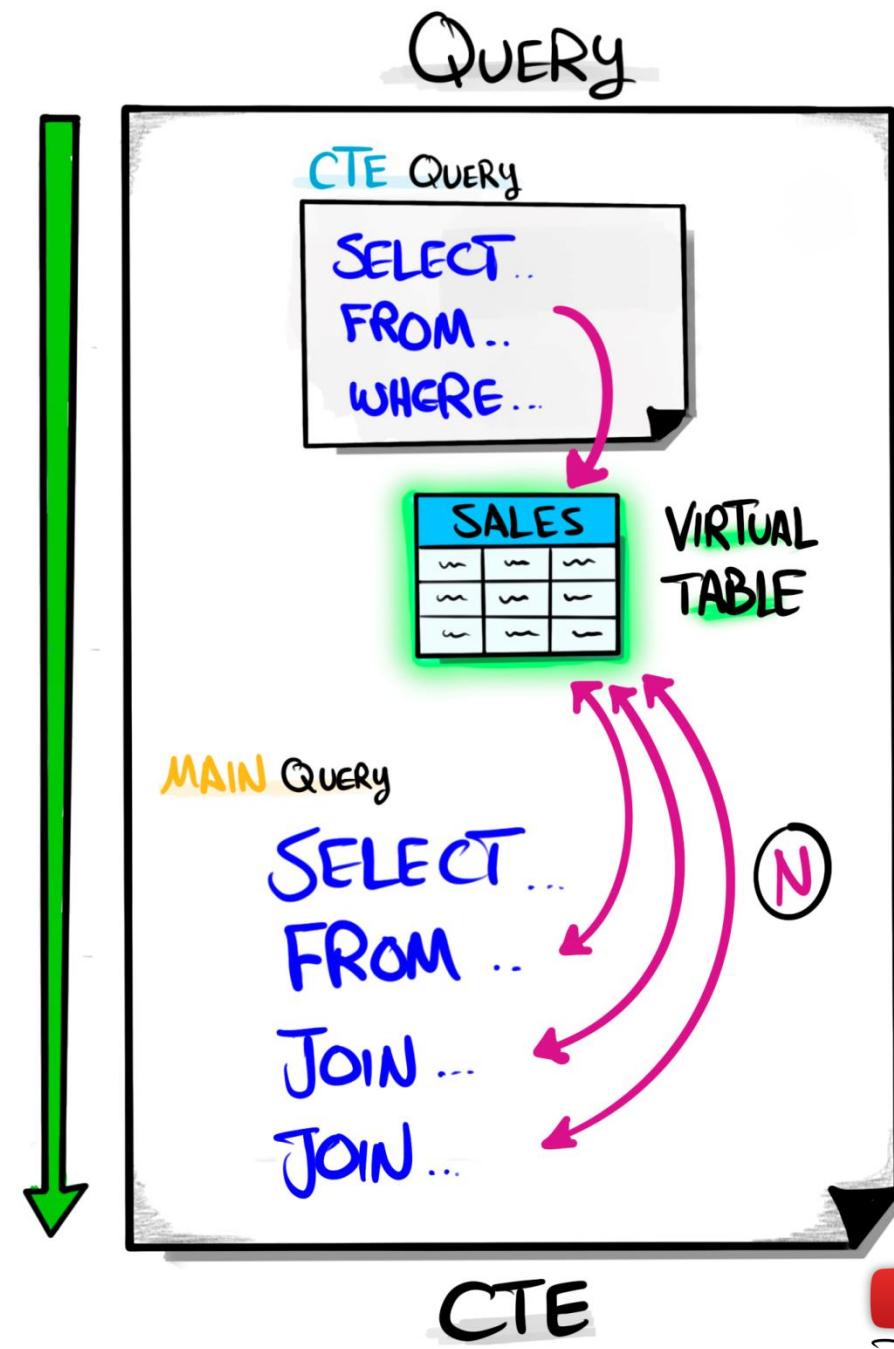
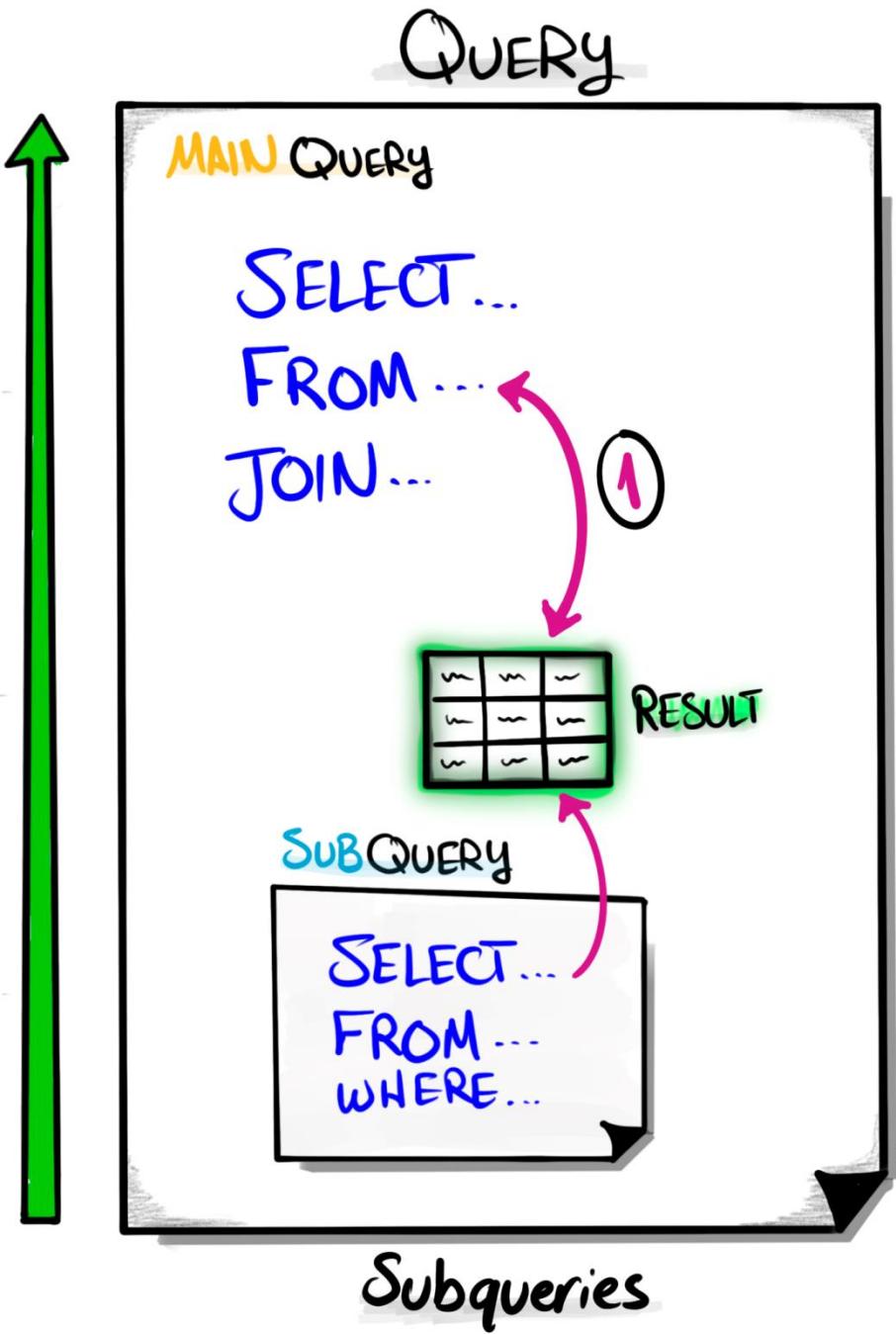
- Create Temporary Result Set.
- Prepare Data Before Joining Tables.
- Dynamic & Complex Filtering.
- Check the Existence of Rows From another Table. (**EXISTS**)
- Row By Row Comparison - Correlated Subquery -



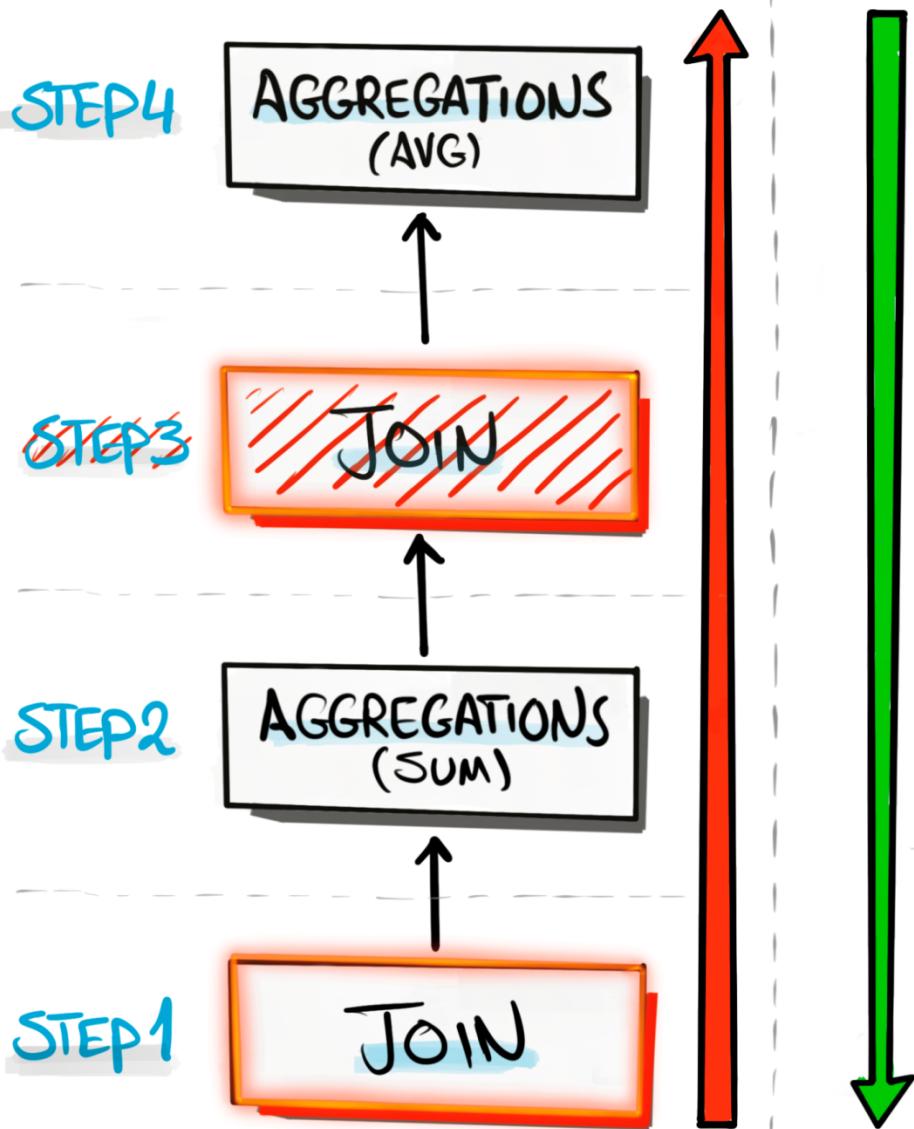
CTE

Common Table Expression

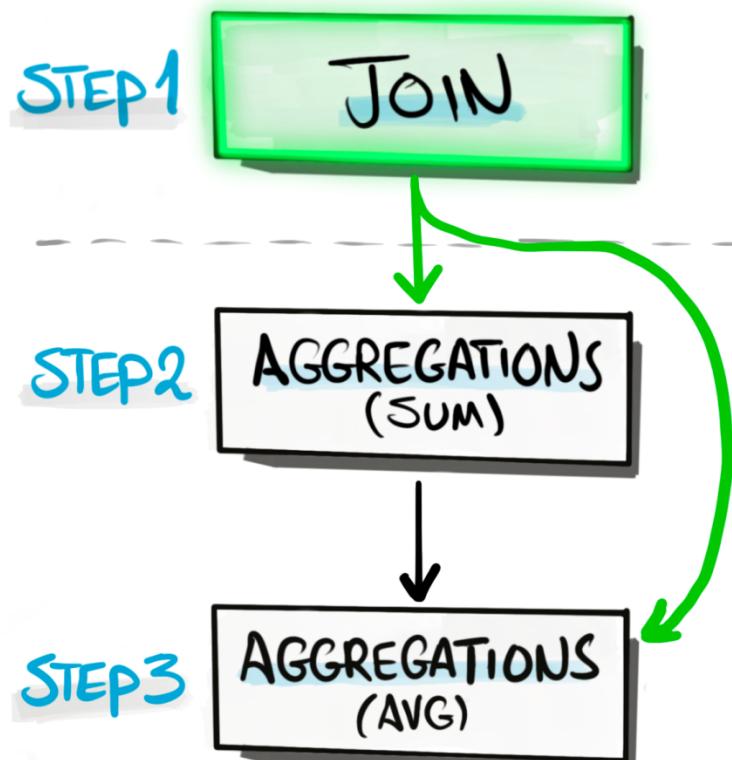




SUBQUERY



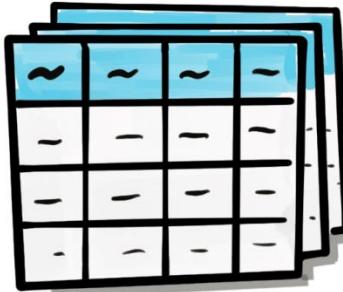
CTE



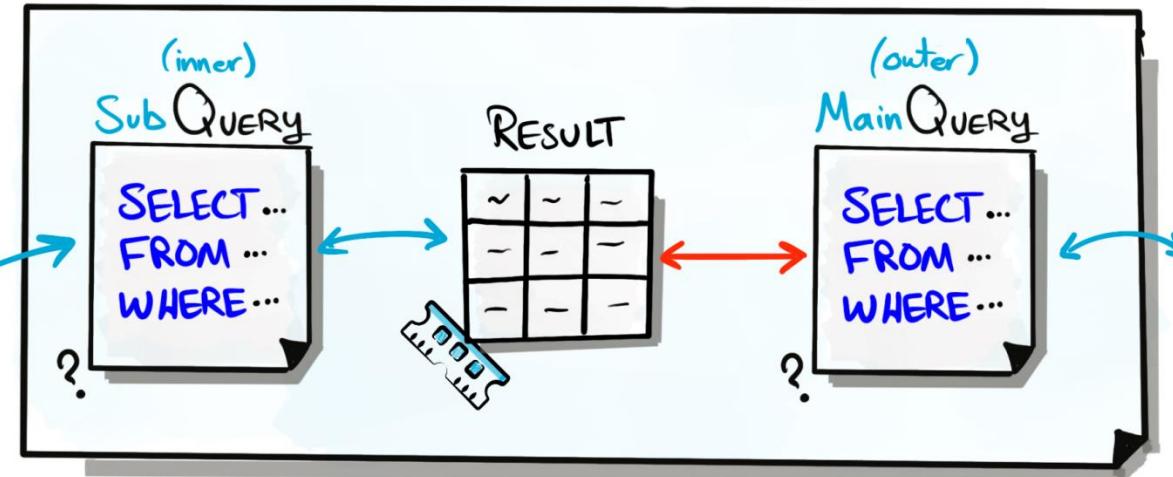


DATABASE

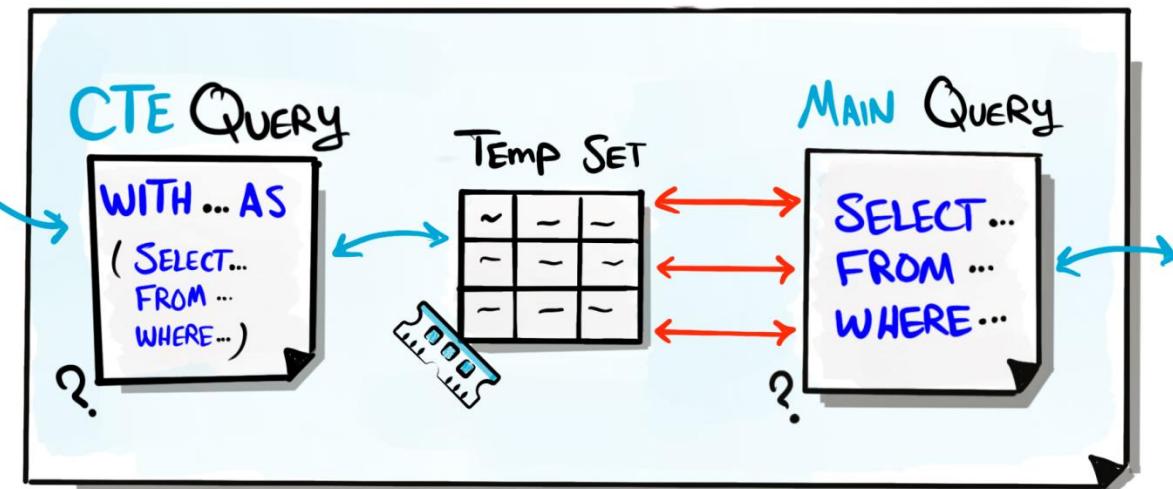
TABLES

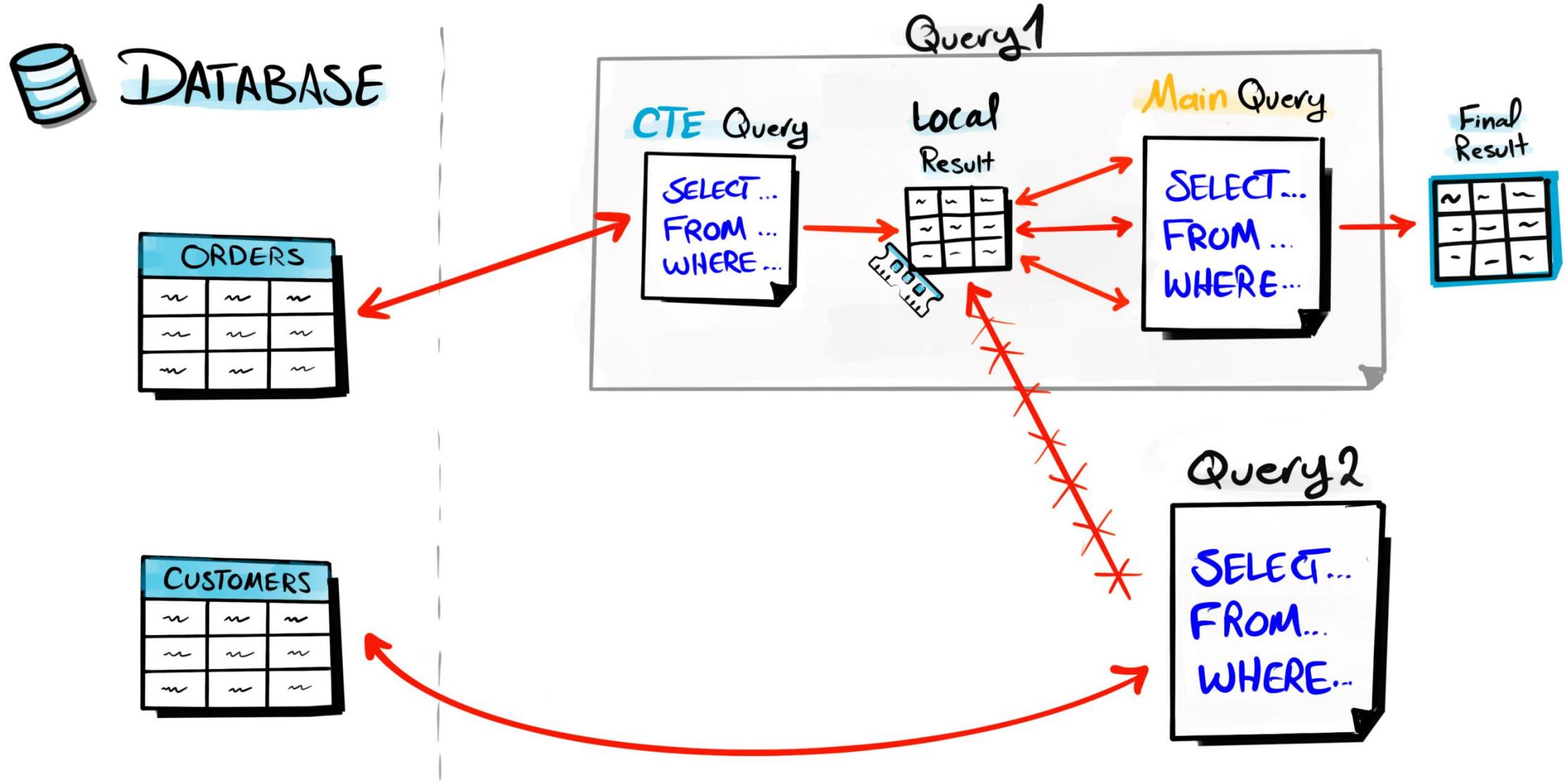


SQL SUBQUERY



SQL CTE



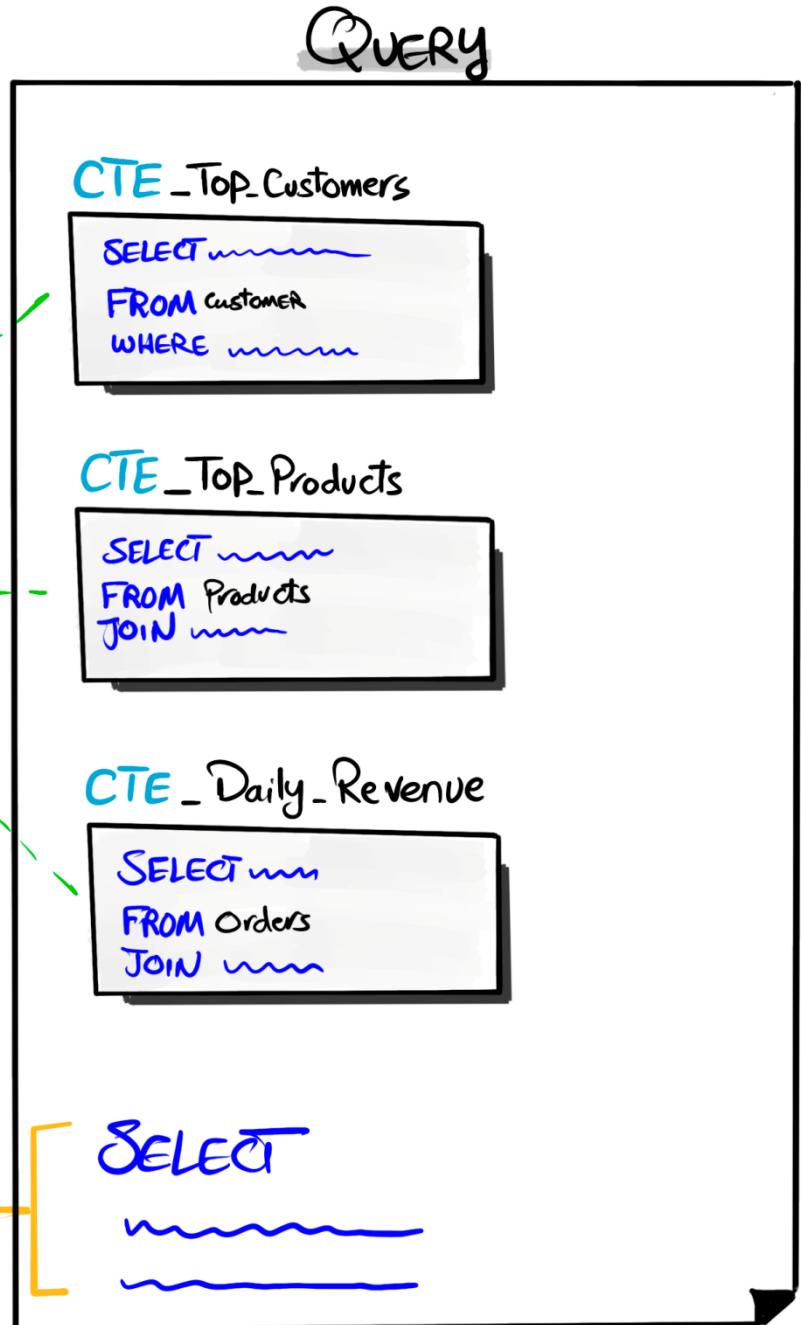


b-d **READABILITY**

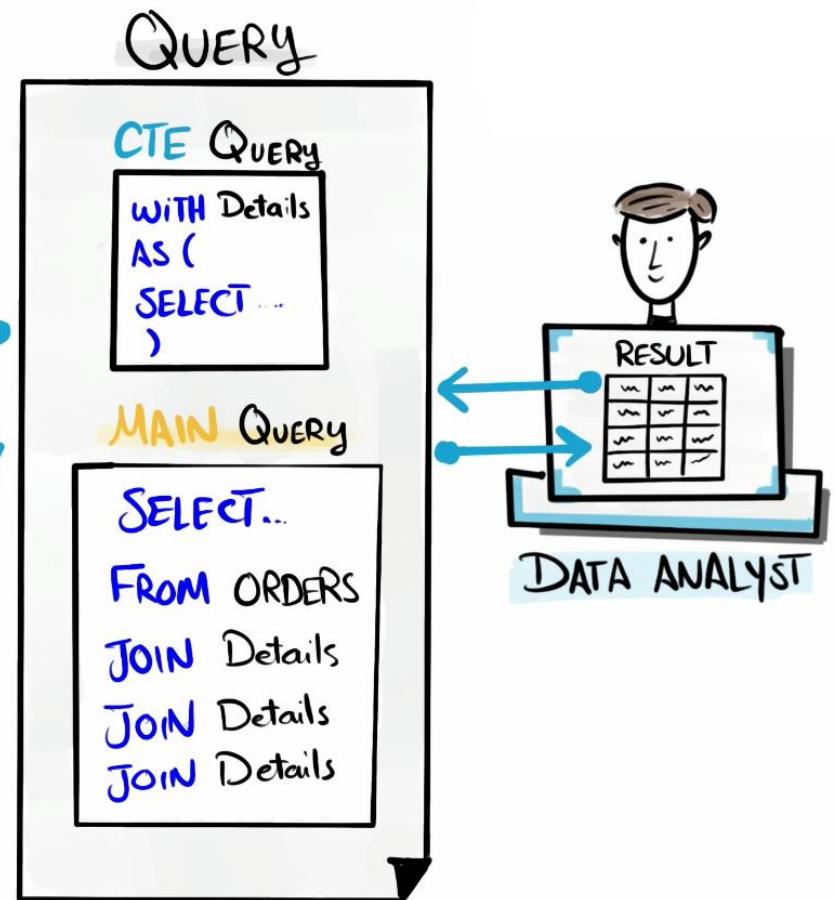
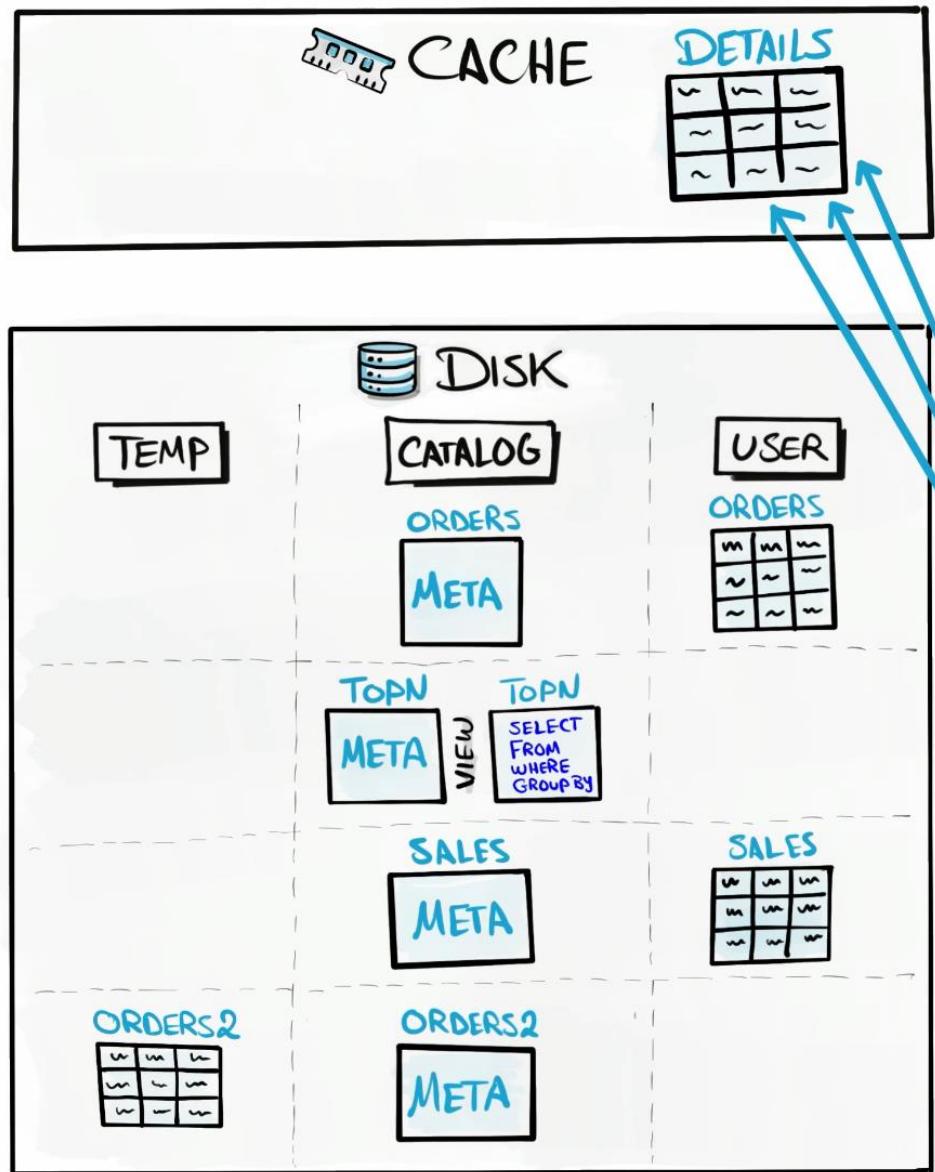
 **MODULARITY**

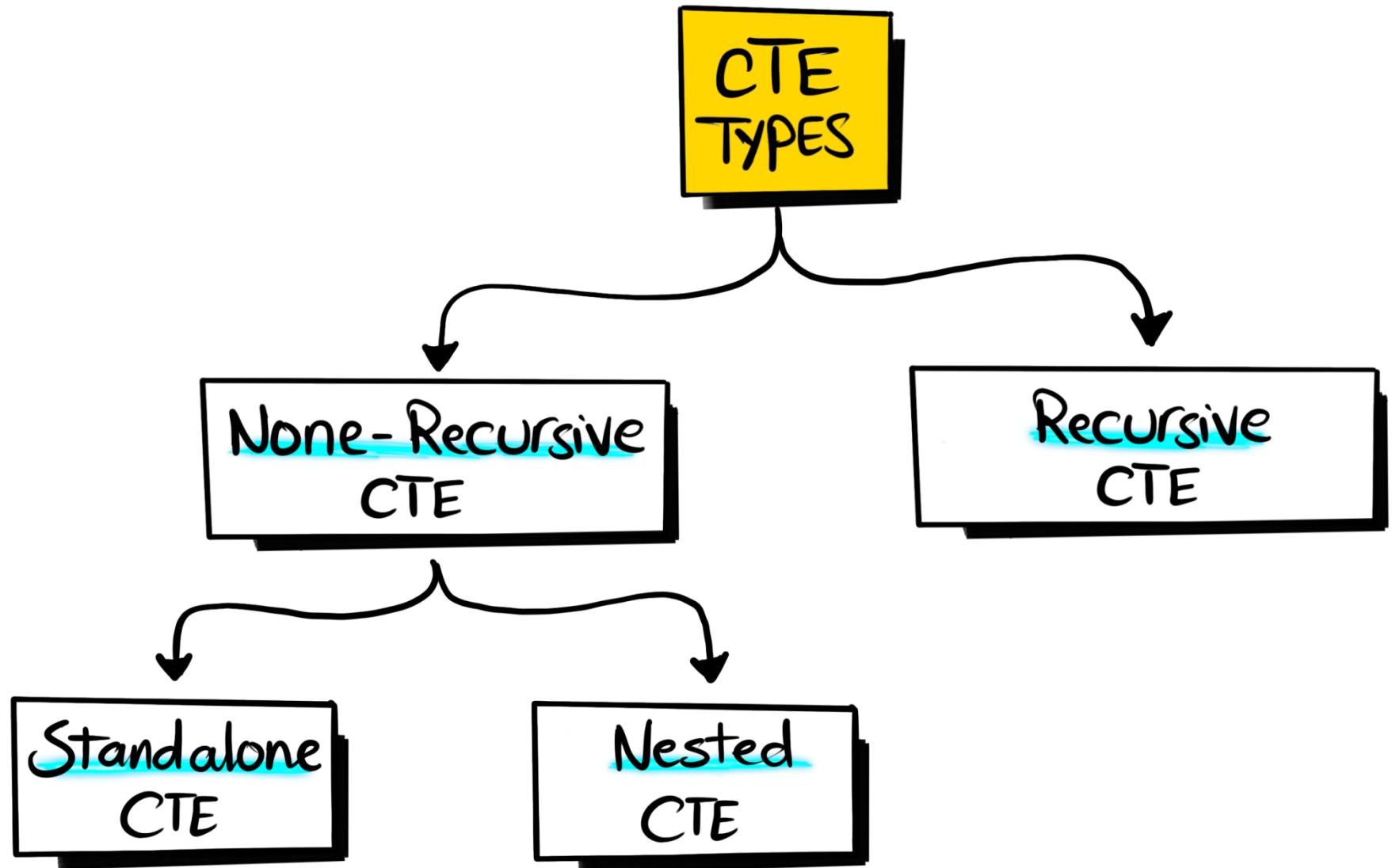
 **REUSABILITY**

**MAIN
QUERY**



SERVER





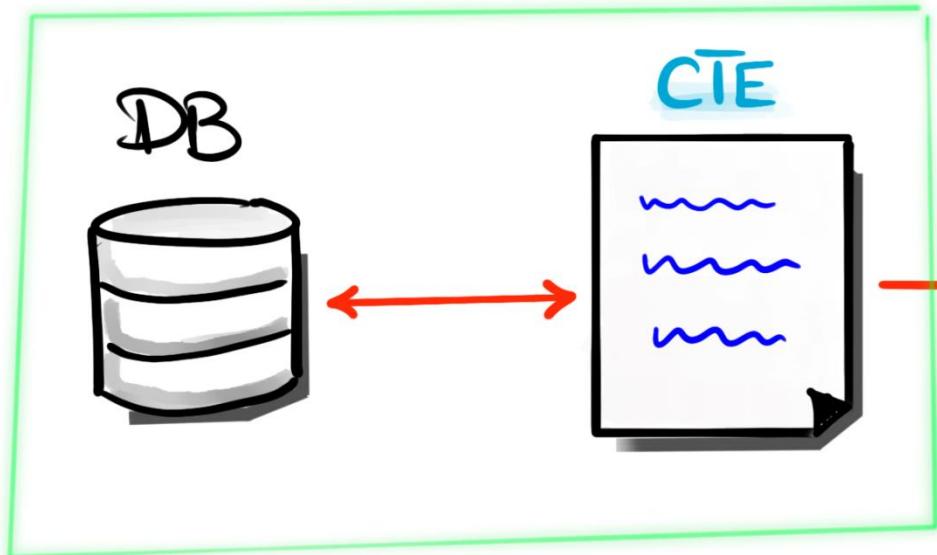
Standalone CTE

Defined and Used independently.

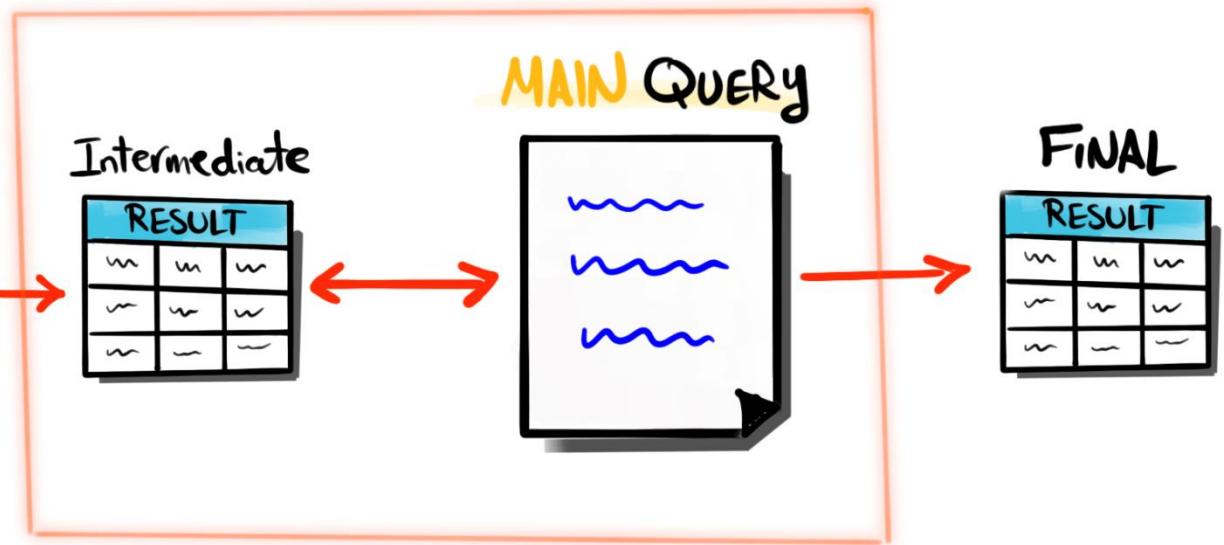
Runs independently as it's self-contained
and doesn't rely on other CTEs or queries.

Standalone CTE

Independent



Dependent



Standalone CTE

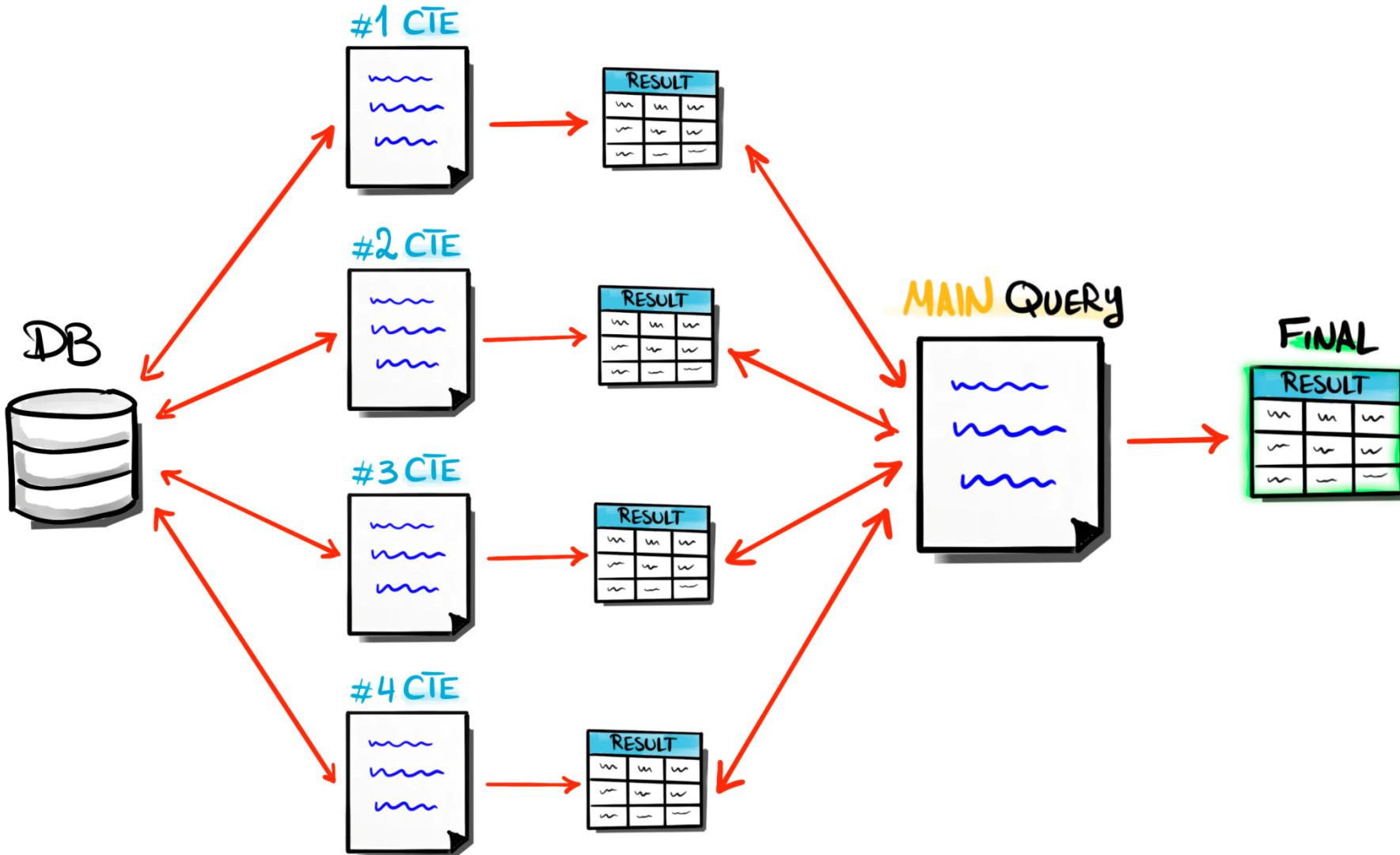
```
WITH CTE-Name AS  
(  
    SELECT ...  
    FROM ...  
    WHERE ...  
)
```

CTE Query
- CTE Definition -

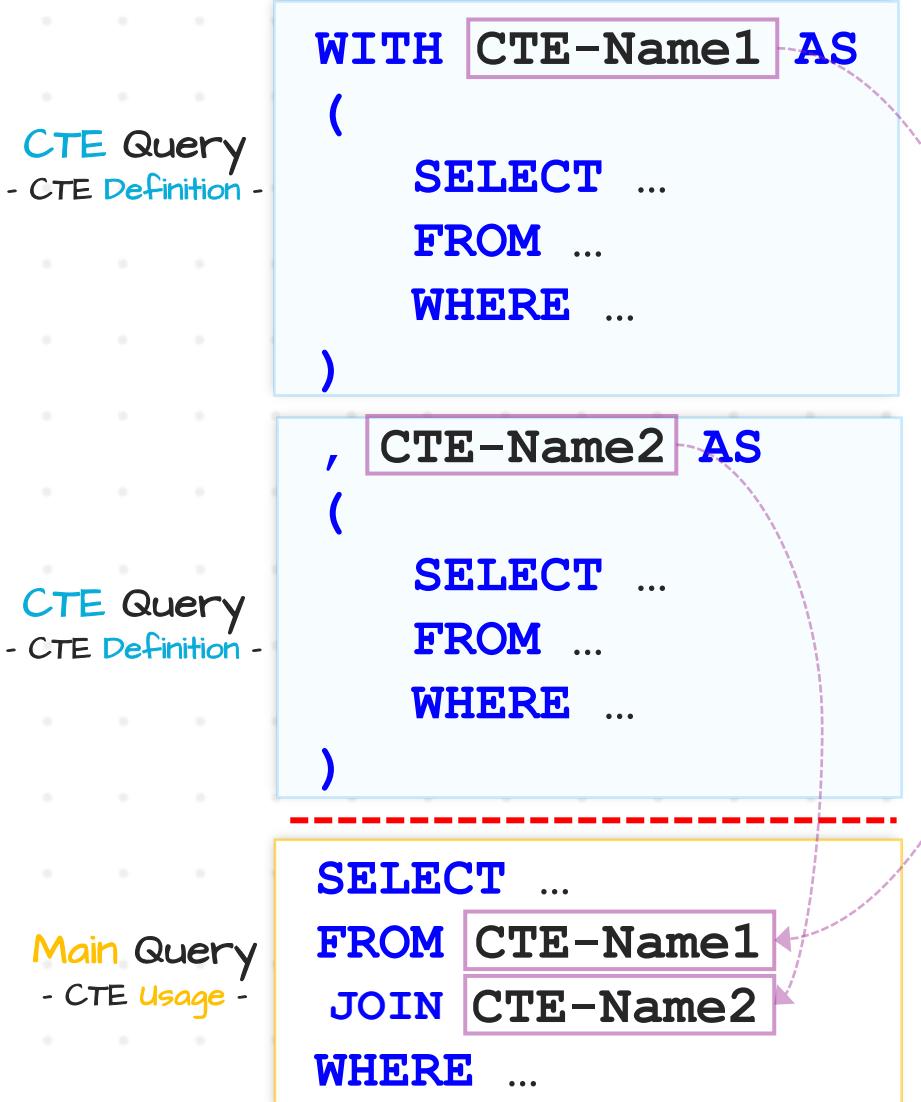
```
SELECT ...  
FROM CTE-Name  
WHERE ...
```

Main Query
- CTE Usage -

Multiple CTEs



Multiple CTEs

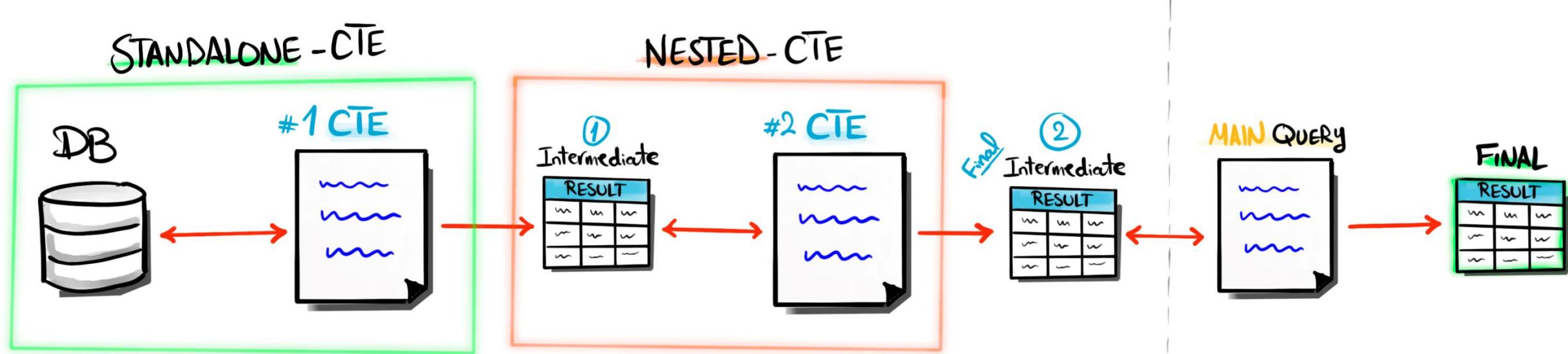


Nested CTE

CTE inside another CTE

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

Nested CTEs



Nested CTEs

Standalone CTE

```
WITH CTE-Name1 AS
(
  SELECT ...
  FROM ...
  WHERE ...
)
```

CTE Query
- CTE Definition -

NESTED CTE

```
, CTE-Name2 AS
(
  SELECT ...
  FROM CTE-Name1
  WHERE ...
)
```

CTE Query
- CTE Definition -

```
SELECT ...
FROM CTE-Name2
WHERE ...
```

Main Query
- CTE Usage -

BEST PRACTICE

**Don't use more than 5 CTEs in one query; otherwise,
your code will be hard to understand and maintain.**

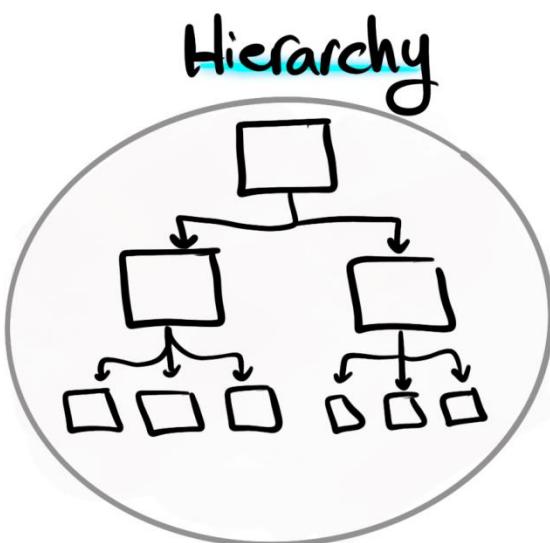
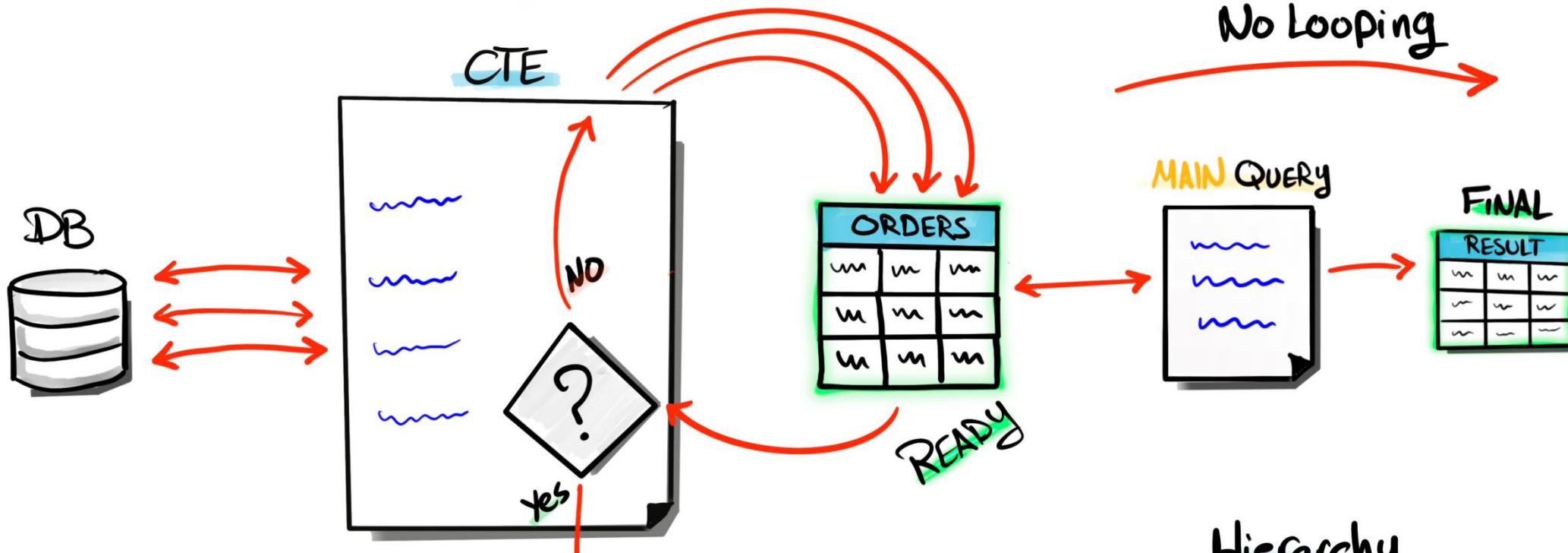
Non-Recursive CTE

is executed only once without any repetition.

Recursive CTE

Self-referencing query that repeatedly processes
data until a specific condition is met.

Recursive CTE



Recursive CTE

CTE Query
- CTE Definition -

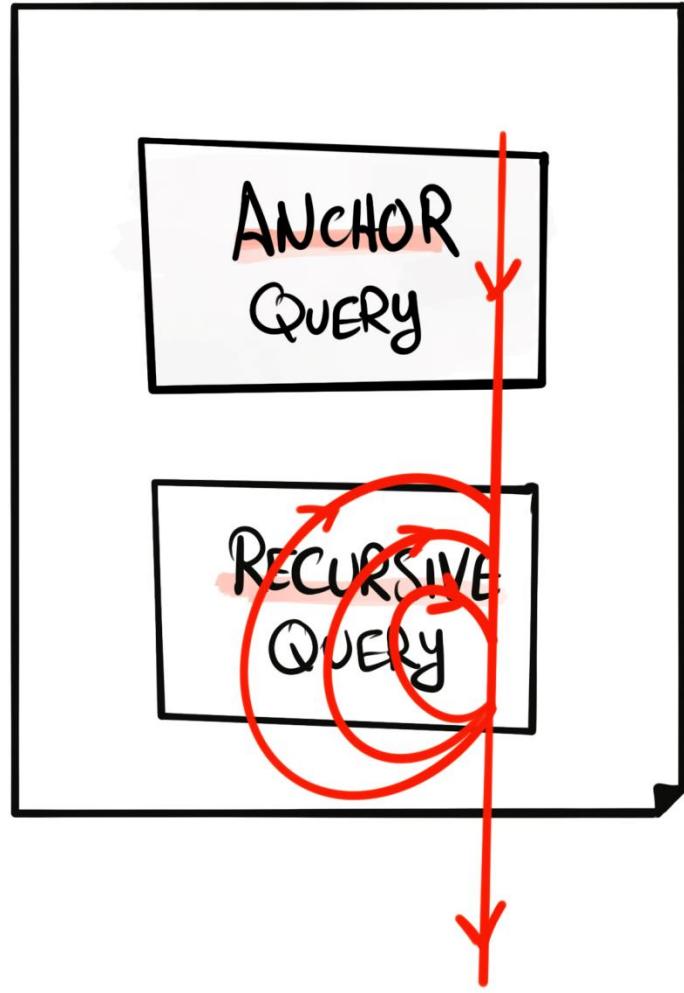
```
WITH CTE-Name AS
(
    SELECT ...
    FROM ...
    WHERE ...
    UNION ALL
    SELECT ...
    FROM CTE-Name
    WHERE [Break Condition]
)
SELECT ...
FROM CTE-Name
WHERE ...
```

Anchor Query

Recursive Query

Main Query
- CTE Usage -

CTE Recursive



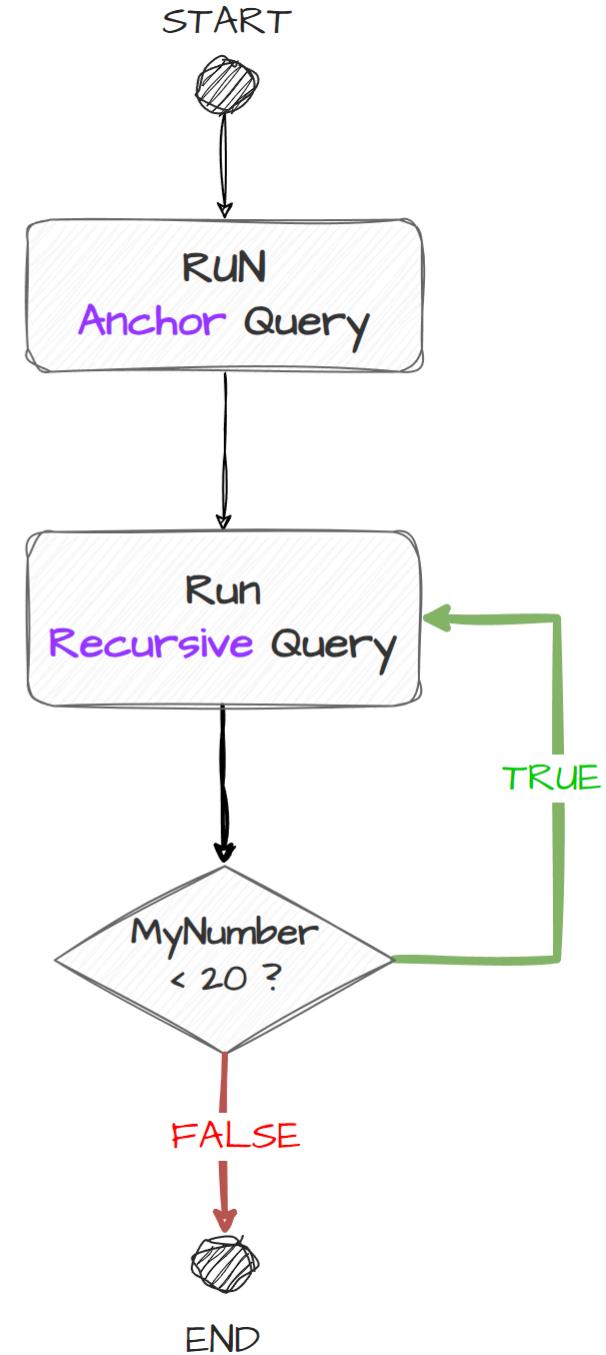
```
WITH Series AS (
    SELECT
        1 AS MyNumber
    UNION ALL
    SELECT
        MyNumber + 1
    FROM Series
    WHERE MyNumber < 20
)
SELECT *
FROM Series
```

Anchor Query

Recursive Query

| MyNumber |
|----------|
| 1 |
| 2 |
| 3 |
| ... |
| 20 |

Recursive Query



```

WITH CTE_Emp_Hierarchy AS
(
    SELECT
        EmployeeID,
        FirstName,
        ManagerID,
        1 AS Level
    FROM Sales.Employees
    WHERE ManagerID IS NULL

    UNION ALL

    SELECT
        e.EmployeeID,
        e.FirstName,
        e.ManagerID,
        Level +1
    FROM Sales.Employees AS e
    INNER JOIN CTE_Emp_Hierarchy ceh
    ON e.ManagerID = ceh.EmployeeID
)

SELECT *
FROM CTE_Emp_Hierarchy

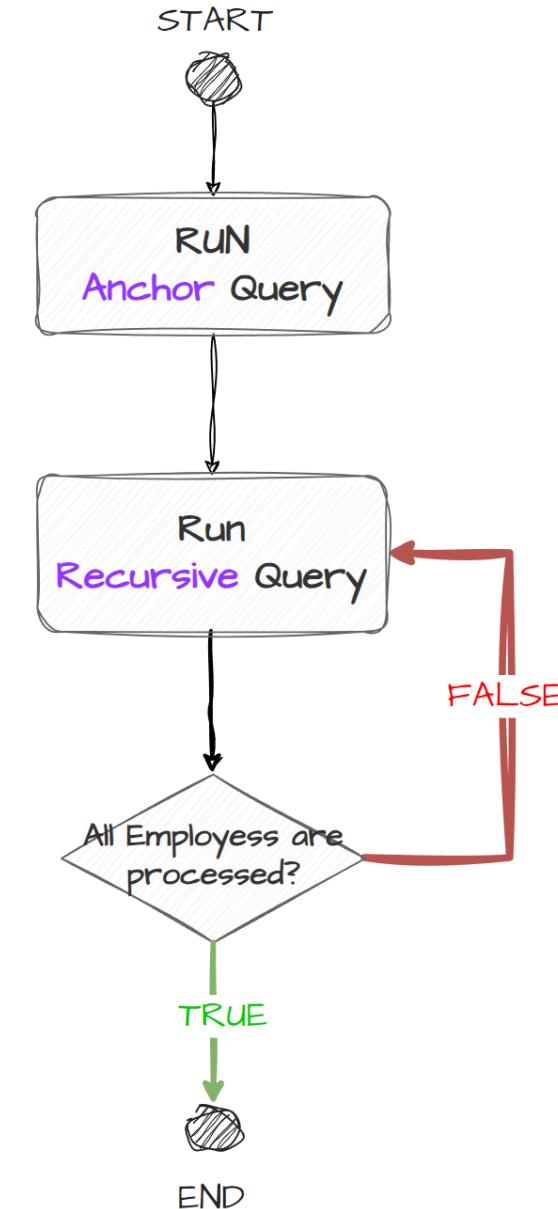
```

Anchor Query

Recursive Query

| Employees | | |
|------------|-----------|-----------|
| EmployeeID | FirstName | ManagerID |
| 1 | Frank | NULL |
| 2 | Kevin | 1 |
| 3 | Mary | 1 |
| 4 | Michael | 2 |
| 5 | Carol | 3 |

| CTE RESULT | | | |
|------------|-----------|-----------|-------|
| EmployeeID | FirstName | ManagerID | Level |
| 1 | Frank | NULL | 1 |
| 2 | Kevin | 1 | 2 |
| 3 | Mary | 1 | 2 |
| 4 | Michael | 2 | 3 |
| 5 | Carol | 3 | 3 |



```

WITH CTE_Emp_Hierarchy AS
(
    SELECT
        EmployeeID,
        FirstName,
        ManagerID,
        1 AS Level
    FROM Sales.Employees
    WHERE ManagerID IS NULL

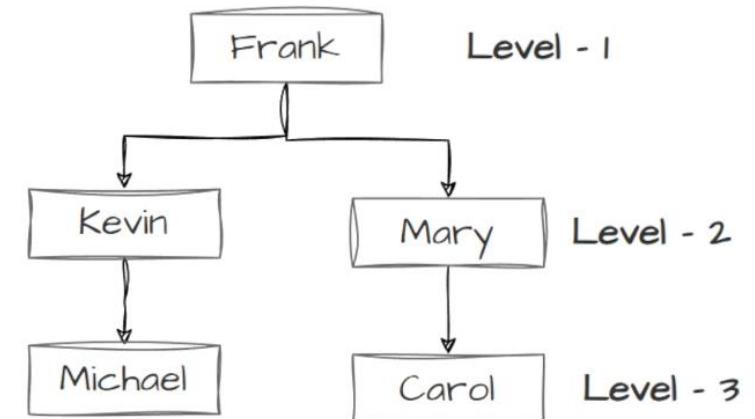
    UNION ALL

    SELECT
        e.EmployeeID,
        e.FirstName,
        e.ManagerID,
        Level +1
    FROM Sales.Employees AS e
    INNER JOIN CTE_Emp_Hierarchy ceh
    ON e.ManagerID = ceh.EmployeeID
)

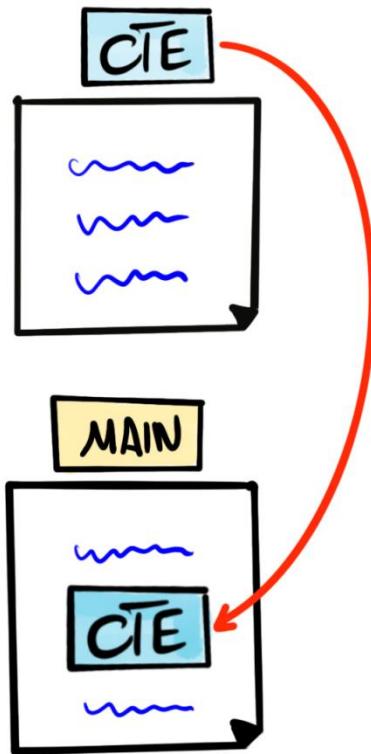
SELECT *
FROM CTE_Emp_Hierarchy

```

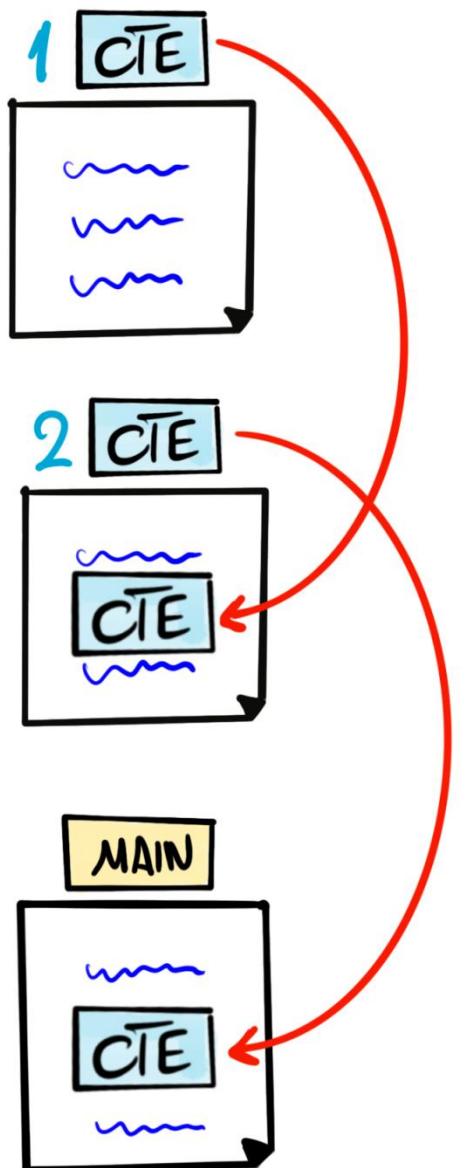
| CTE RESULT | | | |
|------------|-----------|-----------|-------|
| EmployeeID | FirstName | ManagerID | Level |
| 1 | Frank | NULL | 1 |
| 2 | Kevin | 1 | 2 |
| 3 | Mary | 1 | 2 |
| 4 | Michael | 2 | 3 |
| 5 | Carol | 3 | 3 |



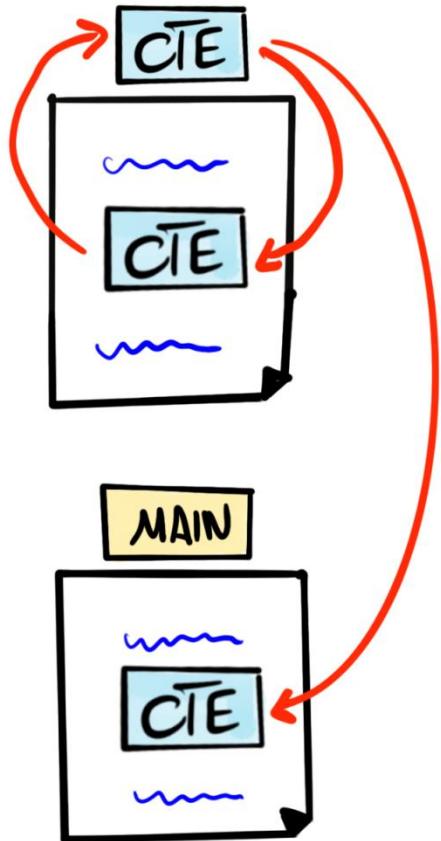
STANDALONE



NESTED



RECURSIVE





Views

Database Object

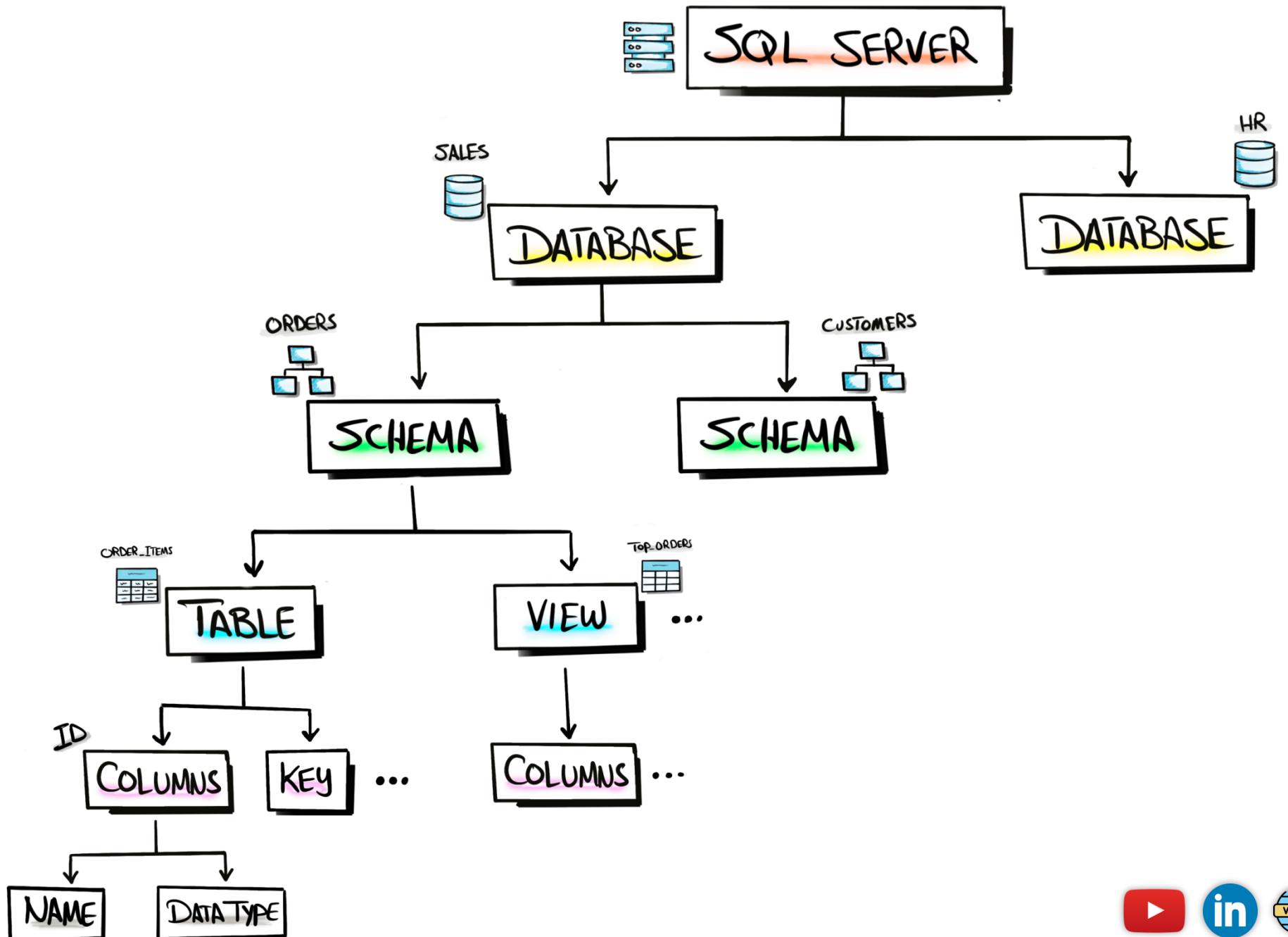
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Views

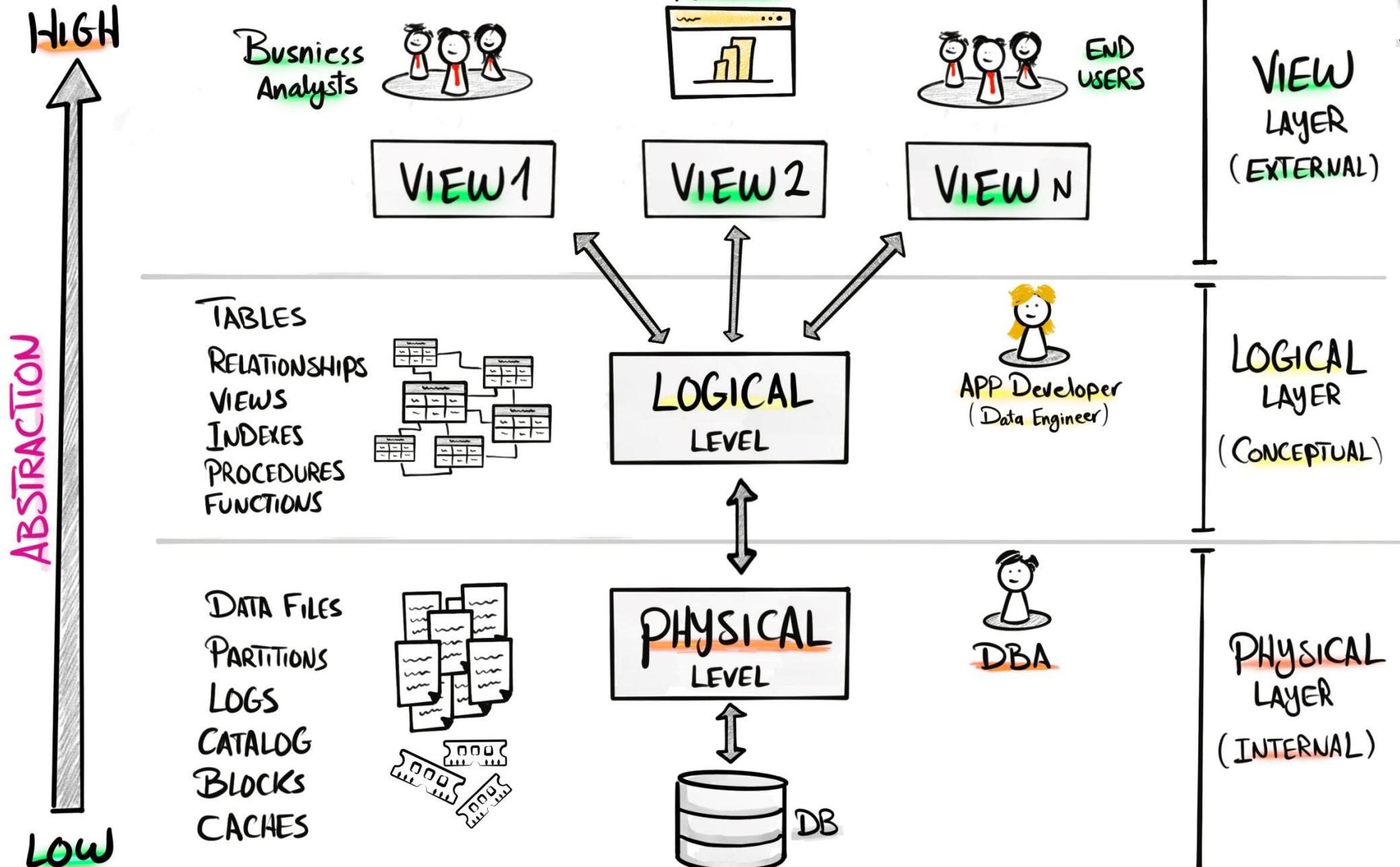


DDL

Data Definition Language

- CREATE
- ALTER
- DROP

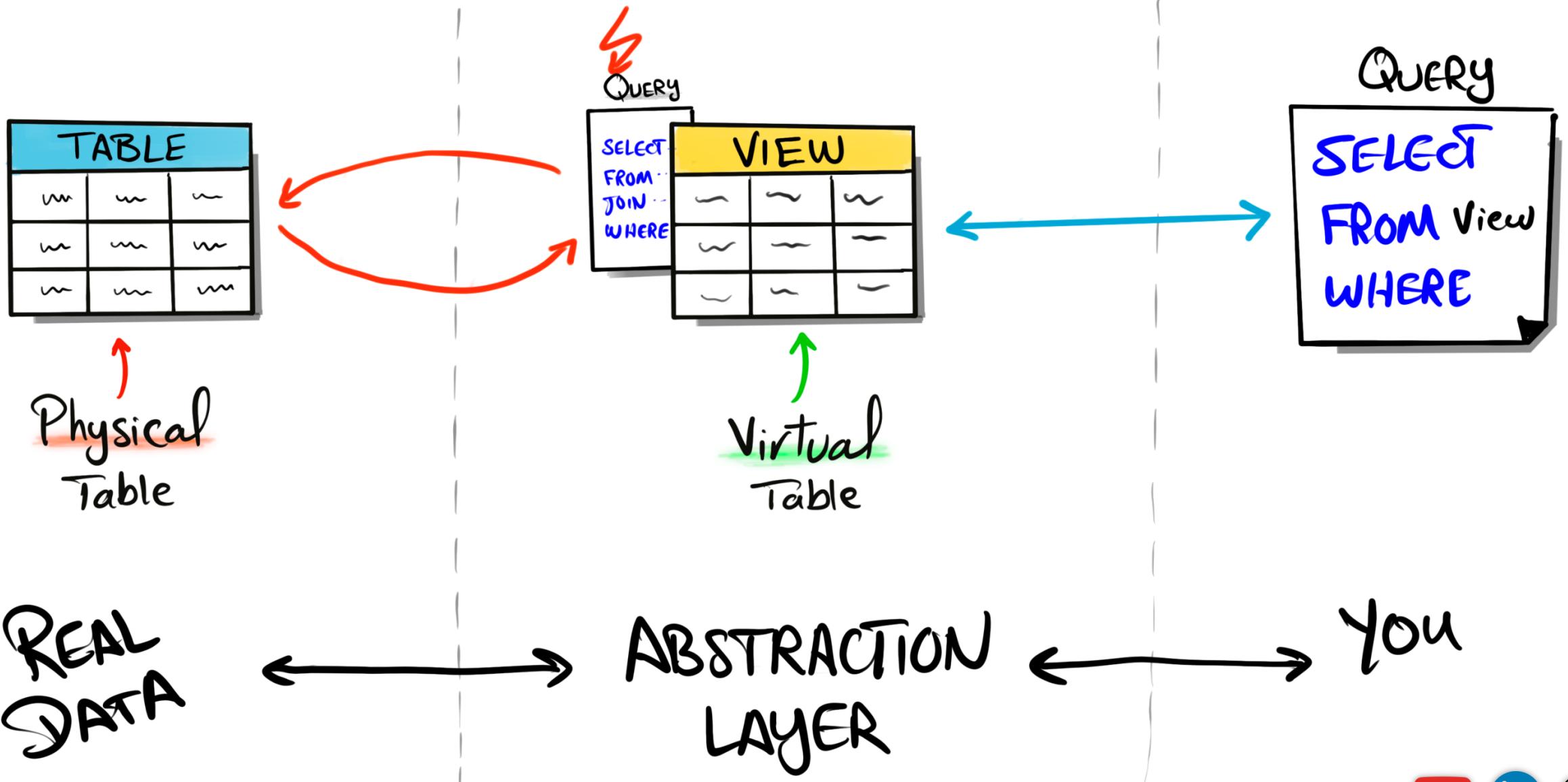




VIEW

virtual table based on the result set of a query,
without storing the data in database.

Views are persisted SQL queries in the database.



VIEW

TABLE

No Persistence

Persisted Data

Easy To Maintain

Hard To Maintain

Slow Response

Fast Response

Read

Read / Write

Flexible

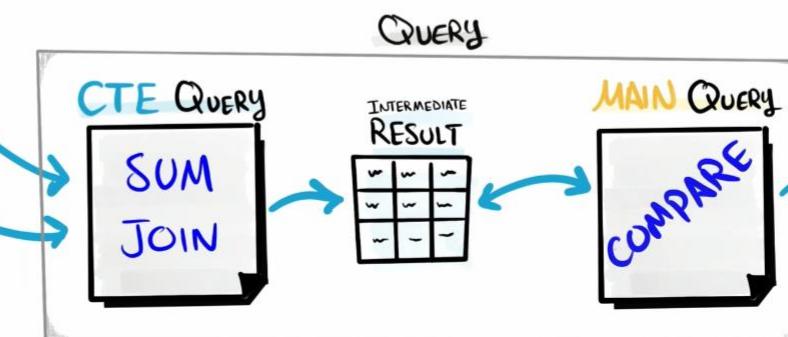
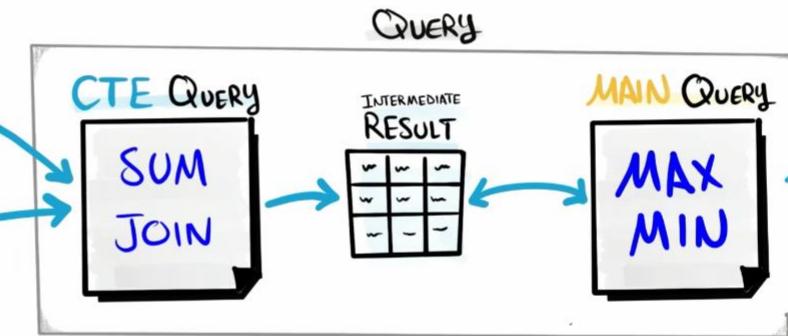
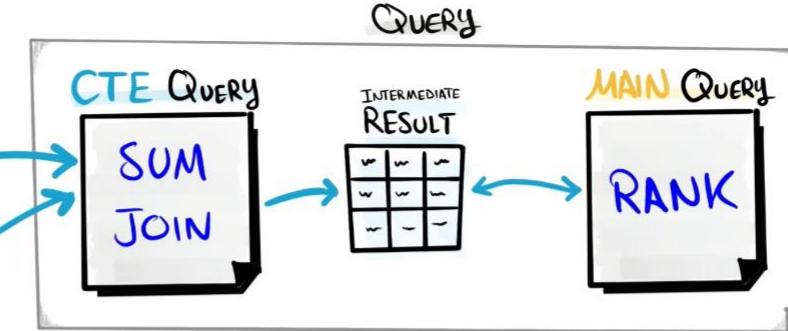
Hard To Change

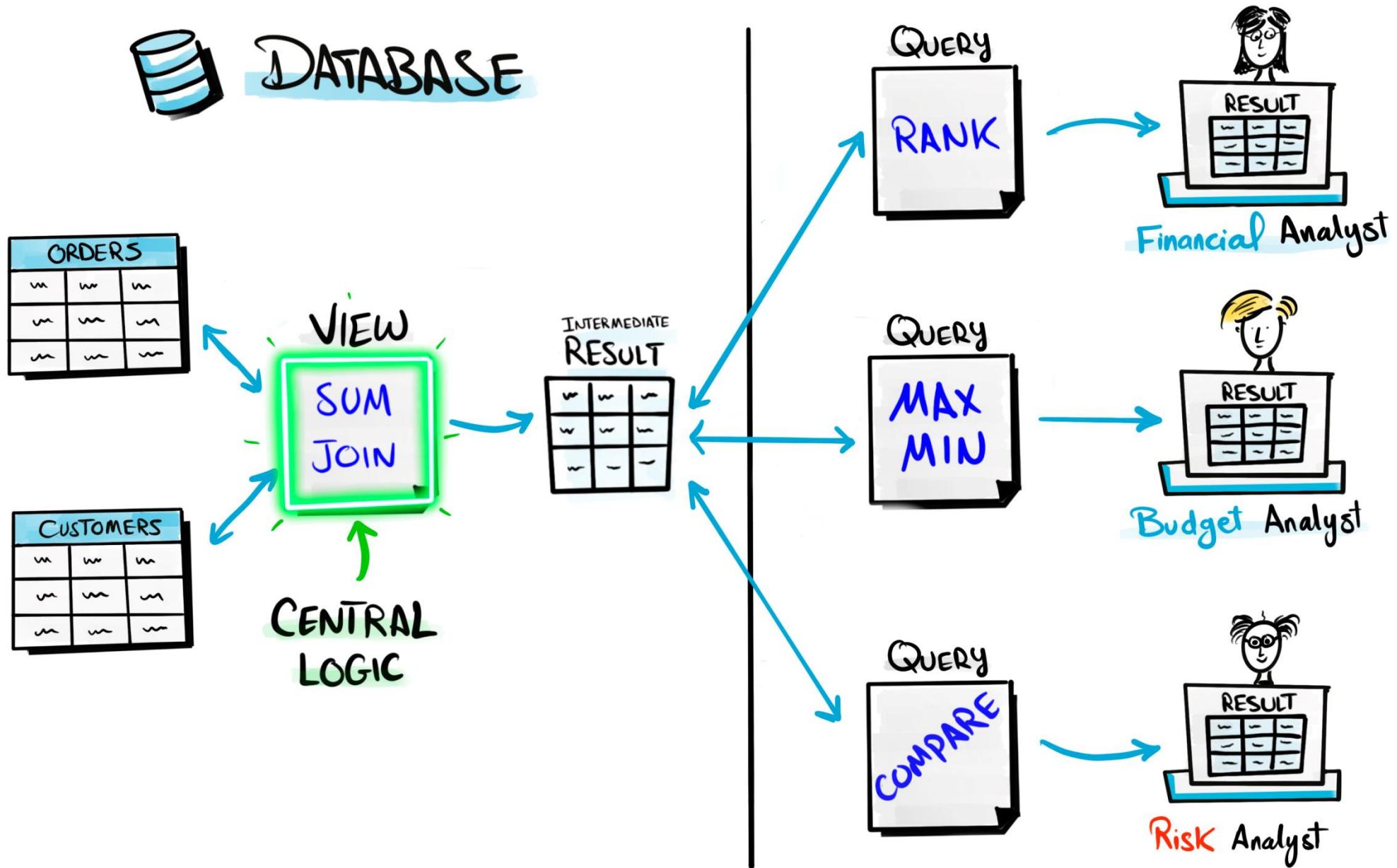


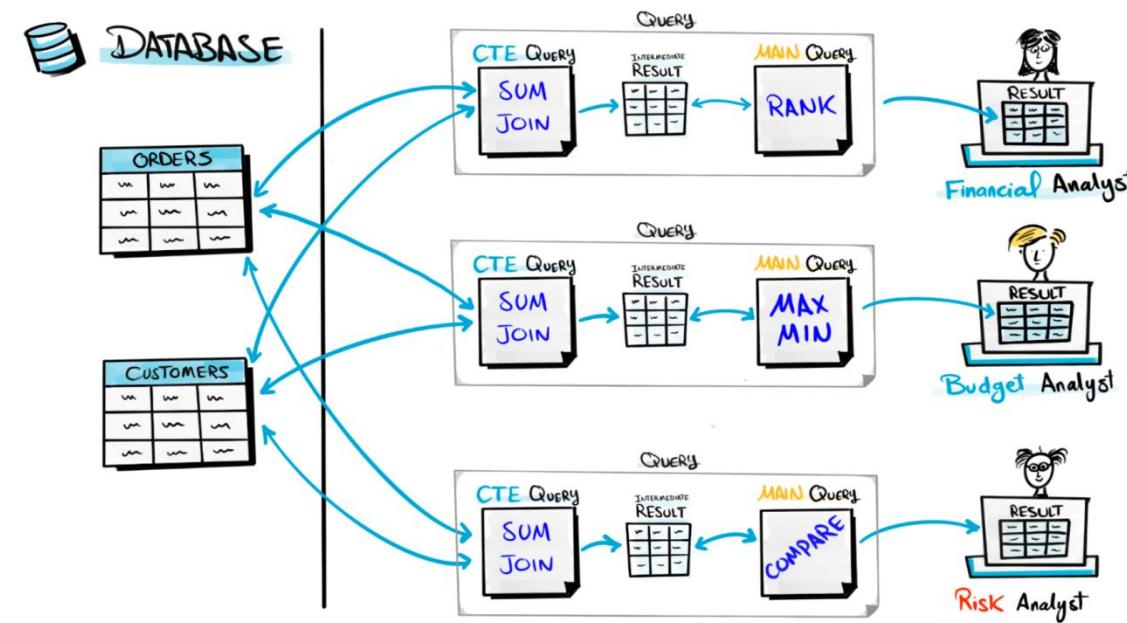
DATABASE

| ORDERS | | |
|--------|---|---|
| w | w | w |
| w | w | w |
| w | w | w |

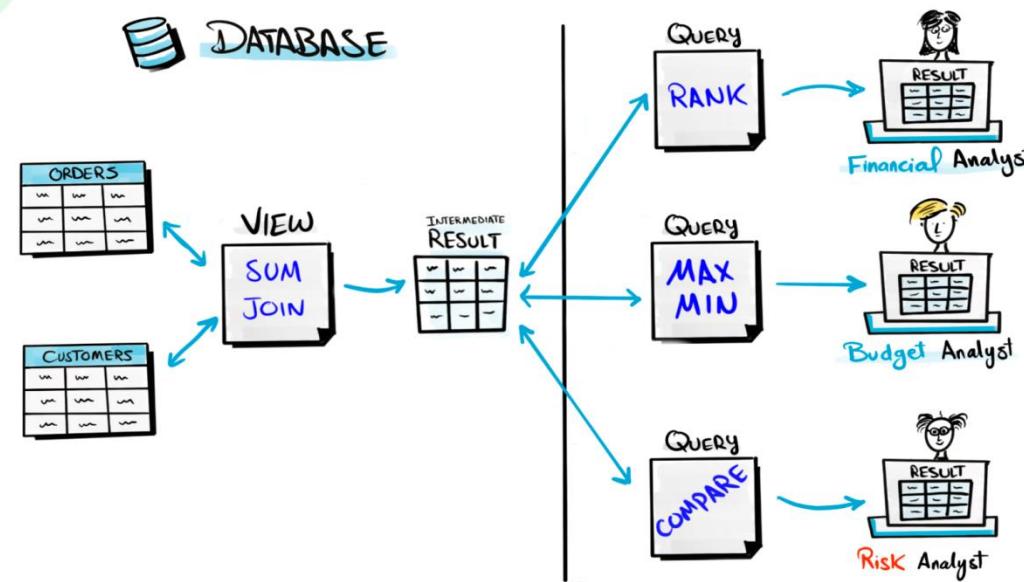
| CUSTOMERS | | |
|-----------|---|---|
| w | w | w |
| w | w | w |
| w | w | w |







✓ Reduce Redundancy & Complexity



VIEWS

Reduce Redundancy
in Multi-Queries

Improve Reusability
in Multi-Queries

Persisted logic

Need to Maintain
- CREATE/DROP -

CTE

Reduce Redundancy
in 1 Query

Improve Reusability
in 1 Query

Temporary Logic
- on the Fly -

No Maintenance
- Auto cleanup -

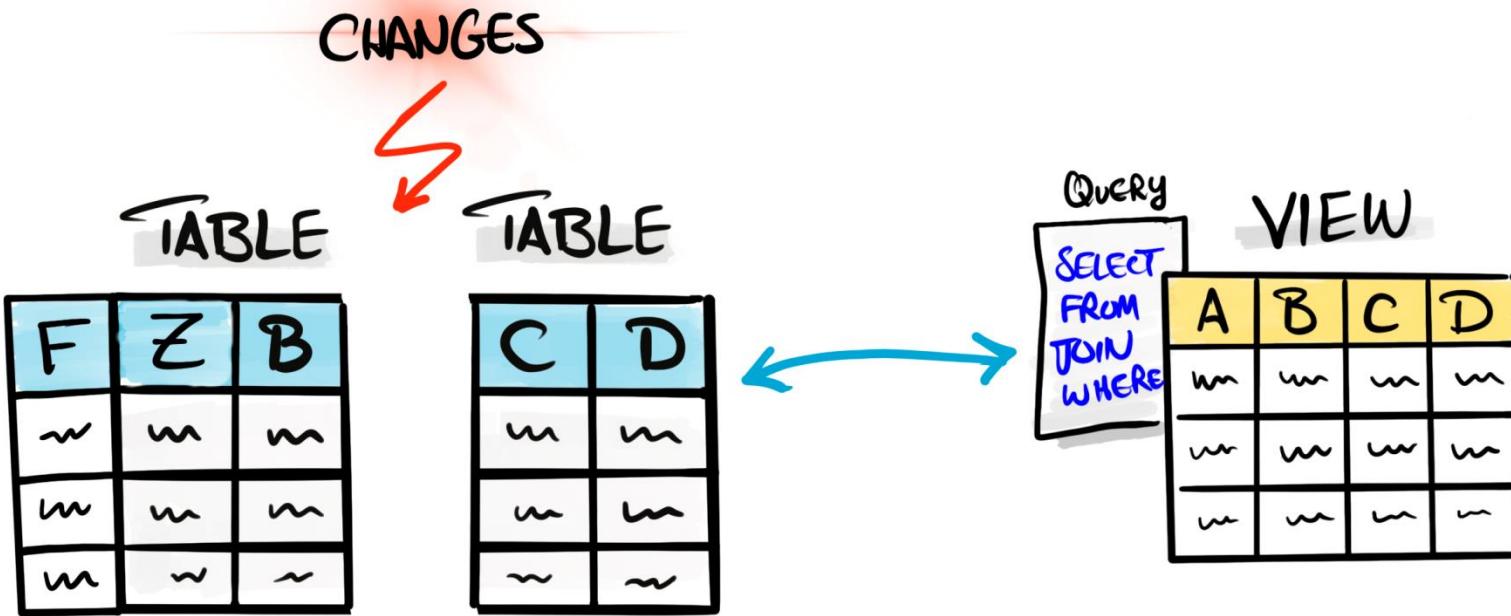
VIEWS

DDL
Statement

Query

```
CREATE VIEW VIEW-NAME AS
(
    SELECT ...
    FROM ...
    WHERE ...
)
```

Flexibility & Dynamic



Query

SELECT
FROM
JOIN...
WHERE
=

VIEW

Query

SELECT
FROM
JOIN...
WHERE
=

Query

SELECT
FROM
JOIN...
WHERE
=

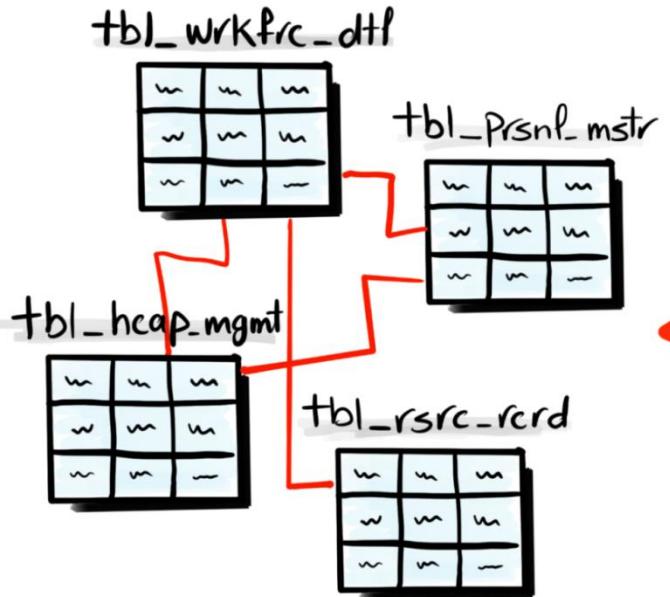
Query

SELECT
FROM
JOIN...
WHERE
=



Hide Complexity

TABLES



VIEW

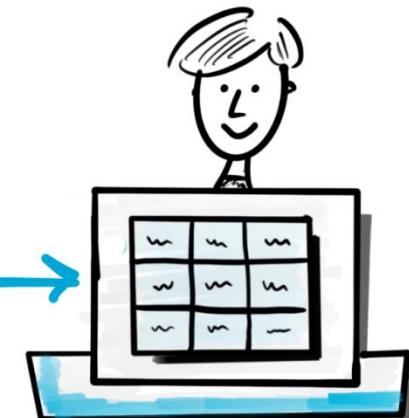
w
w
w
JOIN
JOIN
JOIN
JOIN
JOIN

Employee_Overview

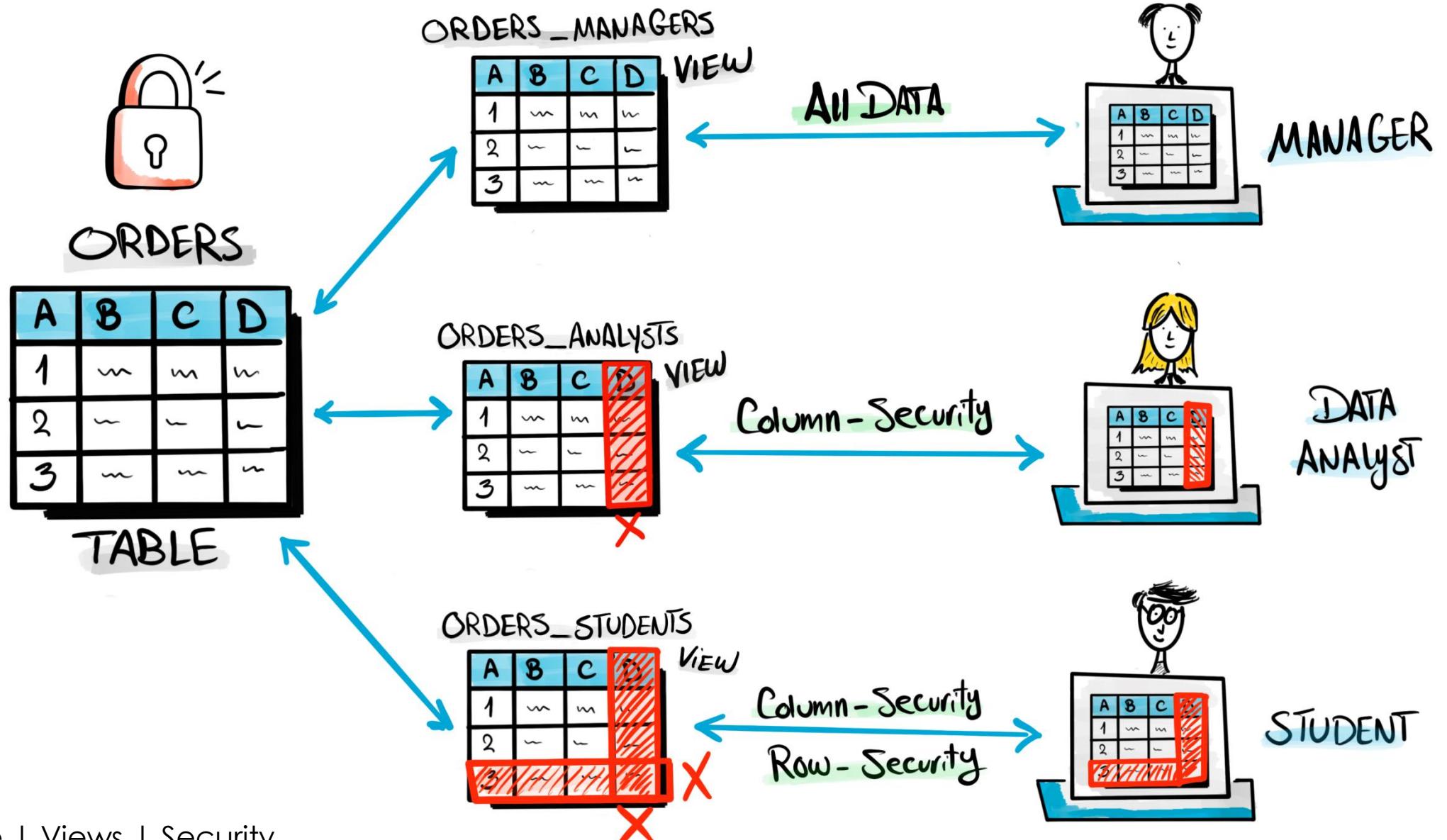
| | | |
|---|---|---|
| w | w | w |
| w | w | w |
| w | w | - |

QUERY

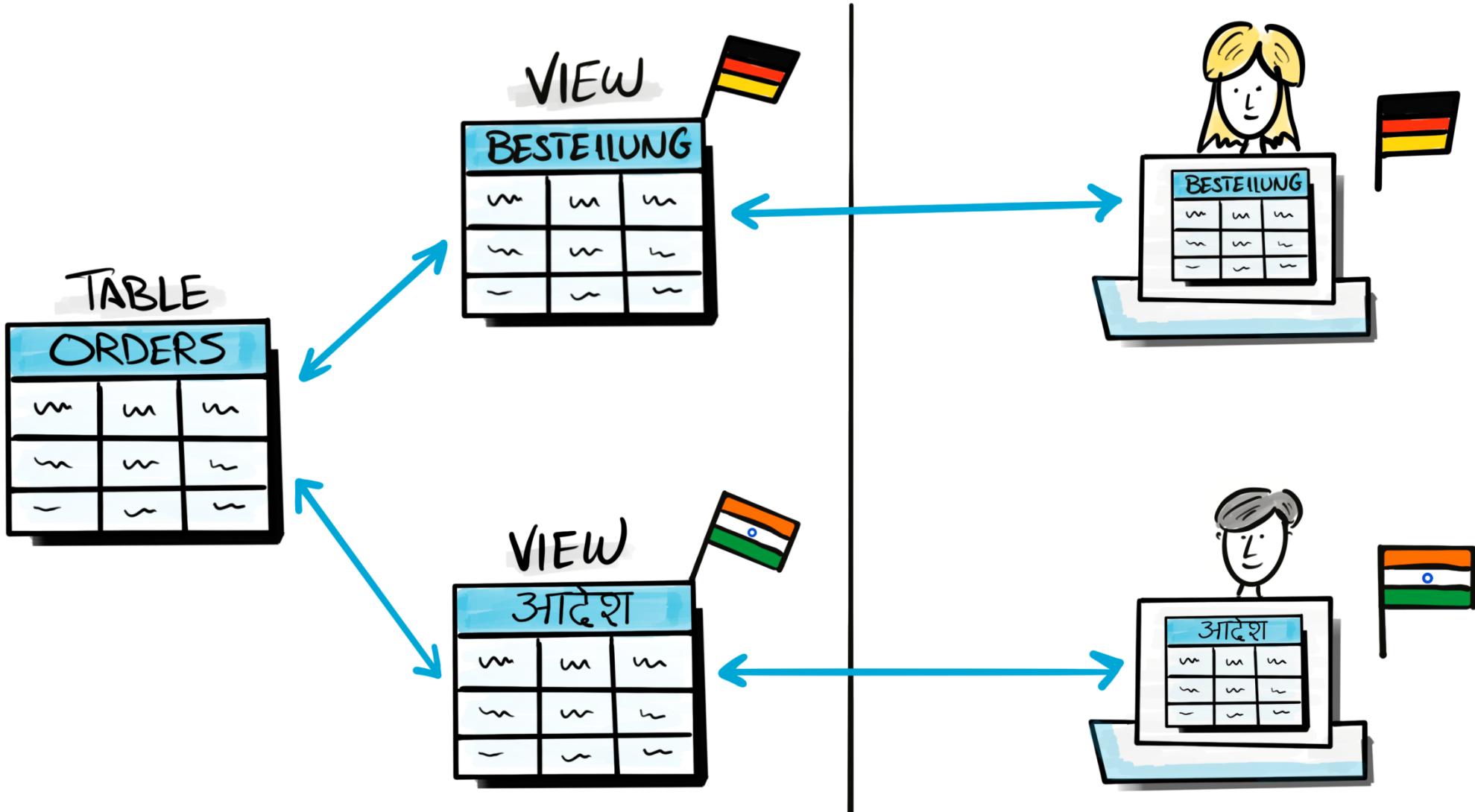
SELECT
FROM



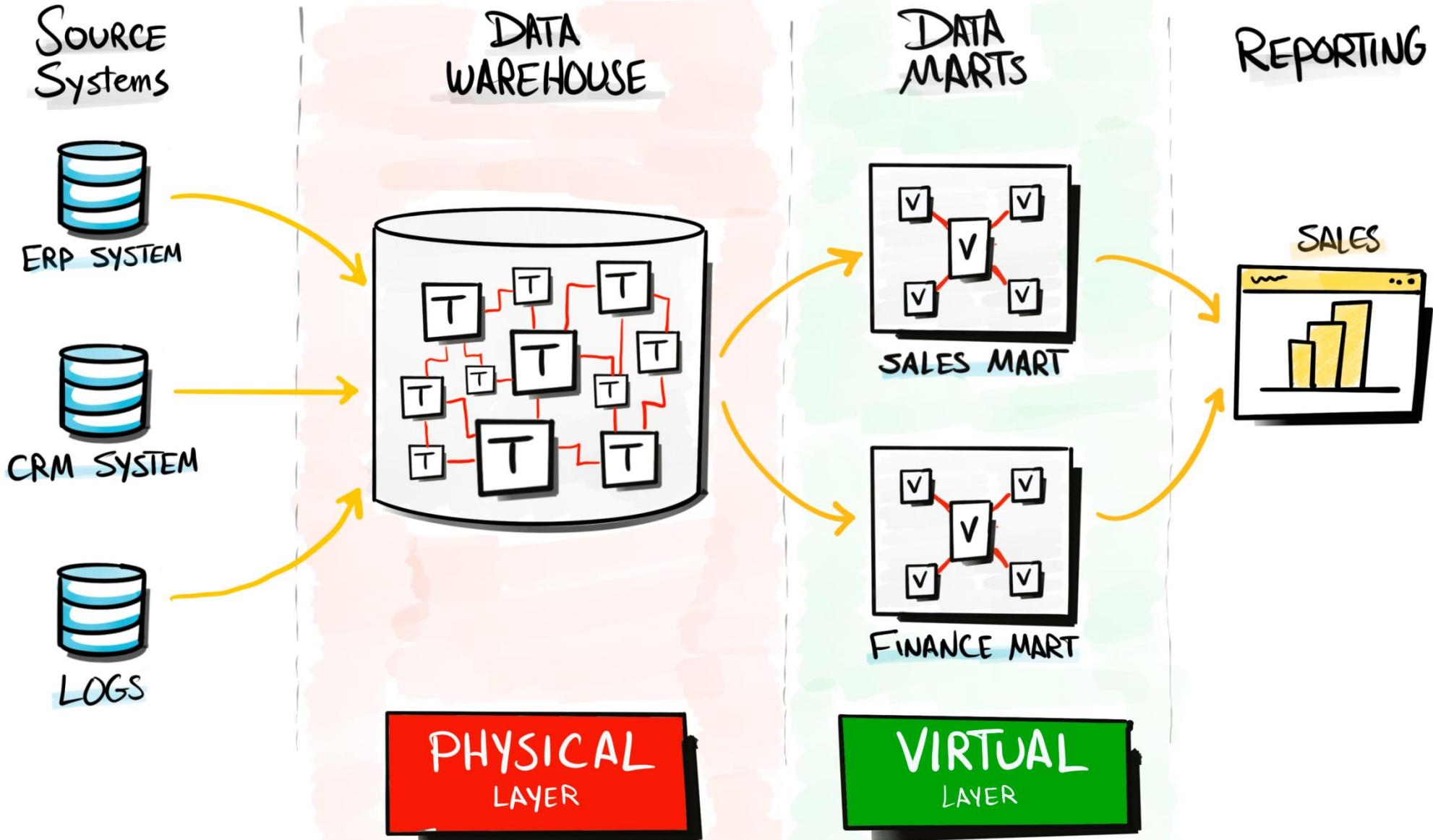
Security



Multiple Languages



Virtual Data Marts



VIEWS

- Virtual Table based on result of Query without storing data.
- We use Views to Persist Complex SQL Query in Database.
- Views are better than CTE - improves reusability in multiple Queries.
- Views are better than Tables - Flexible & ease to maintain.

USE CASES

- Store Central Complex Business Logic to be reused.
- Hide Complexity by offering friendly Views to users.
- Data Security by hiding sensitive rows & Columns.
- Flexibility & Dynamic
- Offer your objects in Multiple Languages.
- Virtual layer (Data Marts) in Data Warehouses.



CTAS

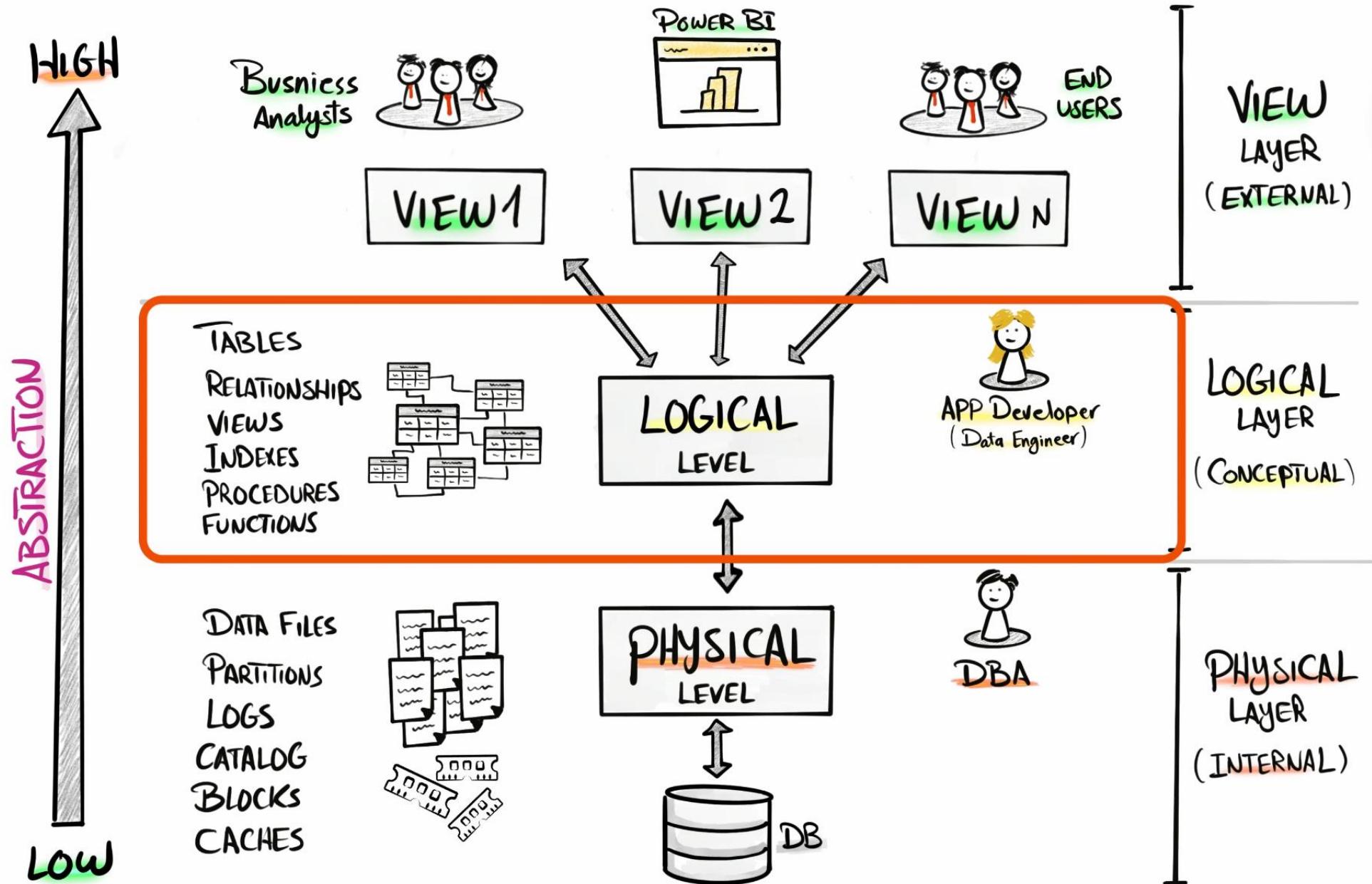
Create Table As SELECT

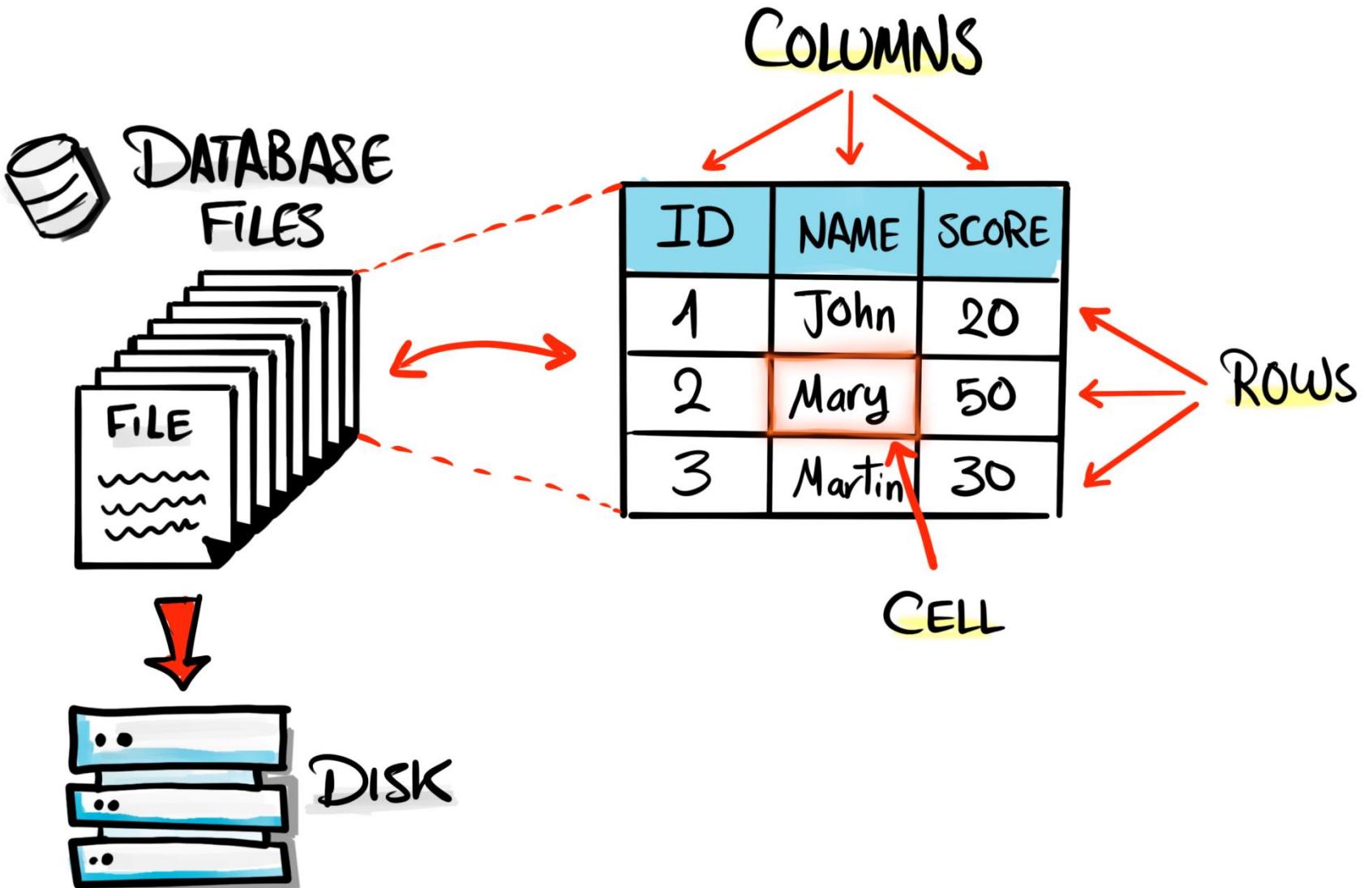
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | CTAS

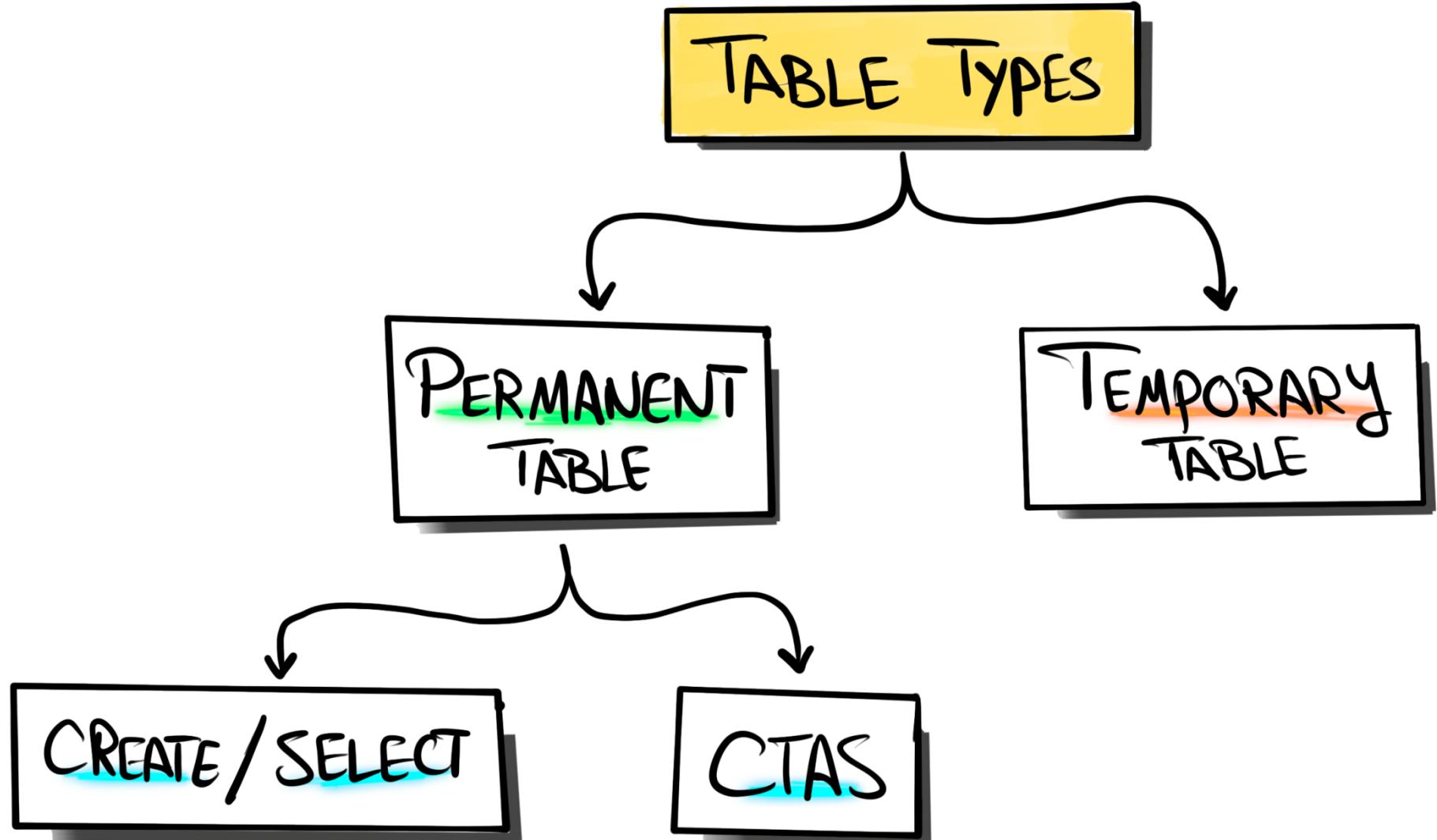


DB TABLE

A table is a structured collection of data,
similar to a spreadsheet or grid (Excel)







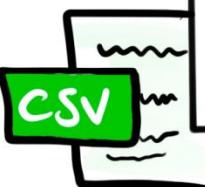
CREATE/INSERT

#1 STEP

CREATE

#2 STEP

INSERT



| TABLE | | |
|-------|---|---|
| w | w | w |
| w | w | w |
| w | w | w |

CTAS

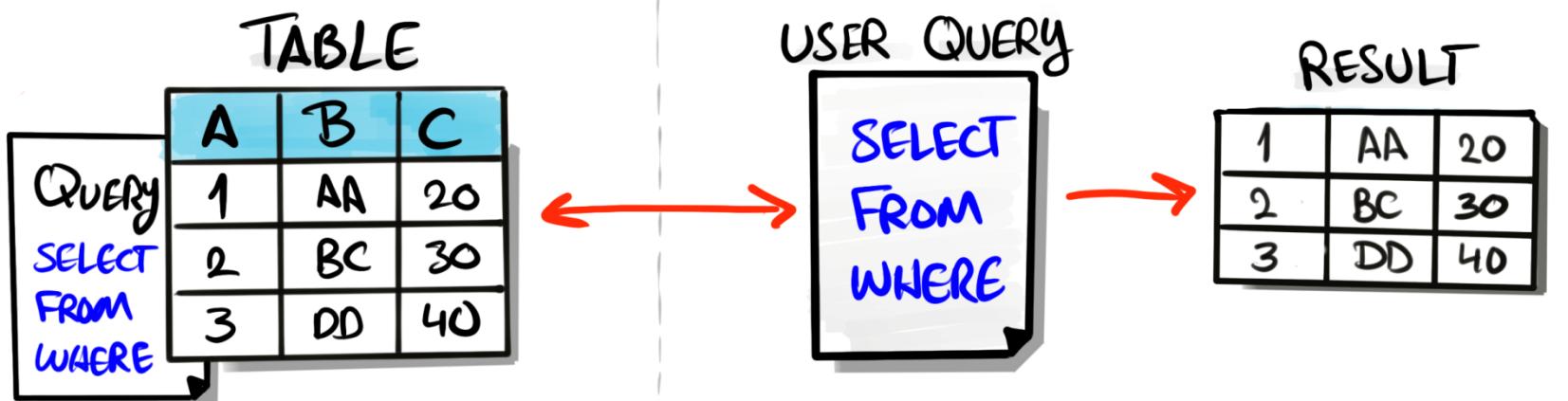
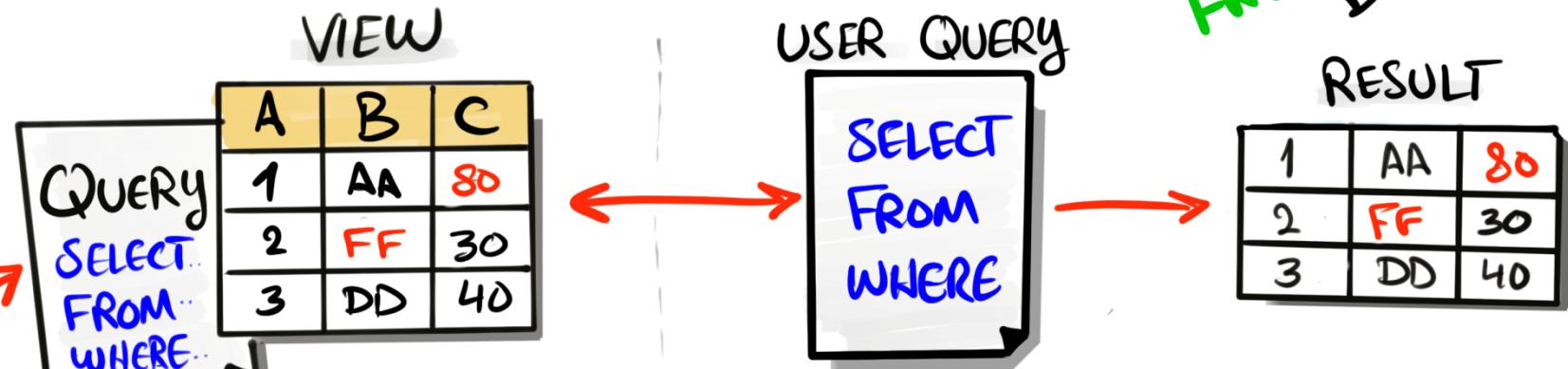
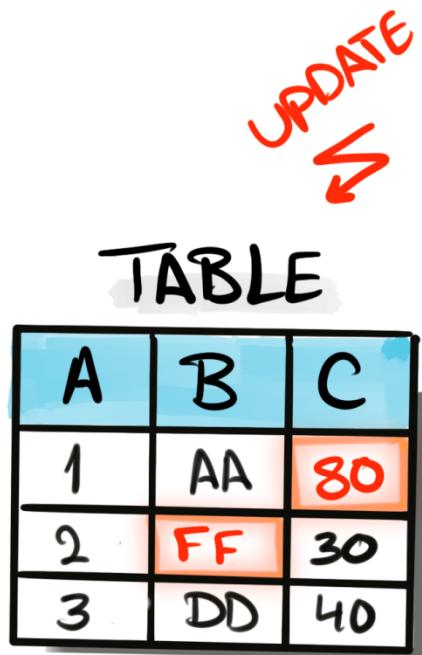
#1 STEP

QUERY

RESULT

| | | |
|---|---|---|
| w | w | w |
| - | - | - |
| w | - | - |

| TABLE | | |
|-------|---|---|
| w | w | w |
| w | w | w |
| w | - | w |



CREATE / INSERT syntax

DDL
Statement

```
CREATE TABLE Table-Name  
(  
    ID INT,  
    Name VARCHAR (50)  
)
```

Insert
Statement

```
INSERT INTO Table-Name  
VALUES (1, 'Frank')
```

CTAS syntax

DDL
Statement

```
CREATE TABLE NAME AS
(
    SELECT ...
    FROM ...
    WHERE ...
)
```

Query

MySQL | Postgres | Oracle

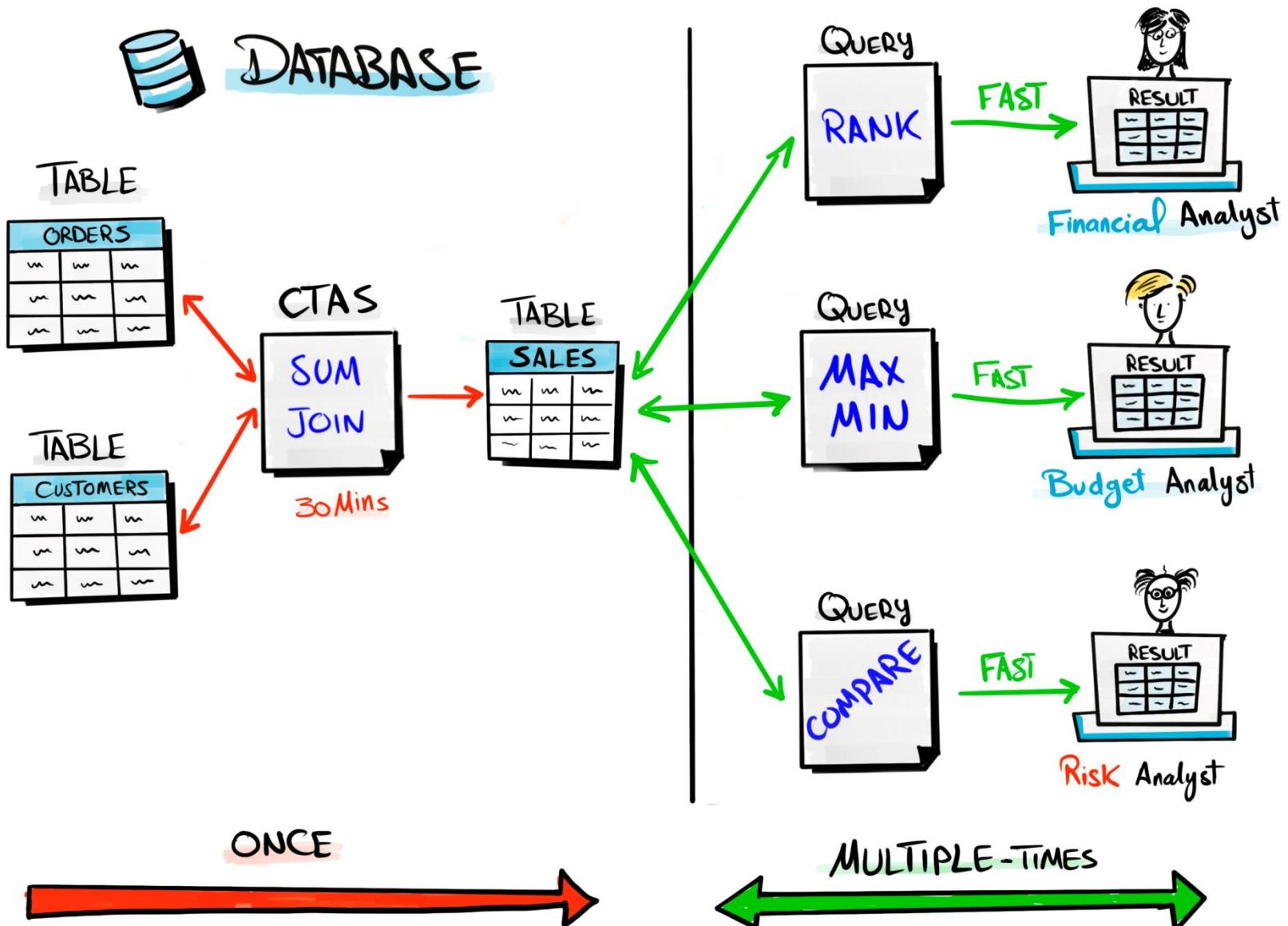
```
SELECT ...
```

```
INTO New-Table
```

```
FROM ...
WHERE ...
```

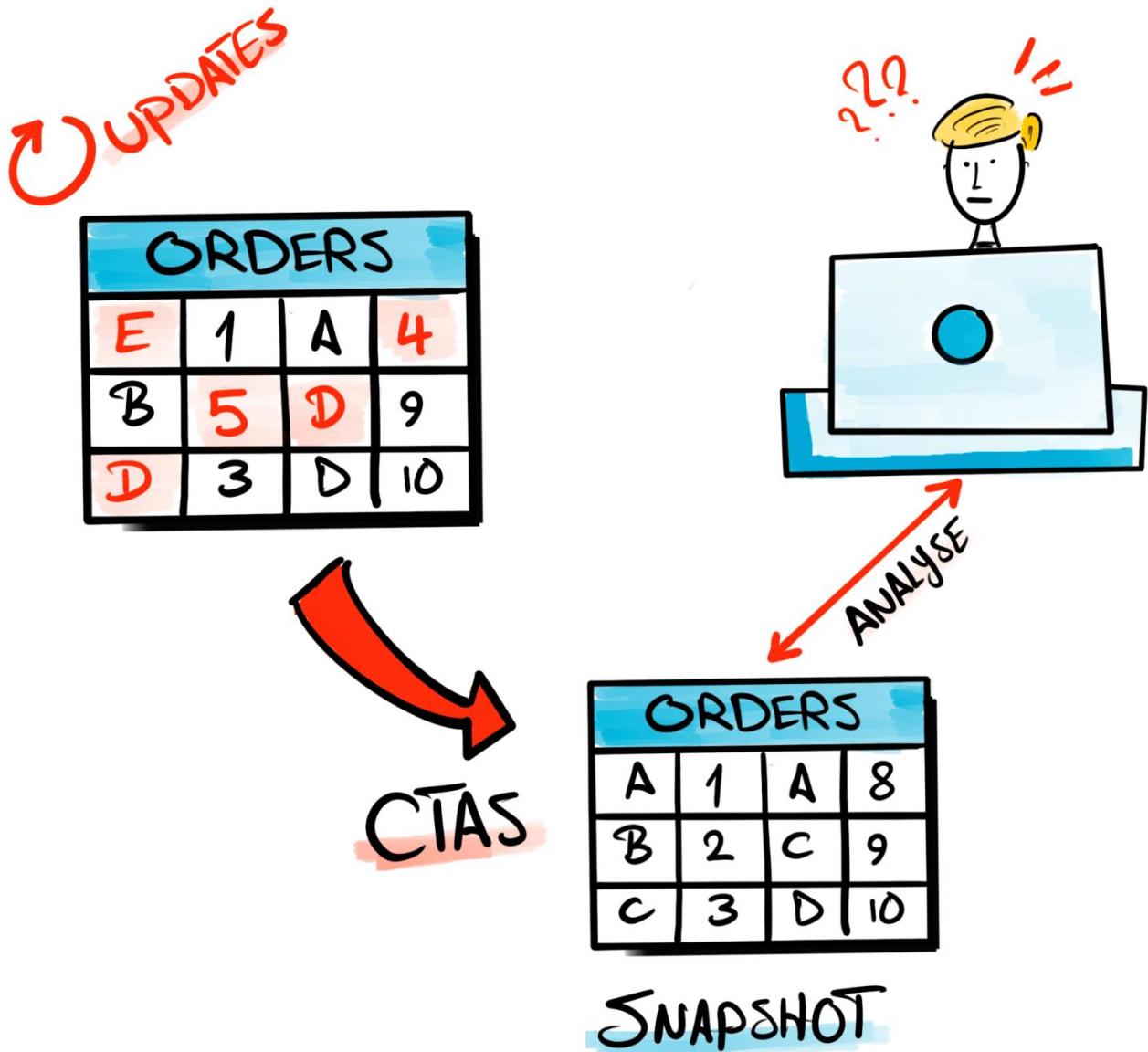
Sql Server

Optimize Performance

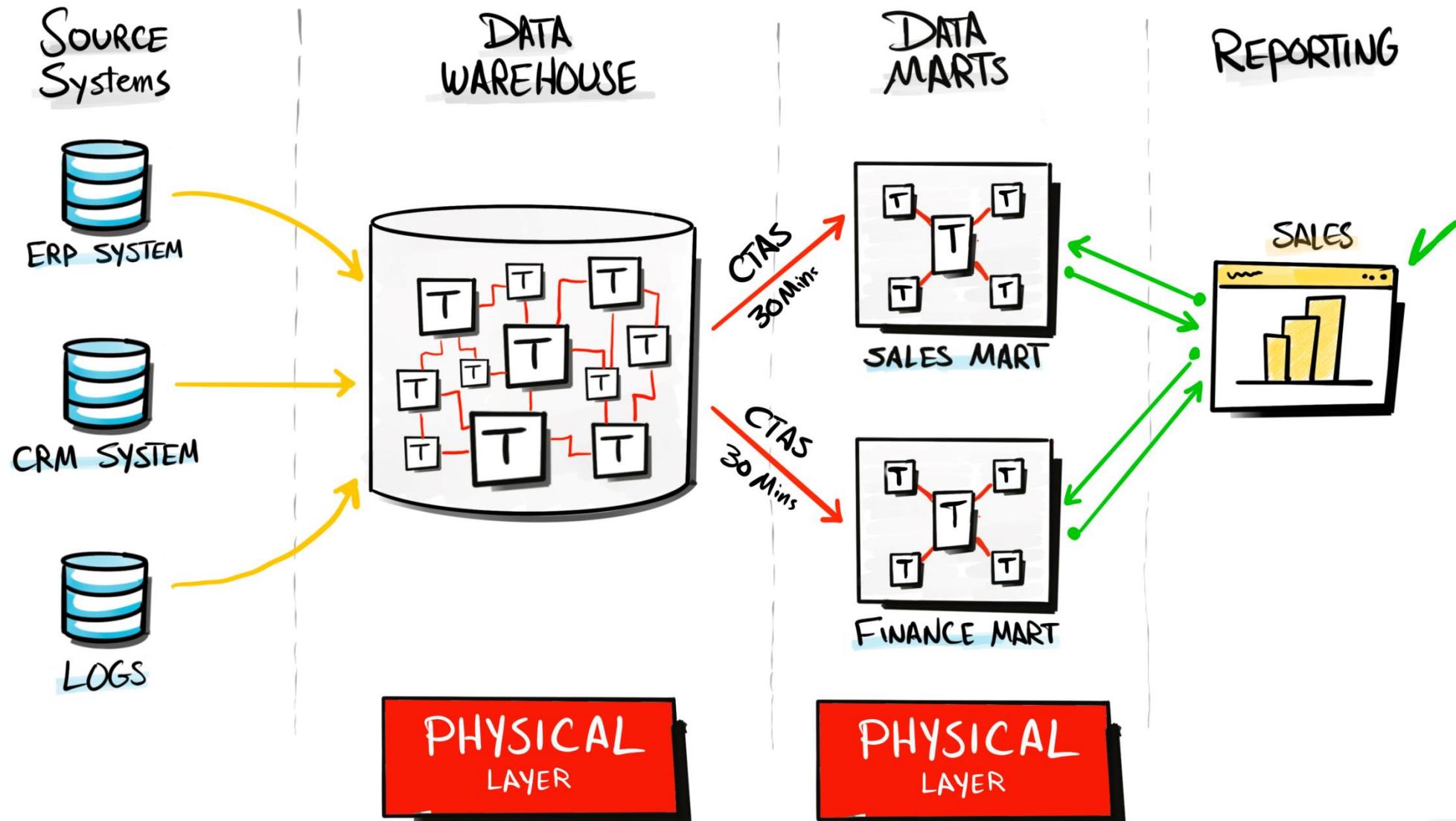


Create Snapshot

You want to preserve the current state of data before performing operations that might change it.



Physical Data Marts



TEMPORARY TABLE

SELECT ...

INTO #New-Table

FROM ...

WHERE ...

Sql Server

PERMENANT CREATE TABLE

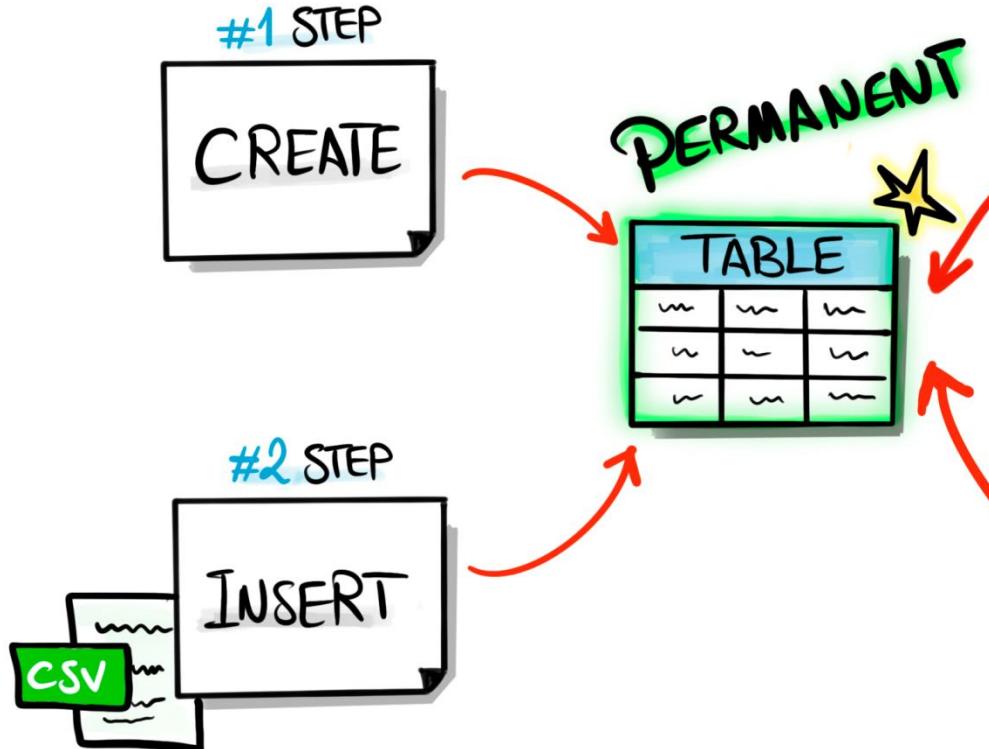
```
CREATE TABLE TABLE-NAME AS  
(  
    SELECT ...  
    FROM ...  
    WHERE ...  
)
```

TEMPORARY TABLE

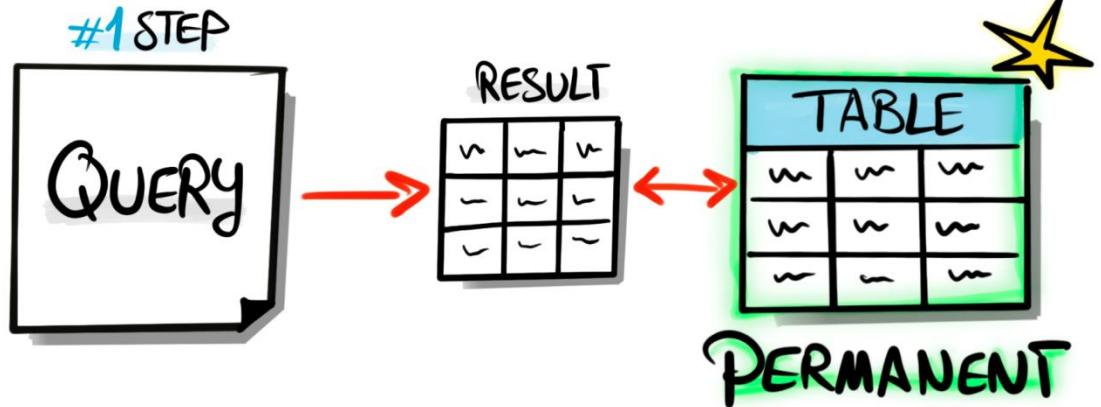
```
CREATE TEMPORARY TABLE TABLE-NAME AS  
(  
    SELECT ...  
    FROM ...  
    WHERE ...  
)
```

MySQL | Postgres | Oracle

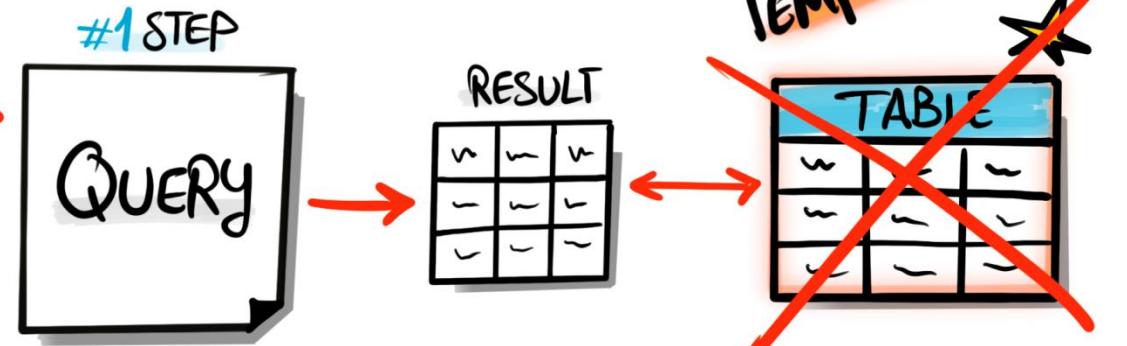
CREATE/INSERT

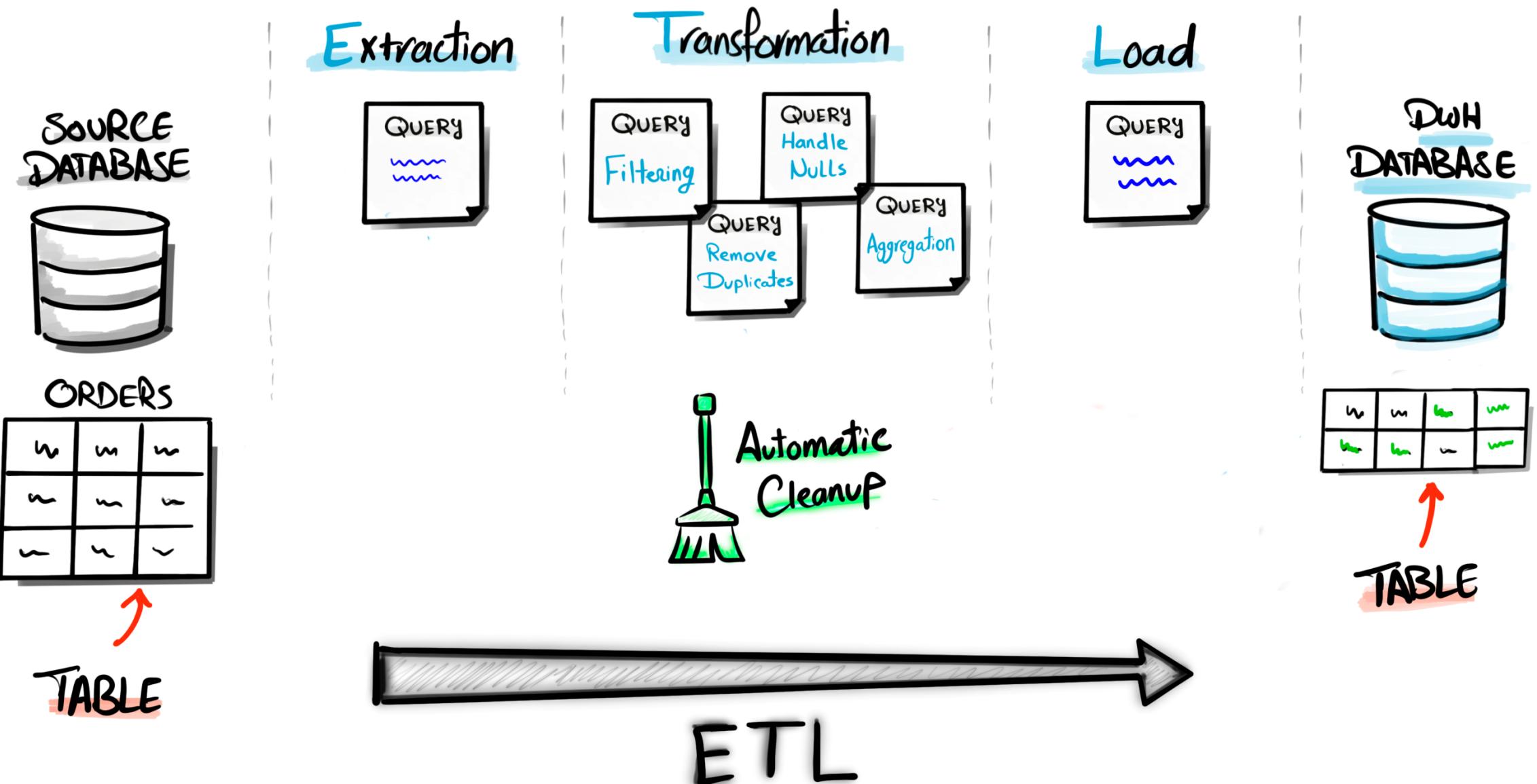


CTAS



TMP





Empty Table

```
CREATE TABLE Table-Name  
(  
    ID INT,  
    Name VARCHAR (50)  
)
```

View

```
CREATE VIEW View-Name AS  
(  
    SELECT ...  
    FROM ...  
    WHERE ...  
)
```

Permanent Table

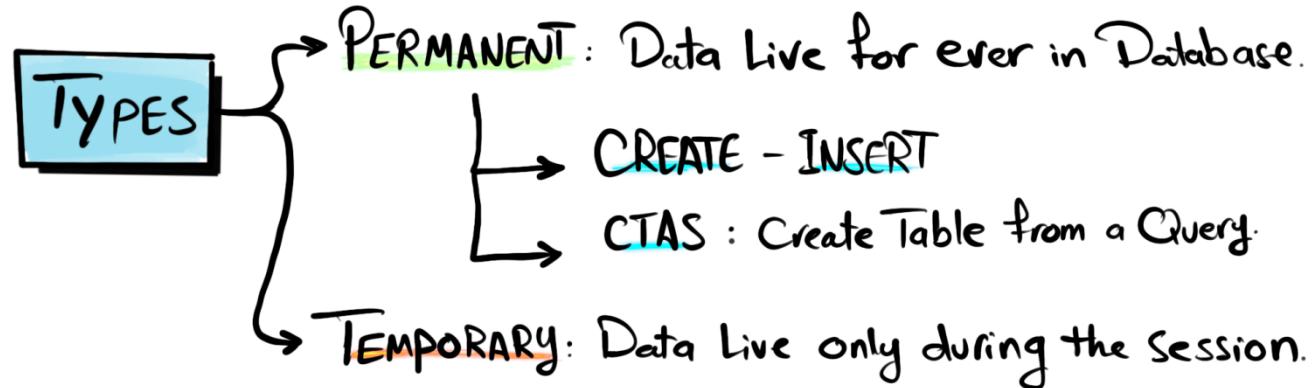
```
SELECT ...  
INTO New-Table  
FROM ...  
WHERE ...
```

Temporary Table

```
SELECT ...  
INTO #New-Table  
FROM ...  
WHERE ...
```

TABLES

Structured Collection of Data like spreadsheet (Columns & Rows)



CTAS USE CASES

- Optimize Performance : Persist Complex SQL logic in Table.
- Creating Snapshot : to analyse Bugs and data issues.

Advantage of TEMP Tables Automatic cleanup of data after session ends.



DATA WITH BARAA

Big Picture All Techniques

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Comparison



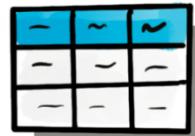


| | SUBQUERY | CTE | TMP | CTAS | VIEW |
|--------------|---|--|--|---|---|
| STORAGE | MEMORY  | | DISK  | | ✗ NO STORAGE |
| LIFE TIME | | TEMPORARY | | | PERMANENT |
| WHEN DELETED | | END OF QUERY | END OF SESSION | | DDL - DROP |
| SCOPE | | SINGLE - QUERY | | | MULTI - QUERIES |
| REUSABILITY | LIMITED 1 PLACE - 1 QUERY | LIMITED Multi PLACES - 1 Query | MEDIUM Multi QUERIES During Session | | HIGH MULTI QUERIES |
| UP2DATE |  |  |  |  |  |



DATABASE

TEMP TABLE



TABLE

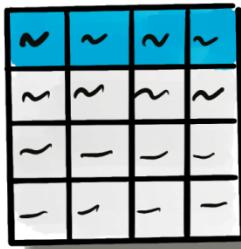
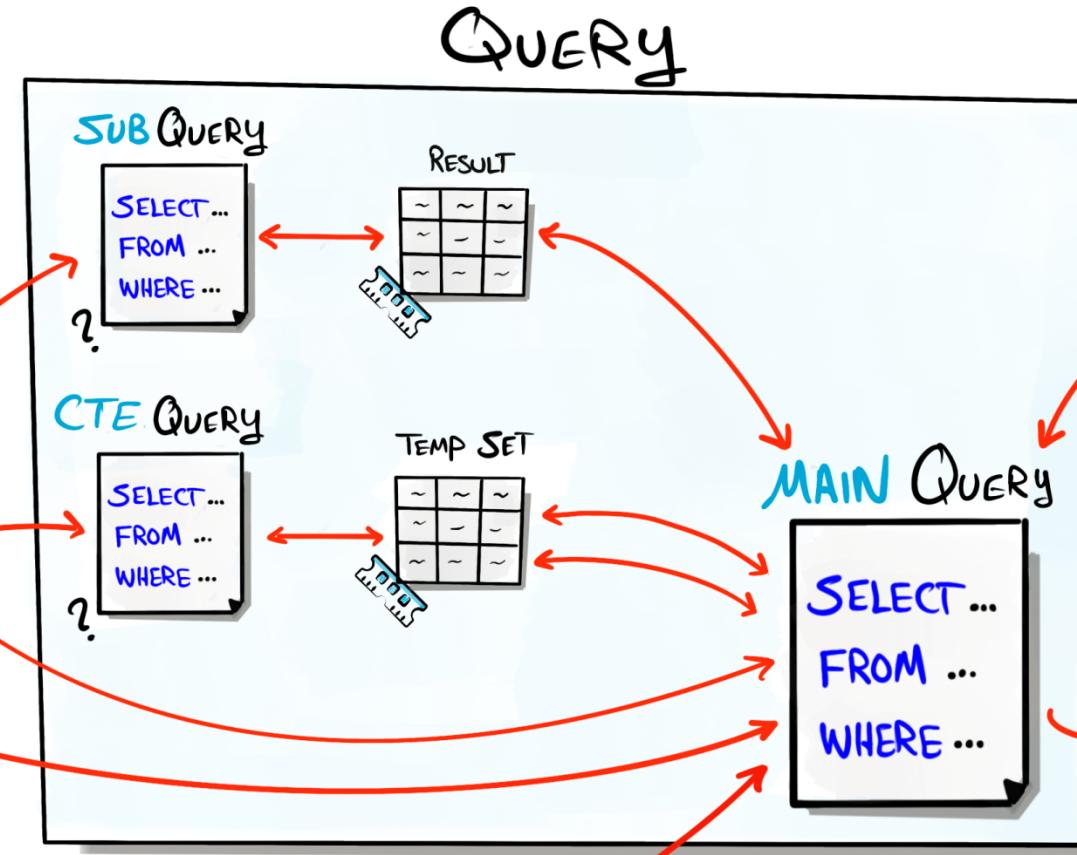


TABLE DDL STATEMENT

VIEW DDL STATEMENT

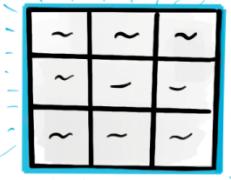


VIEW



DATA SCIENTIST/ANALYST

FINAL RESULT



DATABASE ADMIN

INSERT STATEMENT

INSERT INTO VALUES (...)



DATA WITH BARAA

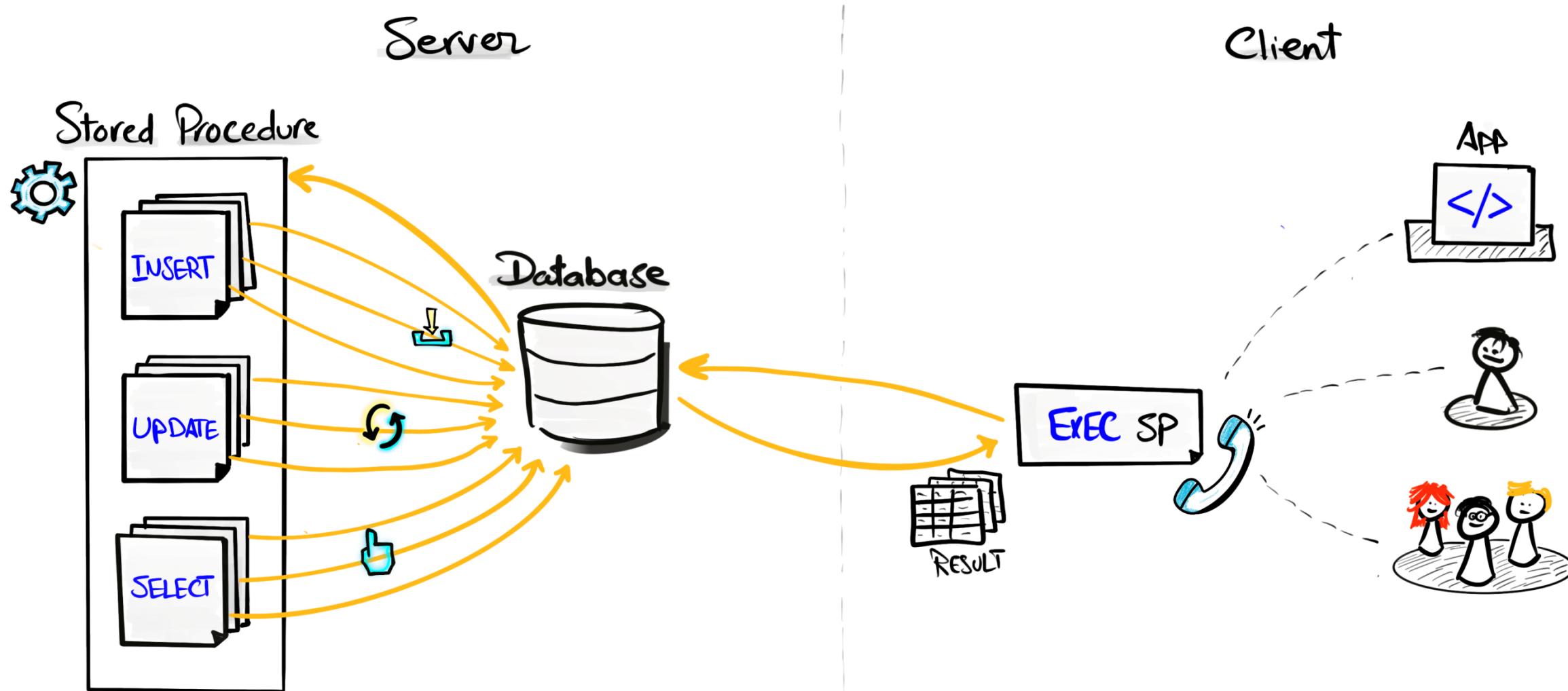


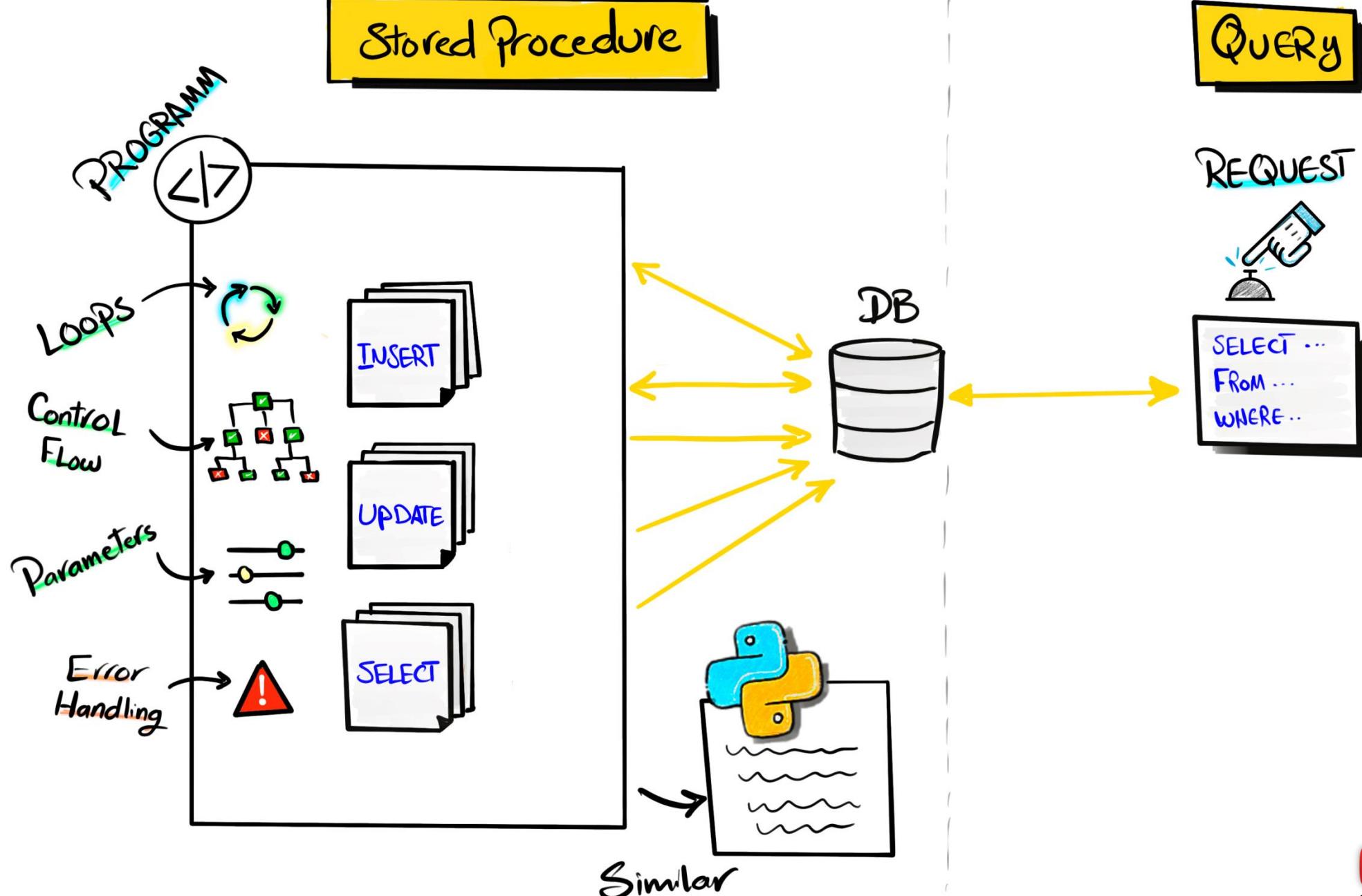
Stored Procedure

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Stored Procedure

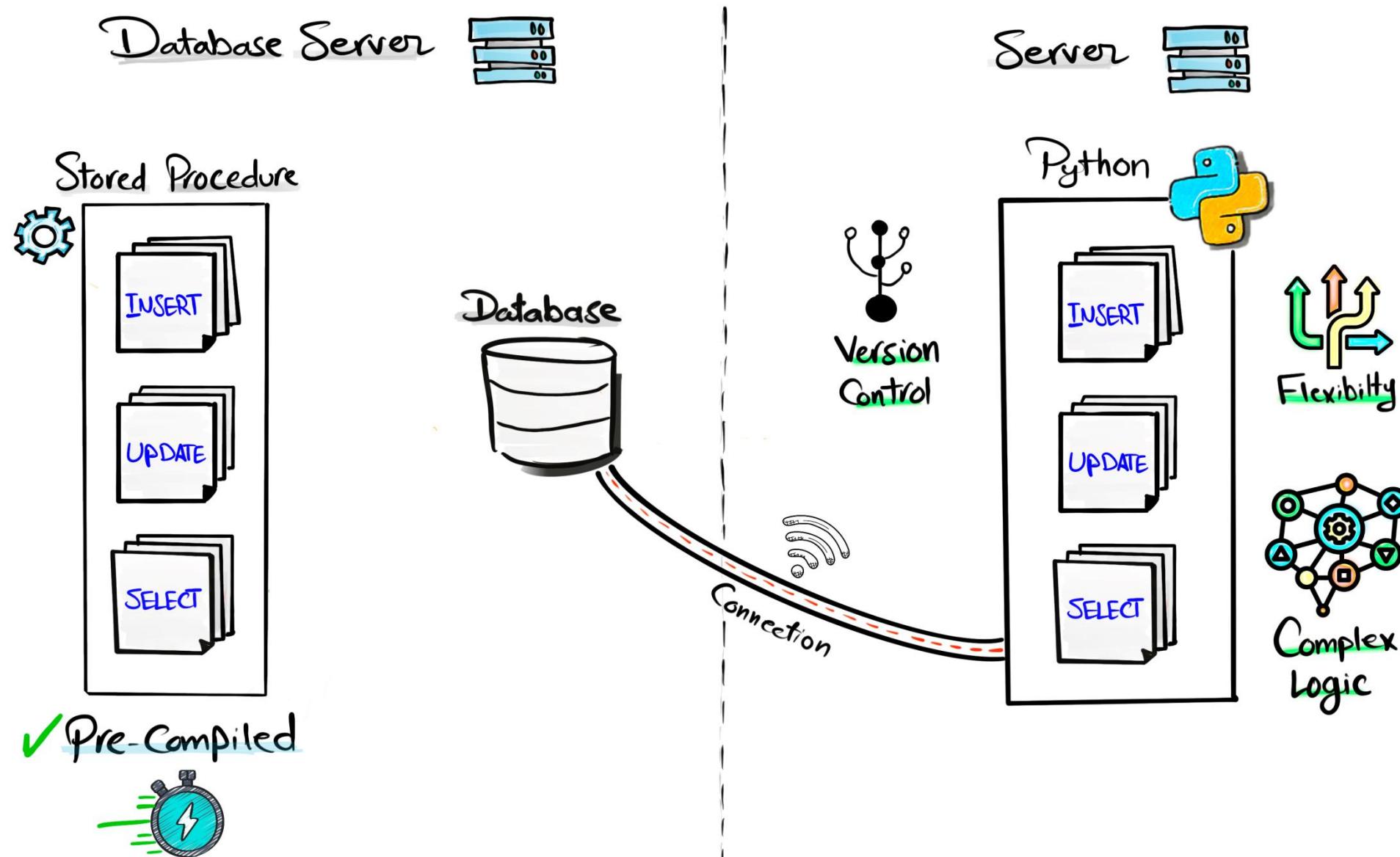


Stored Procedure





Stored Procedure vs Python



Stored Procedure

Stored Procedure
Definition



```
CREATE PROCEDURE ProcedureName AS  
BEGIN  
    -- SQL STATEMENTS GO HERE  
END
```

Stored Procedure
Execution (Call)



```
EXEC ProcedureName
```

Error Handling

```
BEGIN TRY
```

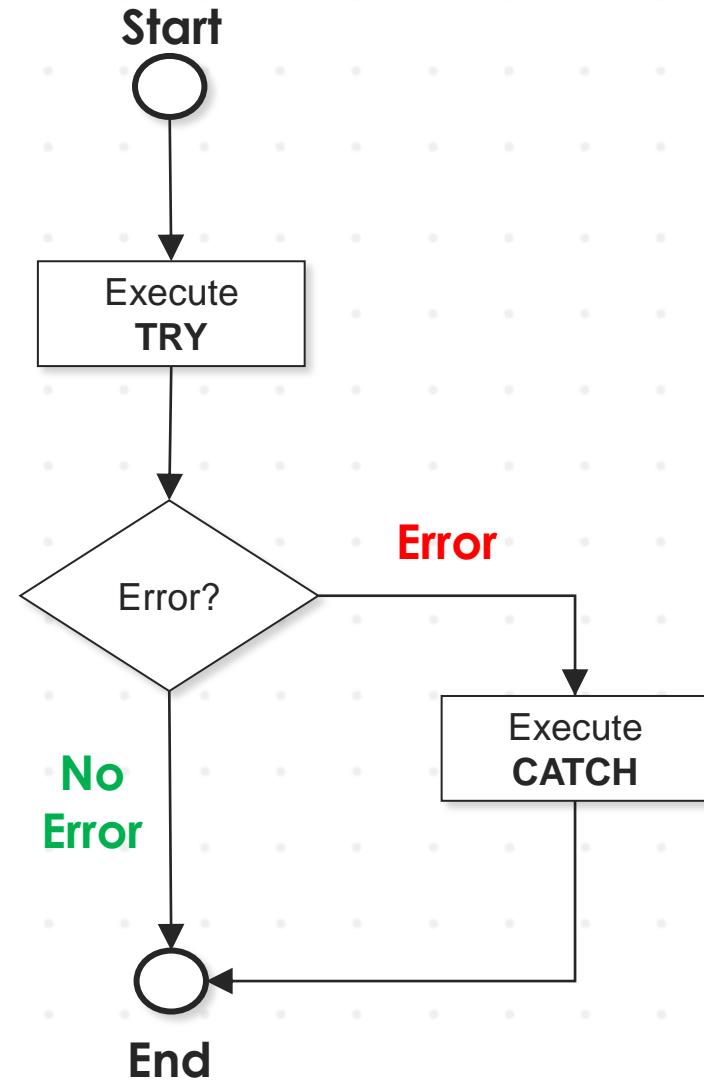
-- SQL statements that might cause an error

```
END TRY
```

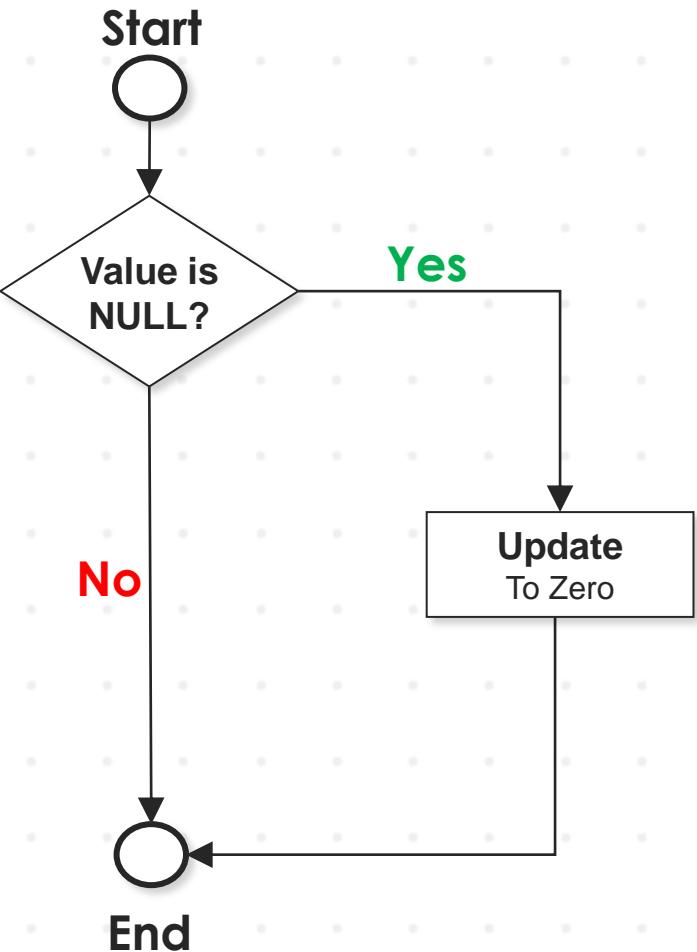
```
BEGIN CATCH
```

-- SQL statements To Handle The Error

```
END CATCH
```



Flow Control



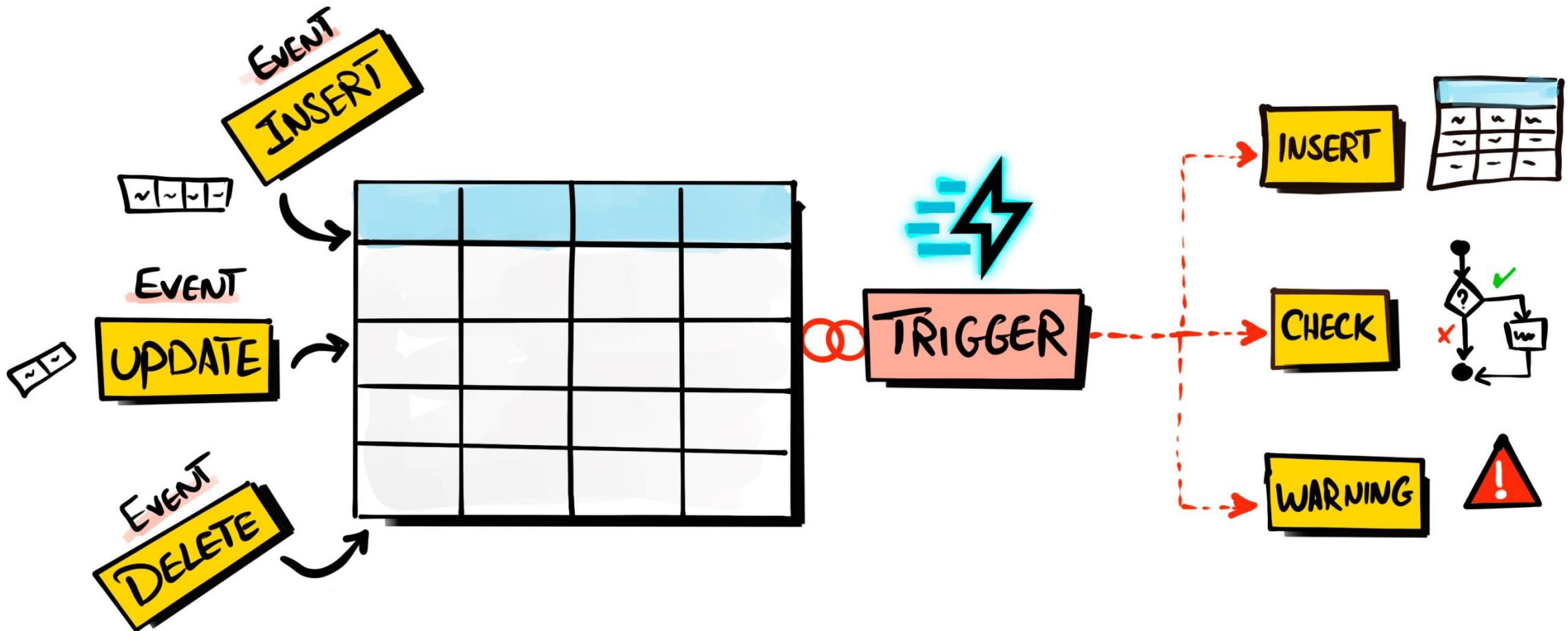


Triggers

Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Triggers



Triggers





TRIGGER TYPES

AFTER

DML
Triggers

DDL
Triggers

LOGGON

INSTEAD OF

INSERT
UPDATE
DELETE

CREATE
ALTER
DROP

Maintaining Logs

Employees

| ID | Name | Dept. | Salary |
|----|-------|-------|--------|
| 1 | Maria | HR | 70K |
| 2 | John | Sales | 80K |
| 3 | Max | HR | 70K |
| ~ | ~ | - | - |

TRIGGER

INSERT

LOGS

| | |
|---|------|
| 1 | 2025 |
| 2 | 2026 |
| 3 | 2027 |
| ~ | ~ |
| ~ | ~ |

Triggers

```
CREATE TRIGGER TriggerName ON TableName  
WHEN -----> AFTER INSERT, UPDATE, DELETE  
      BEGIN  
WHAT ----->      -- SQL STATEMENTS GO HERE  
      END
```