



DATA WITH BARAA

ADVANCED SQL

Techniques

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



SOLUTIONS

Subquery

- CTE -

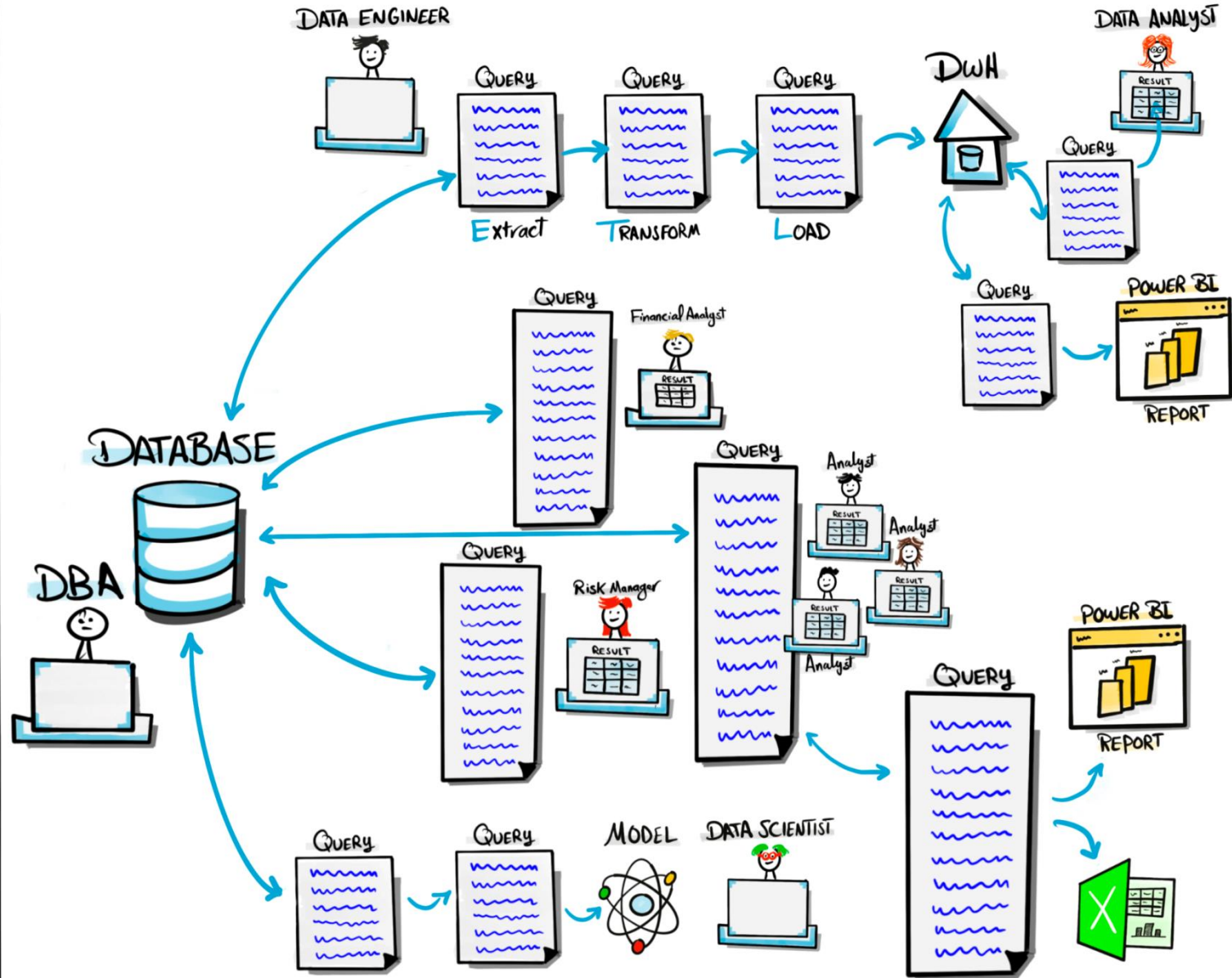
Common Table Expression

Views

Temp Tables

- CTAS -

Create Table As Select



CHALLENGES

Redundancy

Performance Issues

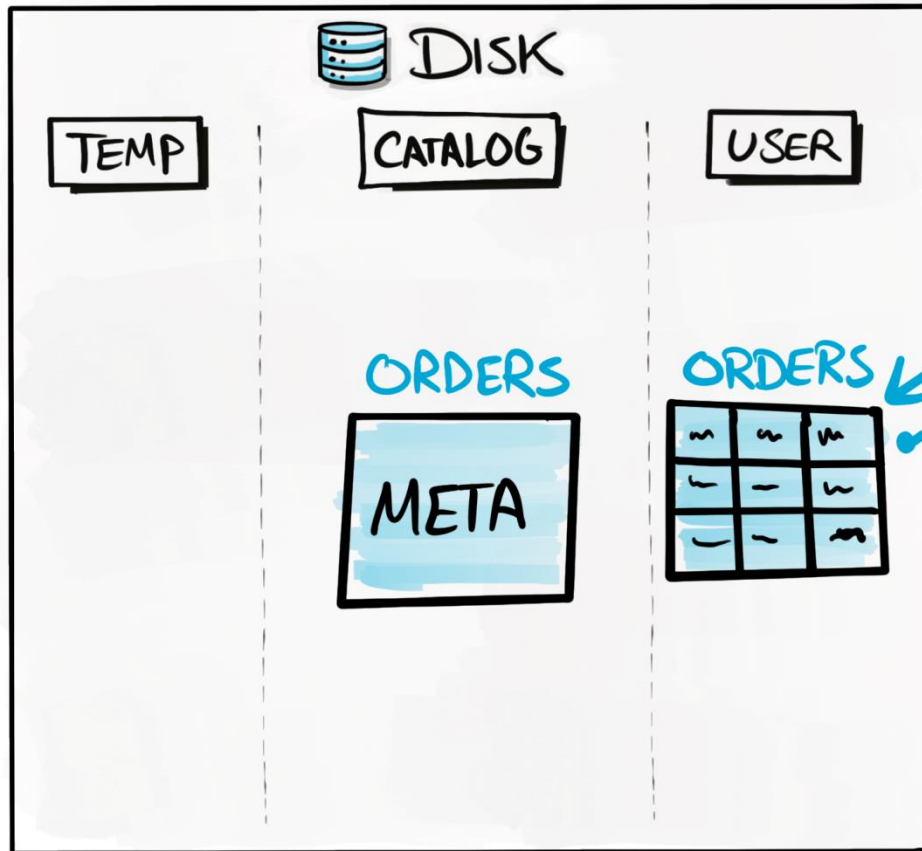
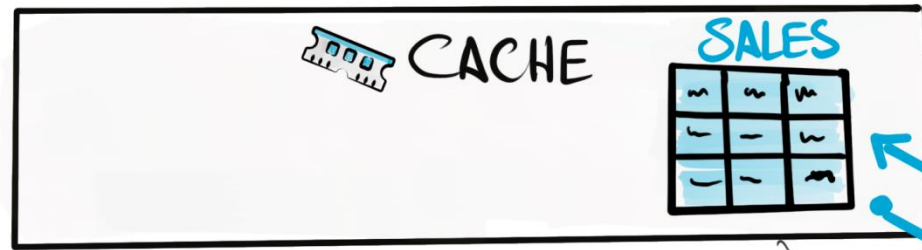
Complexity

Hard To Maintain

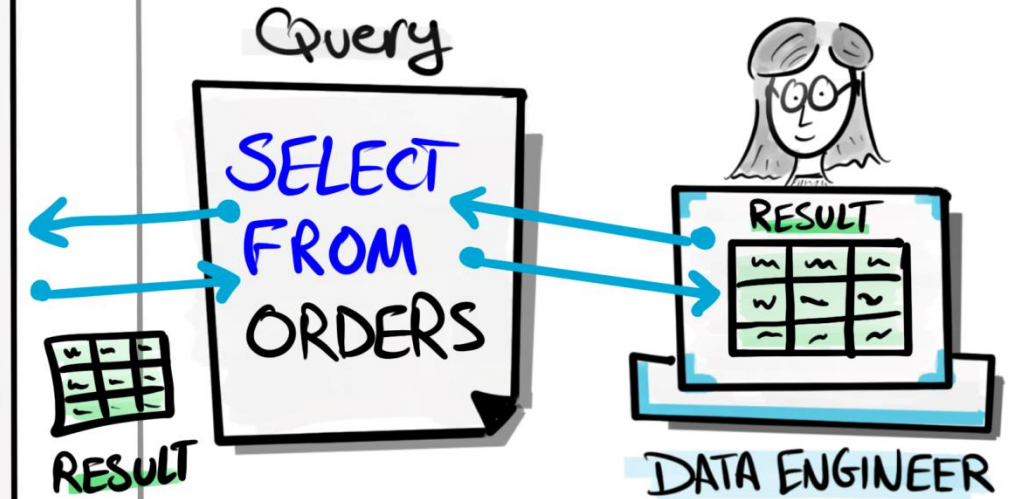
DB Stress

Security

SERVER



CLIENT



?

NO!

Two blue arrows point from the 'SALES' table in the cache to the 'DATABASE ENGINE'. The first arrow is labeled with a question mark '?'. The second arrow is labeled with 'NO!' in red text.

?

A blue arrow points from the 'ORDERS' table in the disk to the 'DATABASE ENGINE'. It is labeled with a question mark '?'.

RESULT

A small table icon with 3 rows and 3 columns, each cell containing a tilde (~). It is labeled 'RESULT' in green text.



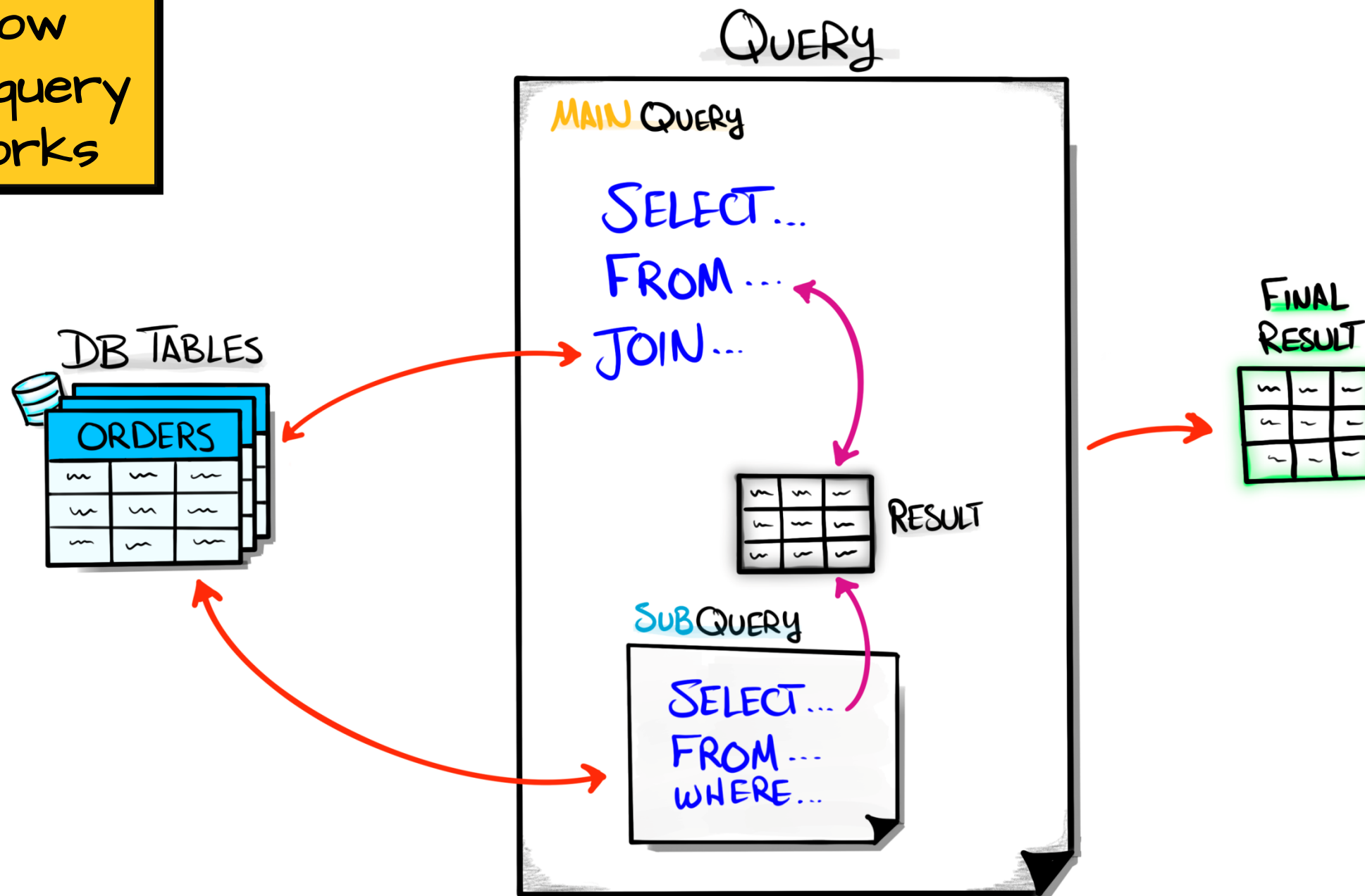
DATA WITH BARAA

Subquery

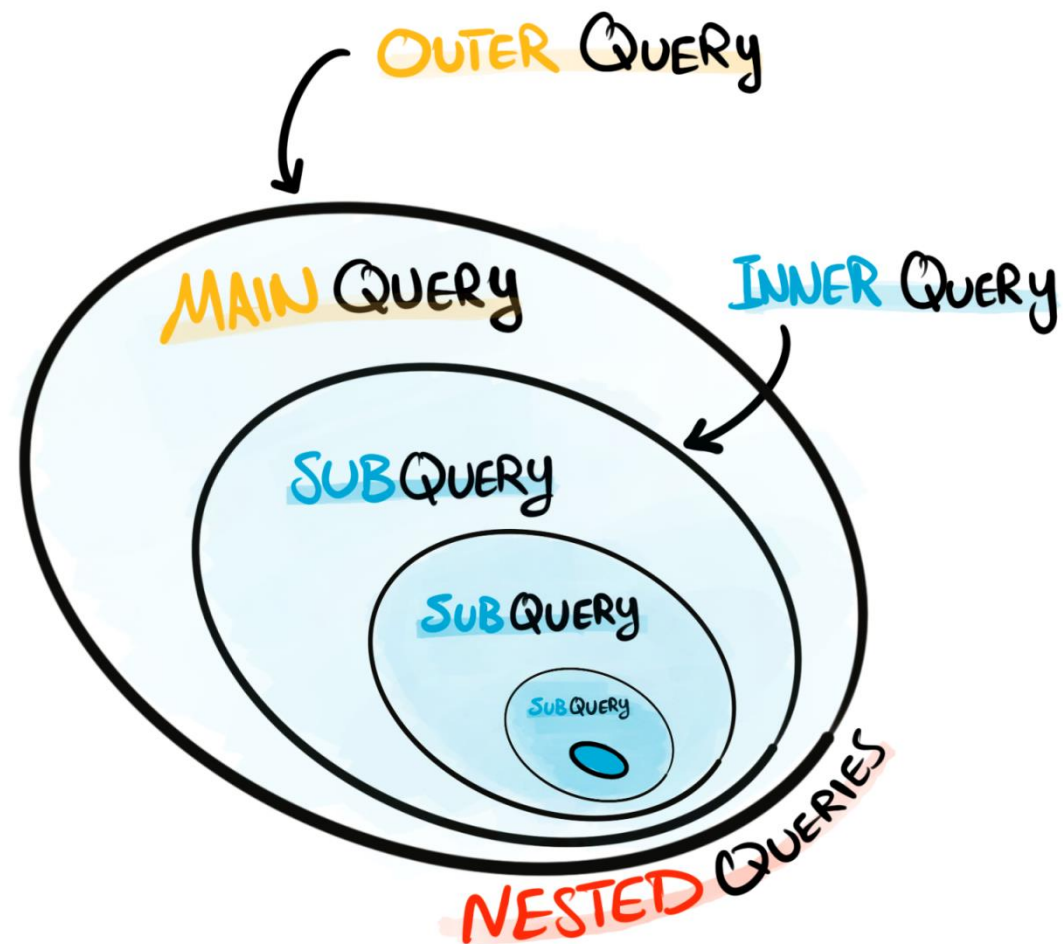
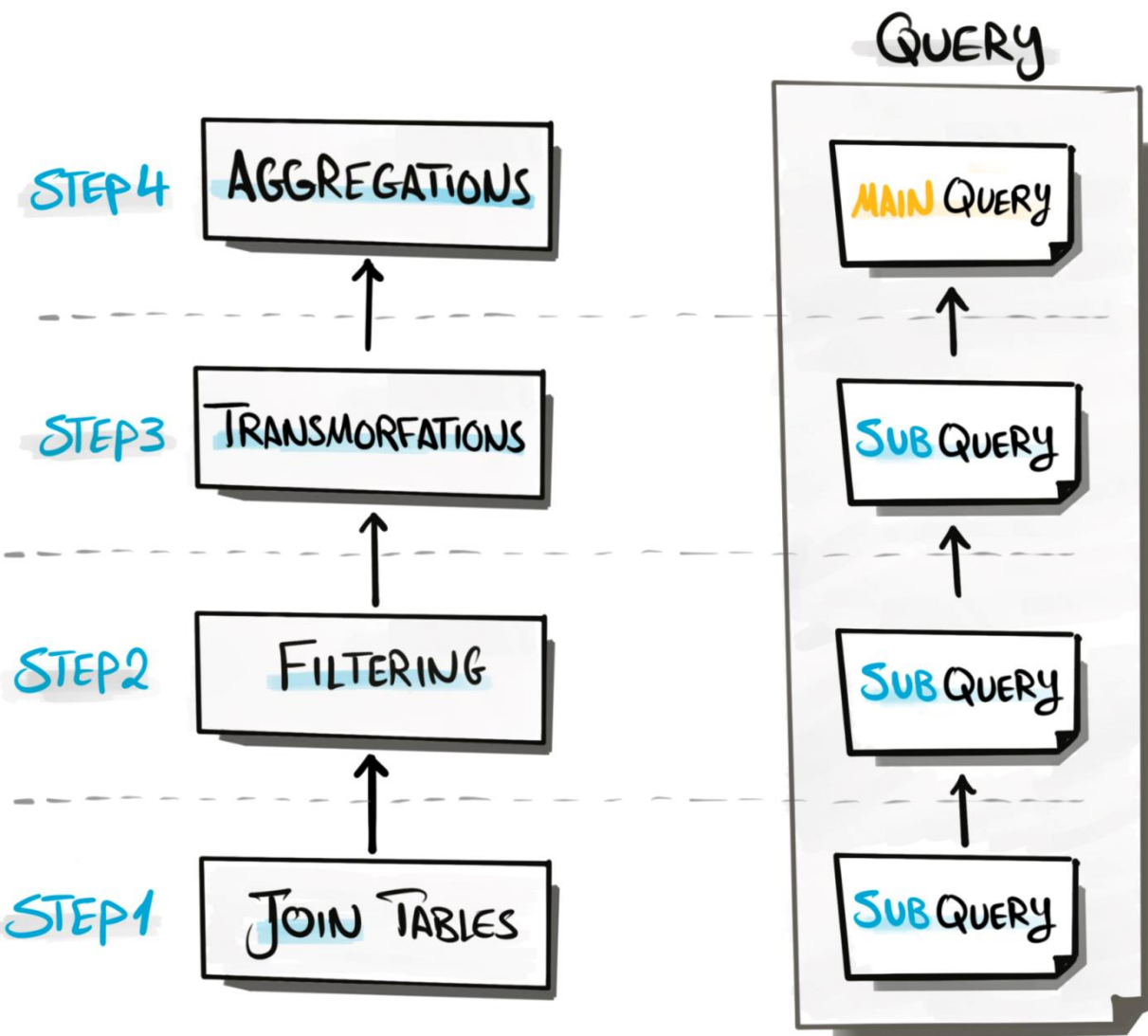
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Subquery

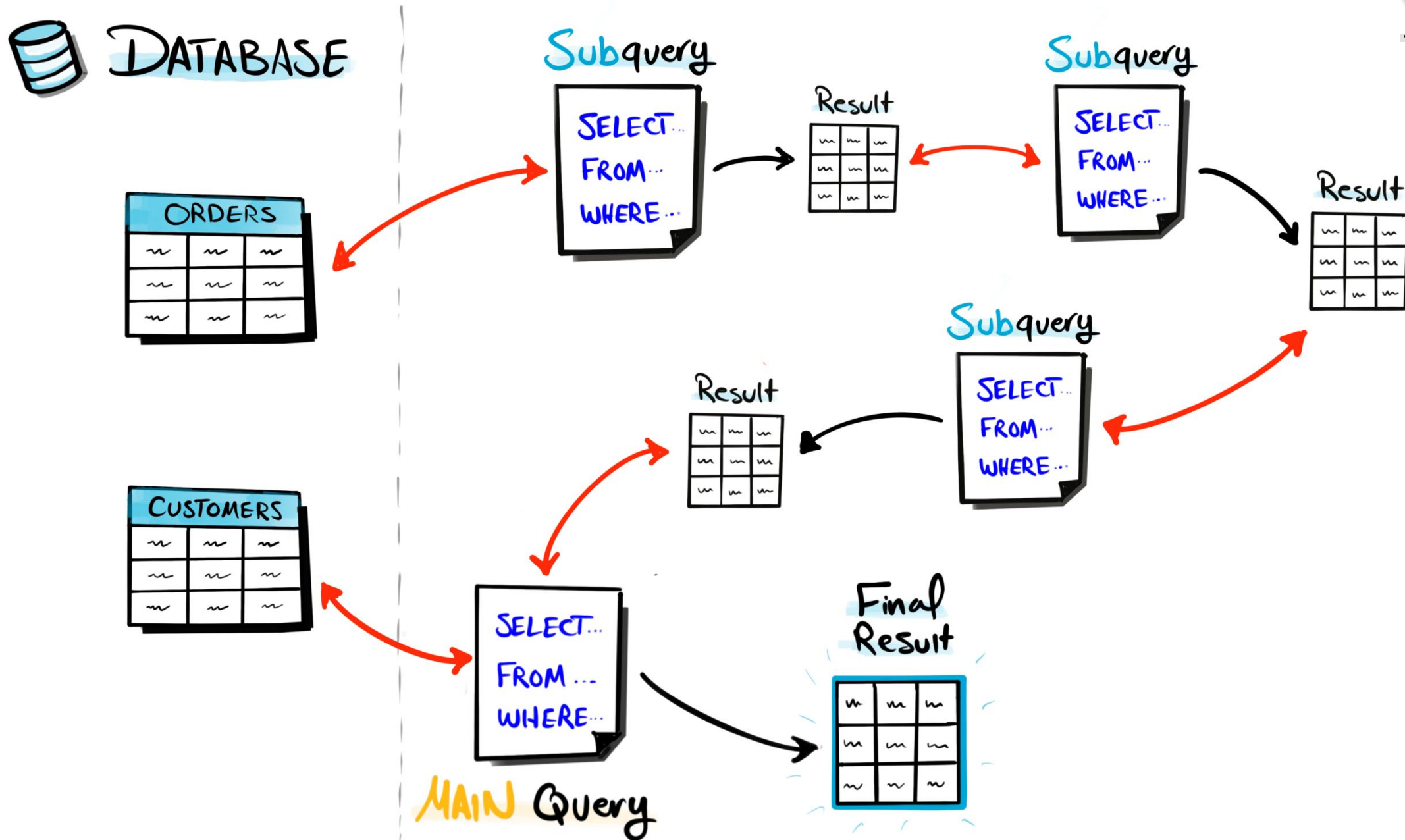


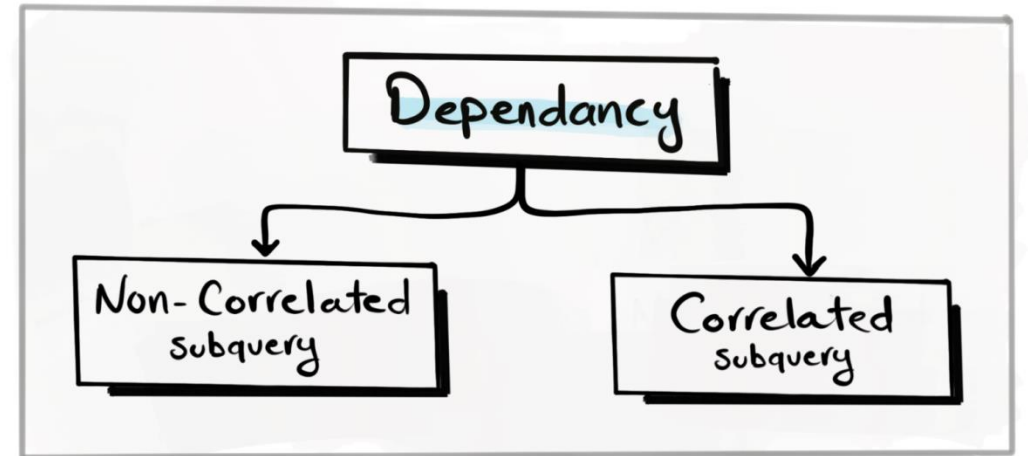
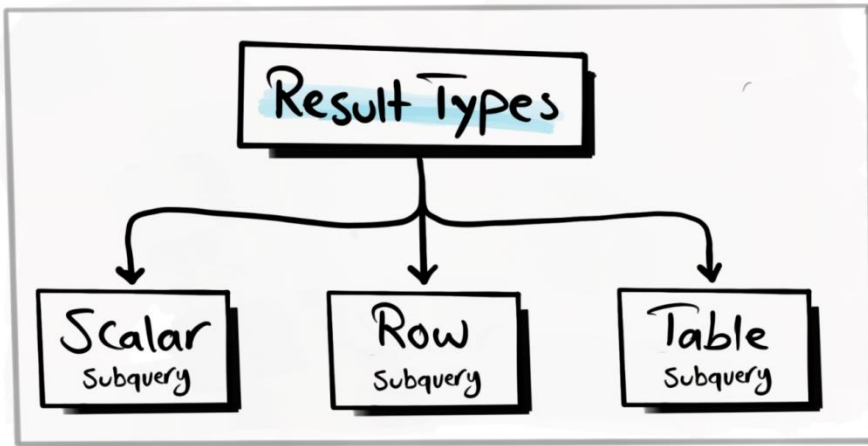
How Subquery Works



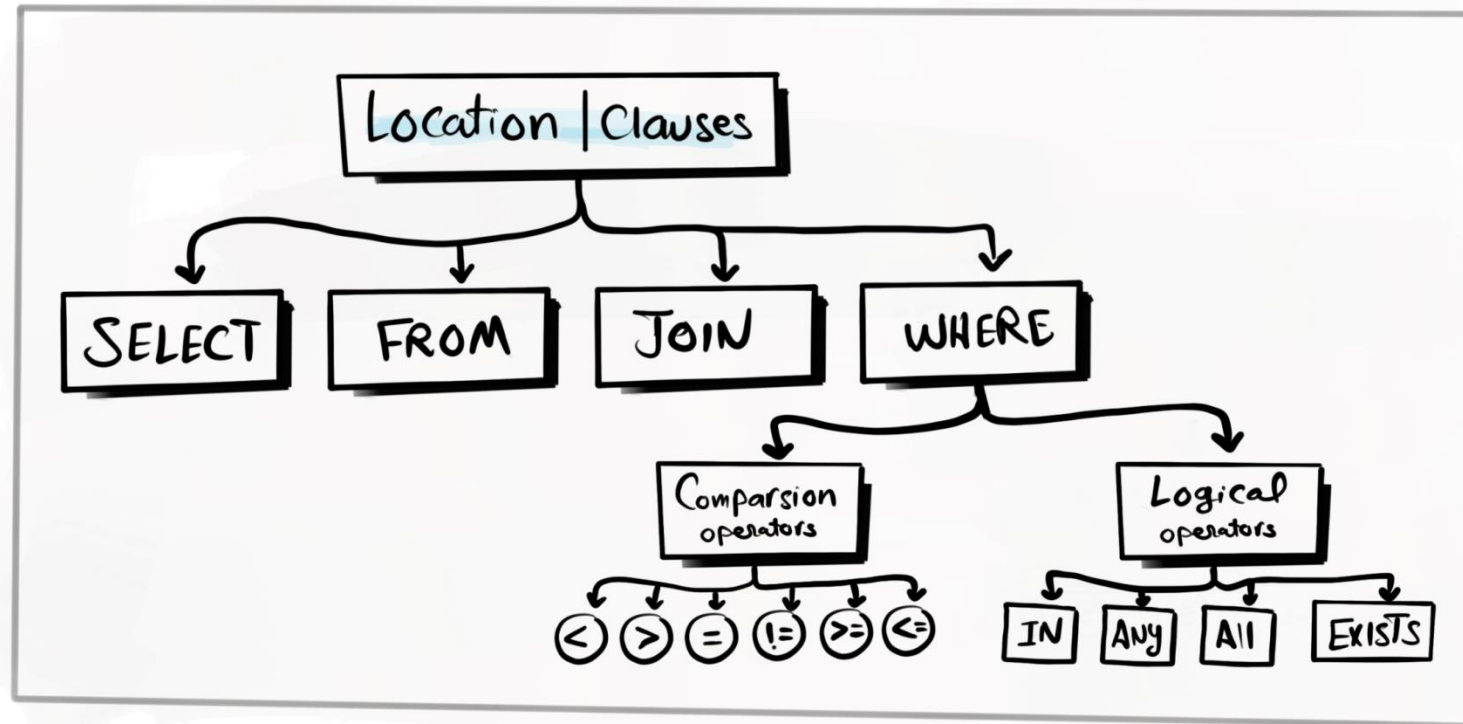
SUBQUERY

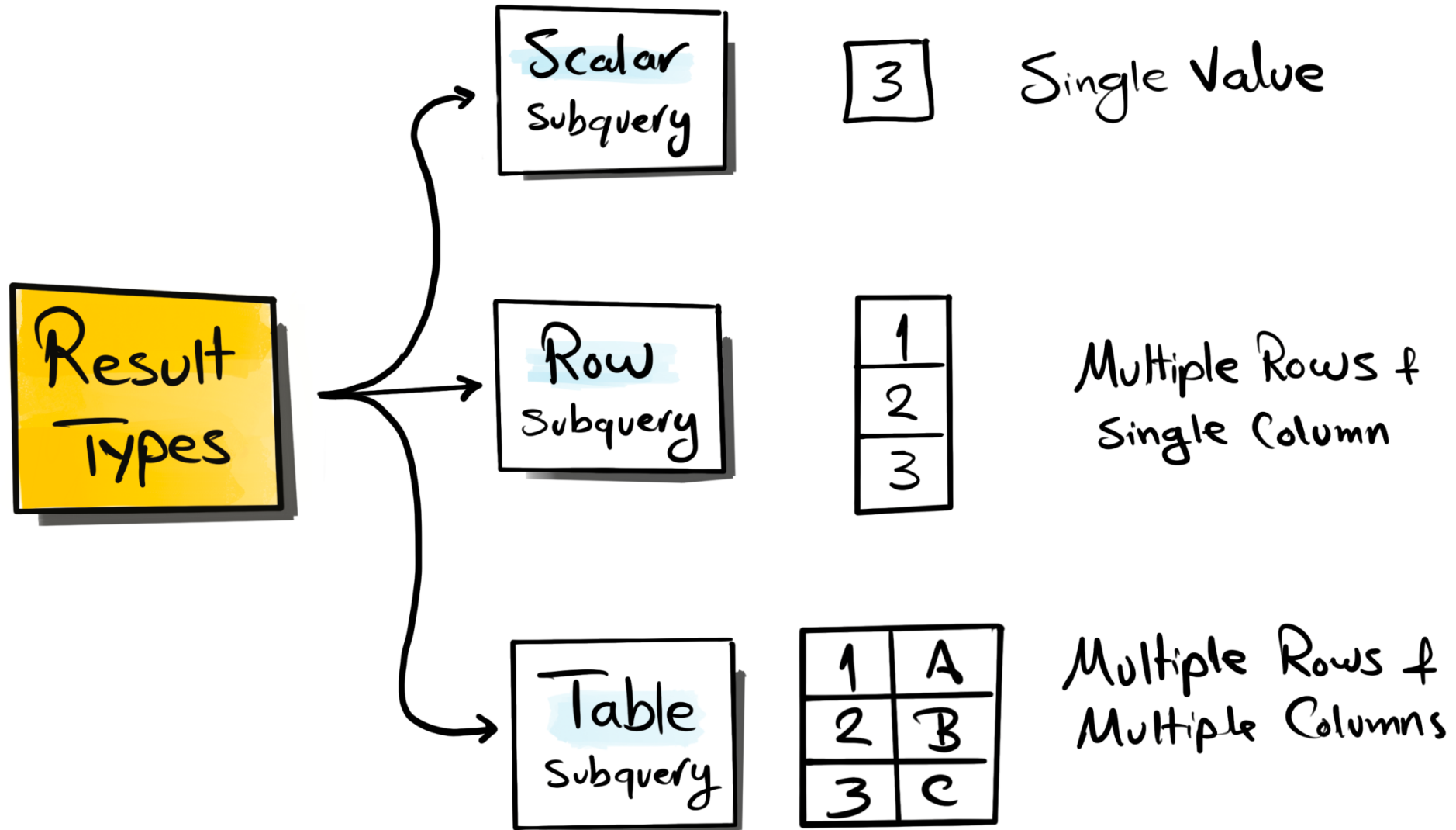






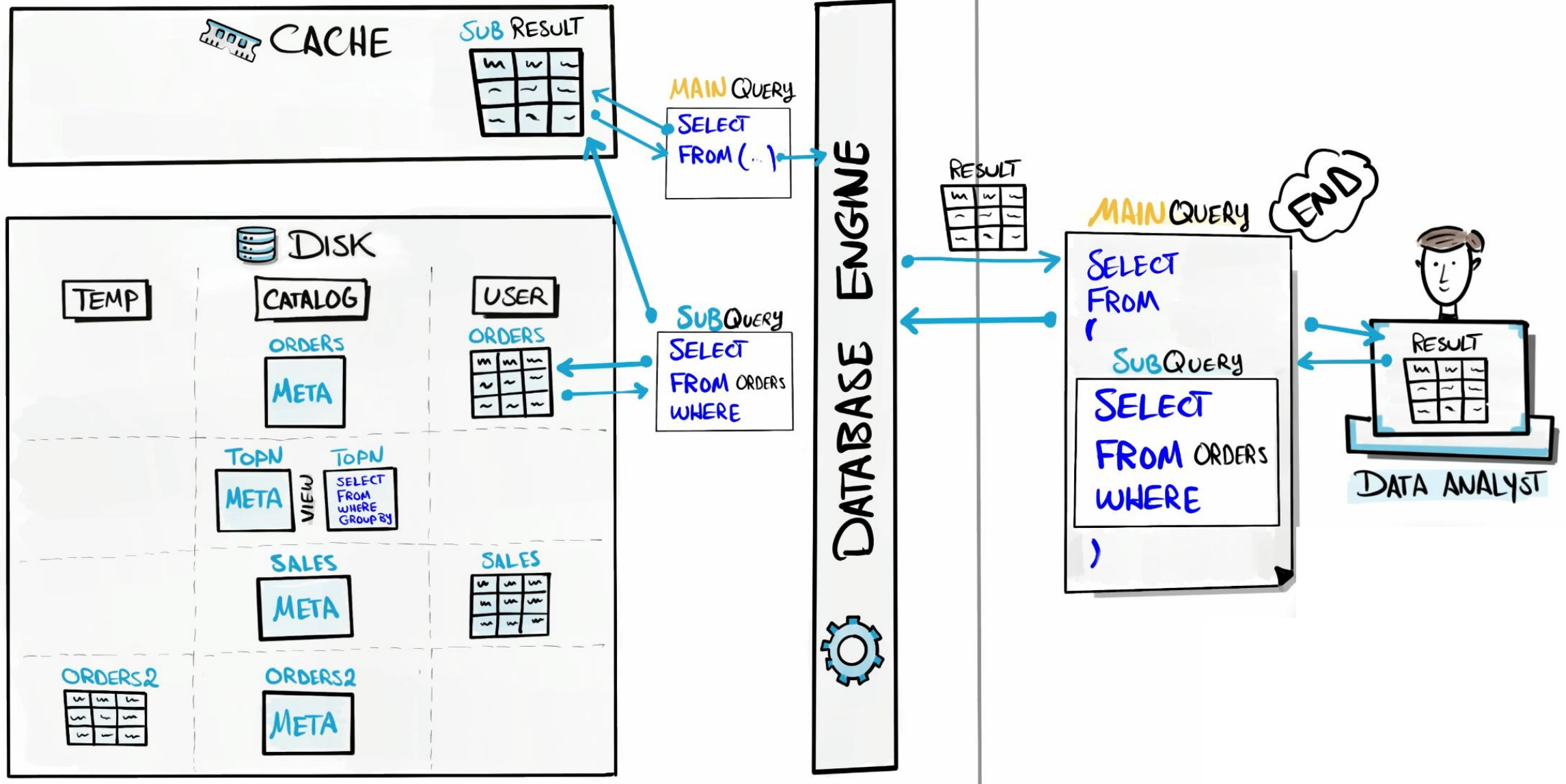
SUBQUERY





SERVER

CLIENT



Subquery in FROM Clause

Main Query

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

Subquery

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 in
FROM Clause



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
```

**FINAL
RESULT**

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

Subquery in SELECT Clause

Main Query

SELECT

Column1,

(**SELECT** column **FROM** table1 **WHERE** condition) **AS** alias

FROM table1

Subquery

Rules

Only Scalar Subqueries are allowed to be used

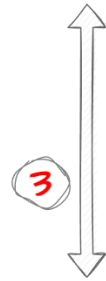
Subquery in
SELECT
Clause

Products		
ProductID	Product	Price
101	Bottle	10
102	Tire	15
103	Socks	20
104	Caps	25
105	Gloves	30

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

Original Query

```
SELECT
    ProductID,
    Product,
    (SELECT COUNT(*)
     FROM Sales.Orders) Total
FROM Sales.Products
```



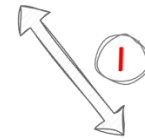
Main Query

```
SELECT
    ProductID,
    Product,
    (10) Total
FROM Sales.Products
```



ProductID	Product	Total
101	Bottle	10
102	Tire	10
103	Socks	10
104	Caps	10
105	Gloves	10

FINAL
RESULT

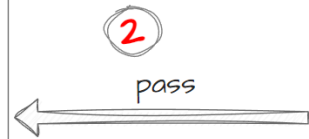


Subquery

```
SELECT COUNT(*)
FROM Sales.Orders
```



COUNT
10



Subquery in WHERE Clause Comparison Operators

Main Query

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

Subquery

Rules

Only Scalar Subqueries are allowed to be used

Subquery in WHERE Clause Comparison Operators

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

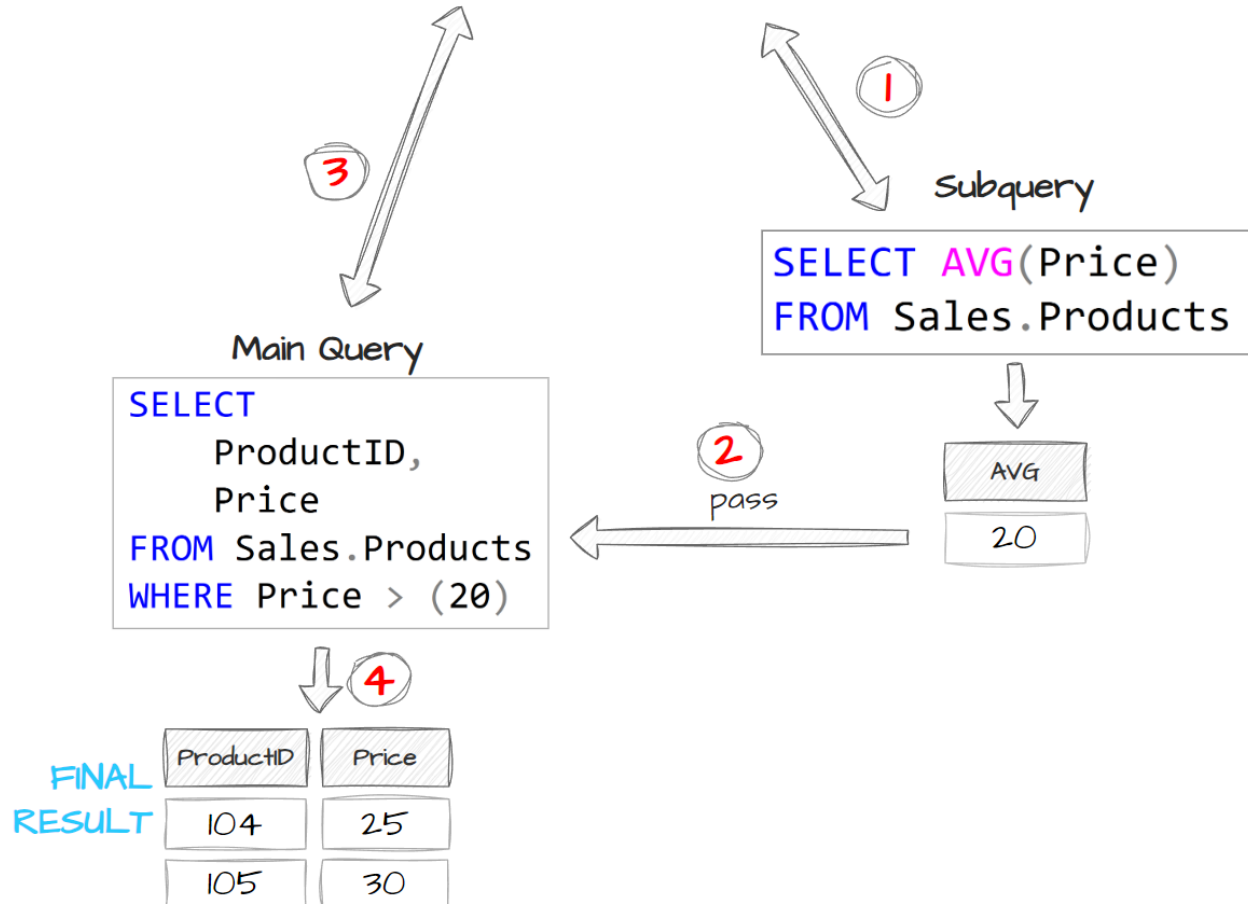
Products			
ProductID	Product	Category	Price
101	Bottle	Accessories	10
102	Tire	Accessories	15
103	Socks	Clothing	20
104	Caps	Clothing	25
105	Gloves	Clothing	30

Original Query

```

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

Subquery in
WHERE Clause
Comparison Operators



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	<code>WHERE Sales IN (SELECT ...)</code>
NOT IN	Checks if a value does not matches any value in a list	<code>WHERE Sales NOT IN (SELECT ...)</code>
EXISTS	Checks if subquery returns any rows	<code>WHERE EXISTS (SELECT ...)</code>
NOT EXISTS	Checks if subquery returns no rows	<code>WHERE NOT EXISTS (SELECT ...)</code>
ANY	Returns true if a value matches any value in a list.	<code>WHERE Sales < ANY (SELECT ...)</code>
ALL	Returns true if a value matches all values in a list.	<code>WHERE Sales > ALL (SELECT ...)</code>

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

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
In Operator



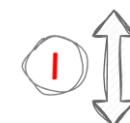
 Main Query

```

SELECT
  OrderID,
  CustomerID,
  Sales
FROM Sales.Orders
WHERE CustomerID IN (1,4)
  
```

FINAL
RESULT


OrderID	CustomerID	Sales
3	1	20
4	1	60
7	1	60
8	4	90

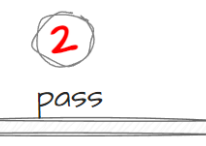


 Subquery

```

SELECT CustomerID
FROM Sales.Customers
WHERE Country = 'Germany'
  
```





 pass

CustomerID
1
4

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

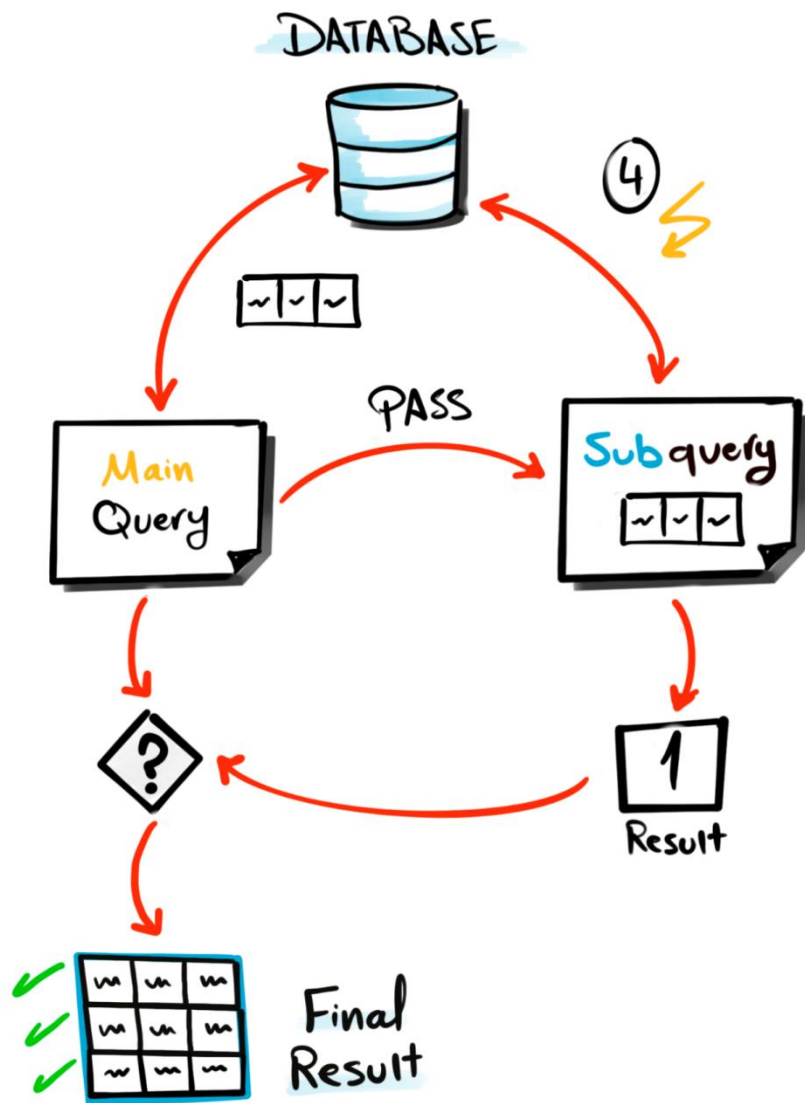
NON-CORRELATED SUBQUERY

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

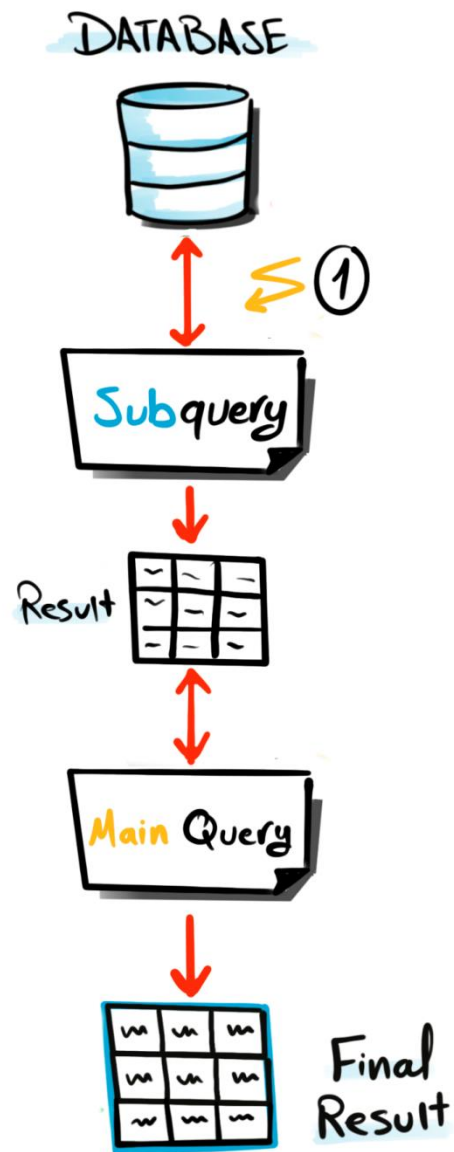
CORRELATED SUBQUERY

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

Correlated Subquery



Non-Correlated Subquery



Non-Correlated Subquery

Correlated Subquery

Definition

Subquery is **independent** of the main query

Subquery is **dependent** of the main query

Execution

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

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

Can be executed on its Own

Can't be executed on its Own.

Easy to use

Easier to read

Harder to read and more **complex**

Performance

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

Executed multiple times leads to **bad** Performance

Usage

Static Comparisons, Filtering with Constants

Row-by-Row Comparisons, Dynamic Filtering

Correlated Subquery in WHERE Clause EXISTS Operator

Main Query

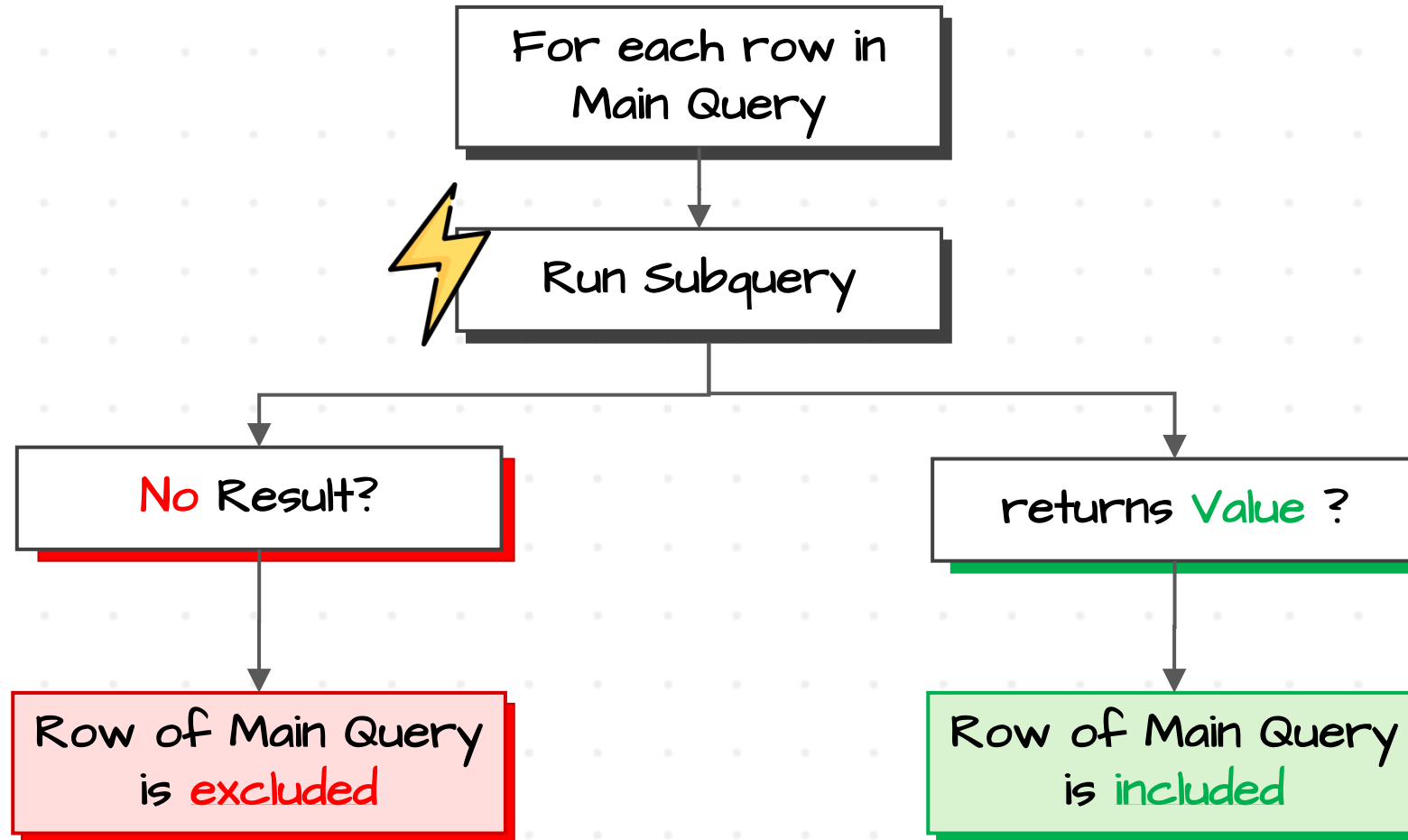
```
SELECT column1, column2,...
```

```
FROM Table2
```

```
WHERE EXISTS ( SELECT 1  
                FROM Table1  
                WHERE Table1.ID = Table2.ID )
```

Subquery

How EXISTS Works?



JOINS

SUBQUERIES

Syntax

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

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

Readability

Not easy to read & maintain

easy to read & maintain

Performance

Fast

Slow

Duplicate

May **lead to duplicate**

Safer
no risk to have duplicates

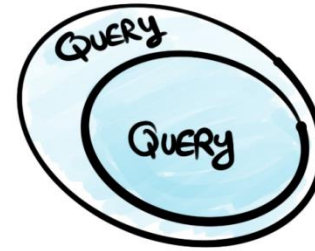
Best Practices

Useful with **Larg tables**

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 -



DATA WITH BARAA

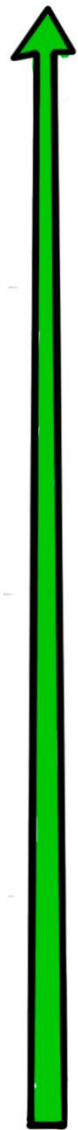
CTE

Common Table Expression

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



QUERY



MAIN QUERY

SELECT...
FROM...
JOIN...

①

~	~	~
~	~	~
~	~	~

RESULT

SUBQUERY

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

Subqueries

QUERY



CTE QUERY

SELECT..
FROM..
WHERE...

SALES		
~	~	~
~	~	~
~	~	~

VIRTUAL
TABLE

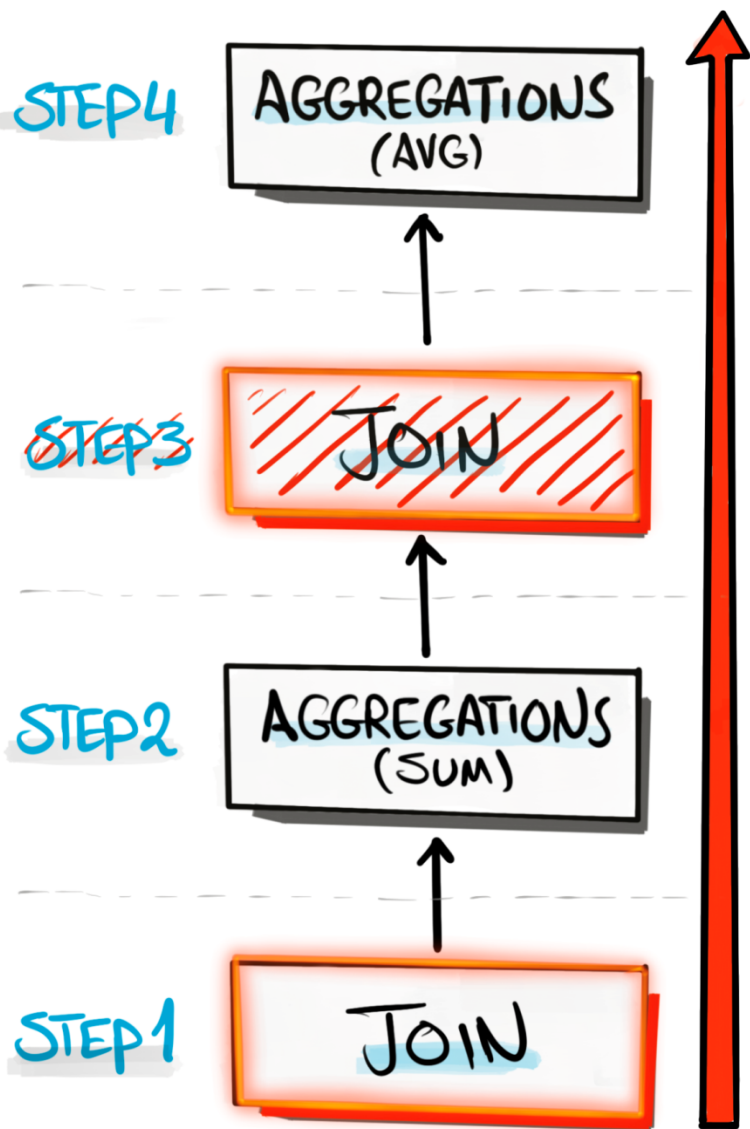
MAIN QUERY

SELECT...
FROM..
JOIN..
JOIN...

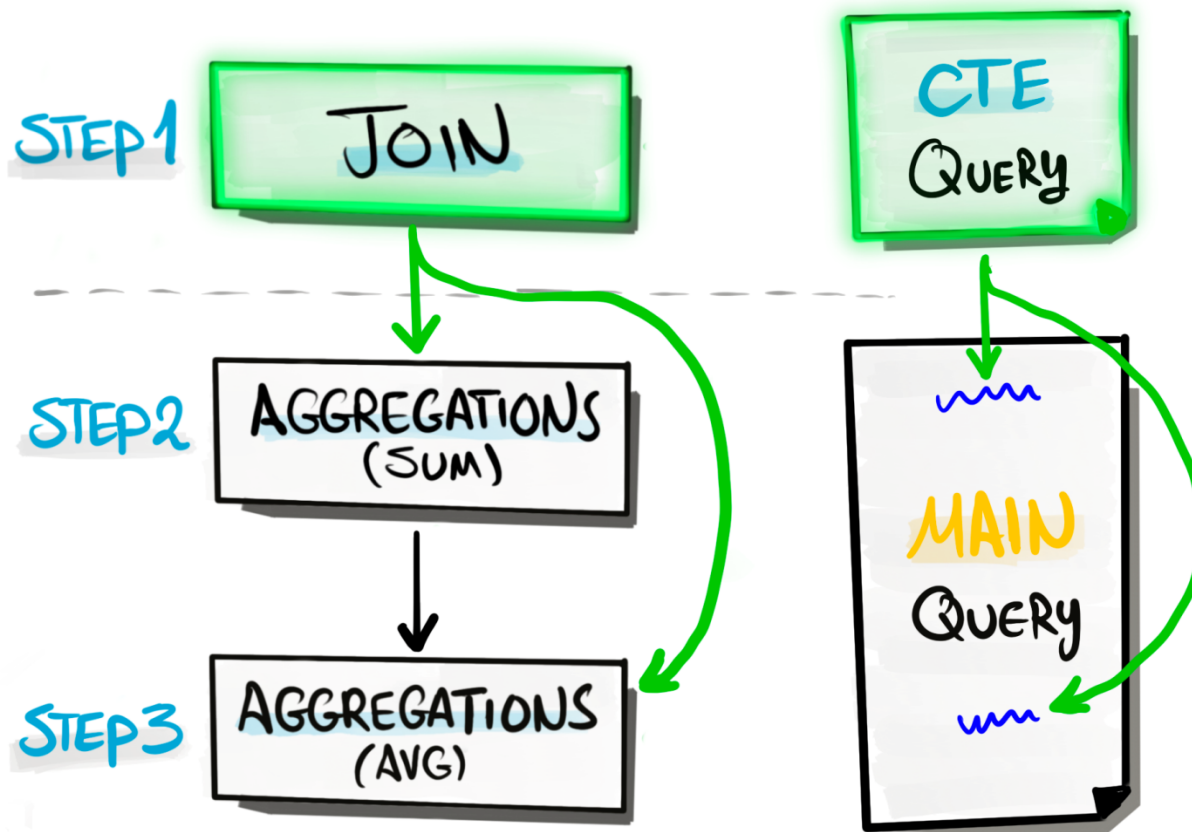
②

CTE

SUBQUERY



CTE



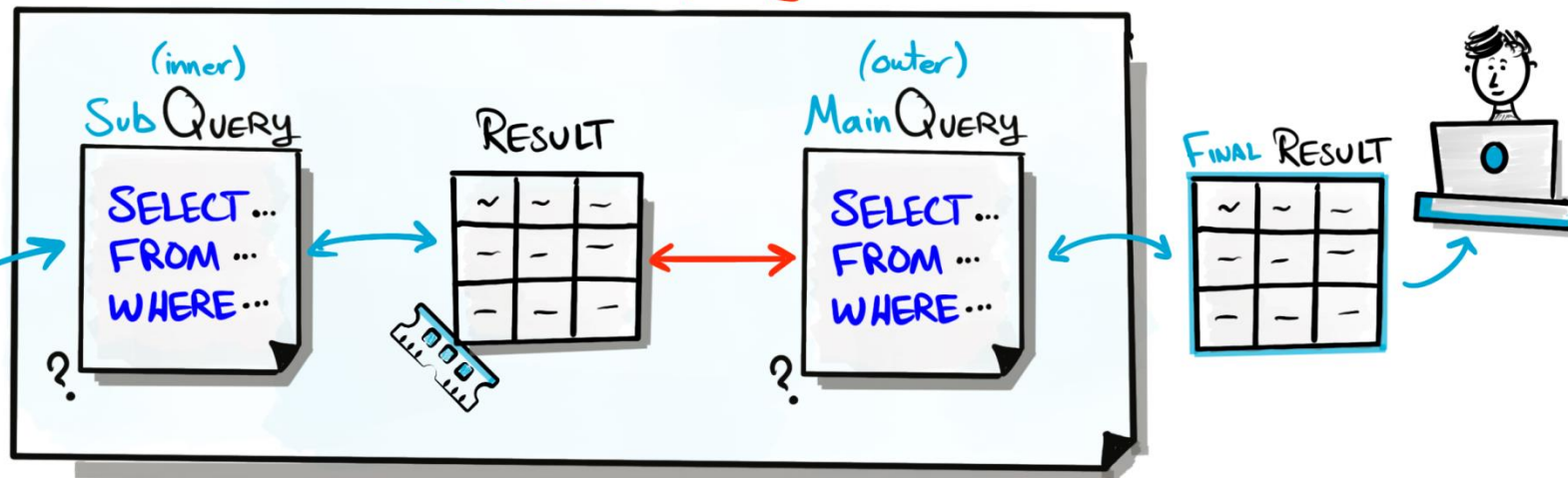


DATABASE

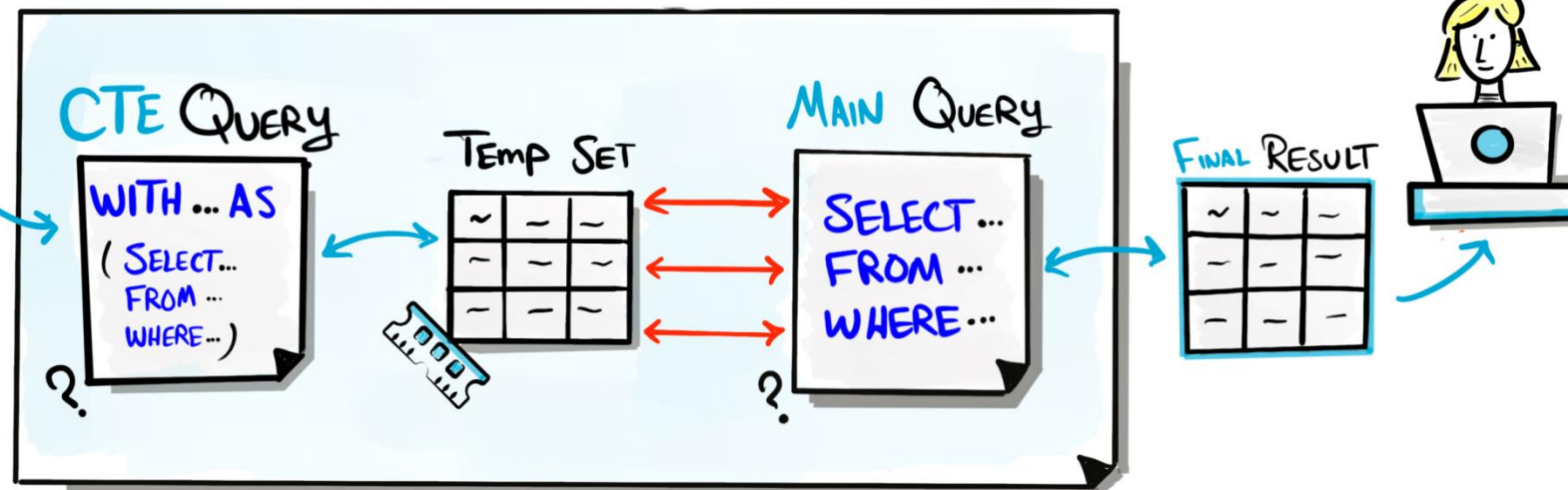
TABLES

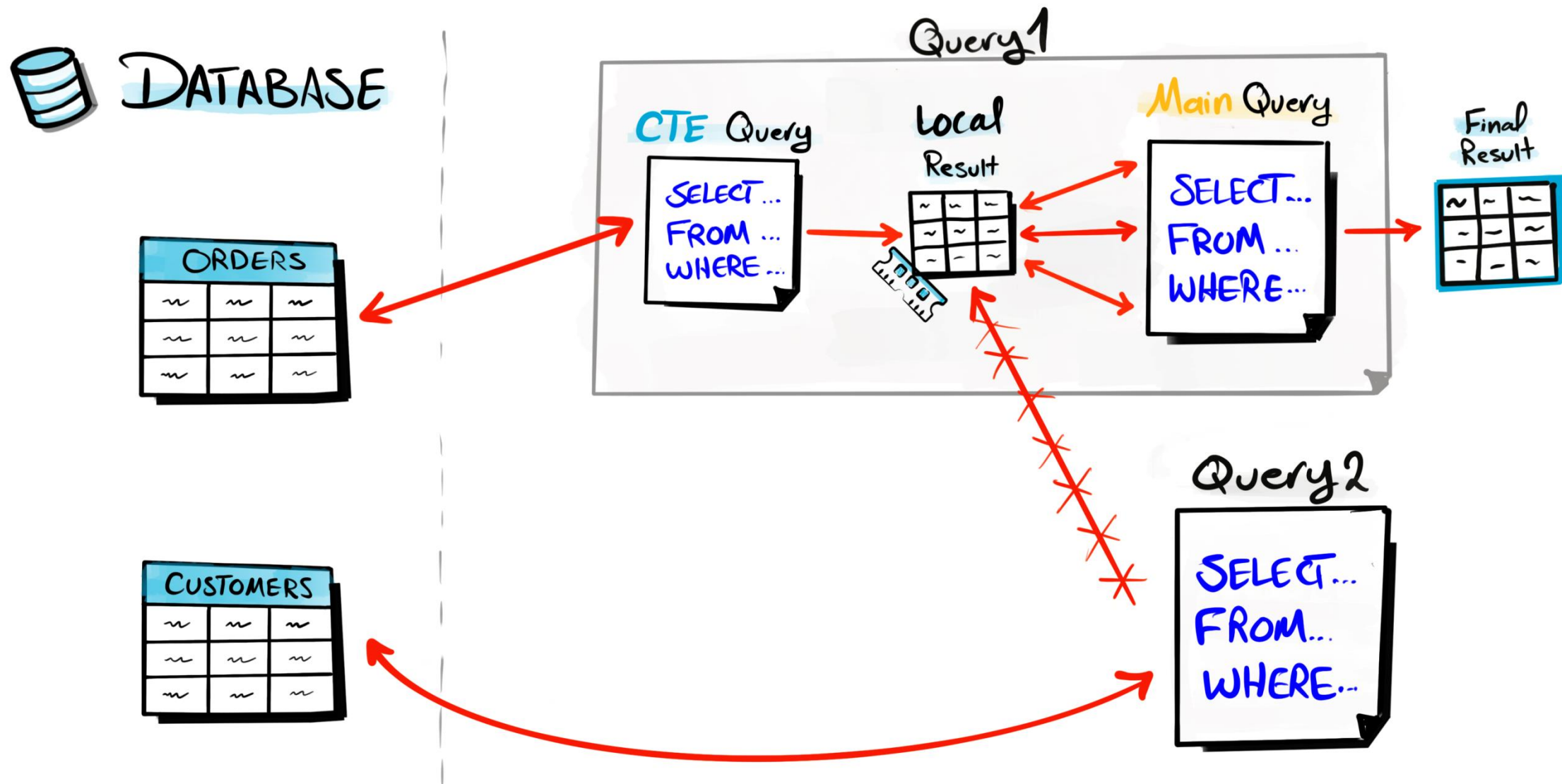
~	~	~	~
-	-	-	-
-	-	-	-
-	-	-	-

SQL SUBQUERY



SQL CTE





QUERY

 READABILITY

 MODULARITY

 REUSABILITY

CTE_Top_Customers

```
SELECT ~~~~~  
FROM customer  
WHERE ~~~~~
```

CTE_Top_Products

```
SELECT ~~~~~  
FROM Products  
JOIN ~~~~~
```

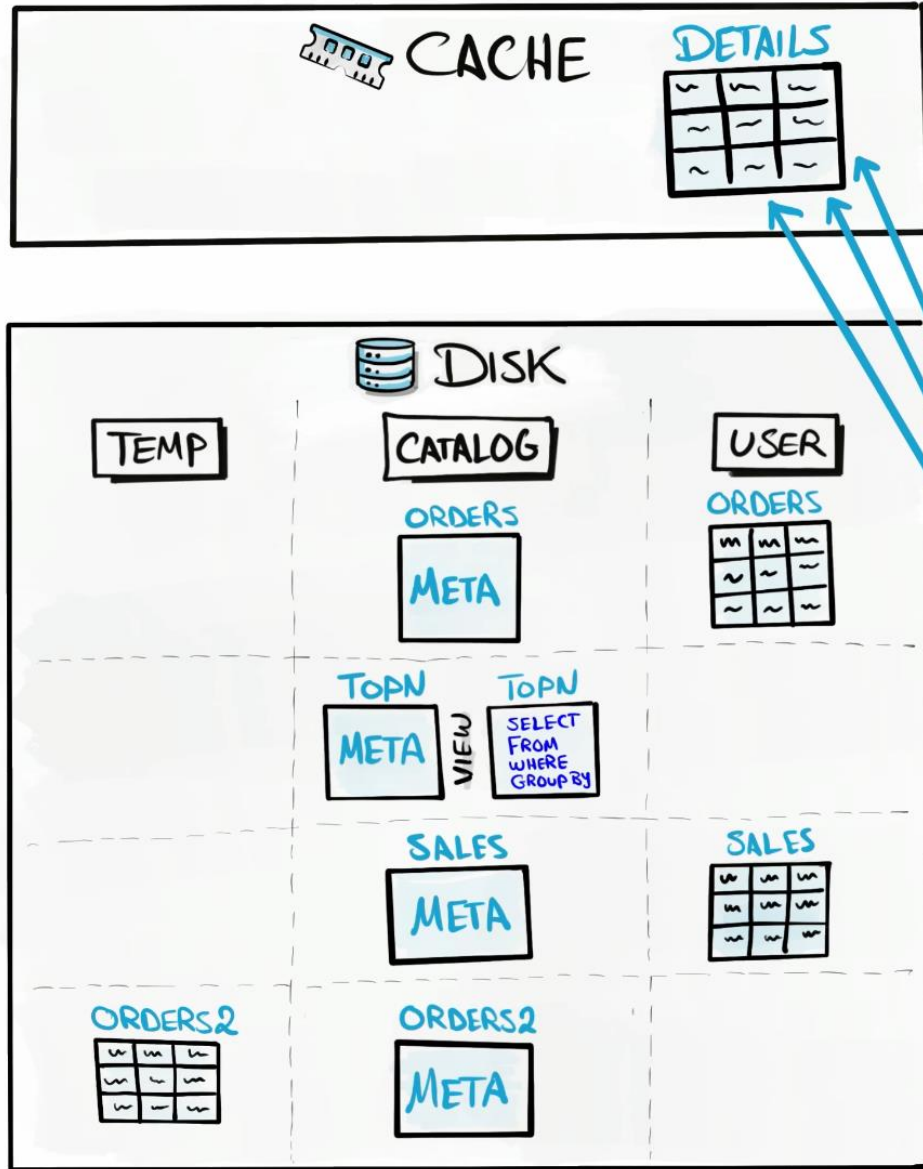
CTE_Daily_Revenue

```
SELECT ~~~~~  
FROM Orders  
JOIN ~~~~~
```

MAIN
QUERY

```
SELECT  
~~~~~  
~~~~~
```

SERVER



MAIN Query
SELECT..
FROM ORDERS
JOIN Details
JOIN Details
JOIN Details

DATABASE ENGINE



CLIENT

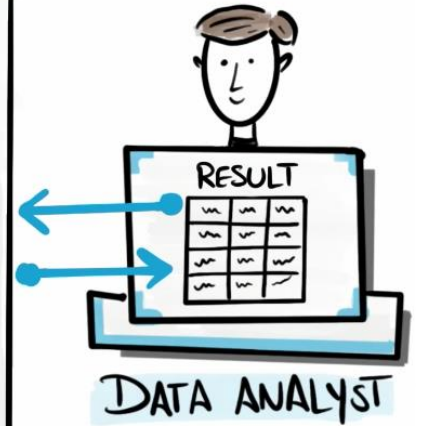
QUERY

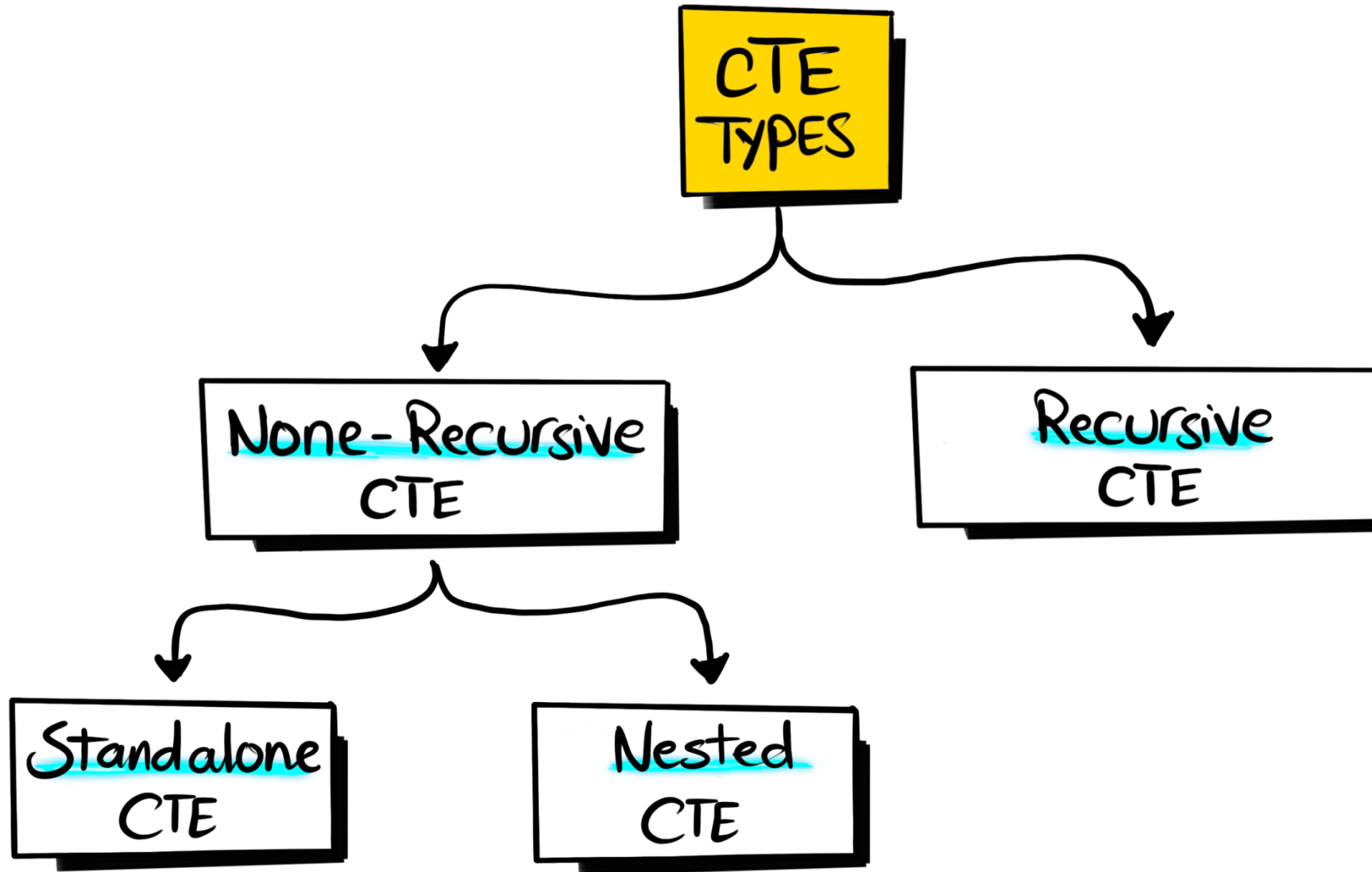
CTE Query

```
WITH Details  
AS (  
  SELECT ...  
)
```

MAIN Query

```
SELECT..  
FROM ORDERS  
JOIN Details  
JOIN Details  
JOIN Details
```





Standalone CTE

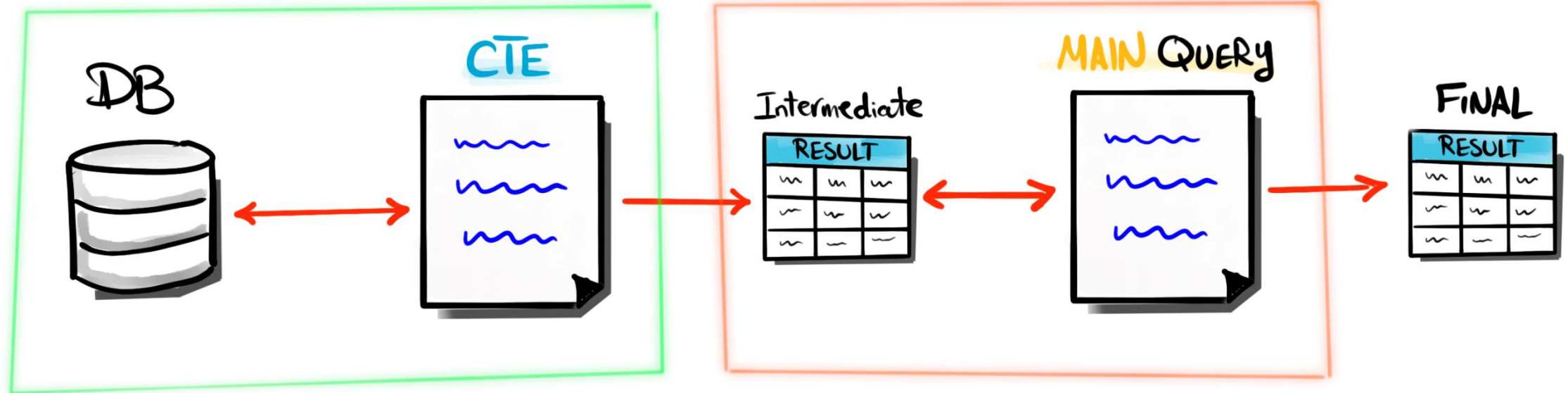
Defined and Used independently.

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

Standalone CTE

Independent

Dependent



Standalone CTE

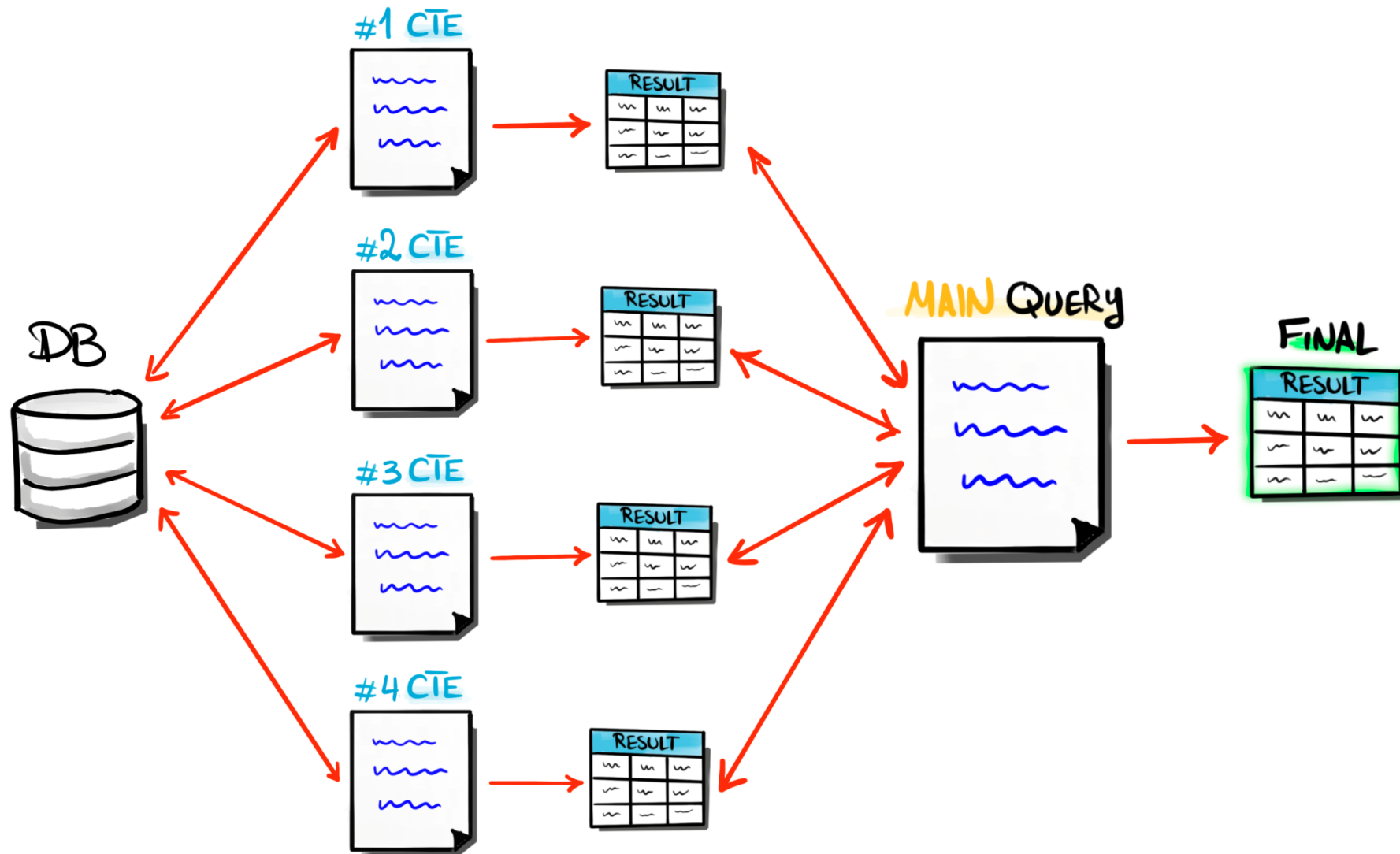
```
WITH CTE-Name AS  
(  
  SELECT ...  
  FROM ...  
  WHERE ...  
)
```

CTE Query
- CTE Definition -

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

Main Query
- CTE Usage -

Multiple CTEs



Multiple CTEs

CTE Query
- CTE Definition -

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

CTE Query
- CTE Definition -

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

Main Query
- CTE Usage -

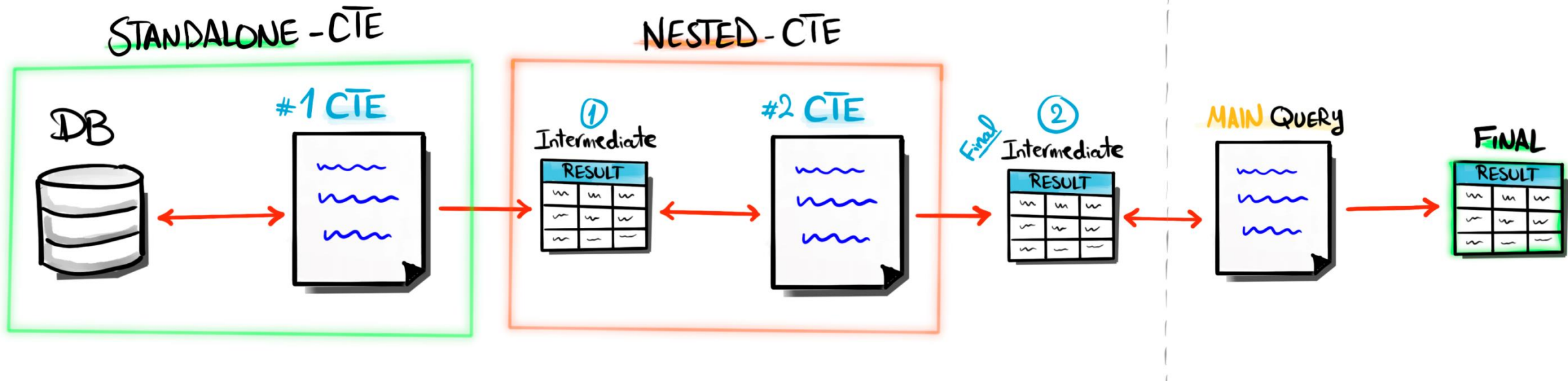
```
SELECT ...  
FROM CTE-Name1  
JOIN CTE-Name2  
WHERE ...
```

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 -



DATA WITH BARAA

BEST PRACTICE

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

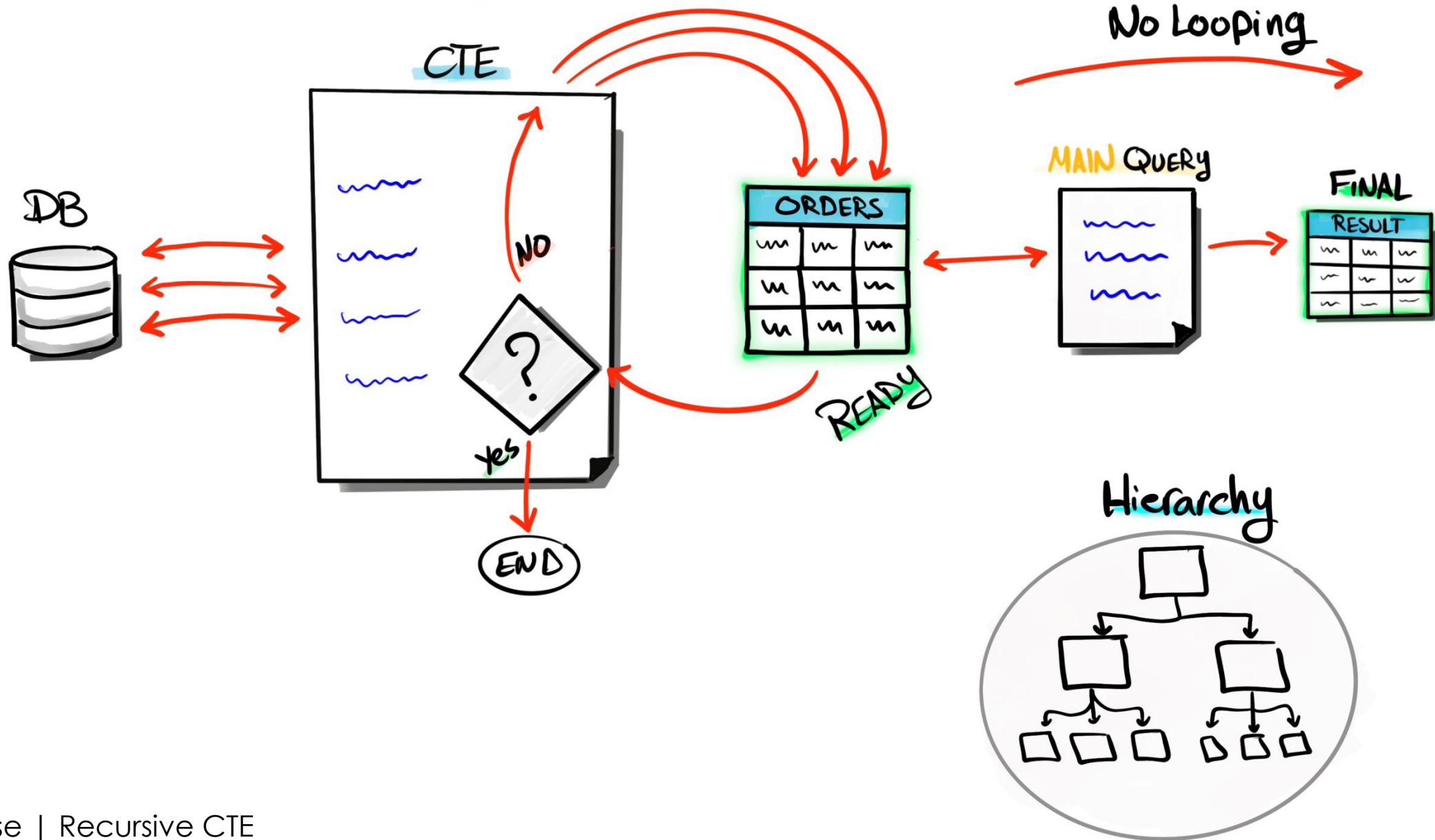
Non-Recursive CTE

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]  
)
```

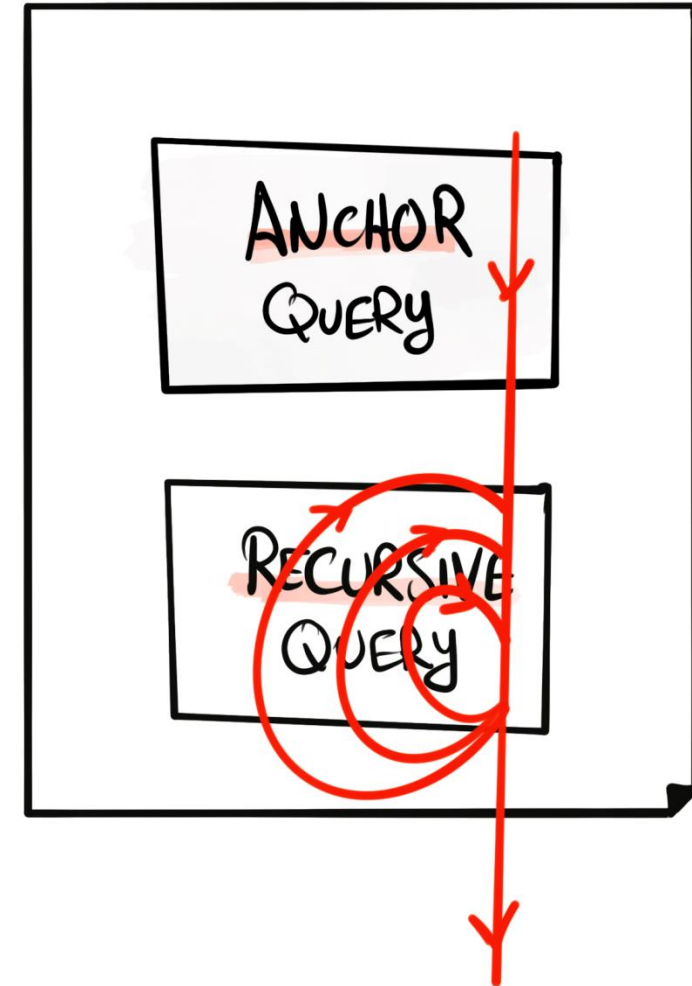
Anchor
Query

Recursive
Query

Main Query
- CTE Usage -

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

CTE Recursive



```
WITH Series AS (
```

```
  SELECT
```

```
  1 AS MyNumber
```

Anchor
Query

```
  UNION ALL
```

```
  SELECT
```

```
  MyNumber + 1
```

```
  FROM Series
```

```
  WHERE MyNumber < 20
```

Recursive
Query

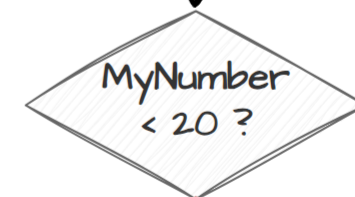
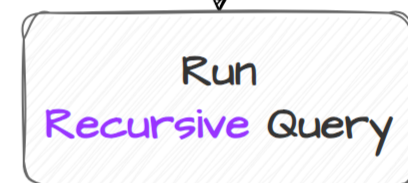
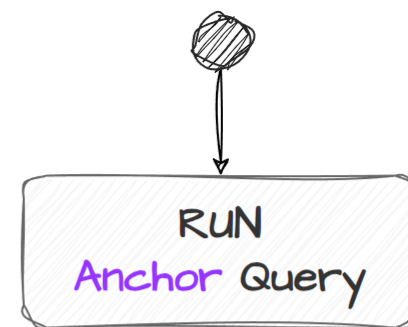
```
)
```

```
SELECT *
```

```
FROM Series
```

MyNumber
1
2
3
...
20

START



TRUE

FALSE



END

```
WITH CTE_Emp_Hierarchy AS
```

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

Anchor
Query

```
    UNION ALL
```

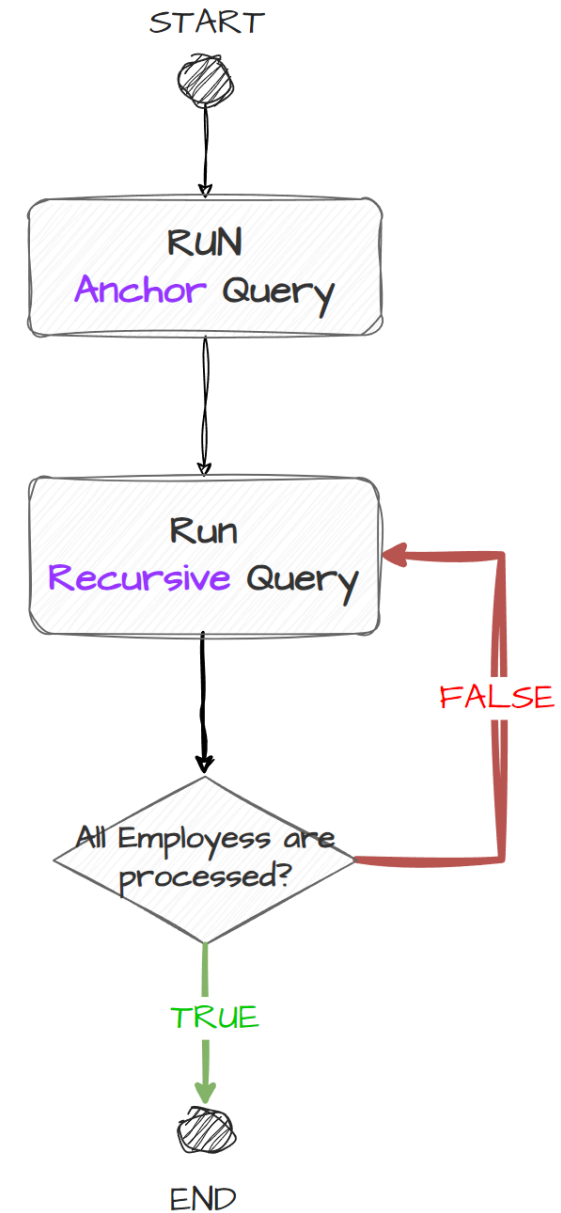
```
    SELECT  
        e.EmployeeID,  
        e.FirstName,  
        e.ManagerID,  
        Level +1  
    FROM Sales.Employees AS e  
    INNER JOIN CTE_Emp_Hierarchy ceh  
    ON e.ManagerID = ceh.EmployeeID
```

Recursive
Query

```
)  
  
SELECT *  
FROM CTE_Emp_Hierarchy
```

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
```

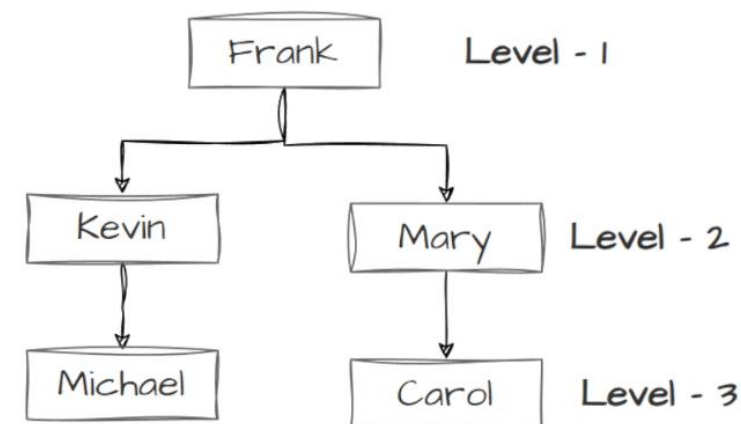
```
    UNION ALL
```

```
    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
```

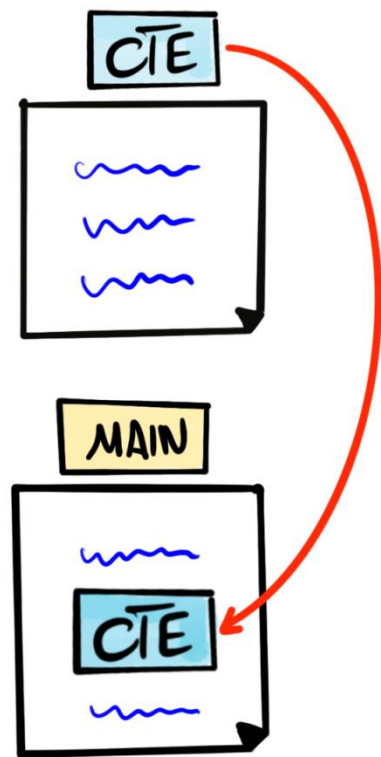
```
)
```

```
SELECT *
FROM CTE_Emp_Hierarchy
```

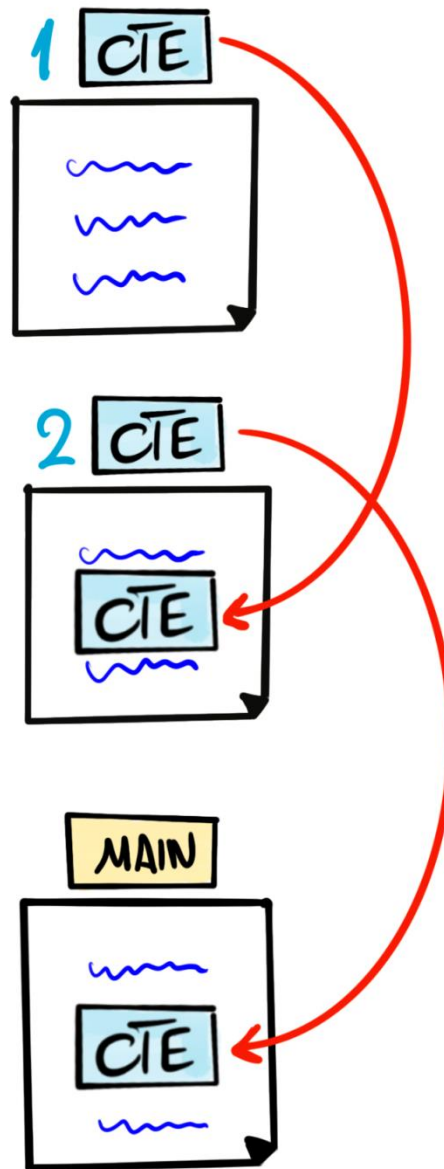
CTE RESULT			
EmployeeID	FirstName	ManagerID	Level
1	Frank	NULL	1
2	Kevin	1	2
3	Mary	1	2
4	Michael	2	3
5	Carol	3	3



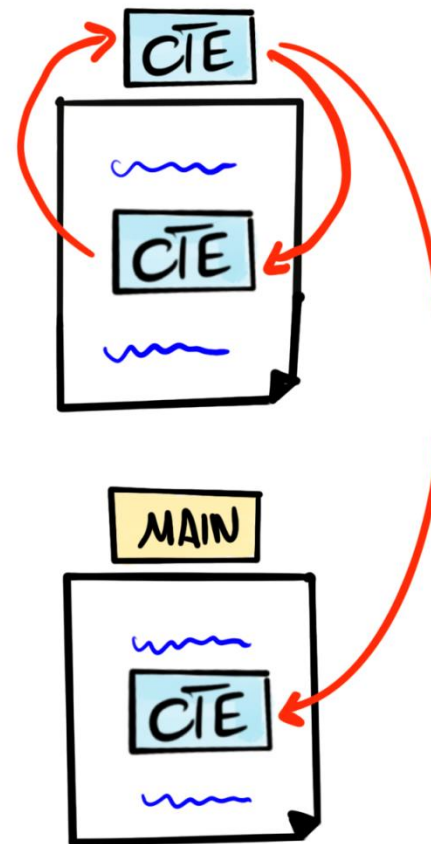
STANDALONE



NESTED



RECURSIVE





DATA WITH BARAA

Views

Database Object

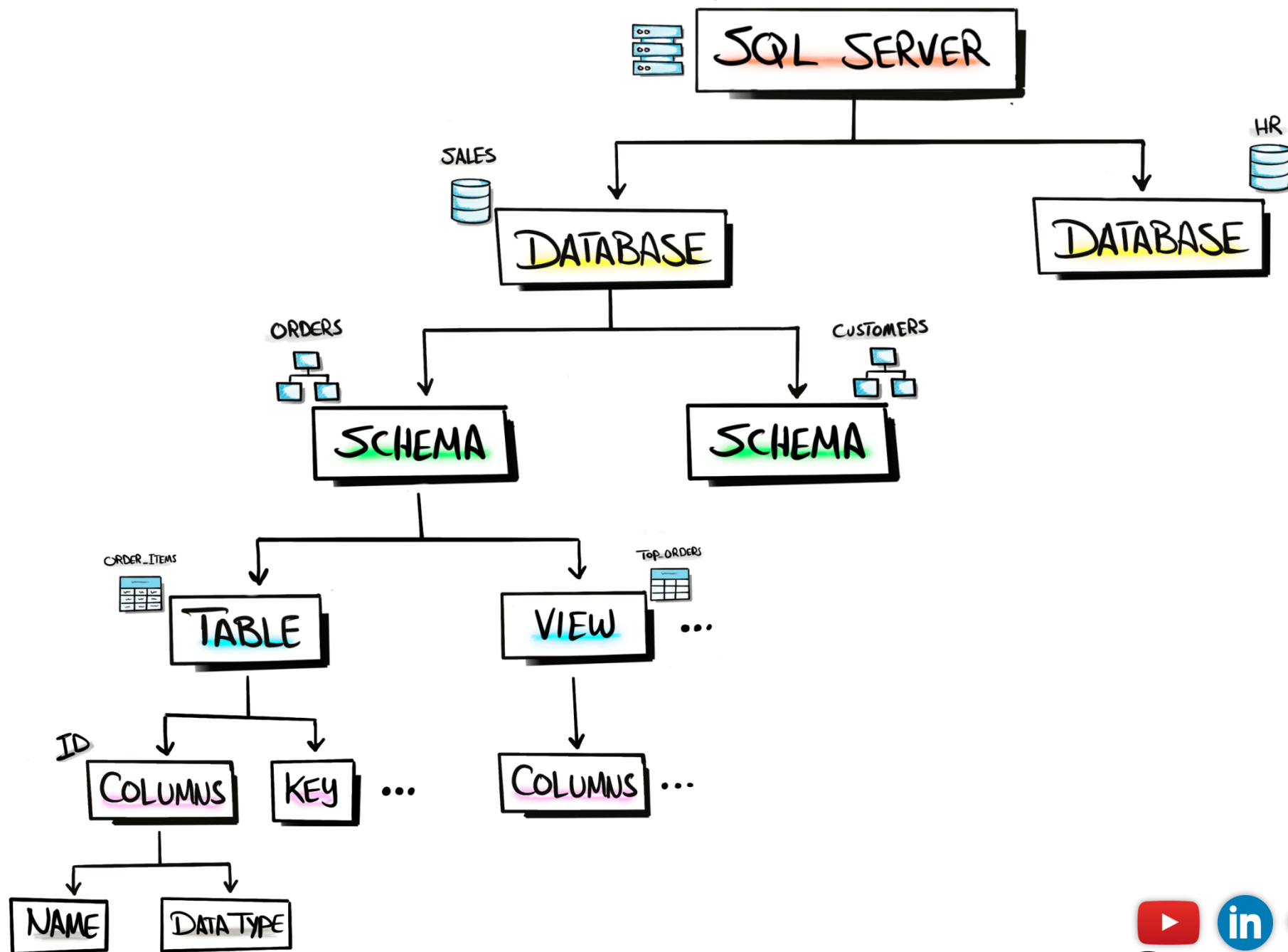
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Views

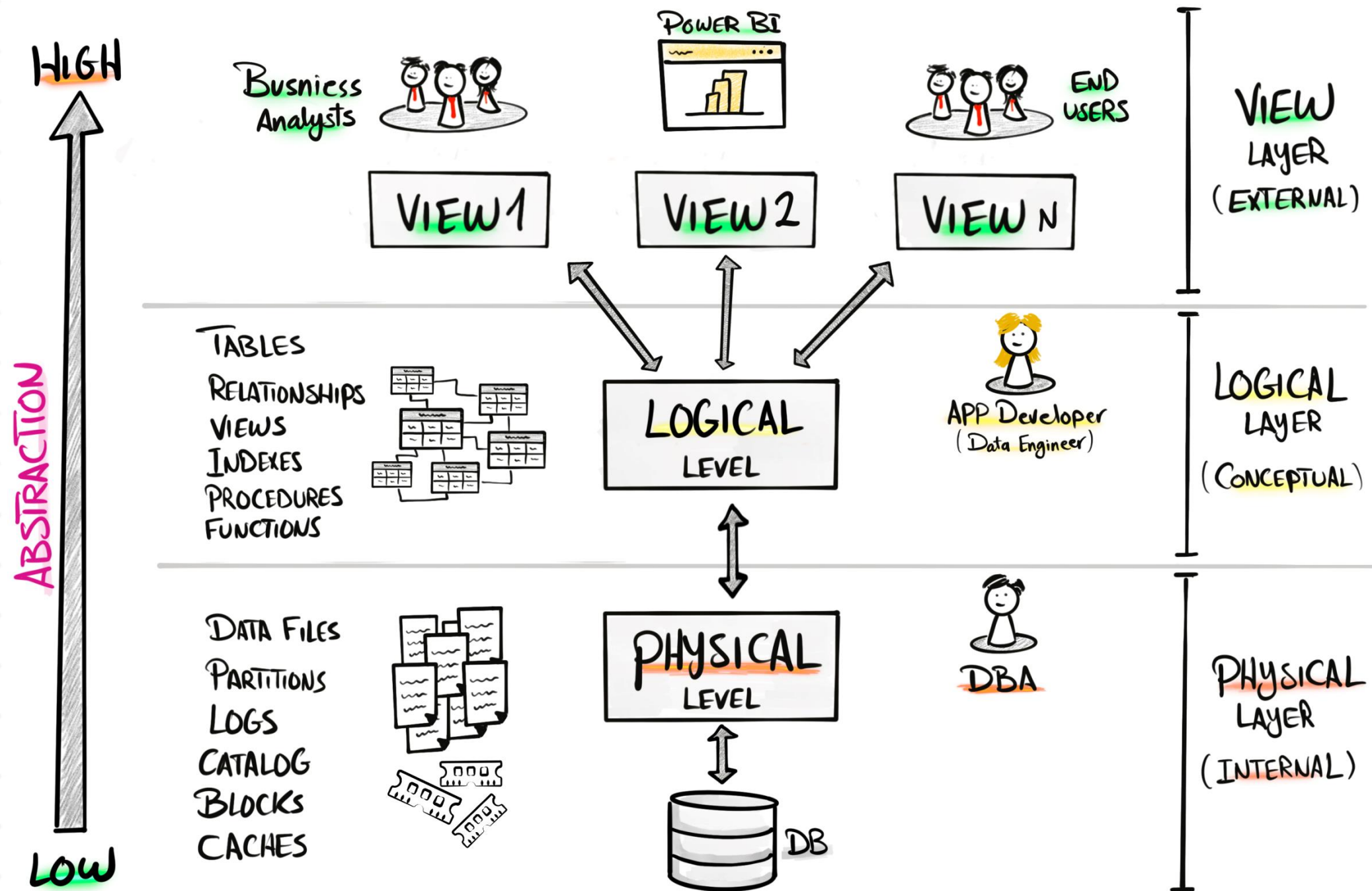


DDL

Data Definition Language

- CREATE
- ALTER
- DROP

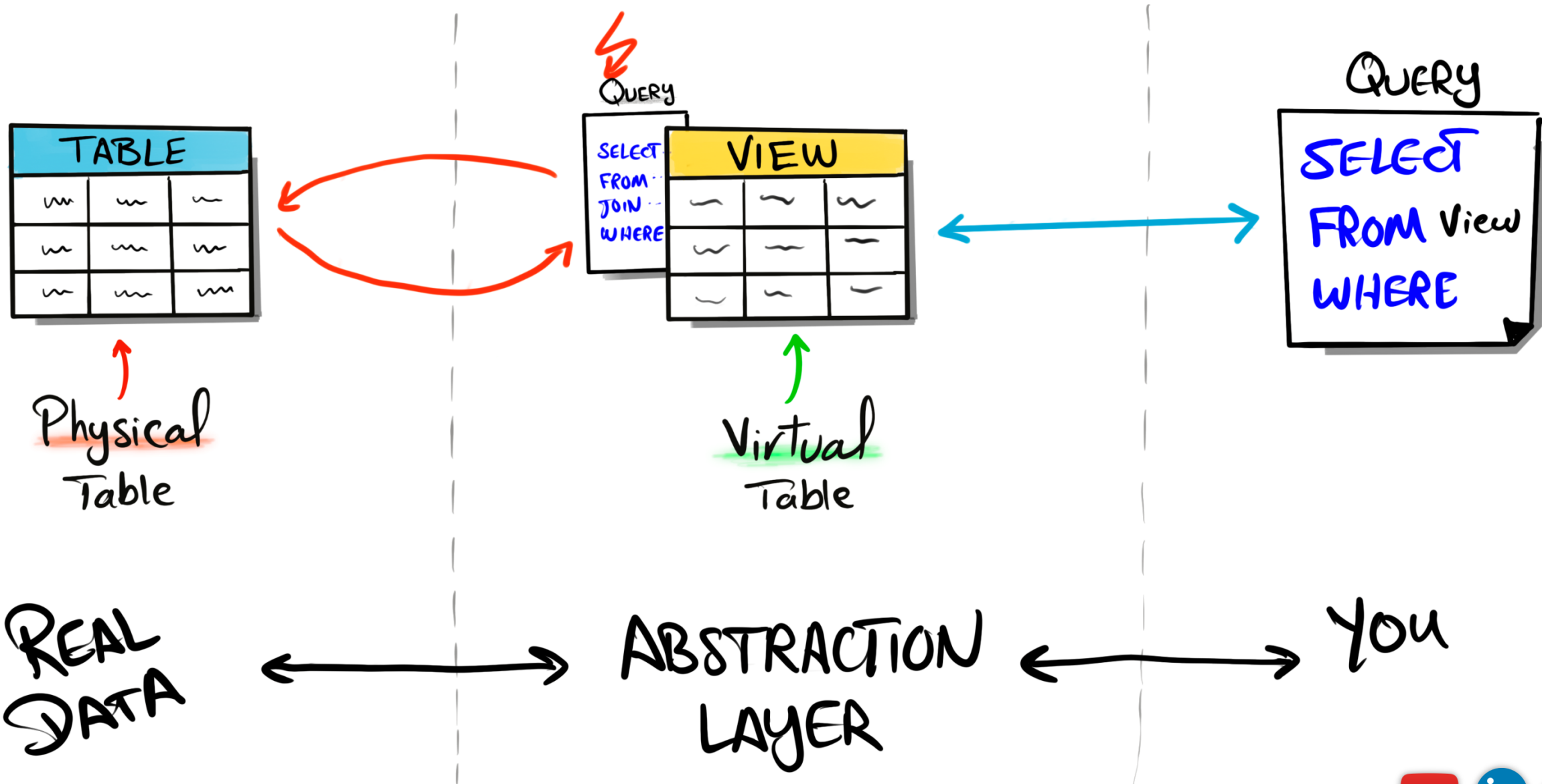




VIEW

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

No Persistence

Easy to Maintain

Slow Response

Read

Flexible

TABLE

Persisted Data

Hard to Maintain

Fast Response

Read/Write

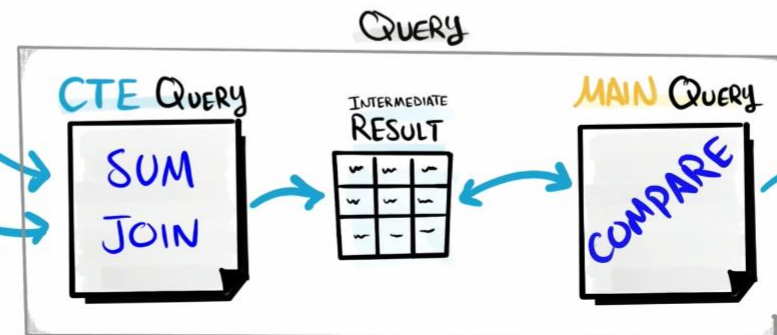
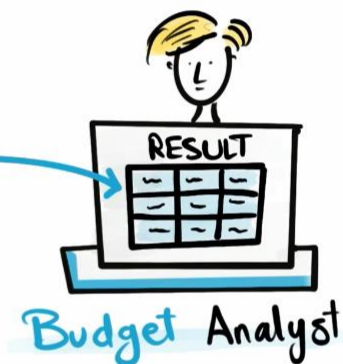
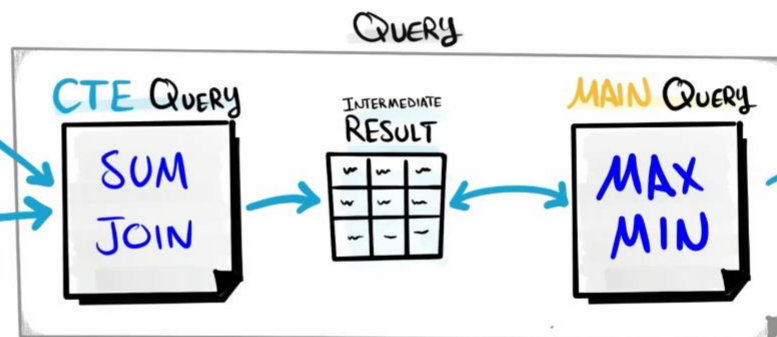
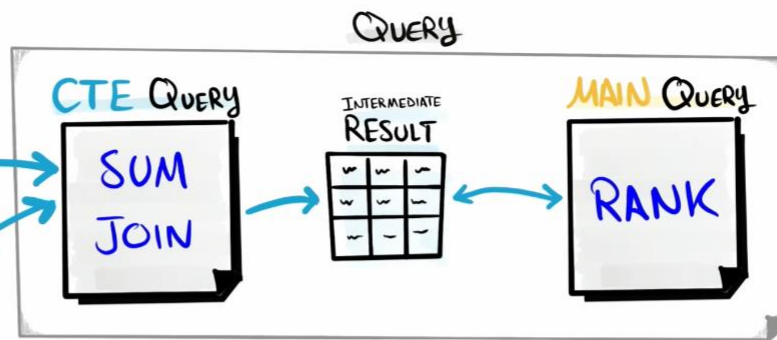
Hard to Change

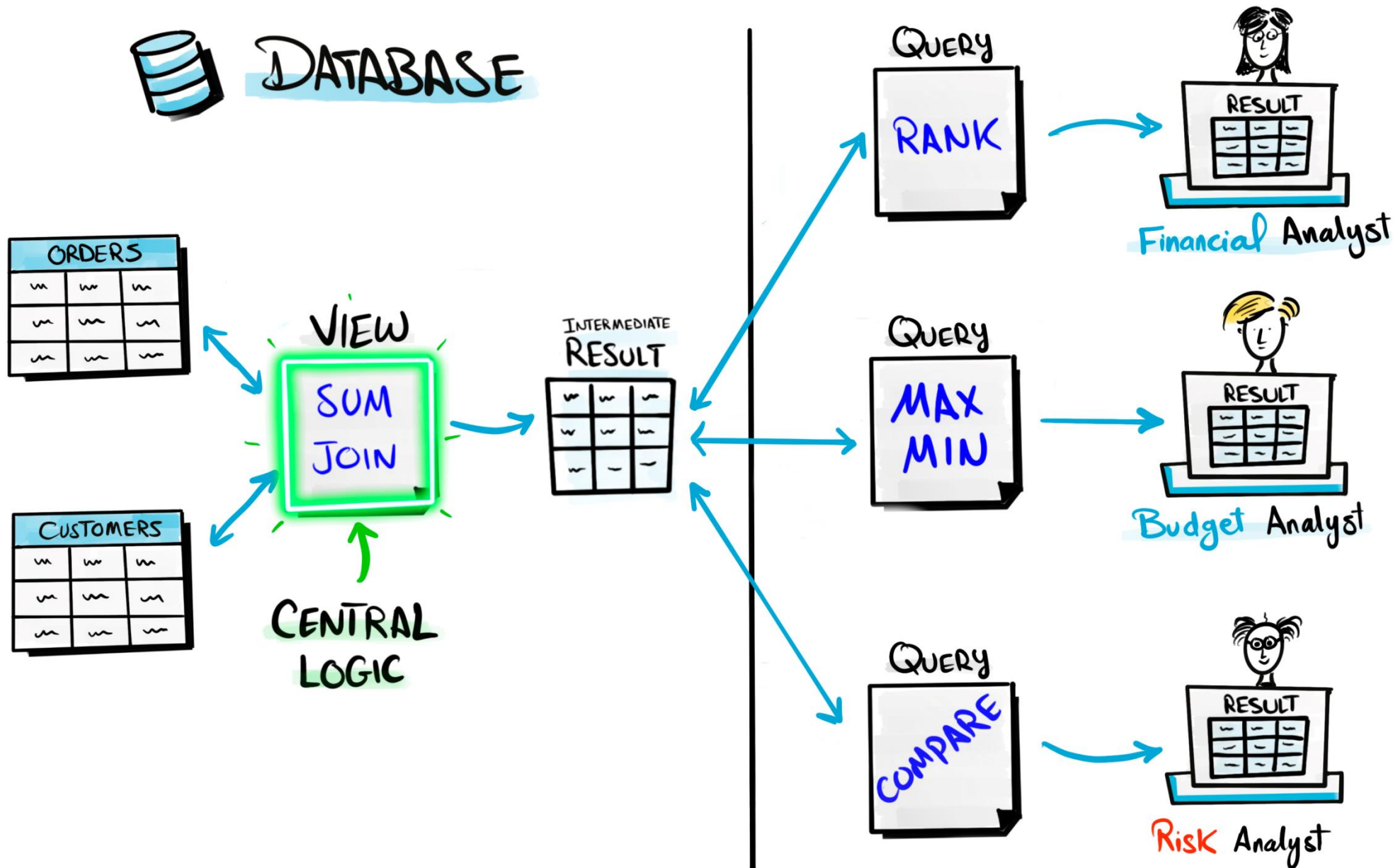


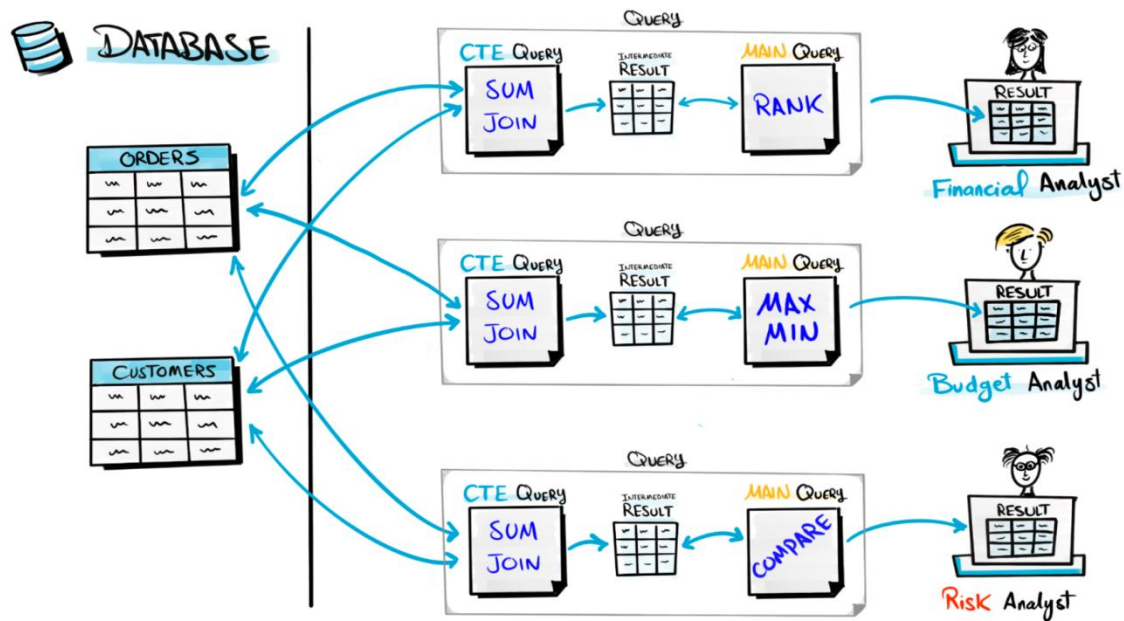
DATABASE

ORDERS		
~	~	~
~	~	~
~	~	~

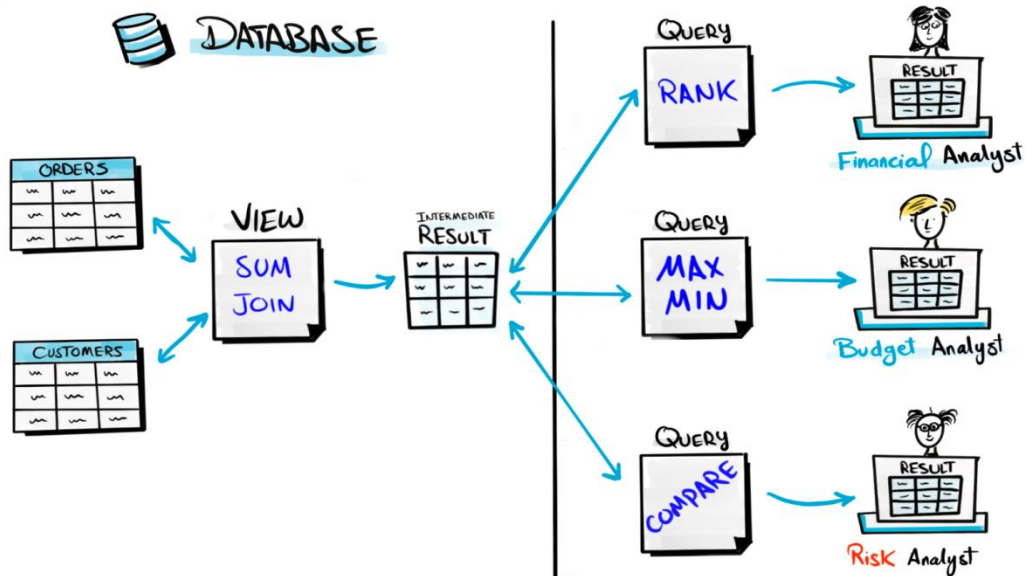
CUSTOMERS		
~	~	~
~	~	~
~	~	~







✓ 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 -

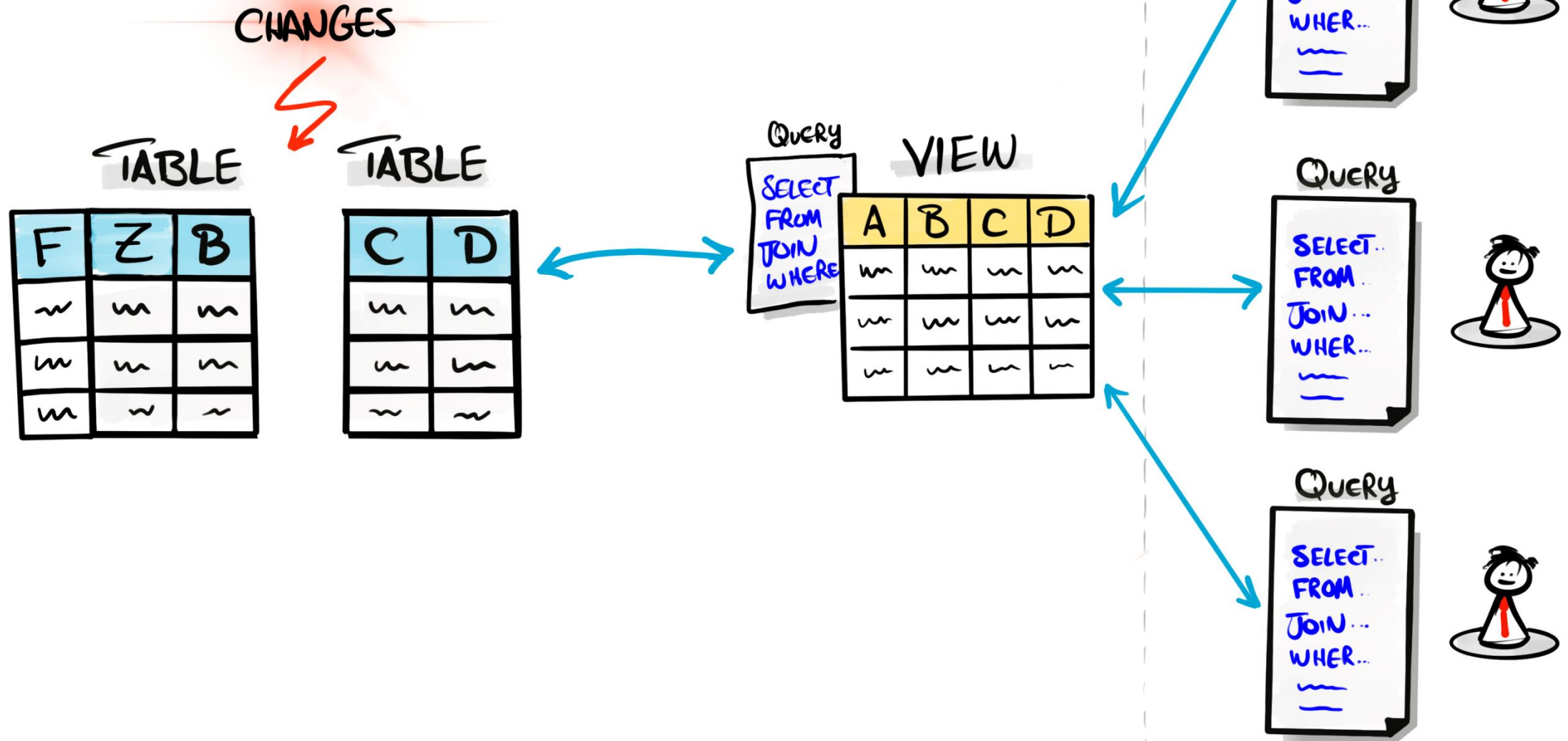
VIEWS

DDL
Statement

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

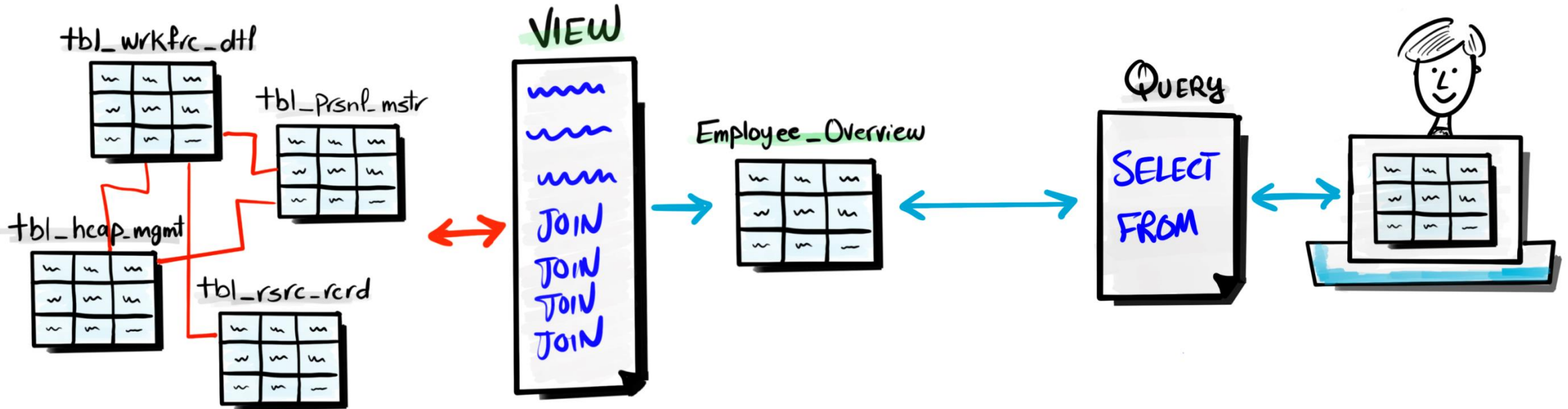
Query

Flexibility & Dynamic

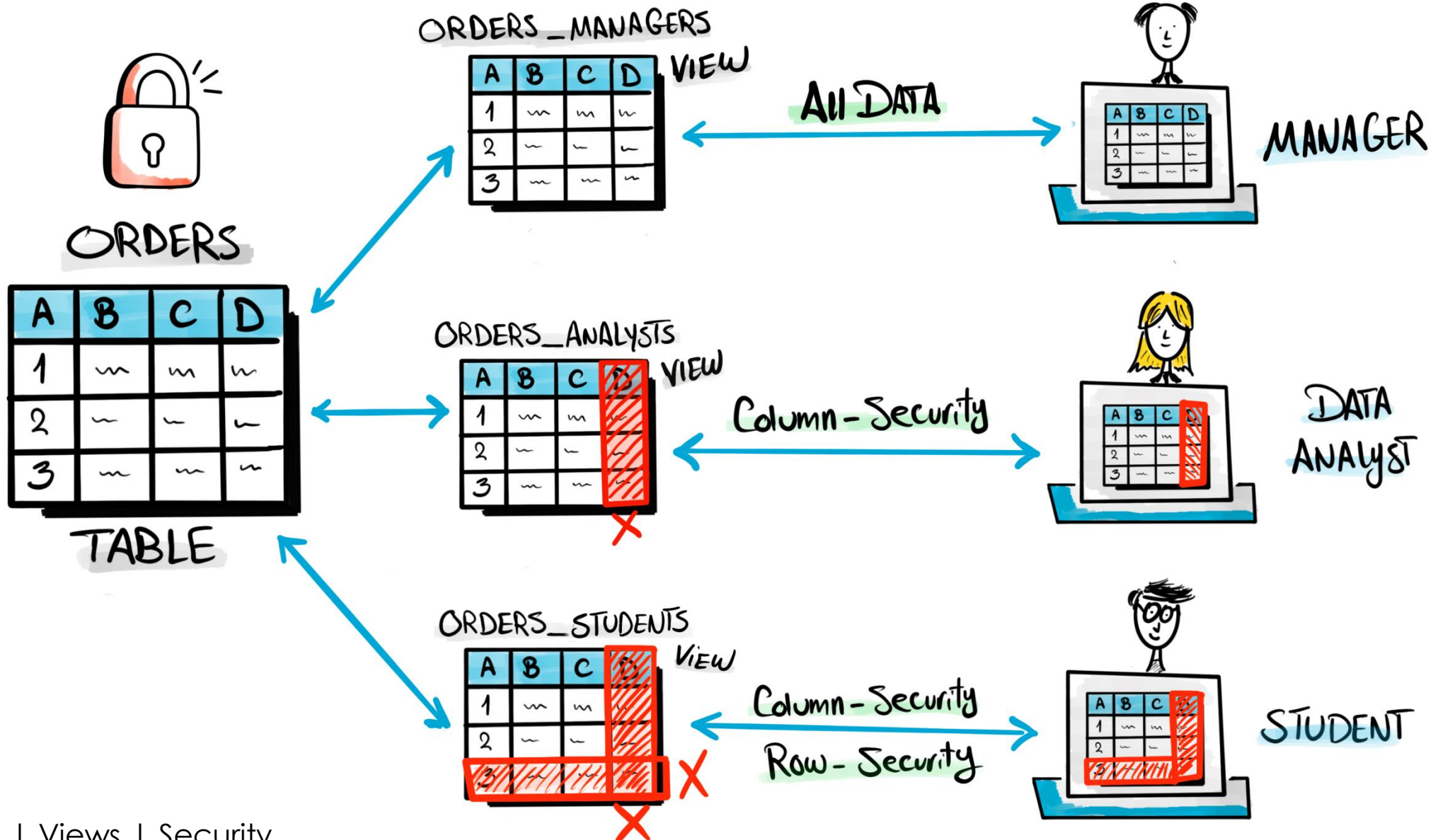


Hide Complexity

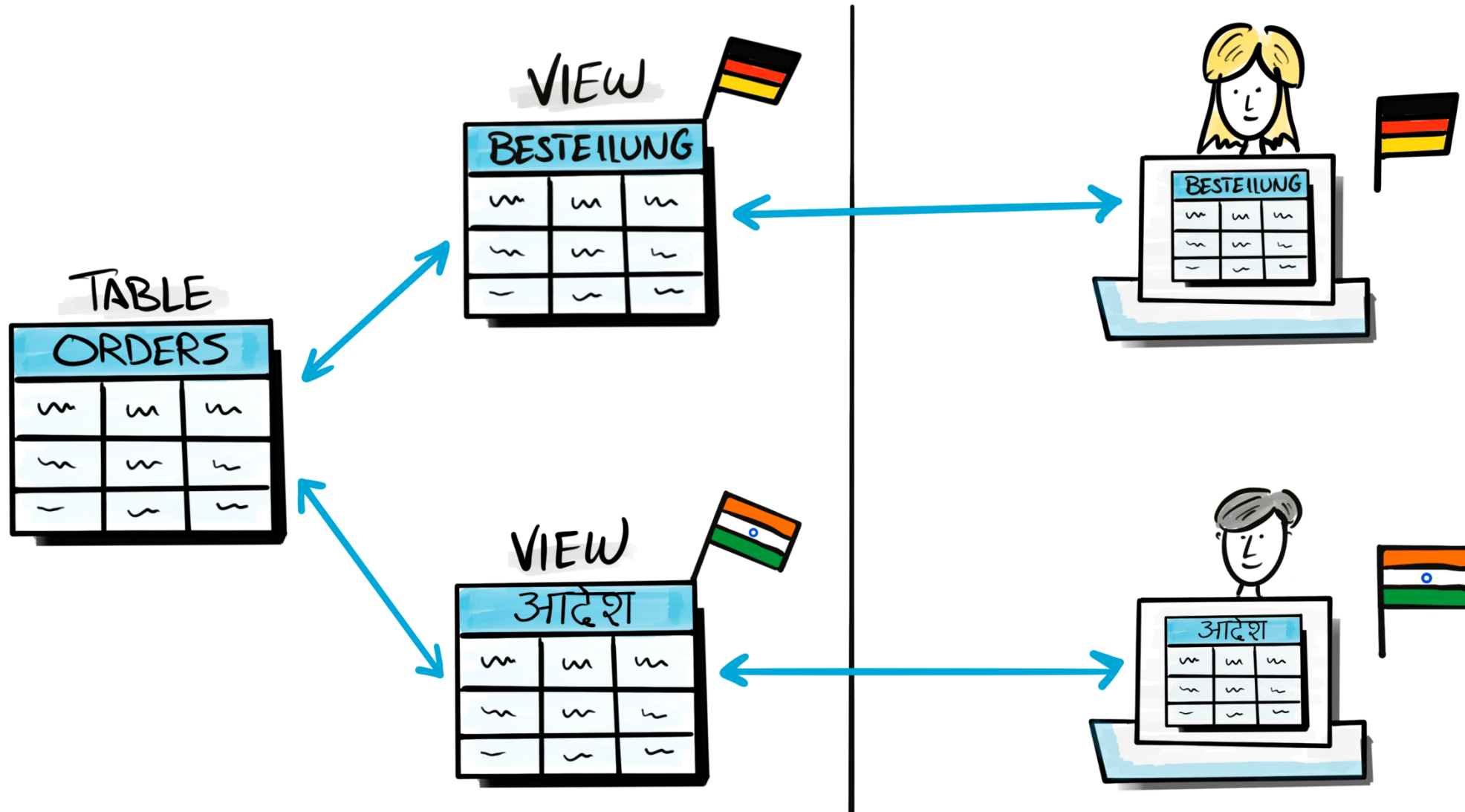
TABLES



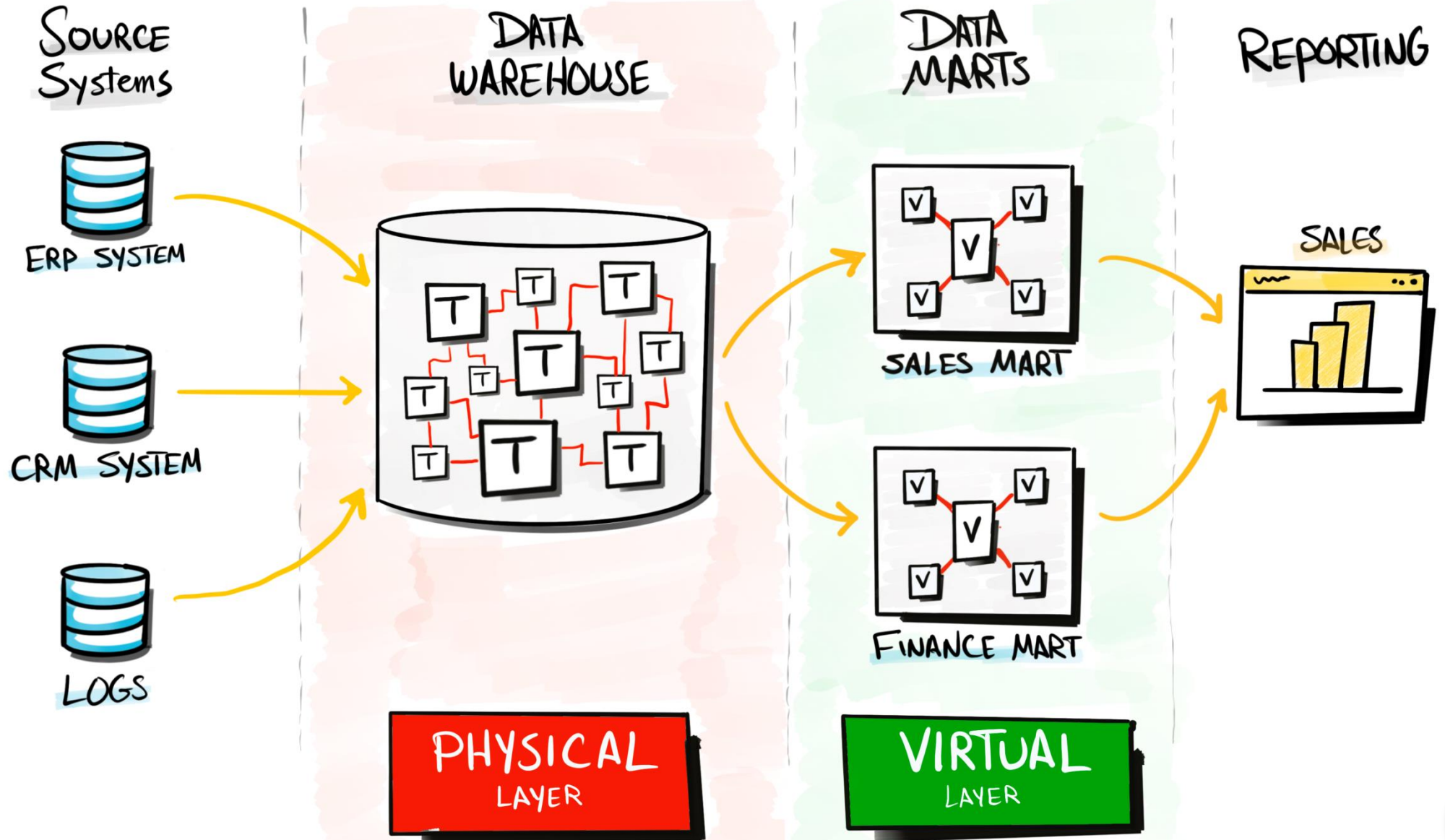
Security



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.



DATA WITH BARAA

CTAS

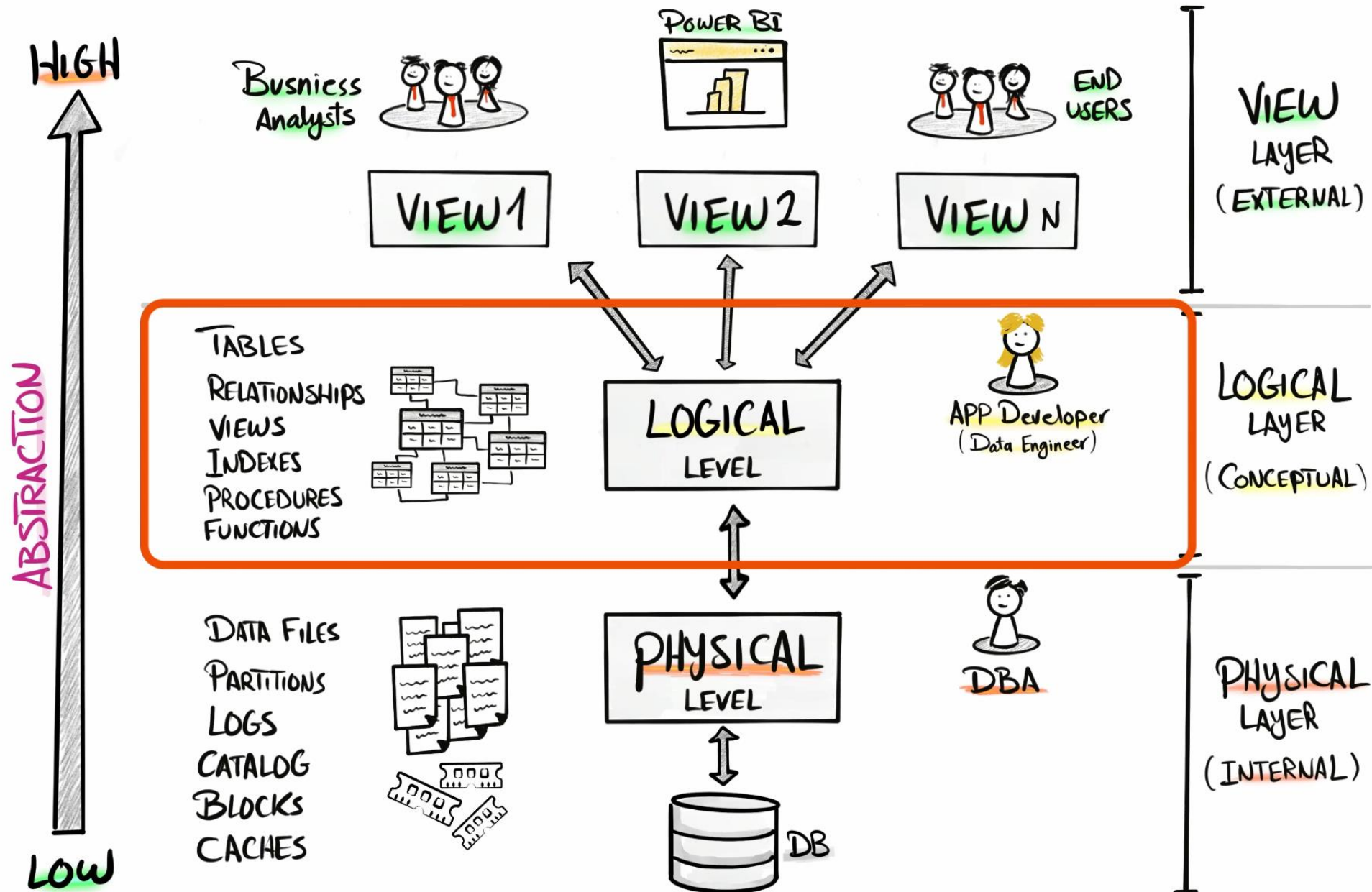
Create Table As SELECT

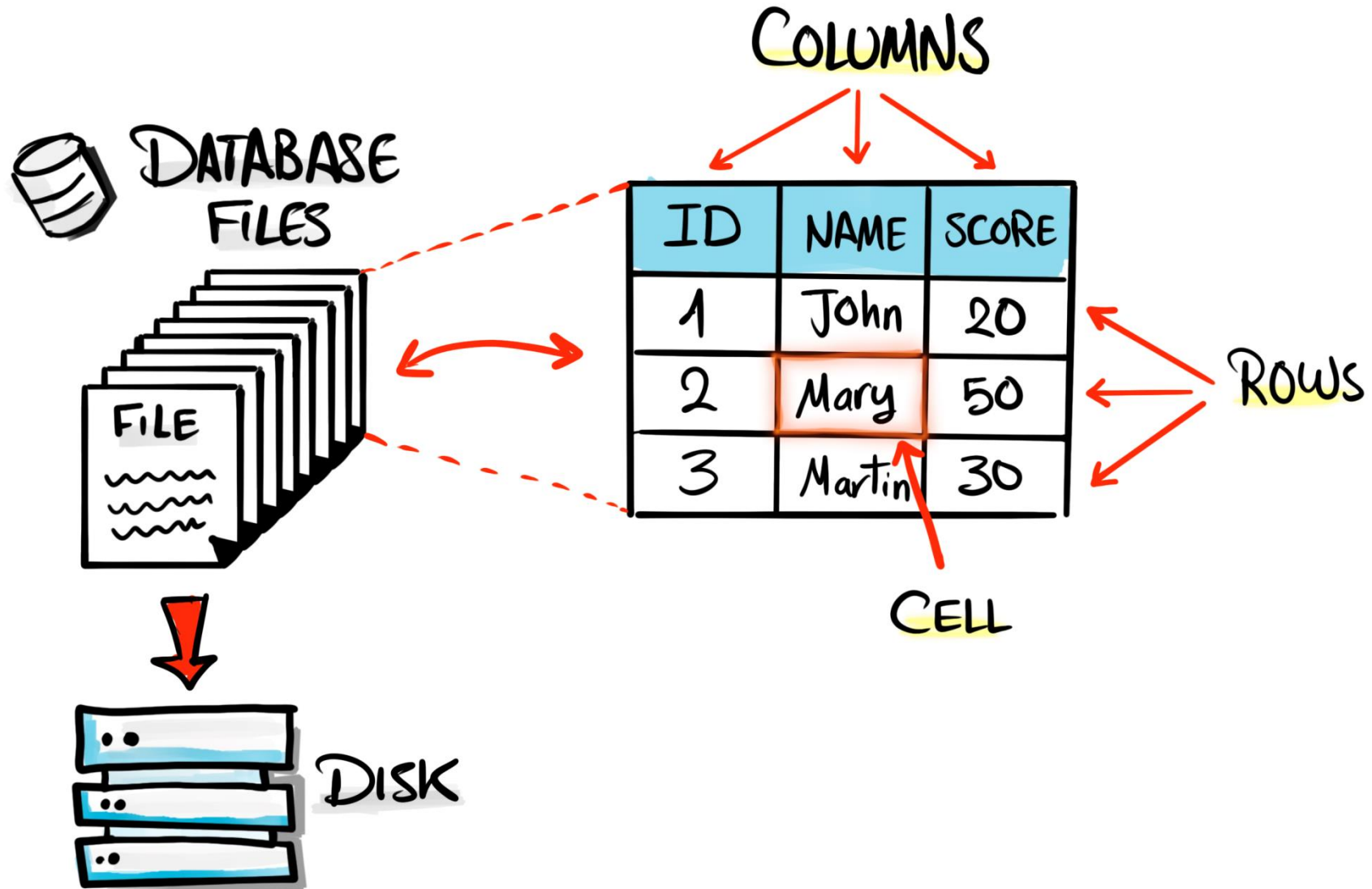
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | CTAS

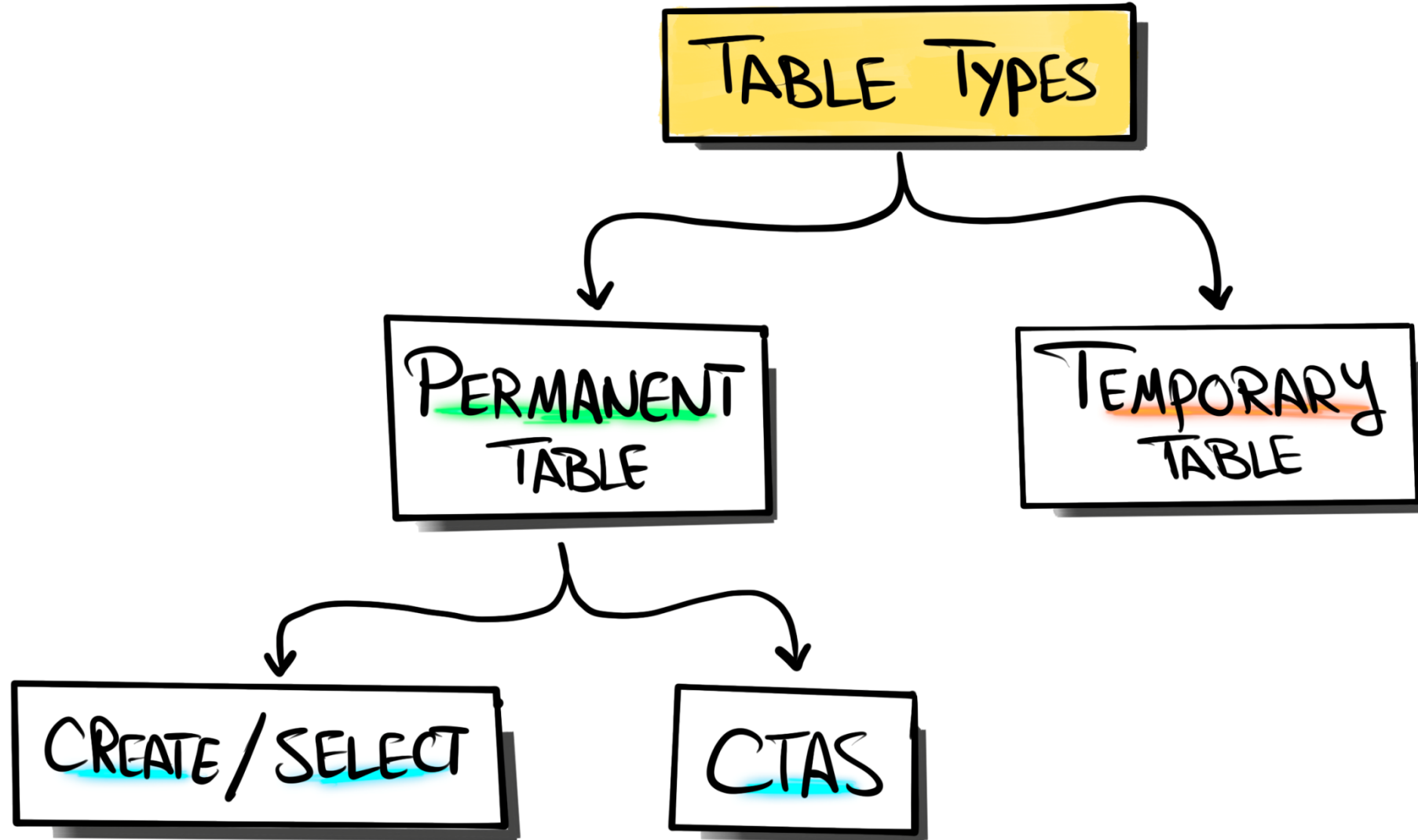


DB TABLE

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







CREATE/INSERT

#1 STEP

CREATE

#2 STEP

INSERT

CSV

TABLE		
~	~	~
~	~	~
~	~	~

CTAS

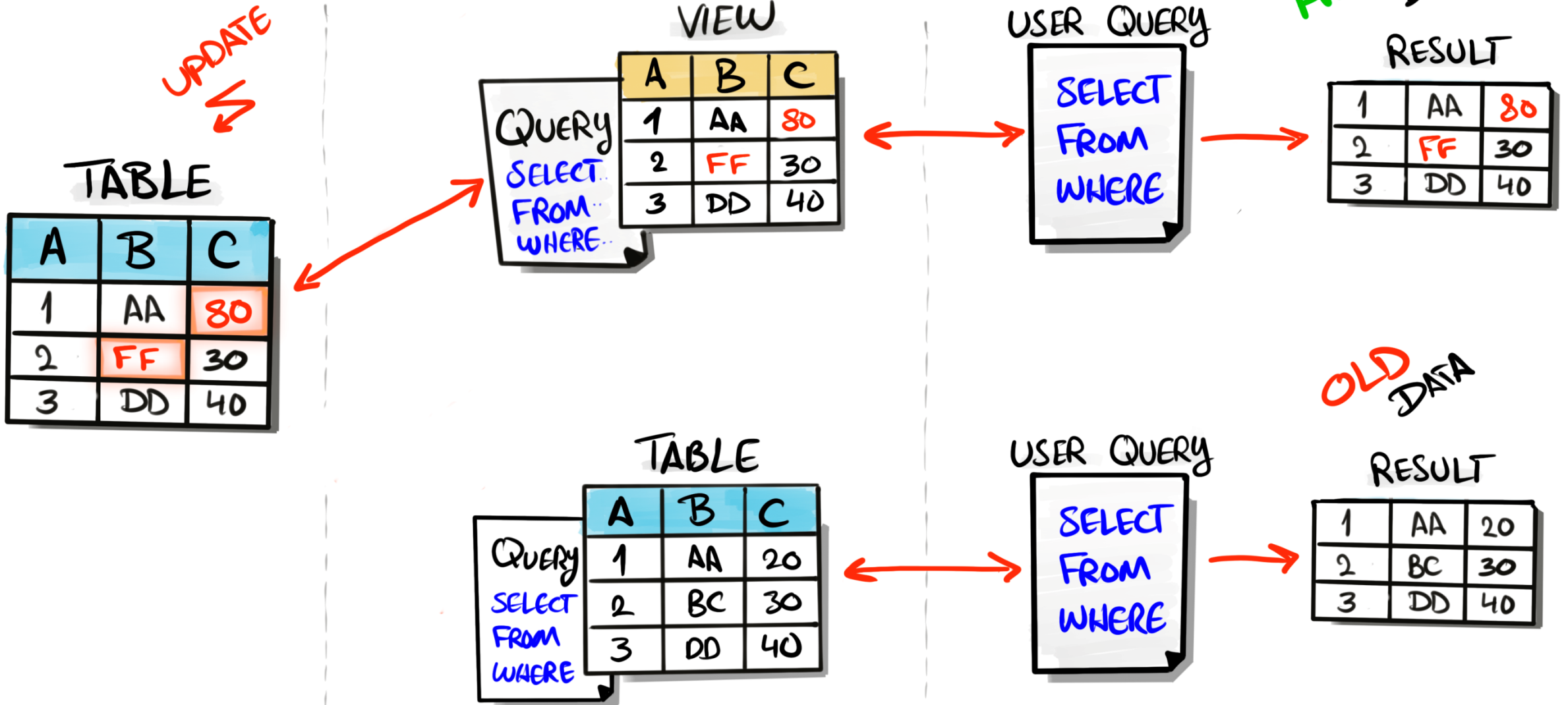
#1 STEP

QUERY

RESULT

~	~	~
~	~	~
~	~	~

TABLE		
~	~	~
~	~	~
~	~	~



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 ...
```

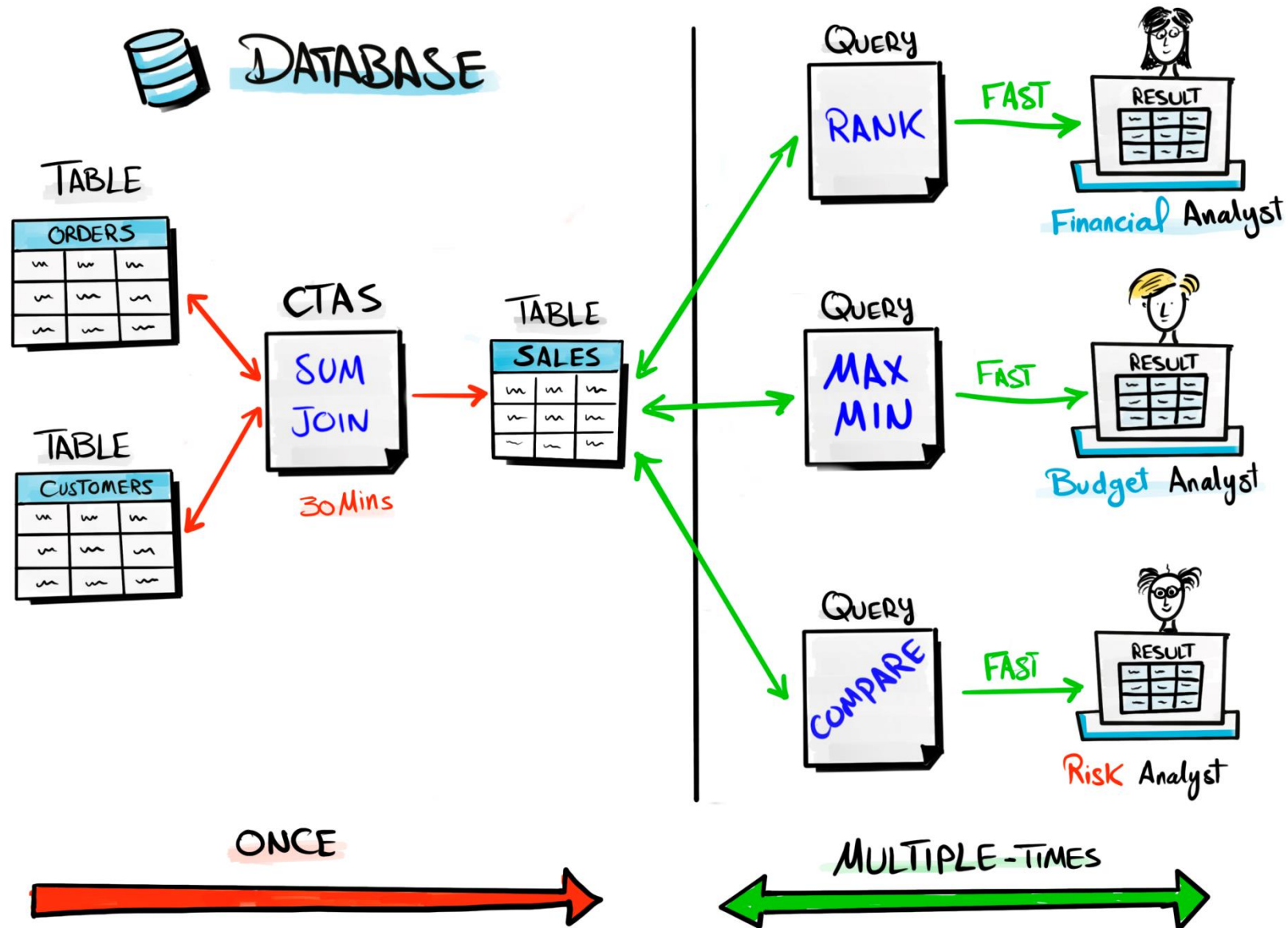
```
INTO New-Table
```

```
FROM ...
```

```
WHERE ...
```

Sql Server

Optimize Performance



Create Snapshot

UPDATES

ORDERS			
E	1	A	4
B	5	D	9
D	3	D	10

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



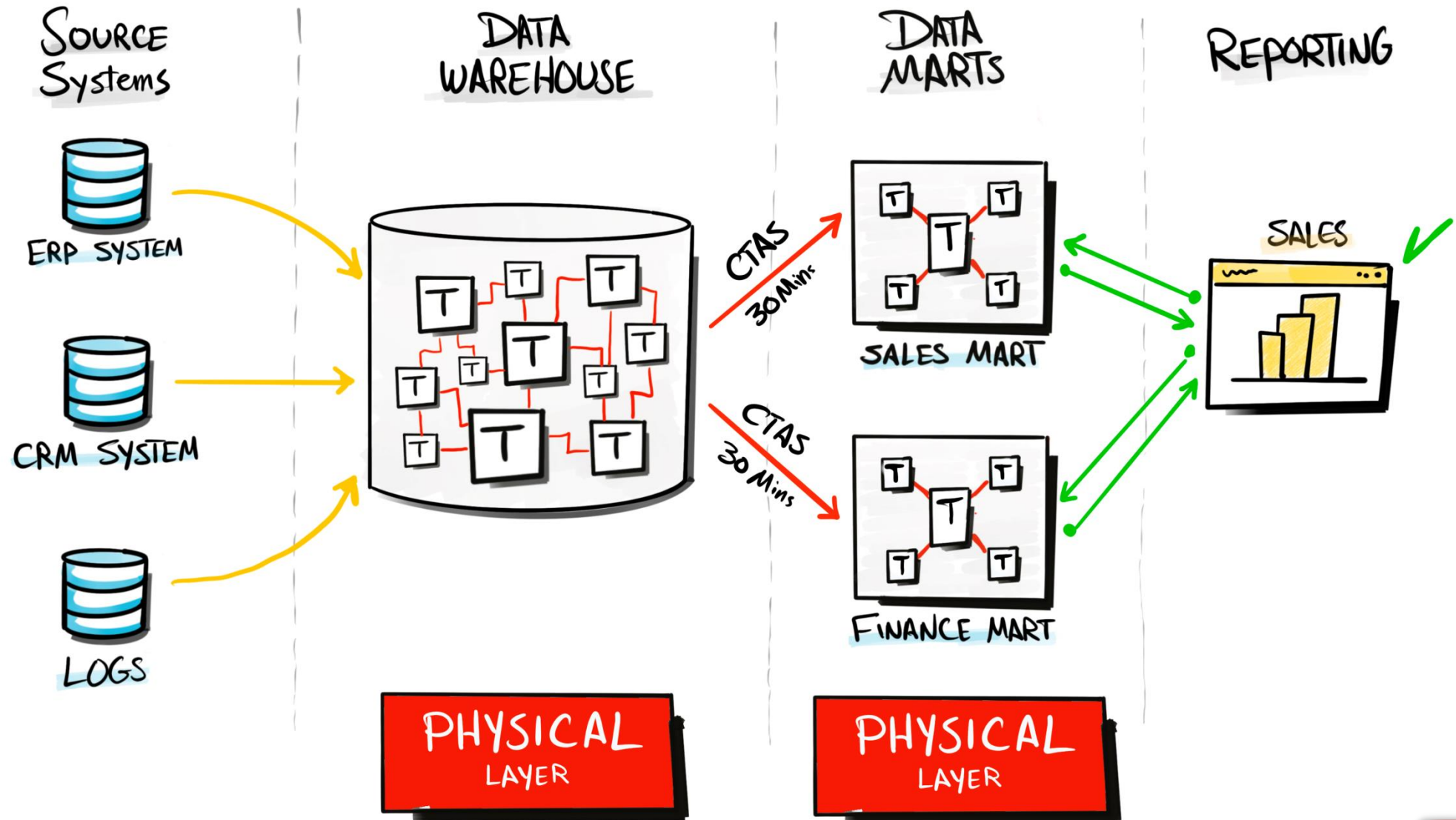
ANALYSE

ORDERS			
A	1	A	8
B	2	C	9
C	3	D	10

SNAPSHOT

CTAS

Physical Data Marts



TEMPORARY TABLE

SELECT ...

INTO **#**New-Table

FROM ...

WHERE ...

Sql Server

PERMENANT CREATE TABLE

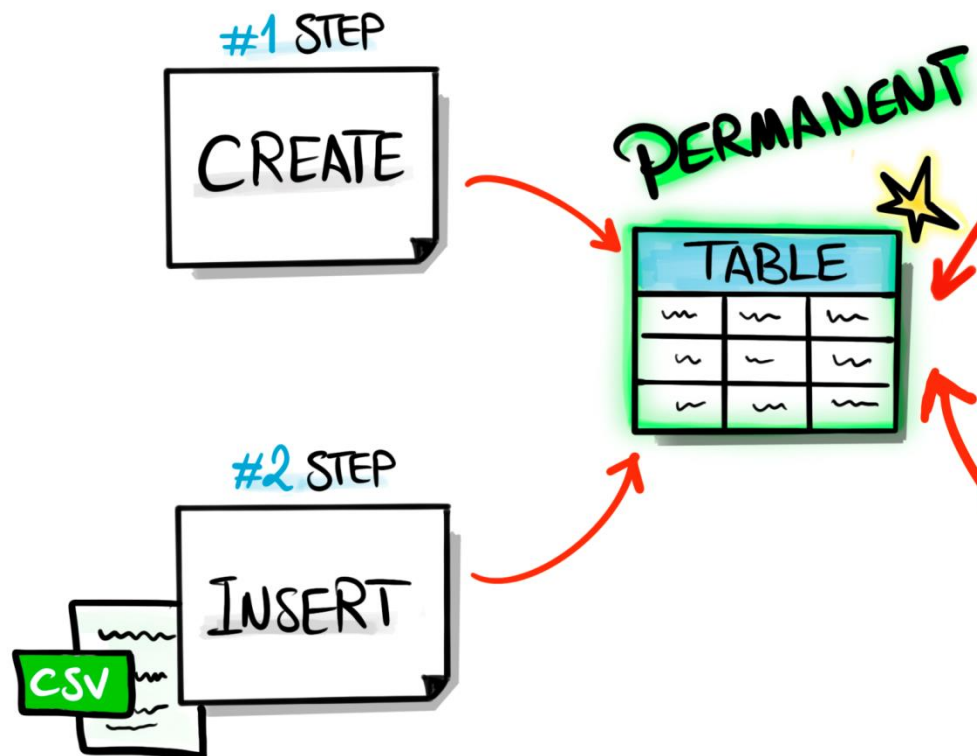
```
CREATE TABLE TABLE-NAME AS  
(  
    SELECT ...  
    FROM ...  
    WHERE ...  
)
```

TEMPORARY TABLE

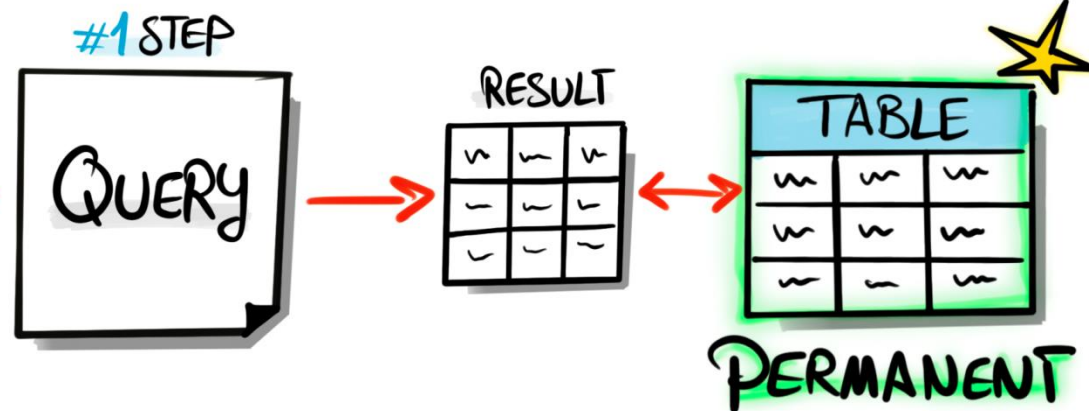
```
CREATE TEMPORARY TABLE TABLE-NAME AS  
(  
    SELECT ...  
    FROM ...  
    WHERE ...  
)
```

MySQL | Postgres | Oracle

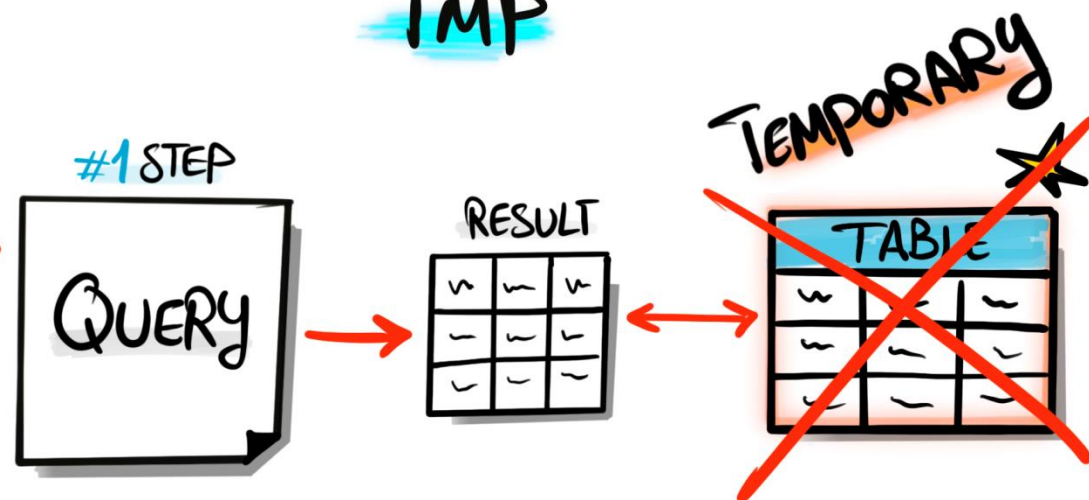
CREATE/INSERT

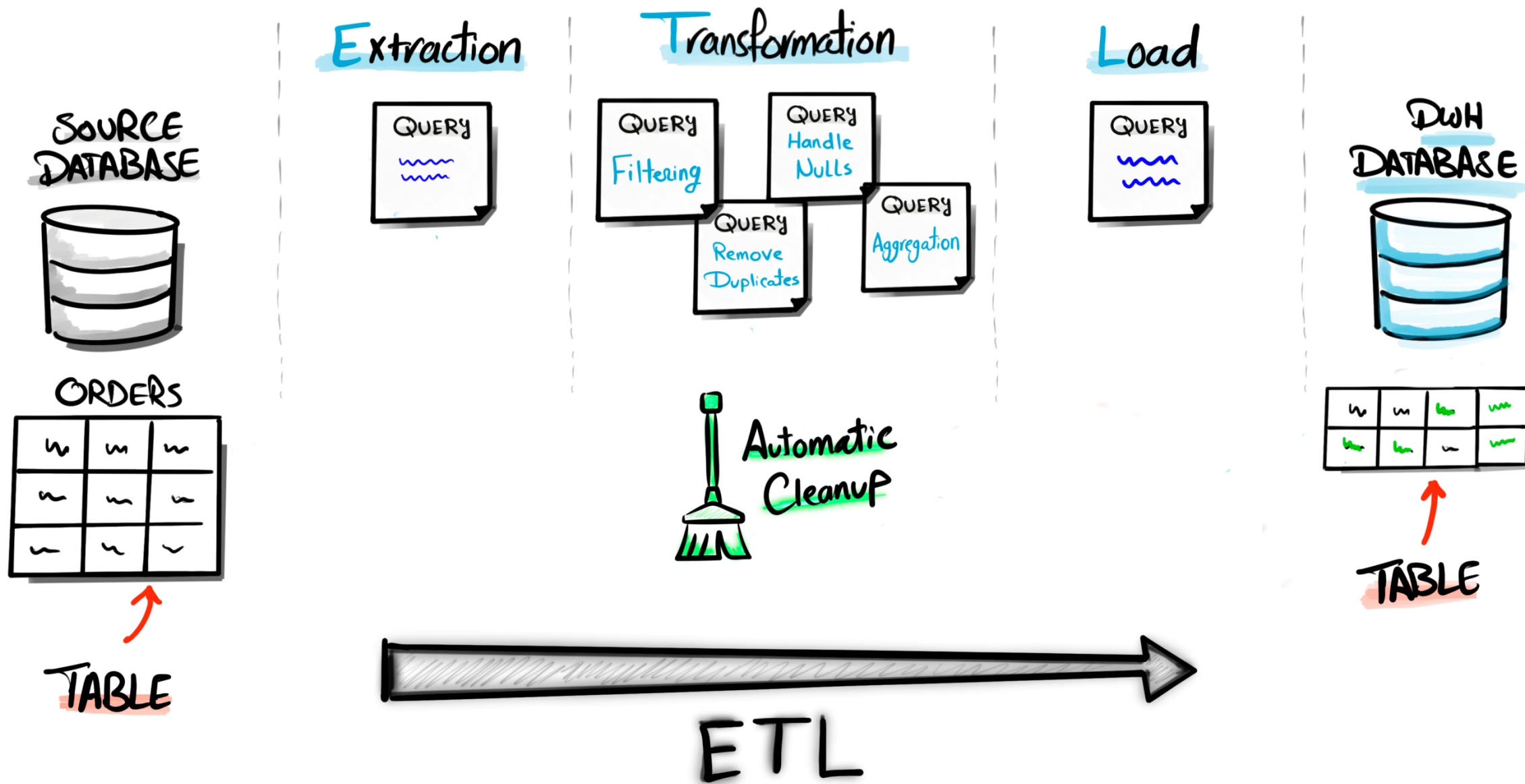


CTAS



TMP





Empty Table

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

View

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

Permenant Table

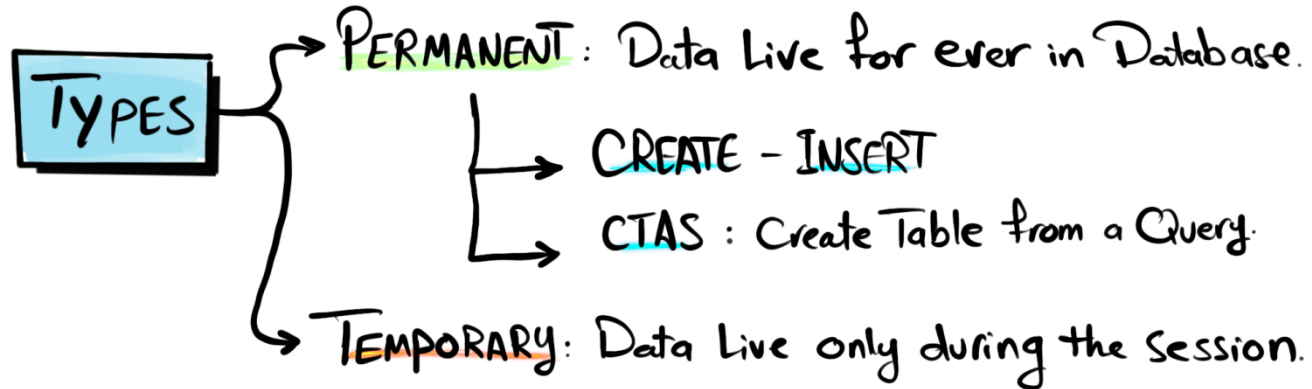
```
SELECT ...  
INTO New-Table  
FROM ...  
WHERE ...
```

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 		X 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

~	~	~
~	~	~
~	~	~

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

TABLE

~	~	~	~
~	~	~	~
~	~	~	~
~	~	~	~

TABLE DDL STATEMENT

VIEW DDL STATEMENT

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

~	~	~
~	~	~
~	~	~

VIEW

QUERY

SUB QUERY

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

RESULT

~	~	~
~	~	~
~	~	~

CTE QUERY

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

TEMP SET

~	~	~
~	~	~
~	~	~

MAIN QUERY

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

DATA SCIENTIST/ANALYST

FINAL RESULT

~	~	~
~	~	~
~	~	~

INSERT STATEMENT

INSERT INTO
VALUES (...)



DATABASE ADMIN



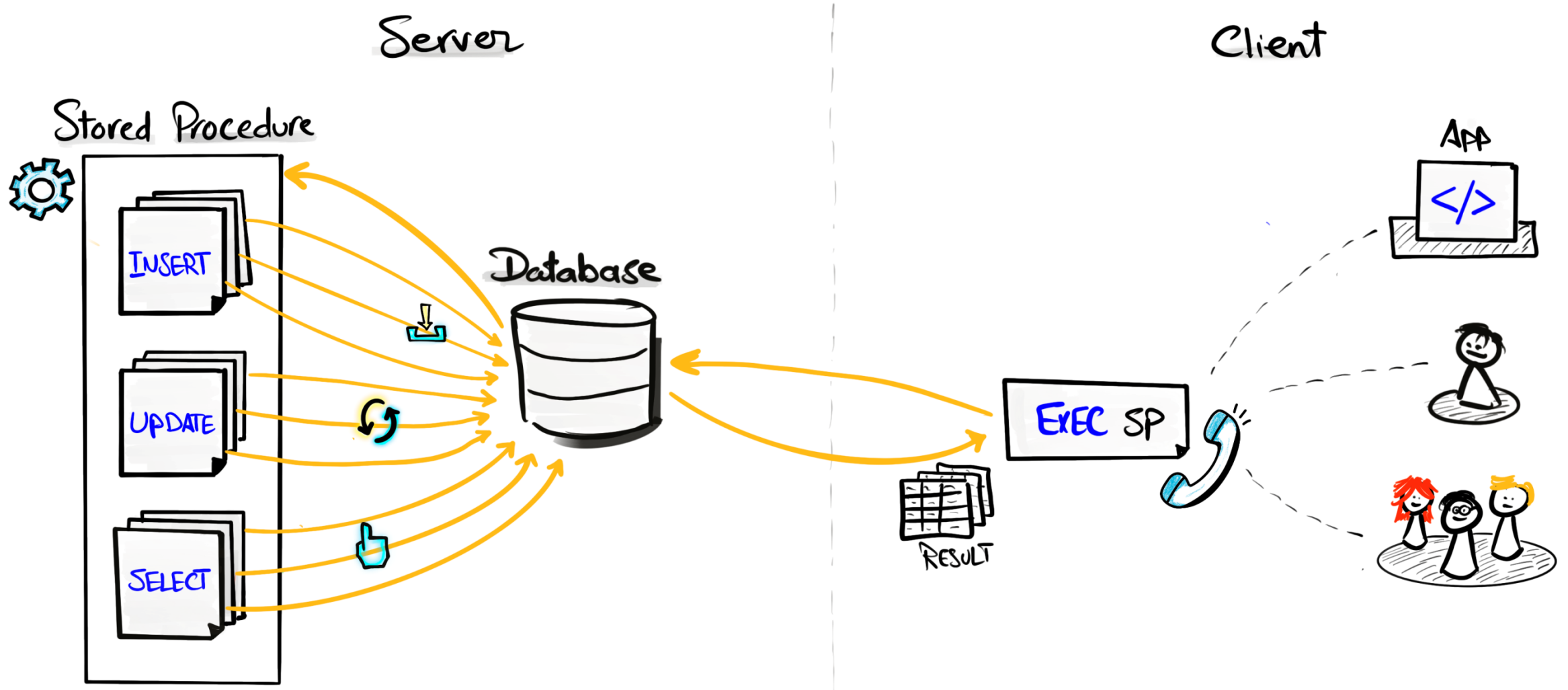
DATA WITH BARAA

Stored Procedure

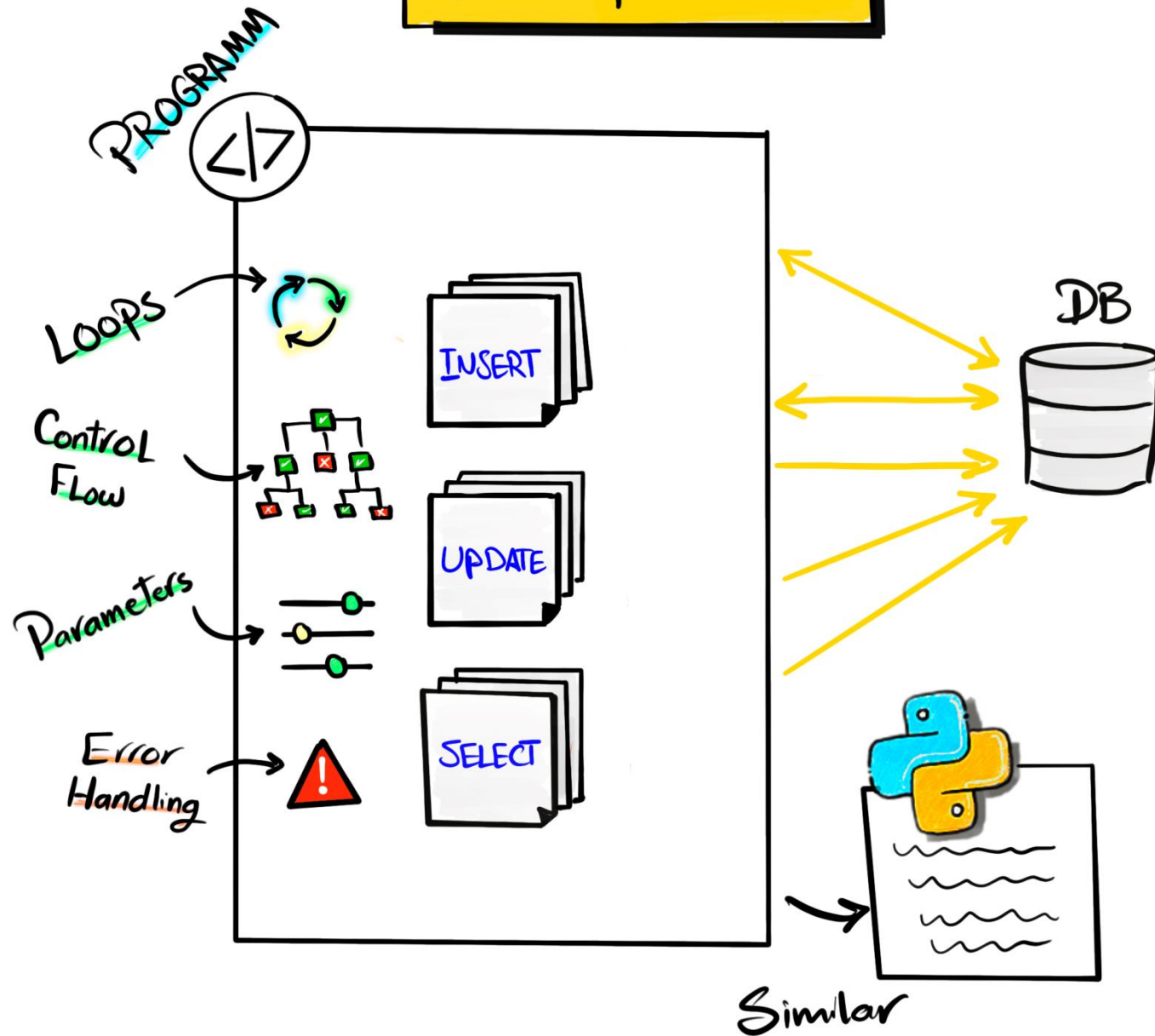
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Stored Procedure



Stored Procedure



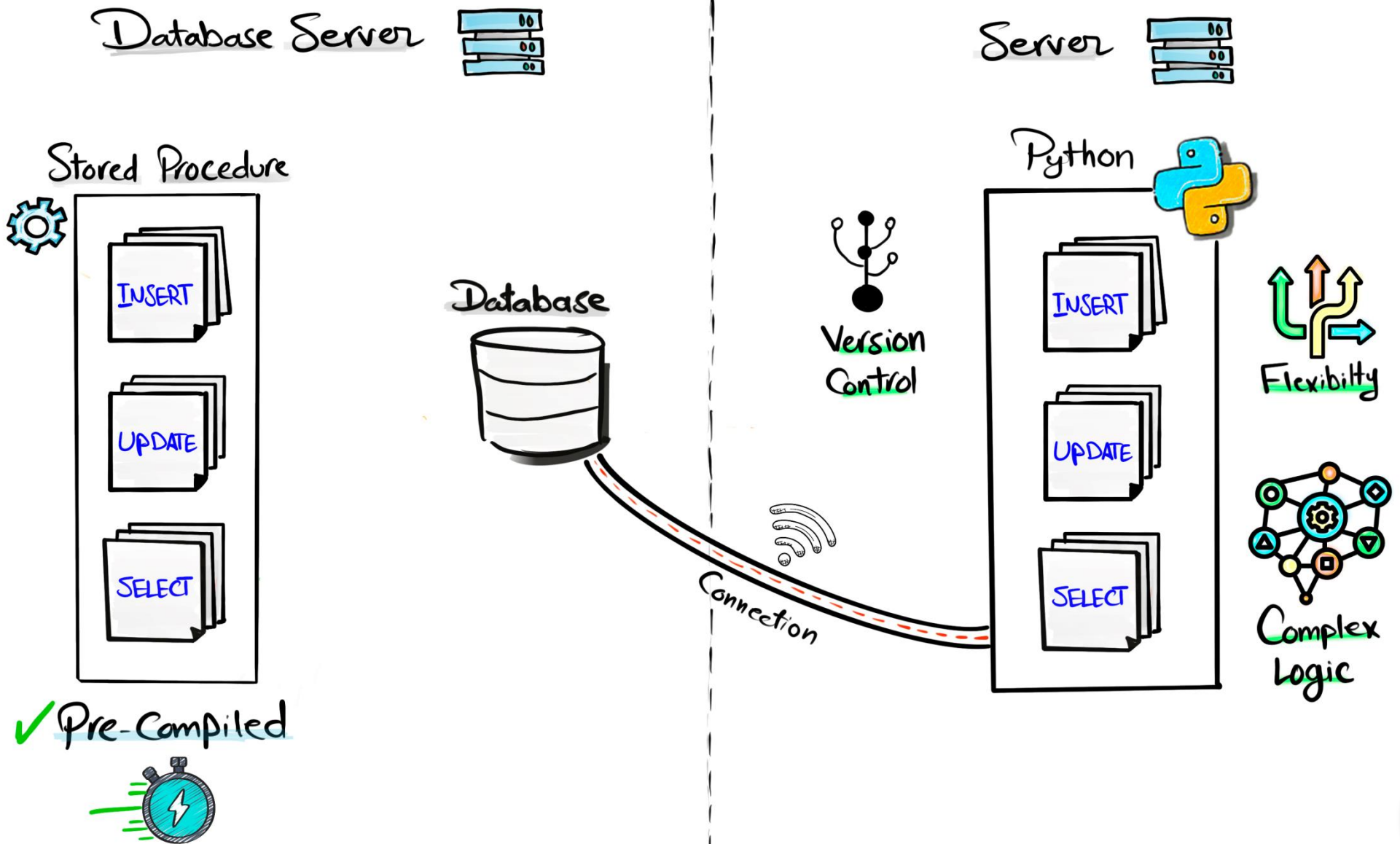
Stored Procedure



Query



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

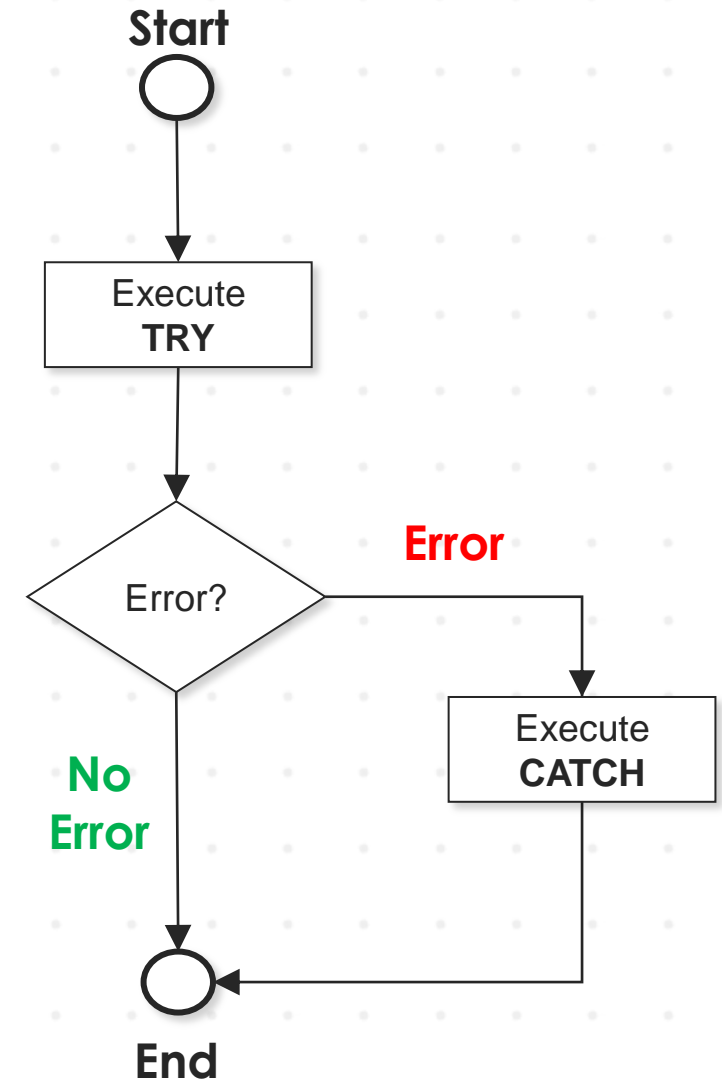
-- SQL statements that might cause an error

END TRY

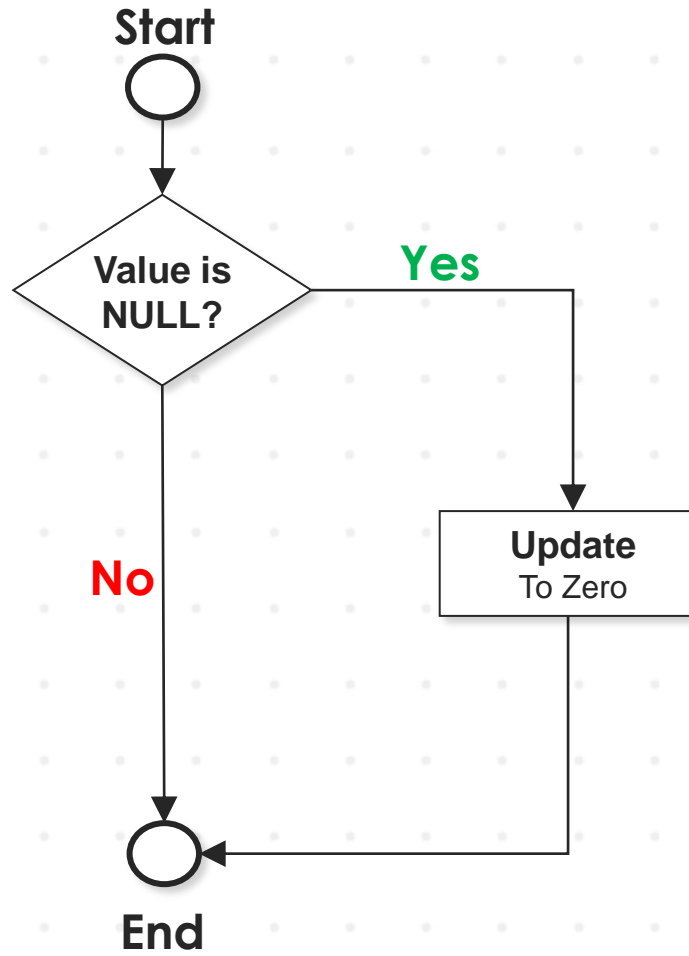
BEGIN CATCH

-- SQL statements To Handle The Error

END CATCH



Flow Control





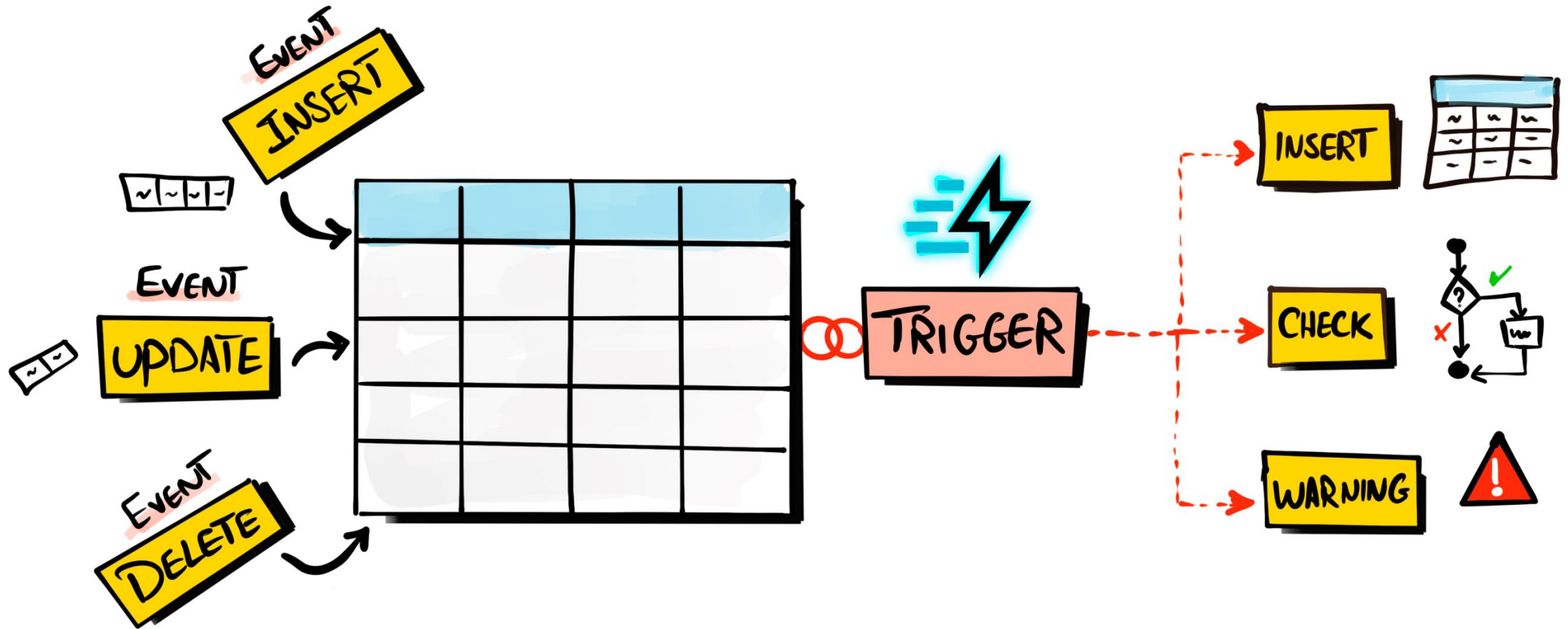
DATA WITH BARAA

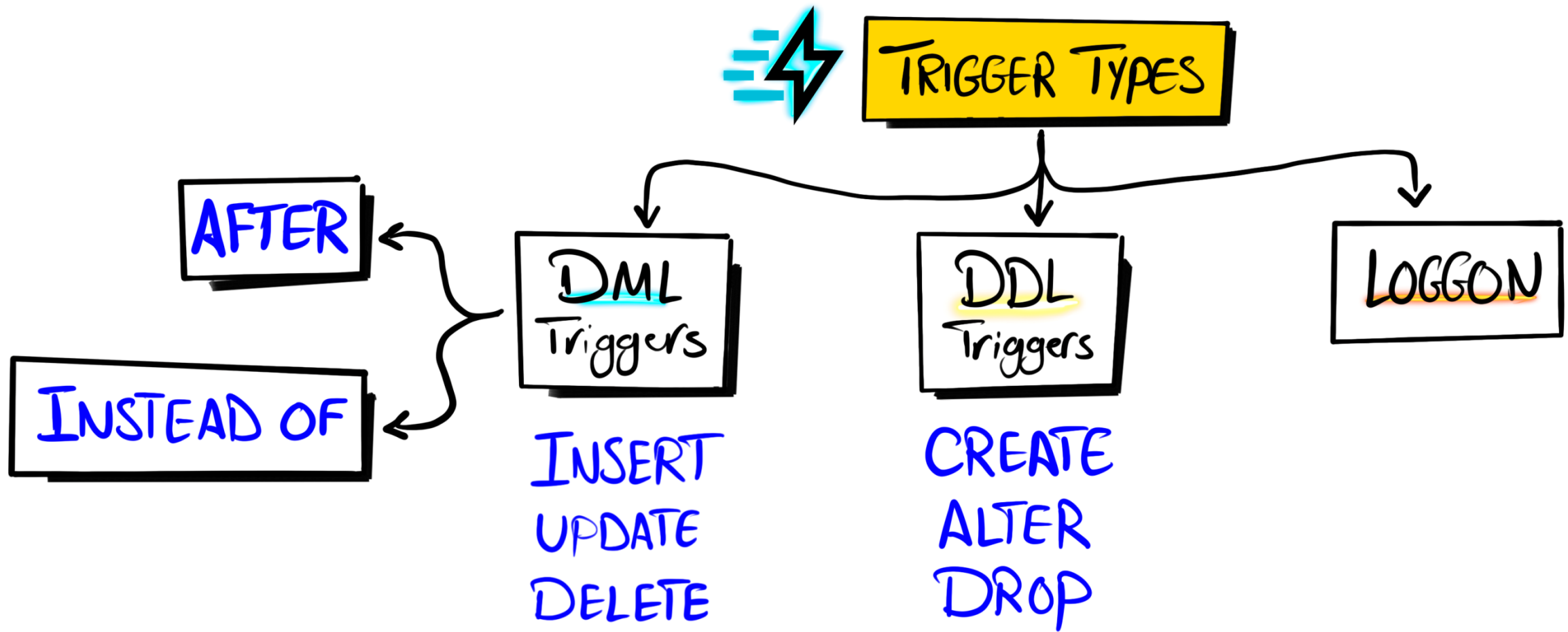
Triggers

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

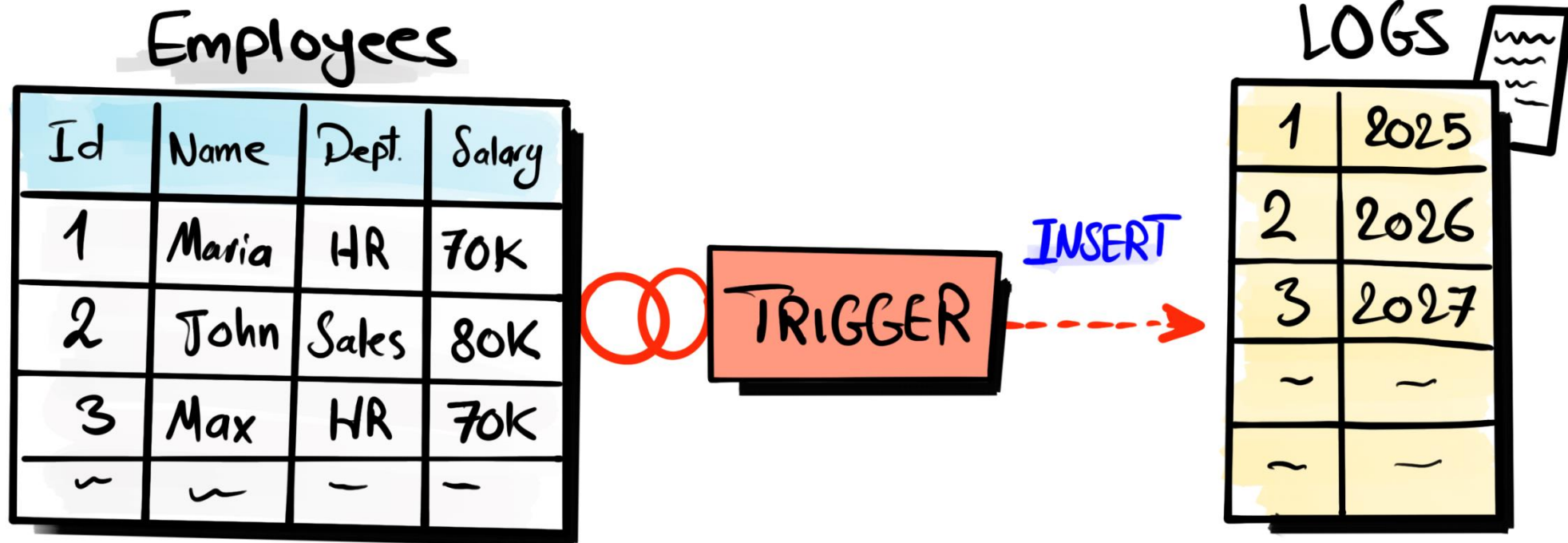


Triggers





Maintaining Logs



Triggers

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