DATA WITH BARAA

ADVANCED SQL

Techniques

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

SOLUTIONS

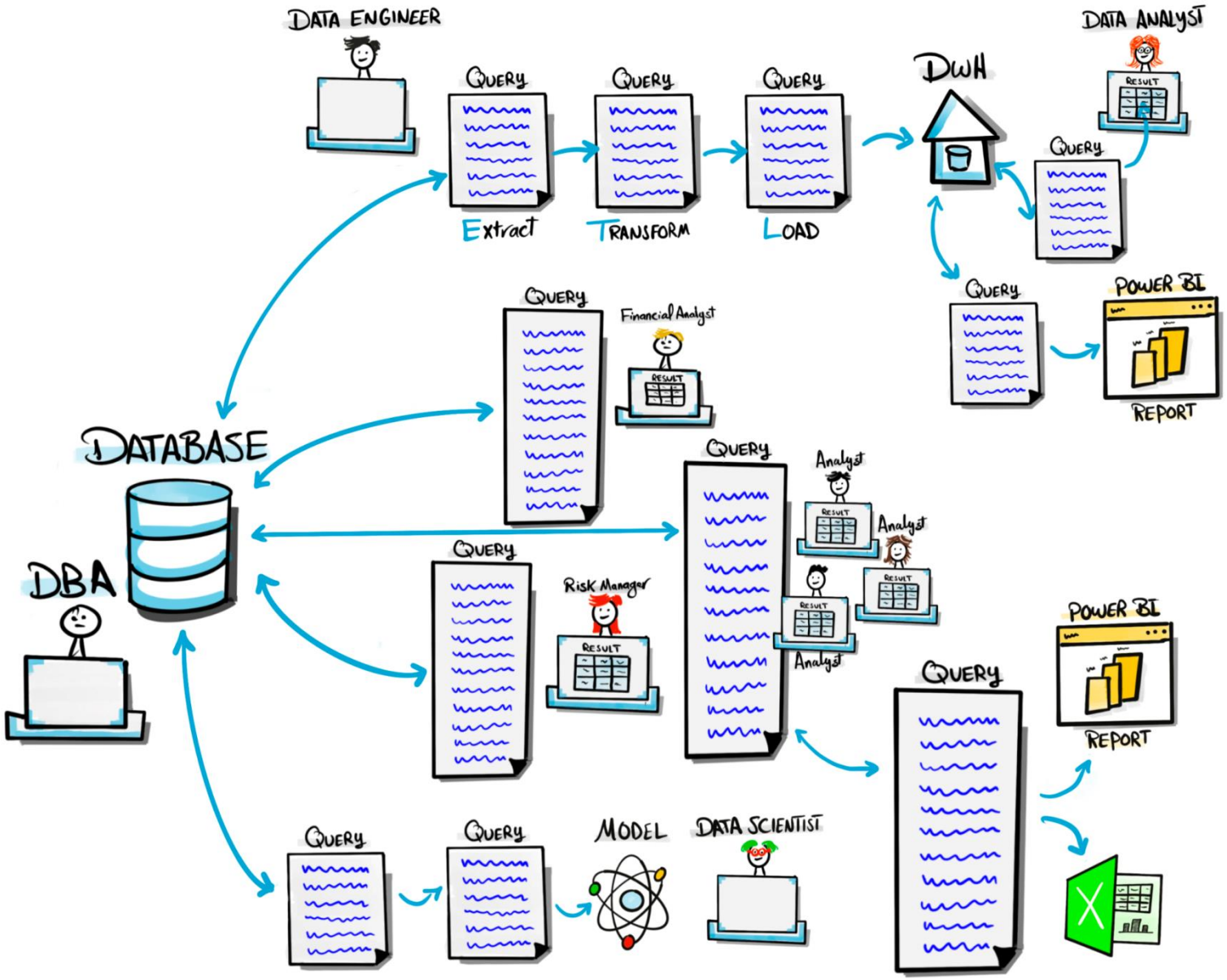
Subquery

- CTE -
Common Table Expression

Views

Temp Tables

-CTAS-
Create Table As Select



CHALLENGS

Redundancy

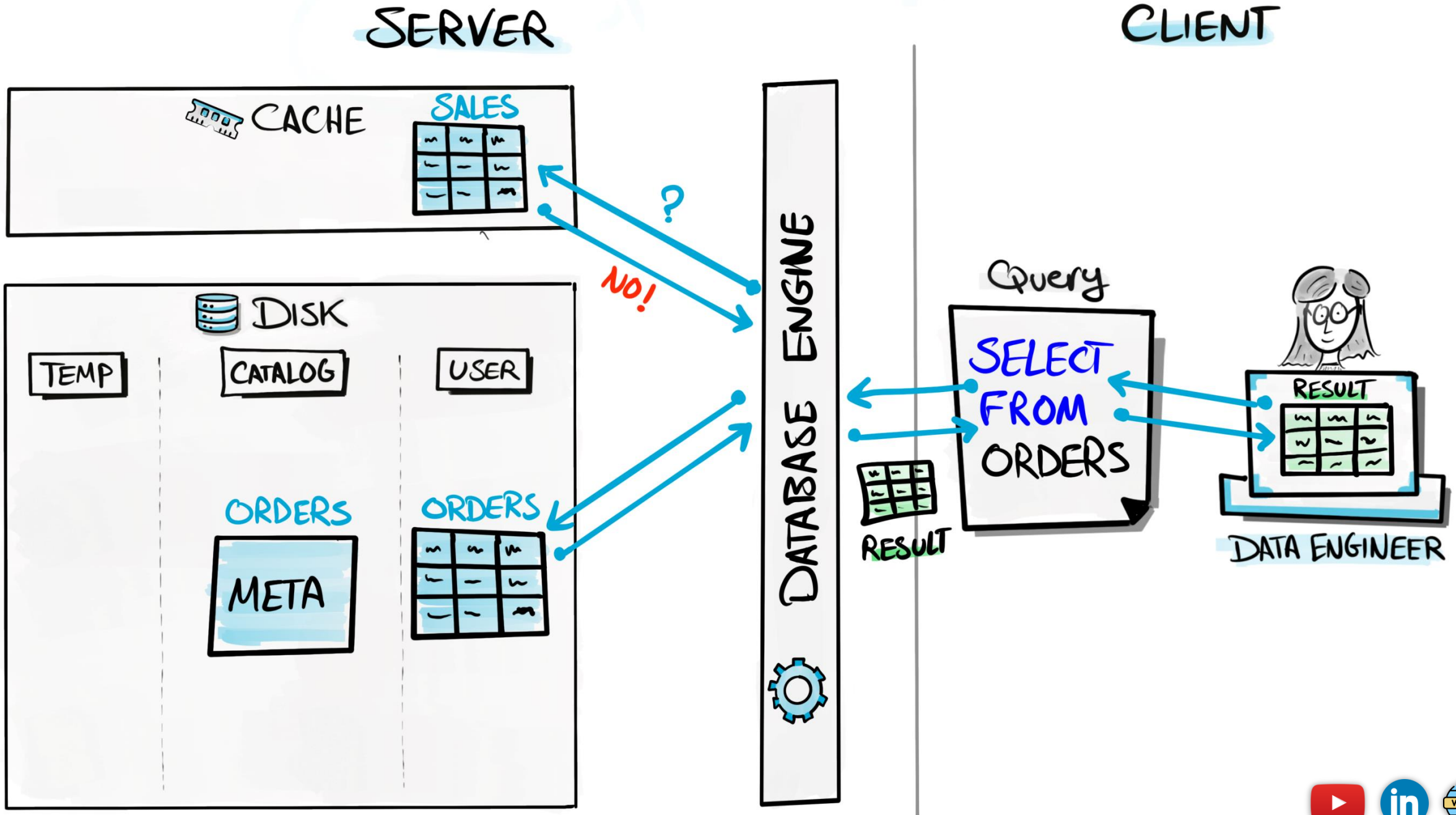
Performance Issues

Complexity

Hard To Maintain

DB Stress

Security

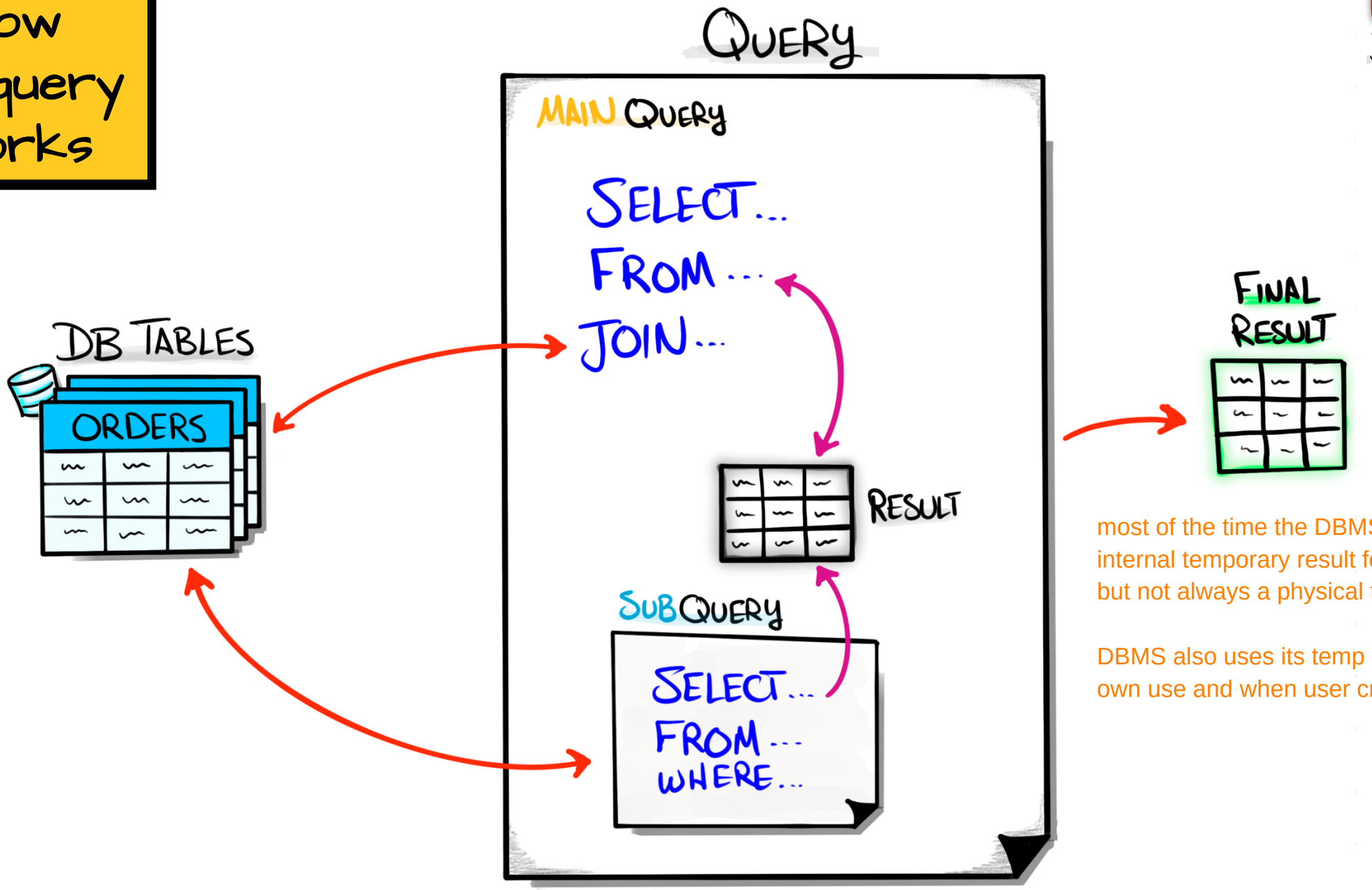


DATA WITH BARAA

Subquery

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

How Subquery Works

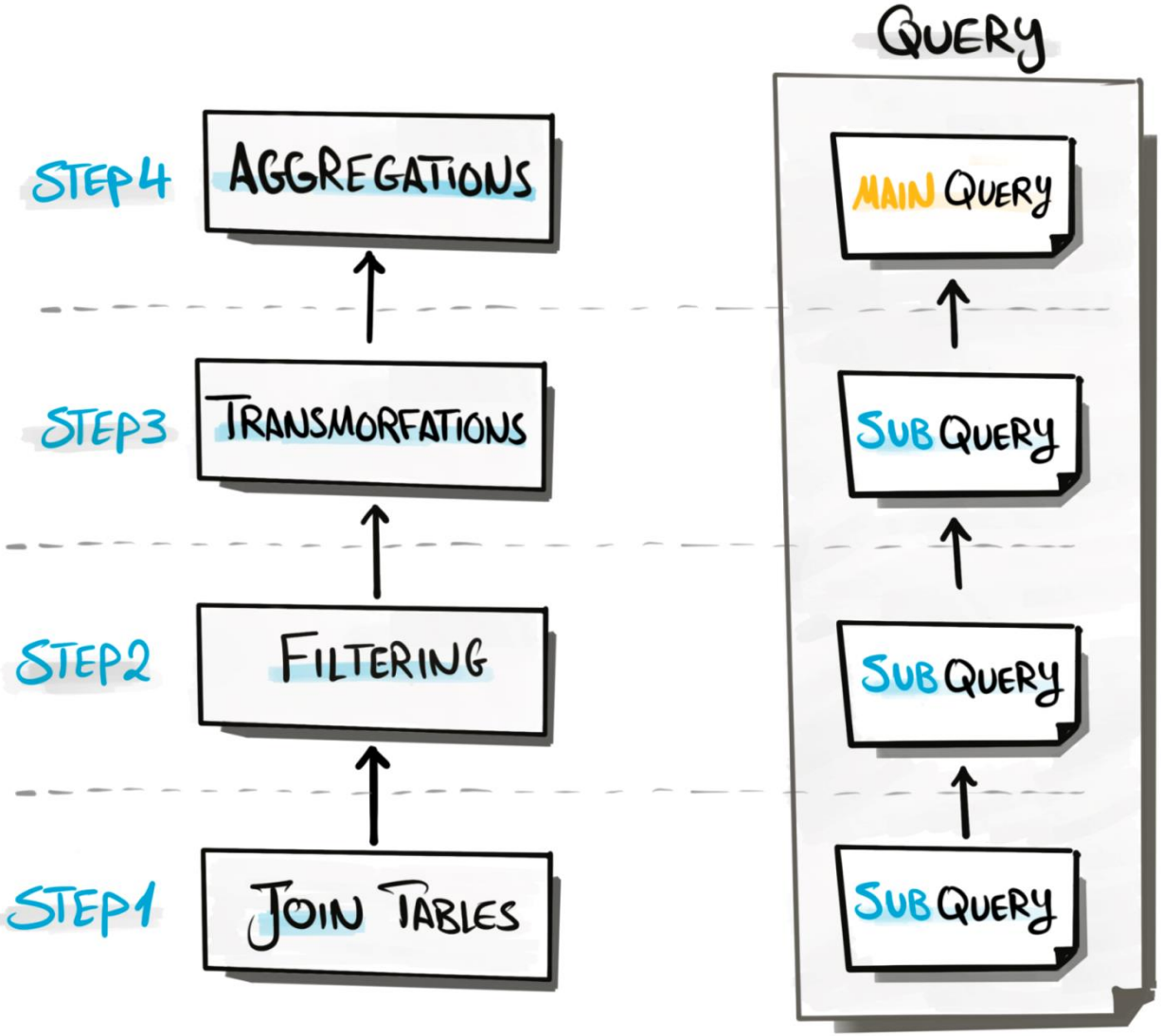


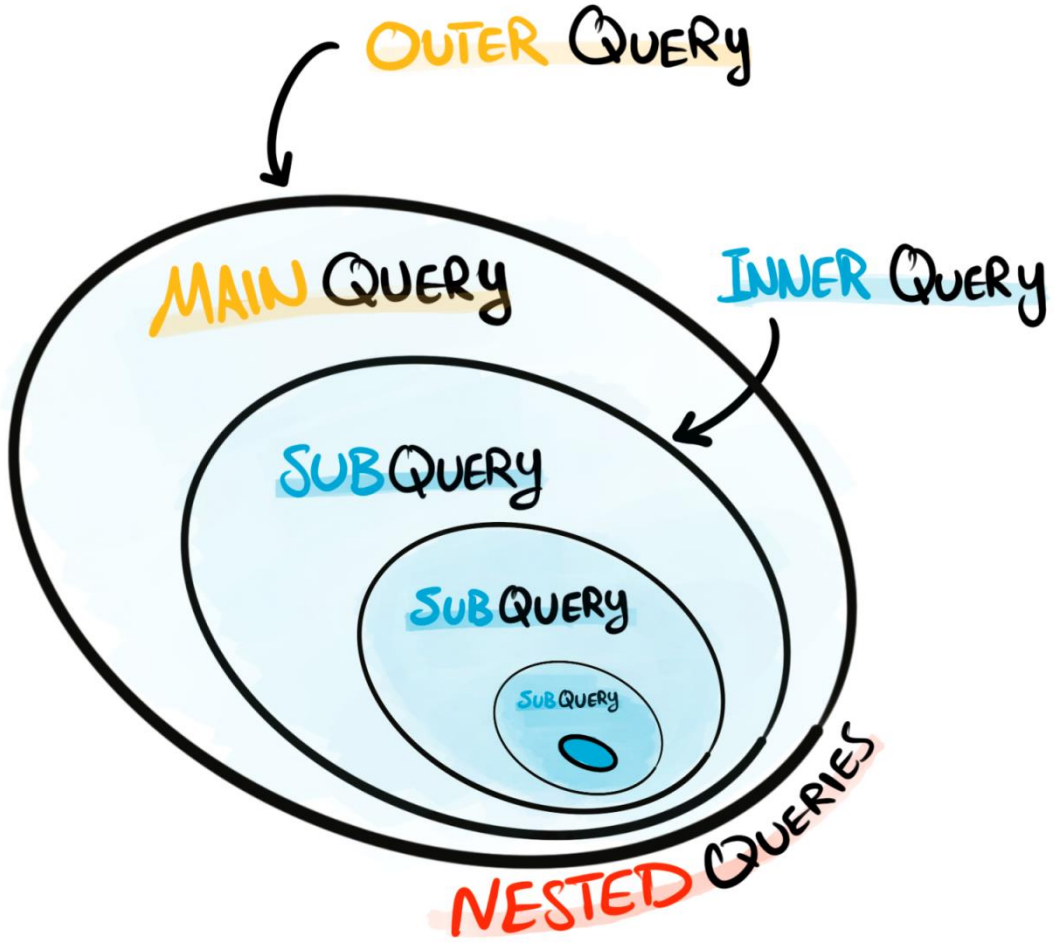
most of the time the DBMS creates an internal temporary result for a subquery, but not always a physical temp table.

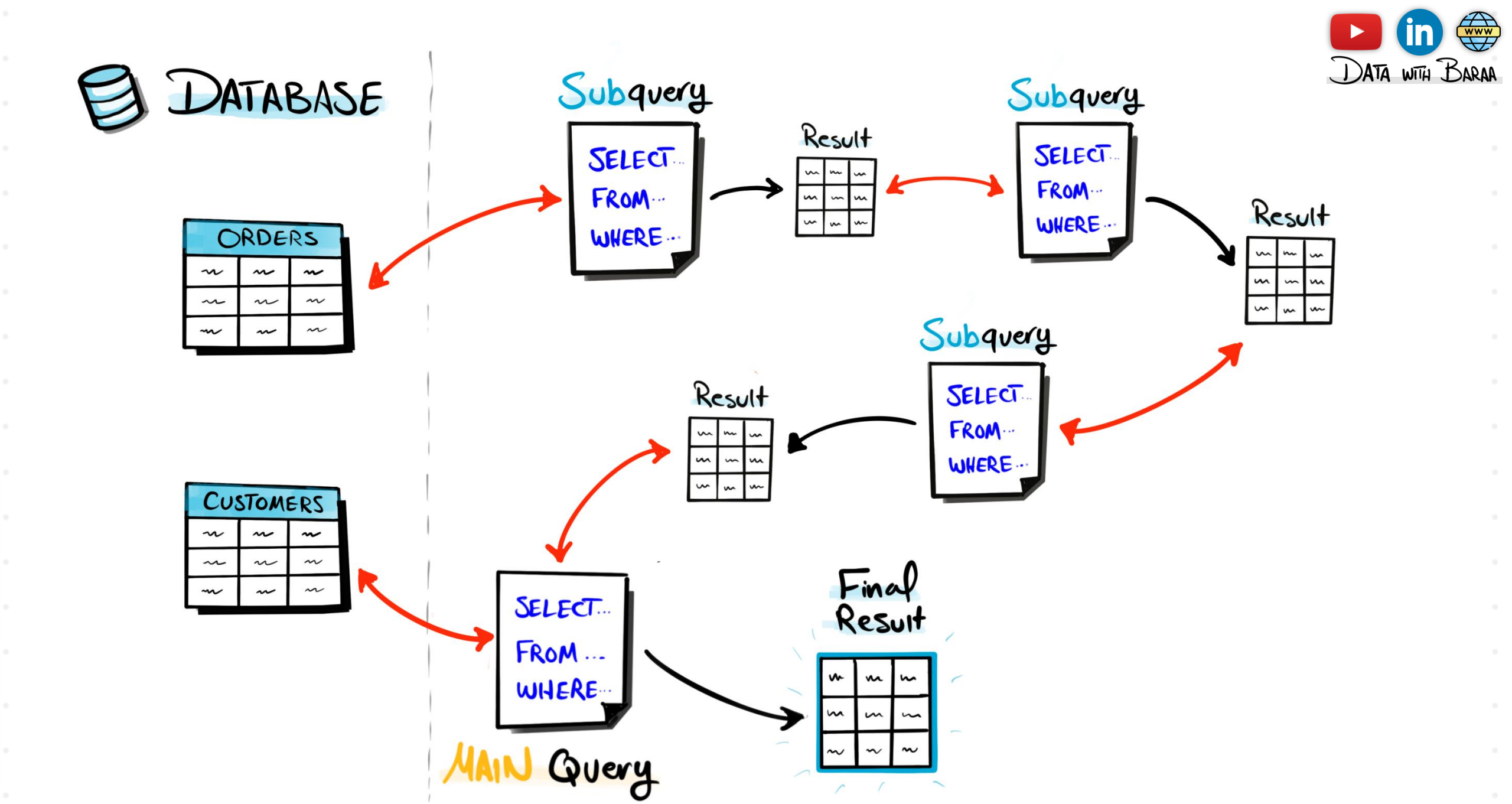
DBMS also uses its temp storage for its own use and when user creates temp tables.

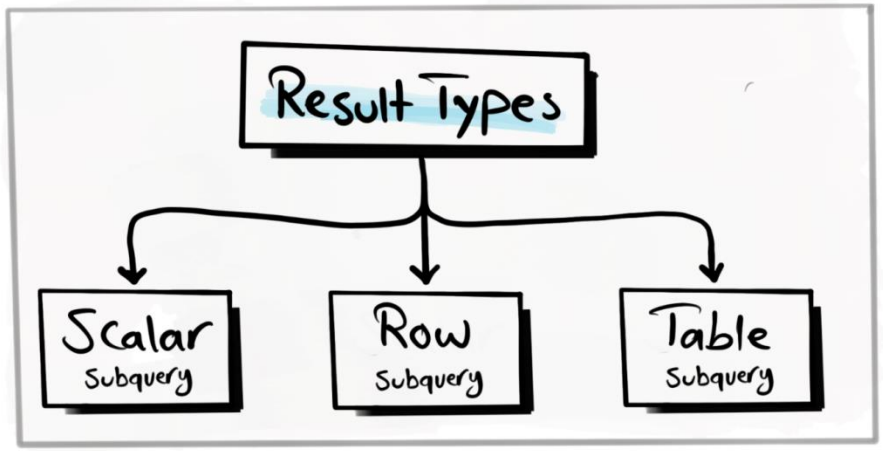
SUBQUERY

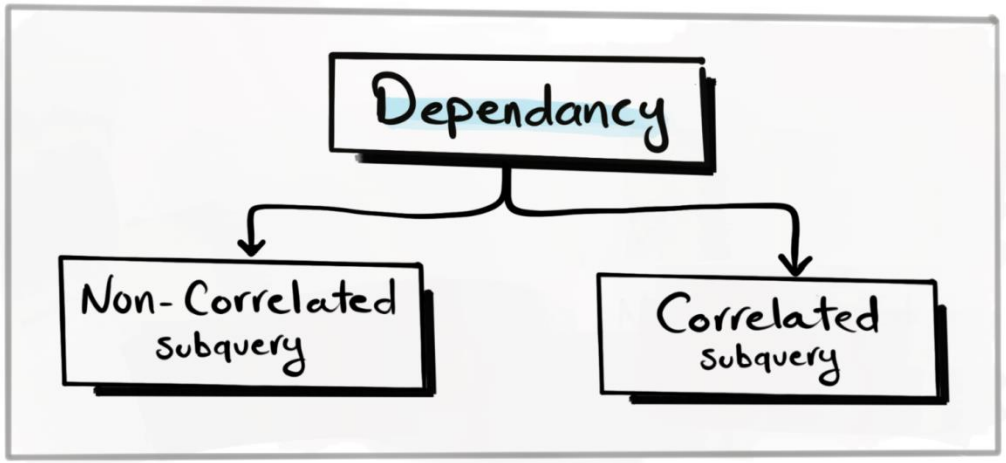
DATA WITH BARAA



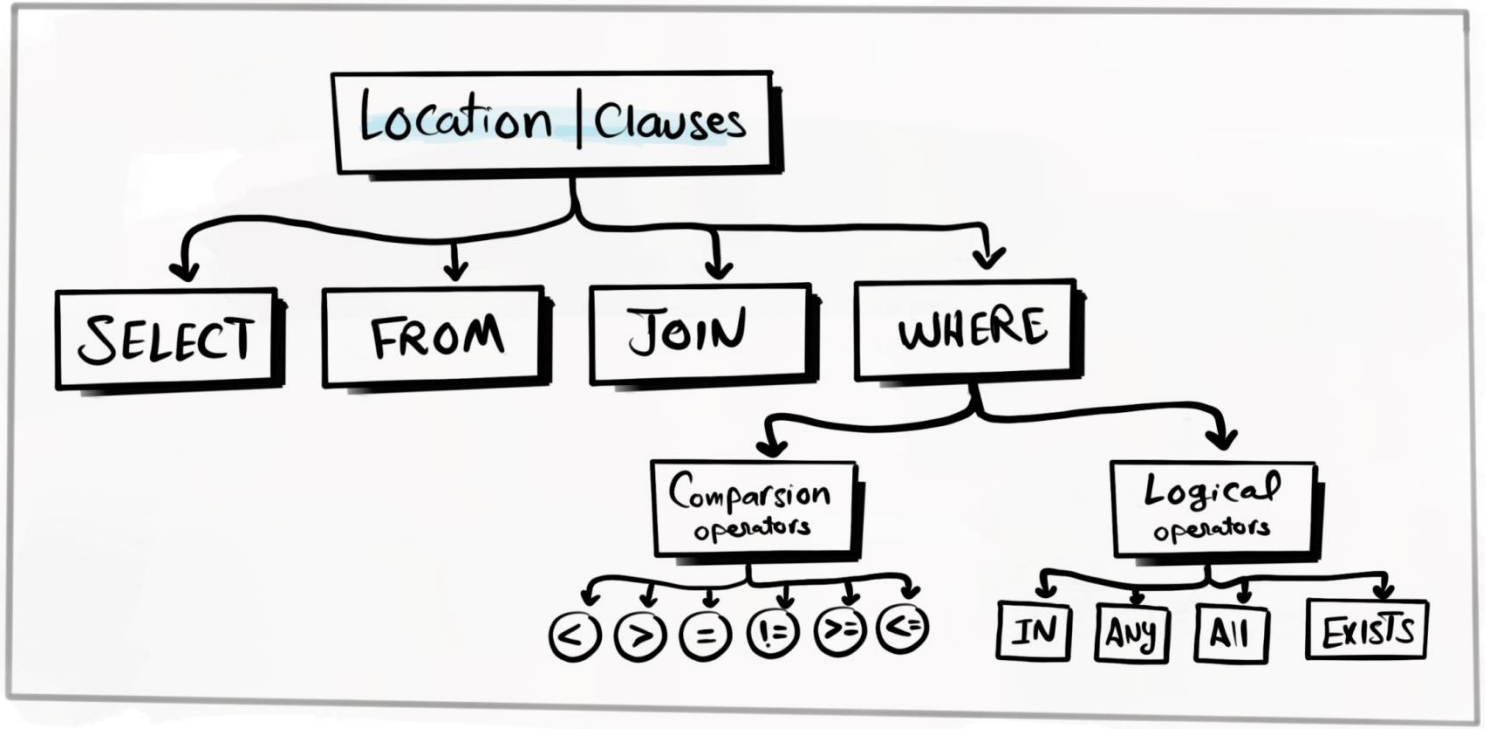



DATABASE
ORDERS
CUSTOMERS
Subquery
SELECT...
FROM...
WHERE...
Result
Subquery
SELECT...
FROM...
WHERE...
Result
Subquery
SELECT...
FROM...
WHERE...
Result
SELECT...
FROM...
WHERE...
MAIN Query
Final Result
DATA WITH BARAA

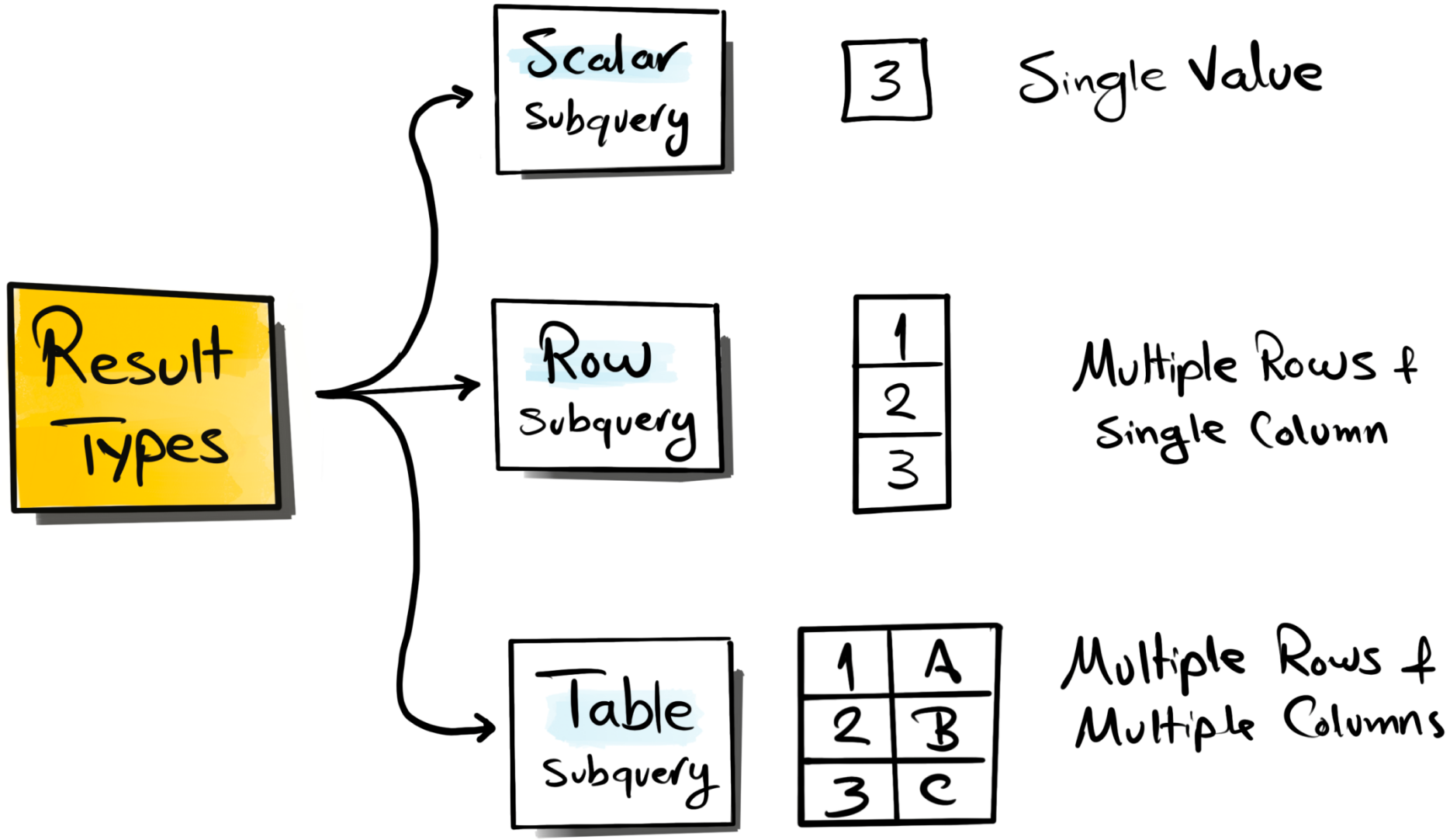


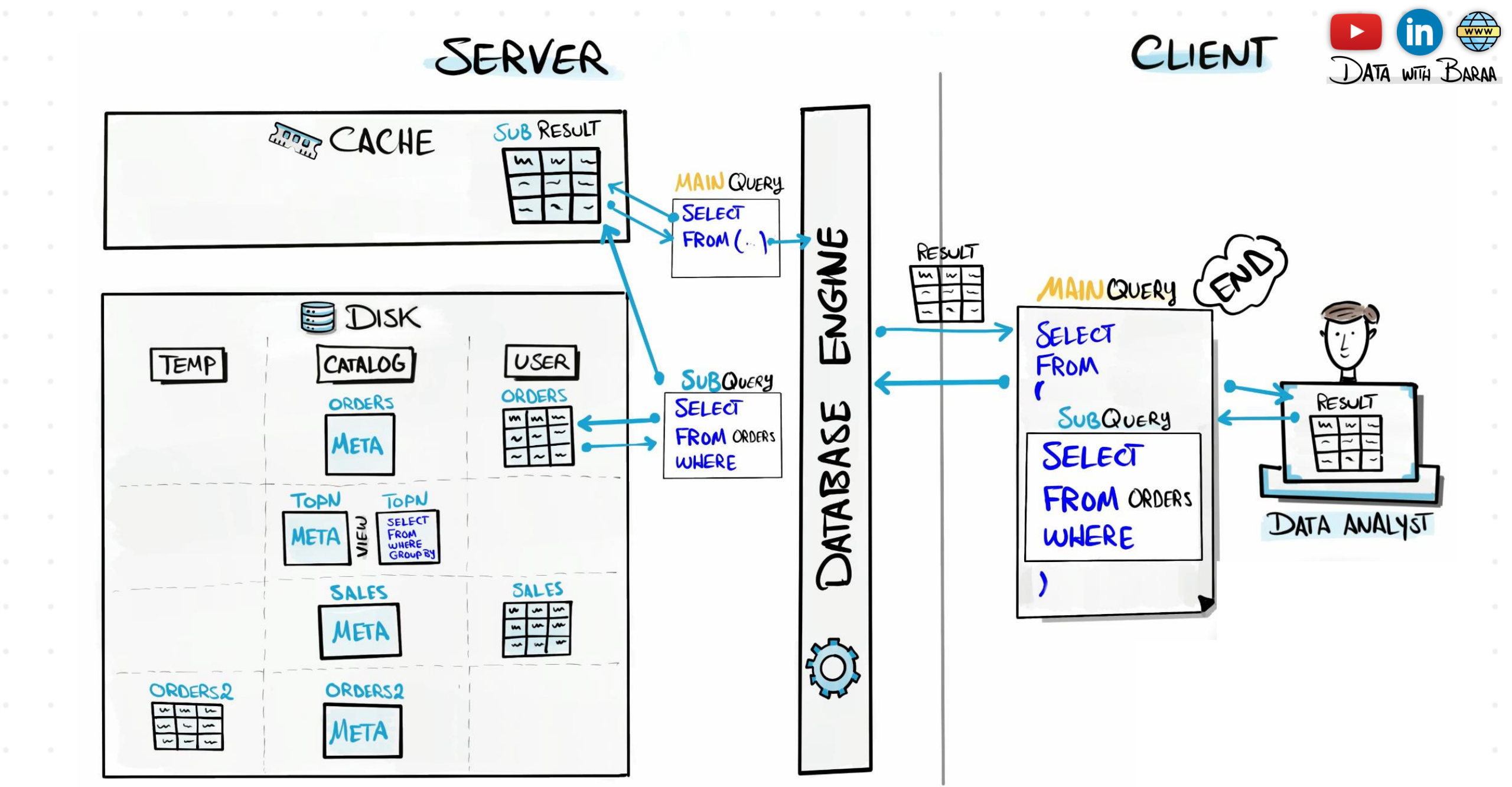


SUBQUERY



DATA WITH BARAA




SERVER
CLIENT
DATA WITH BARAA
CACHE
SUB RESULT
MAIN QUERY
SELECT
FROM (..)
DISK
TEMP
CATALOG
USER
ORDERS
META
ORDERS
TOPN
META
VIEW
TOPN
SELECT FROM WHERE GROUP BY
SALES
META
SALES
ORDERS2
ORDERS2
META
SUBQUERY
SELECT
FROM ORDERS
WHERE
DATABASE ENGINE
RESULT
MAIN QUERY
END
SELECT
FROM
(
SUBQUERY
SELECT
FROM ORDERS
WHERE
)
RESULT
DATA ANALYST

Subquery in FROM Clause

Main Query

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

Subquery

this sub-query runs after where because resultant table will be used in future operations.

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

Original Query

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

Subquery

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

CustomerID	TotalSales
1	110
2	55
3	125
4	90

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
```

1

Subquery

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

COUNT
10

2 pass

3

Main Query

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

4

FINAL RESULT

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

DATA WITH BARAA

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)`

DATA WITH BARAA

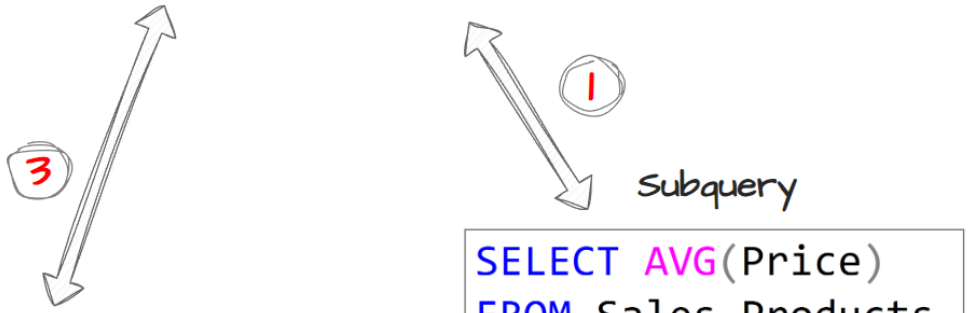
Subquery in WHERE Clause
Comparison Operators

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)
```



FINAL RESULT

ProductID	Price
104	25
105	30

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 ...)`

DATA WITH BARAA

Subquery in WHERE Clause
In Operator

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

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

1

Subquery

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

CustomerID
1
4

2 pass

3

Main Query

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

4

FINAL RESULT

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

Subquery in
WHERE Clause
ALL Operators

Main Query

```
SELECT column1, column2,...
FROM   table1
WHERE  column < ALL( SELECT column FROM table1 WHERE condition )
```

Subquery

DATA WITH BARAA

Subquery in WHERE Clause
ANY Operator

Main Query

```
SELECT column1, column2,...
FROM   table1
WHERE  column < ANY( SELECT column FROM table1 WHERE condition )
```

Subquery

DATA WITH BARAA

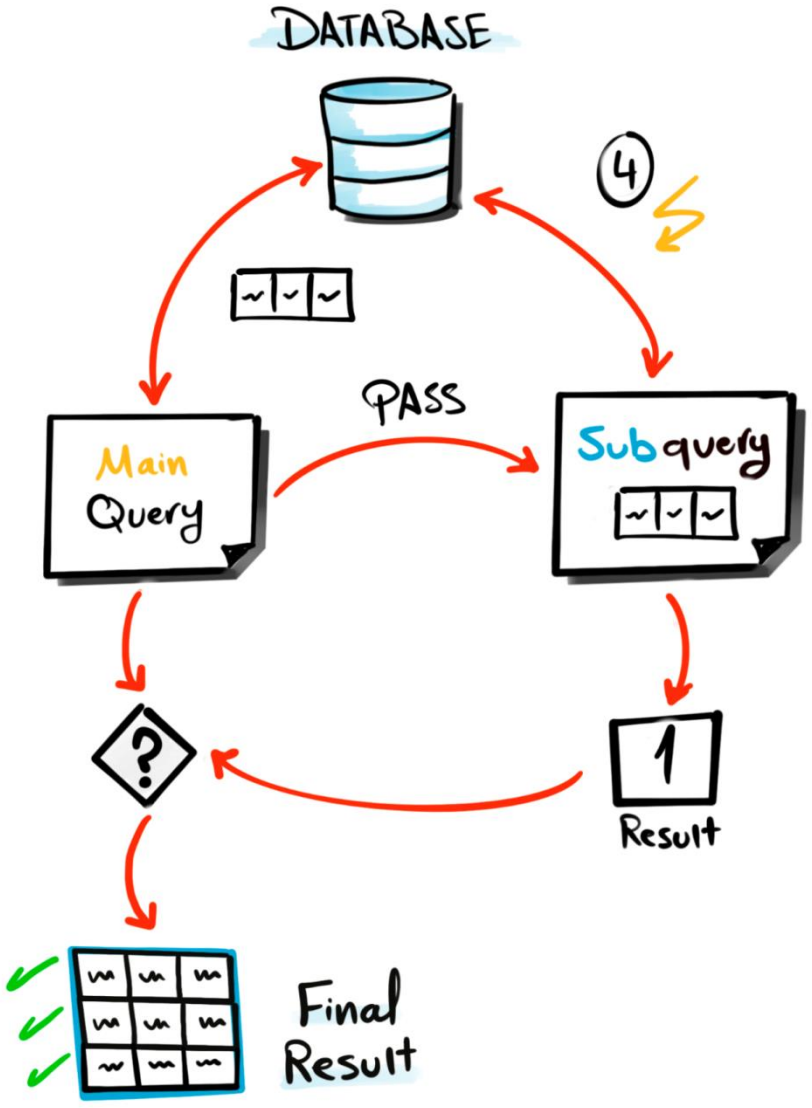
NON-CORRELATED SUBQUERY

A Subquery that can run independtly from the Main Query

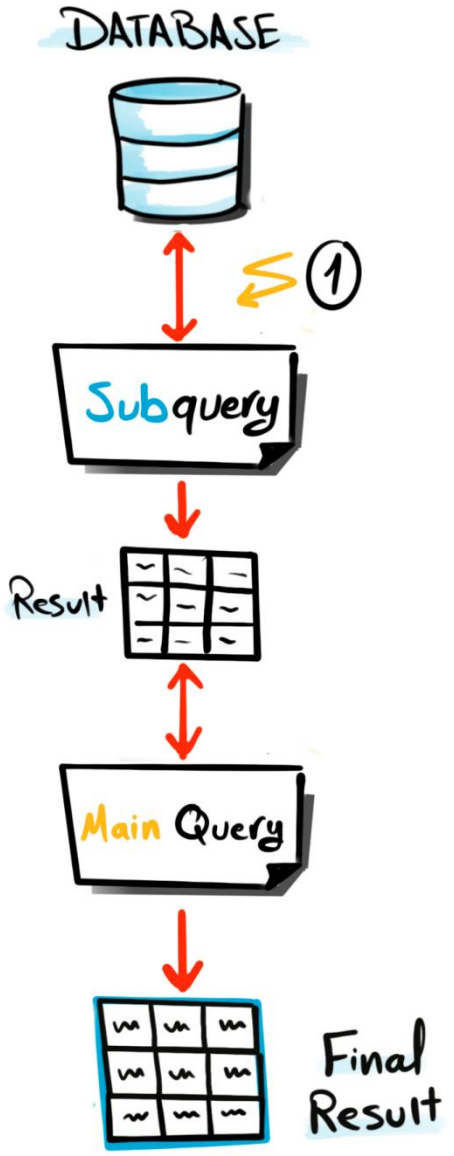
CORRELATED SUBQUERY

A Subquery that relays on values from the Main Query

Correlated Subquery



Non-Correlated Subquery



DATA WITH BARAA

	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 Can be executed on its Own	Executed for each row processed by the main query 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

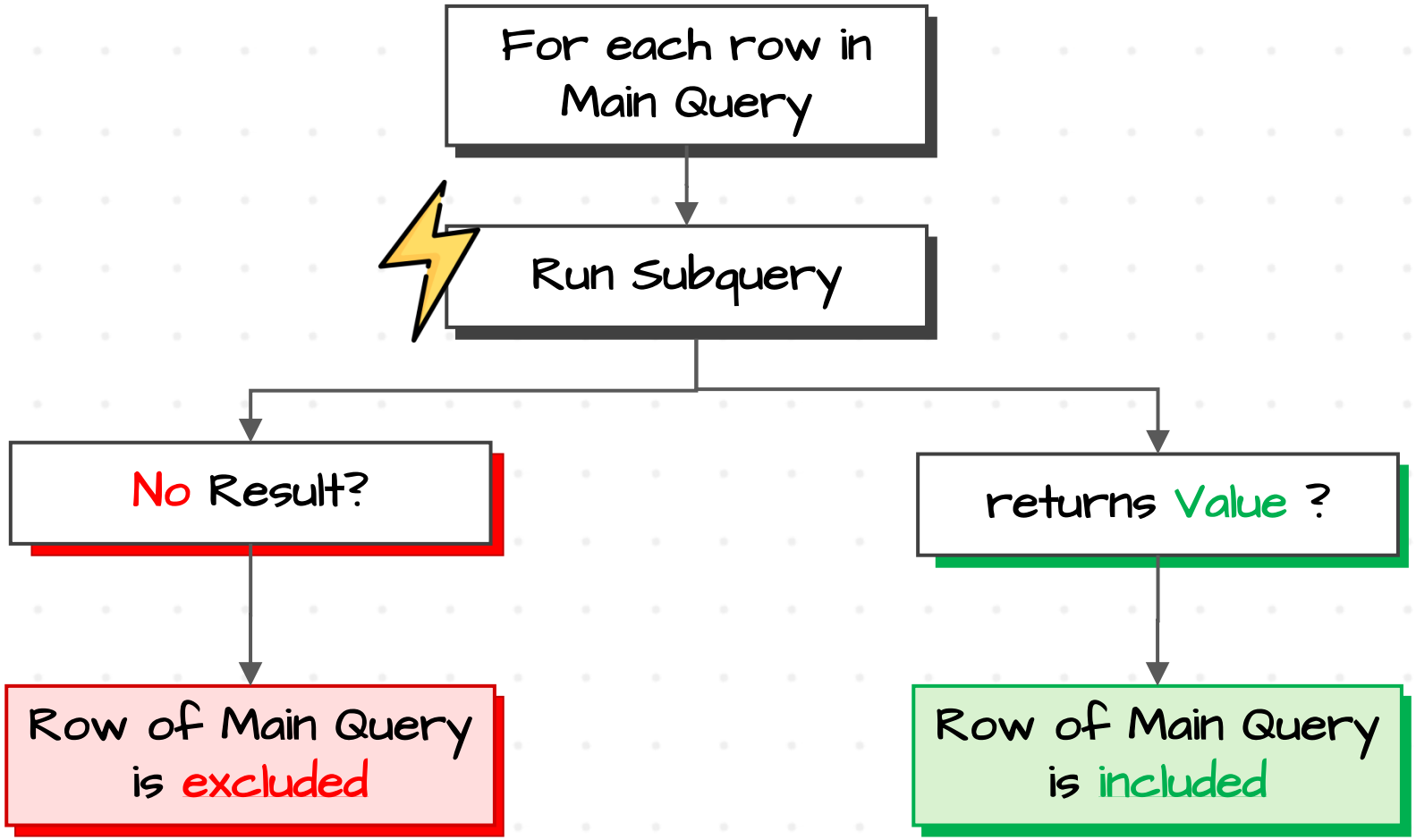
Correlated Subquery in WHERE Clause EXISTS Operator

Main Query

```
SELECT column1, column2,...

FROM   Table2

WHERE  EXISTS (  SELECT 1

                 FROM Table1

                 WHERE Table1.ID = Table2.ID  )
```

Subquery

How EXISTS Works?



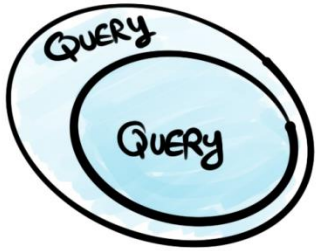
DATA WITH BARAA

	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

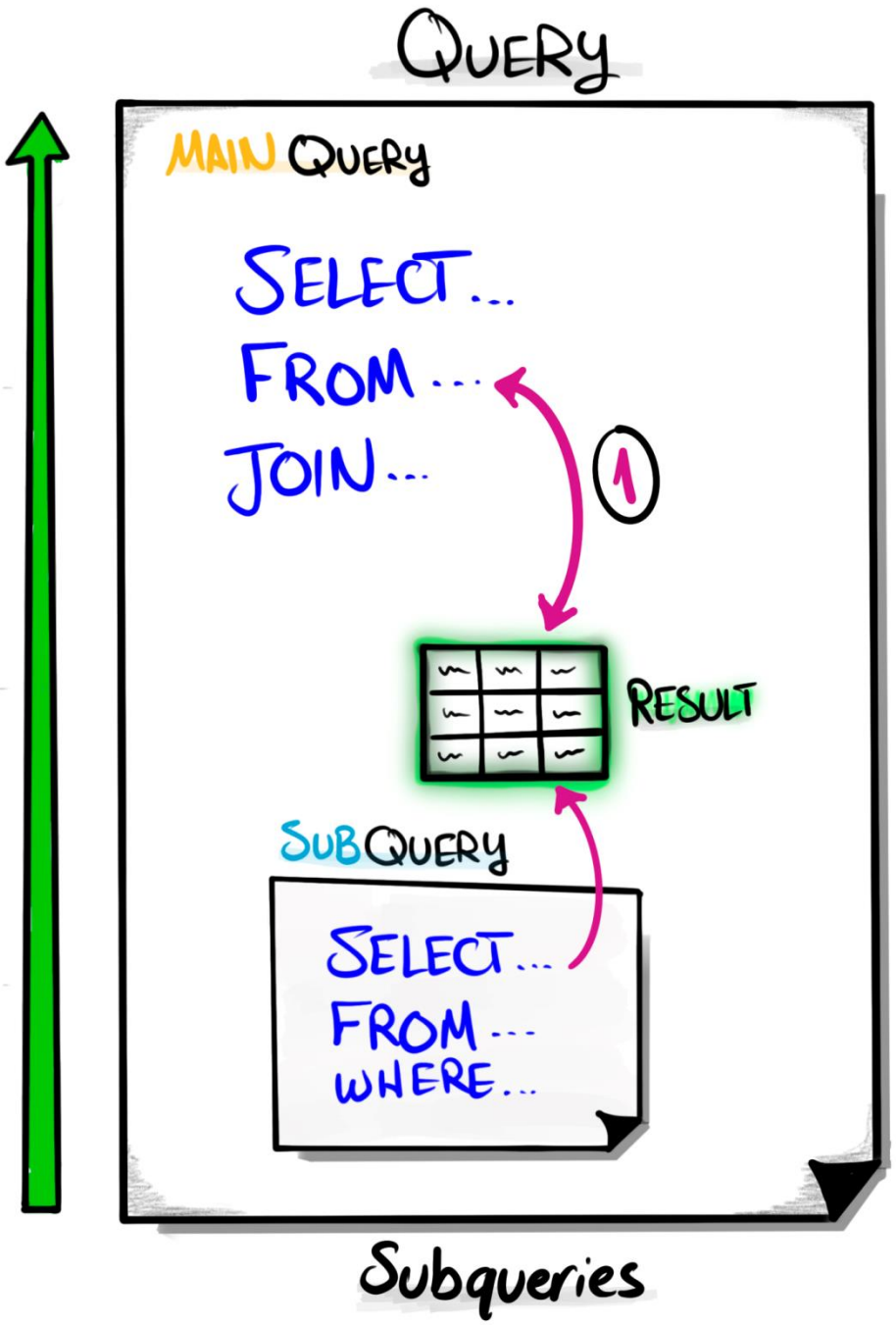
- Create Temporary Result Set.
- Prepare Data Befor Joining Tables.
- Dynamic & Complex Filtering.
- Check the Existence of Rows From another Table. (EXISTS)
- Row By Row Comparison - Correlated Subquery -

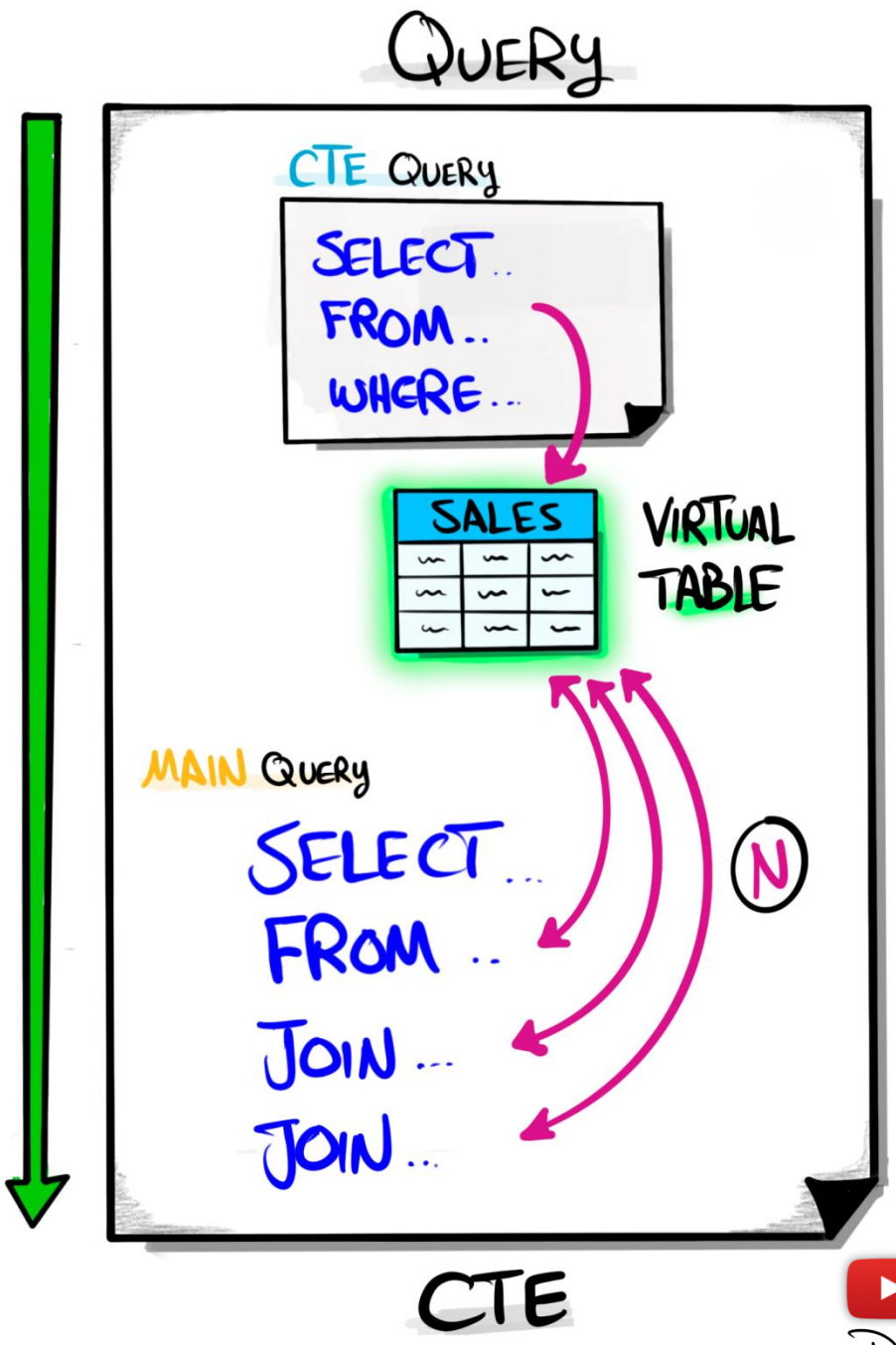
DATA WITH BARAA

CTE

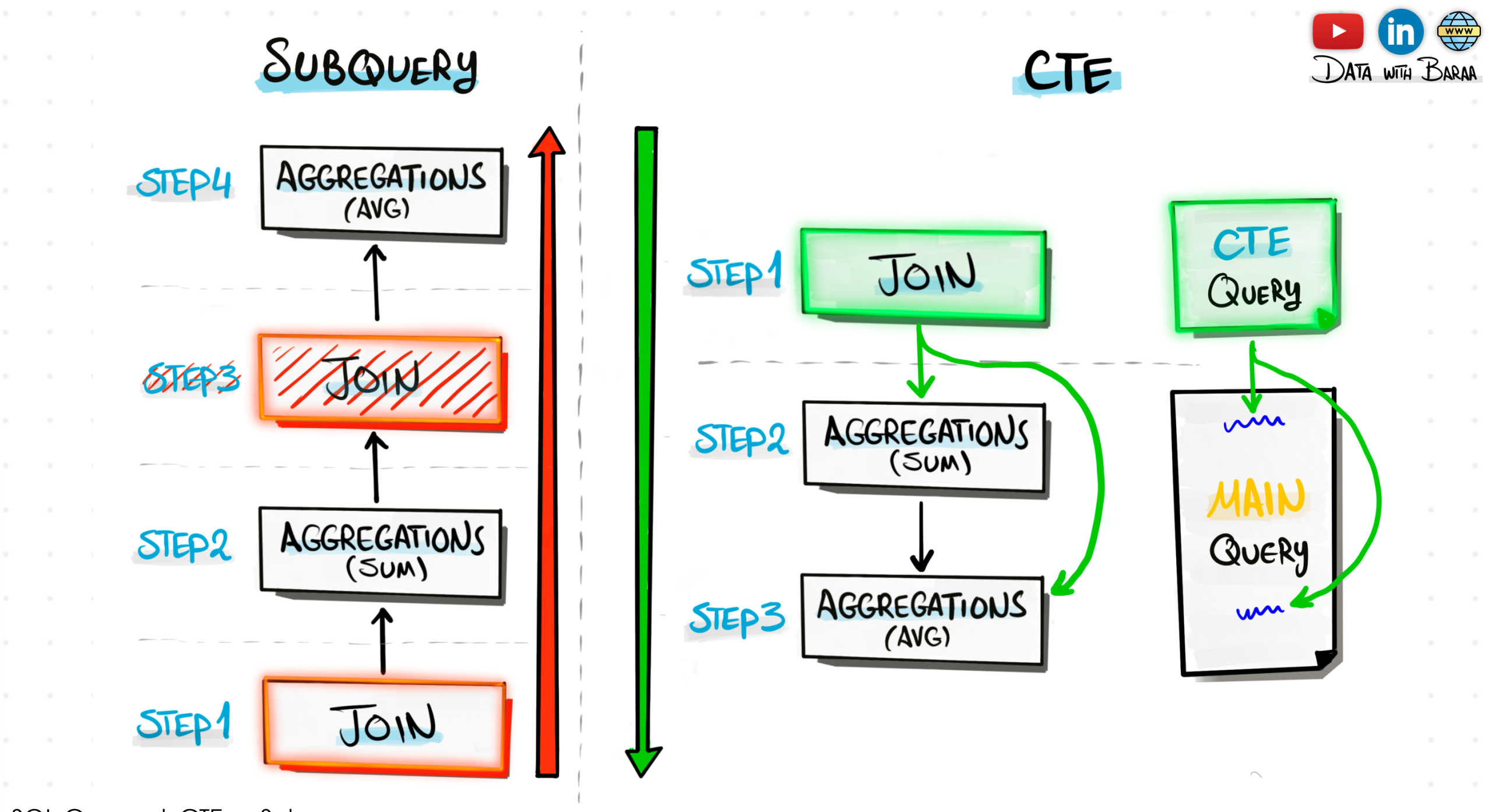
Common Table Expression

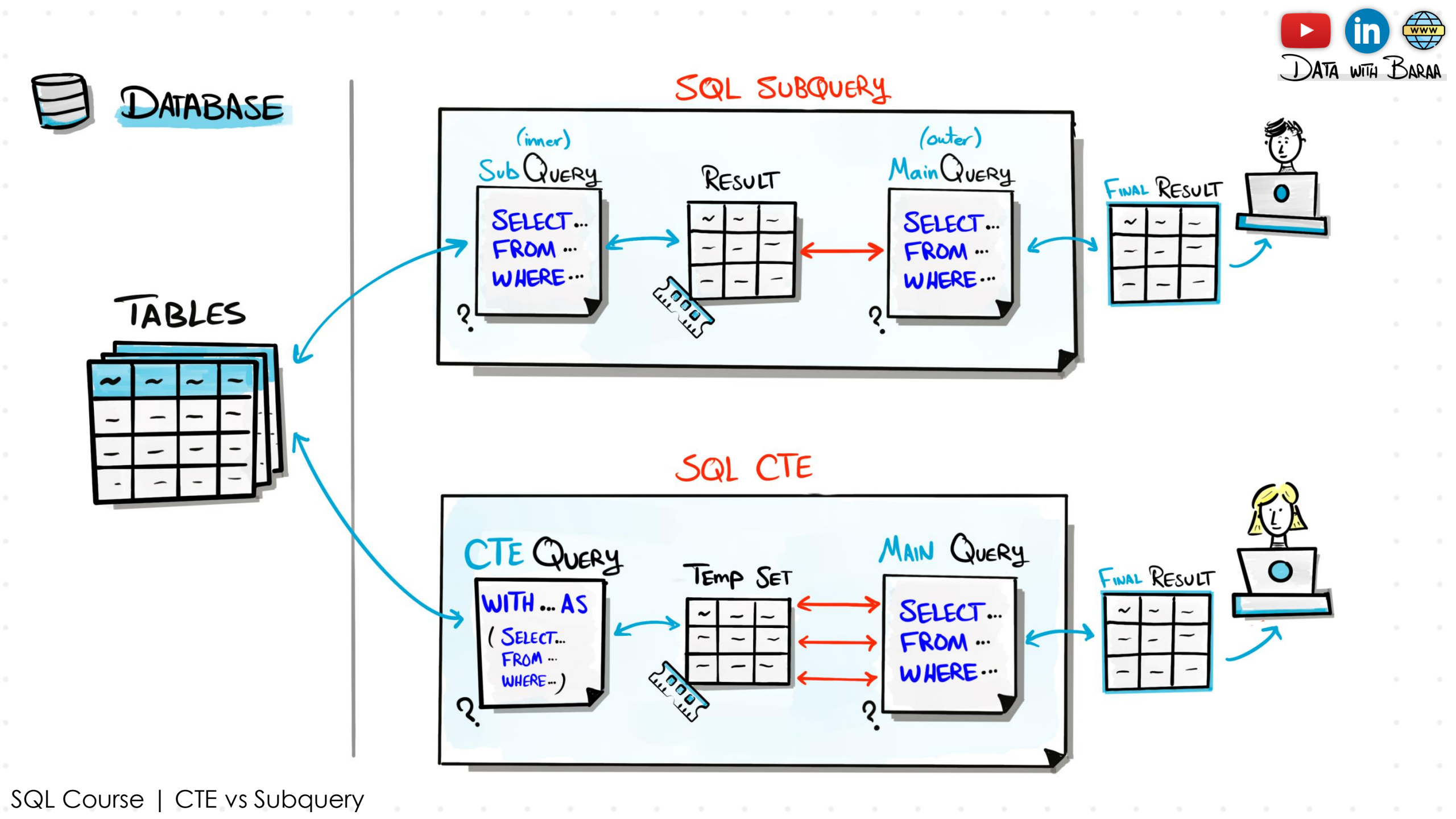
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | CTE

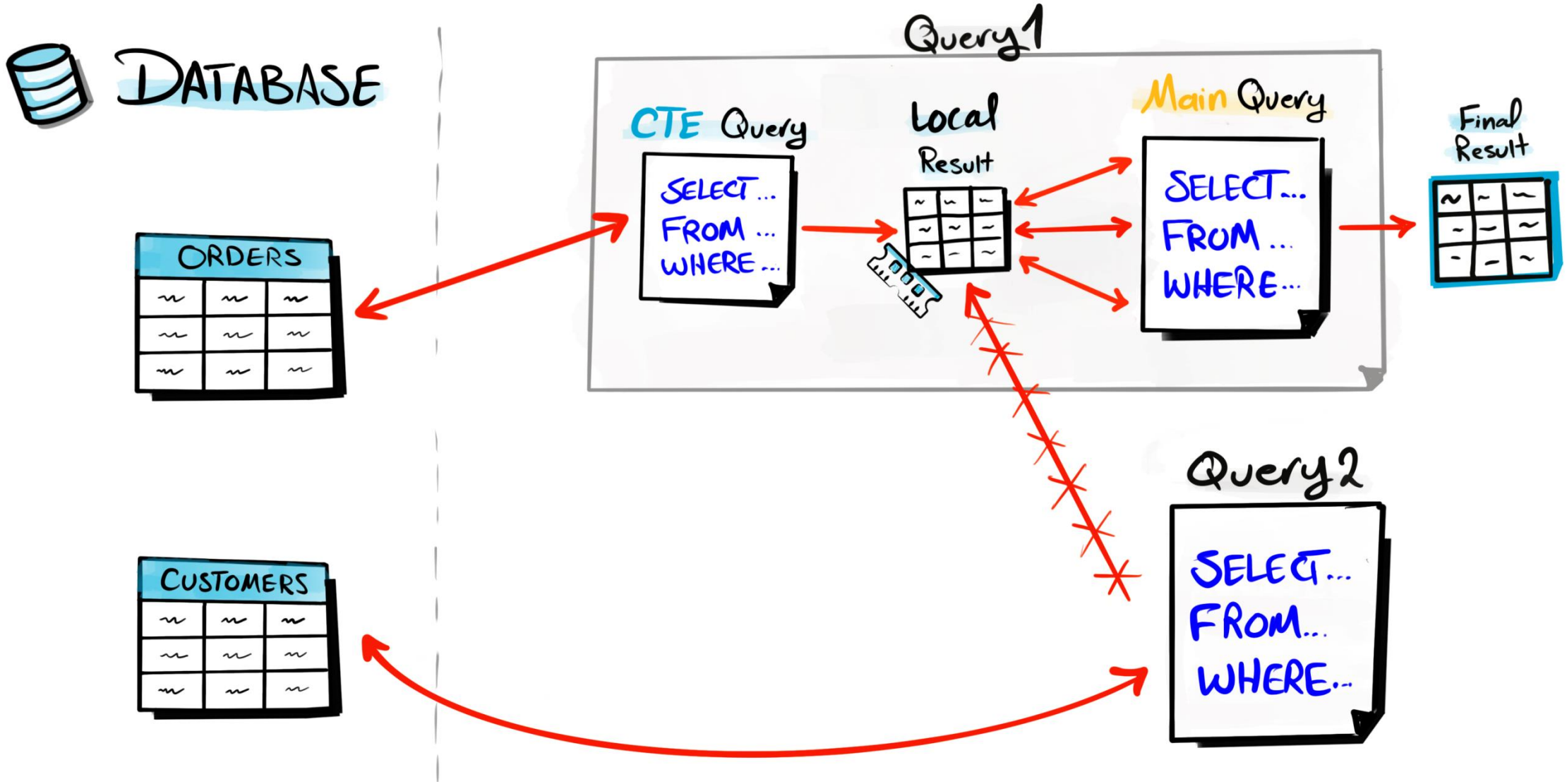


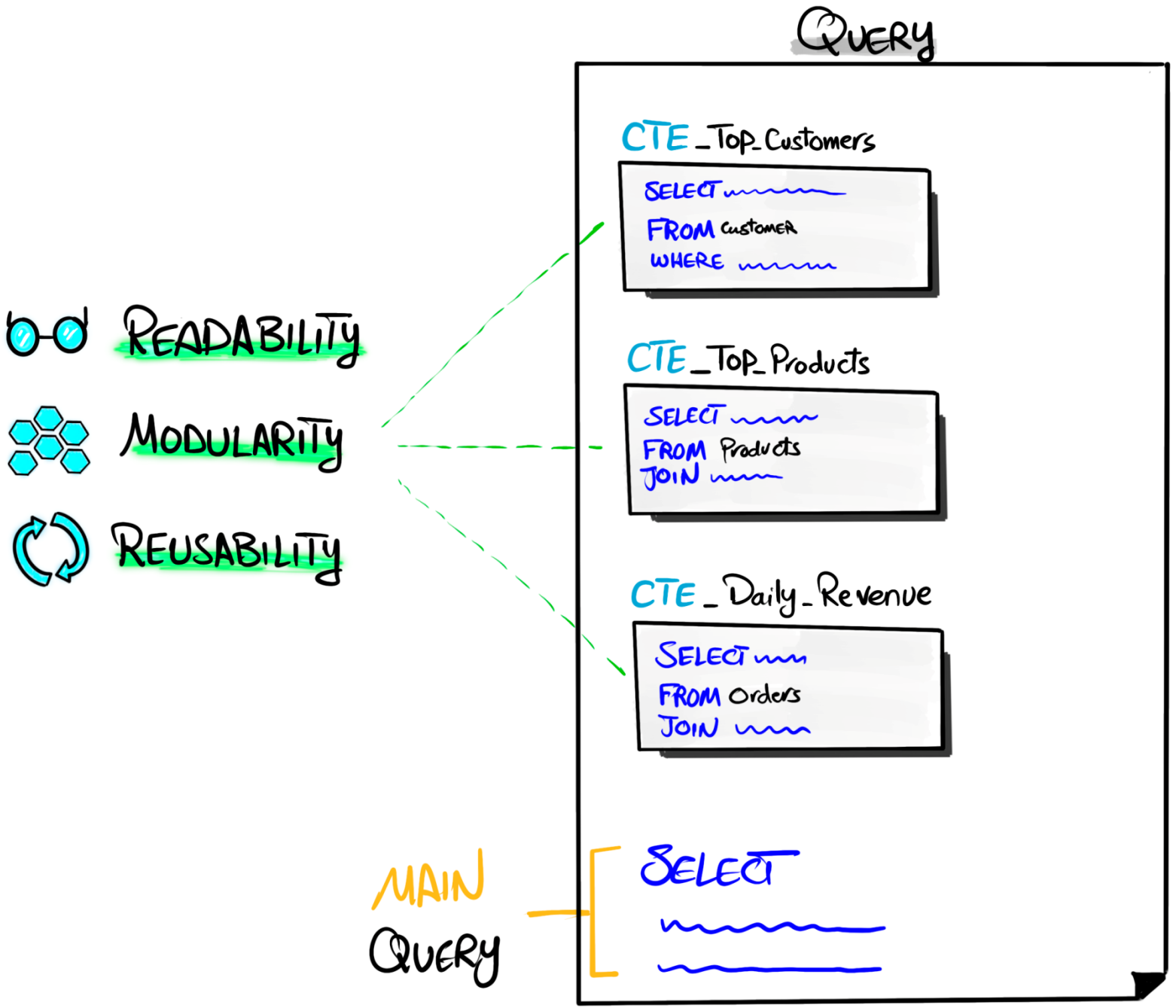


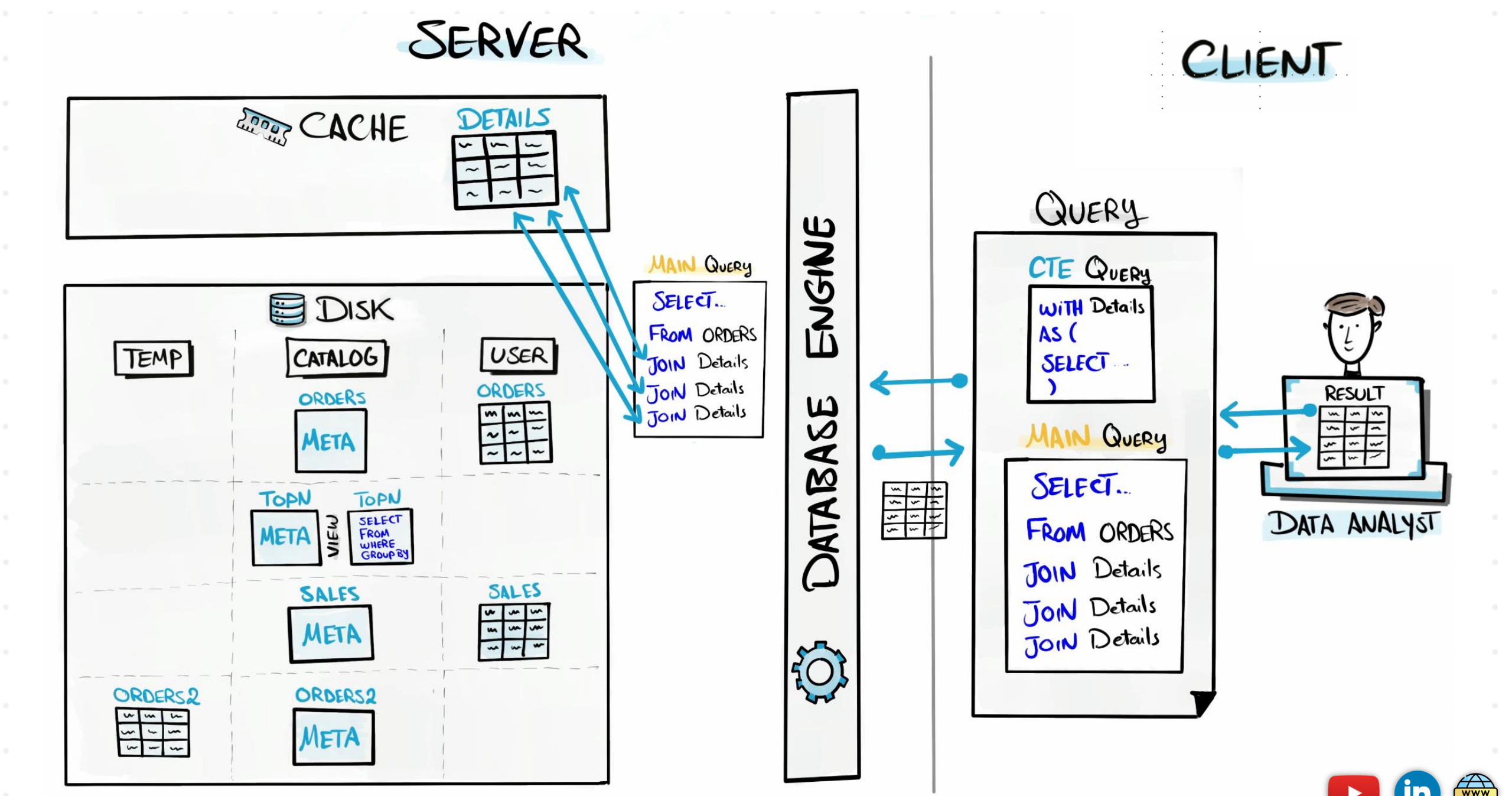
DATA WITH BARAA

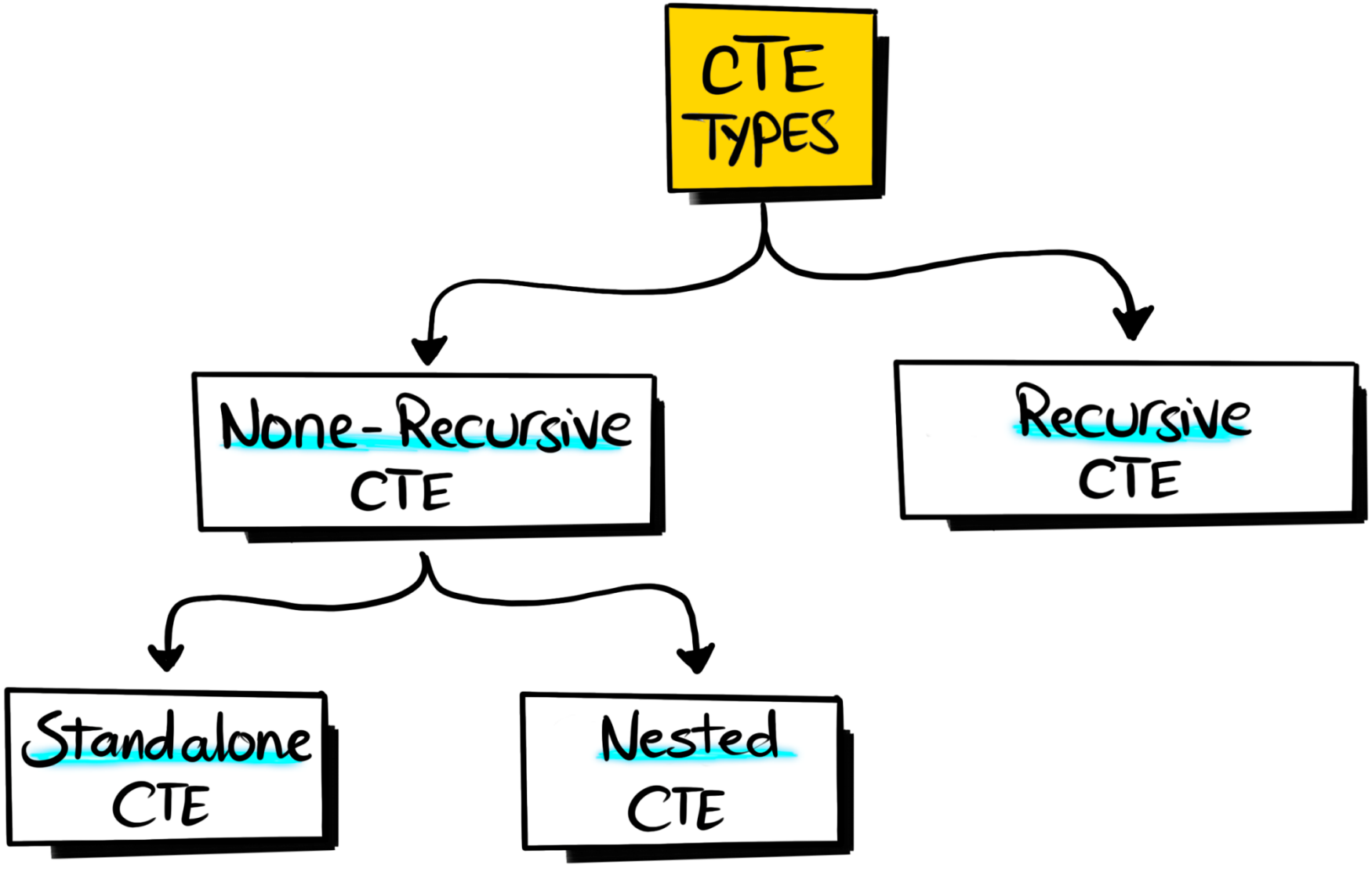

SUBQUERY
STEP4
AGGREGATIONS
(AVG)
STEP3
JOIN
STEP2
AGGREGATIONS
(SUM)
STEP1
JOIN
CTE
STEP1
JOIN
STEP2
AGGREGATIONS
(SUM)
STEP3
AGGREGATIONS
(AVG)
CTE
QUERY
MAIN
QUERY
DATA WITH BARAA


DATABASE
TABLES
SQL SUBQUERY
(inner)
SubQuery
SELECT...
FROM ...
WHERE...
RESULT
(outer)
MainQuery
SELECT...
FROM ...
WHERE...
FINAL RESULT
SQL CTE
CTE Query
WITH ... AS
(SELECT...
FROM ...
WHERE...)
TEMP SET
MAIN QUERY
SELECT...
FROM ...
WHERE...
FINAL RESULT


DATABASE
ORDERS
CUSTOMERS
Query1
CTE Query
SELECT...
FROM...
WHERE...
Local
Result
Main Query
SELECT...
FROM...
WHERE...
Final
Result
Query2
SELECT...
FROM...
WHERE...


QUERY
CTE_Top_Customers
SELECT
FROM customer
WHERE
CTE_Top_Products
SELECT
FROM Products
JOIN
CTE_Daily_Revenue
SELECT
FROM Orders
JOIN
MAIN QUERY
SELECT
READABILITY
MODULARITY
REUSABILITY




CTE TYPES
None-Recursive CTE
Recursive CTE
Standalone CTE
Nested CTE

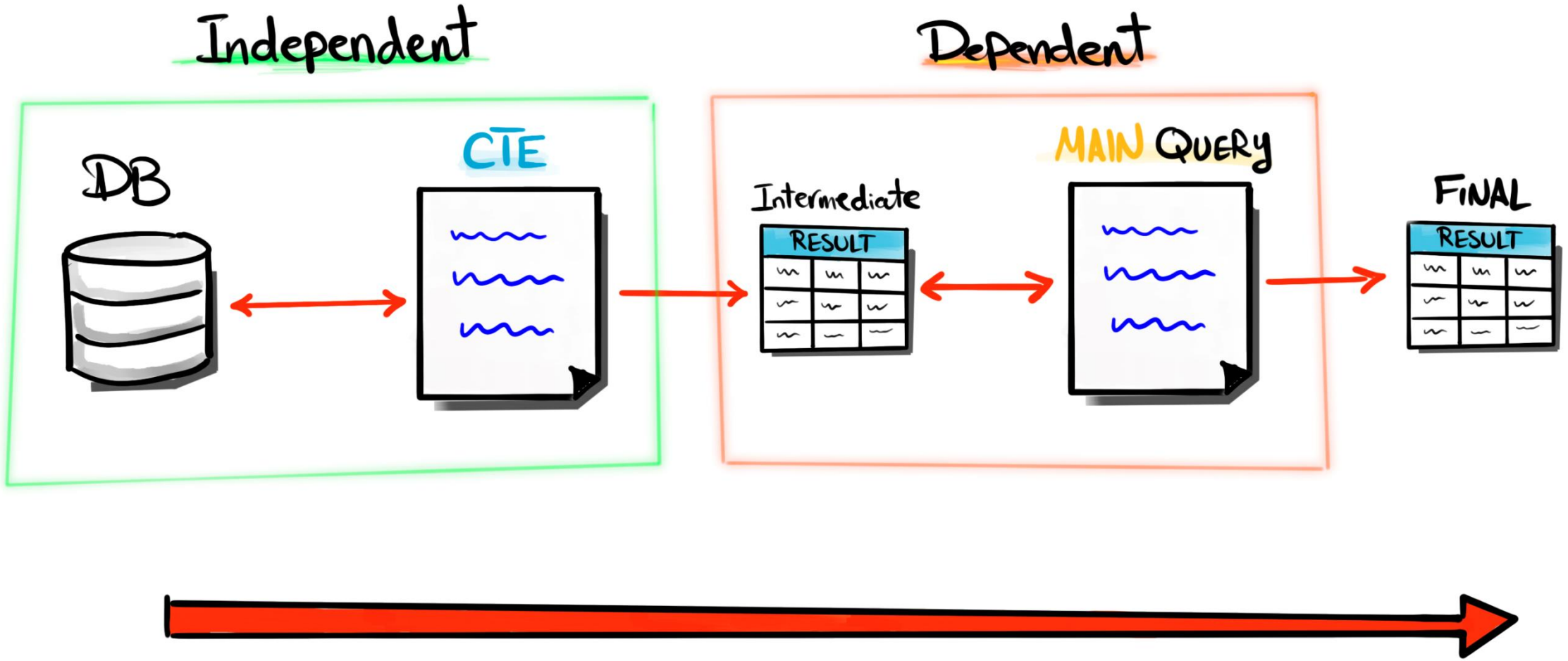
Standalone CTE

Defined and Used independently.

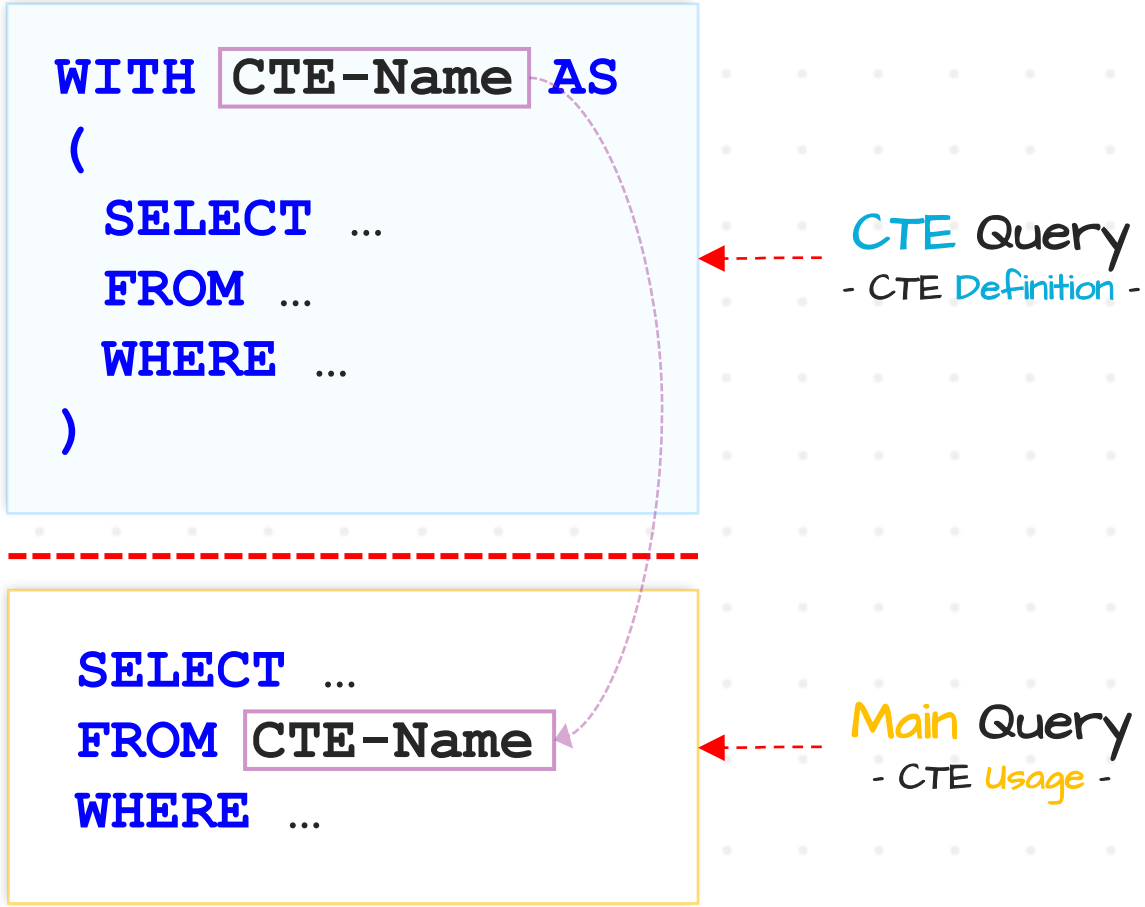
Runs independently as it's self-contained

and doesn't rely on other CTEs or queries.

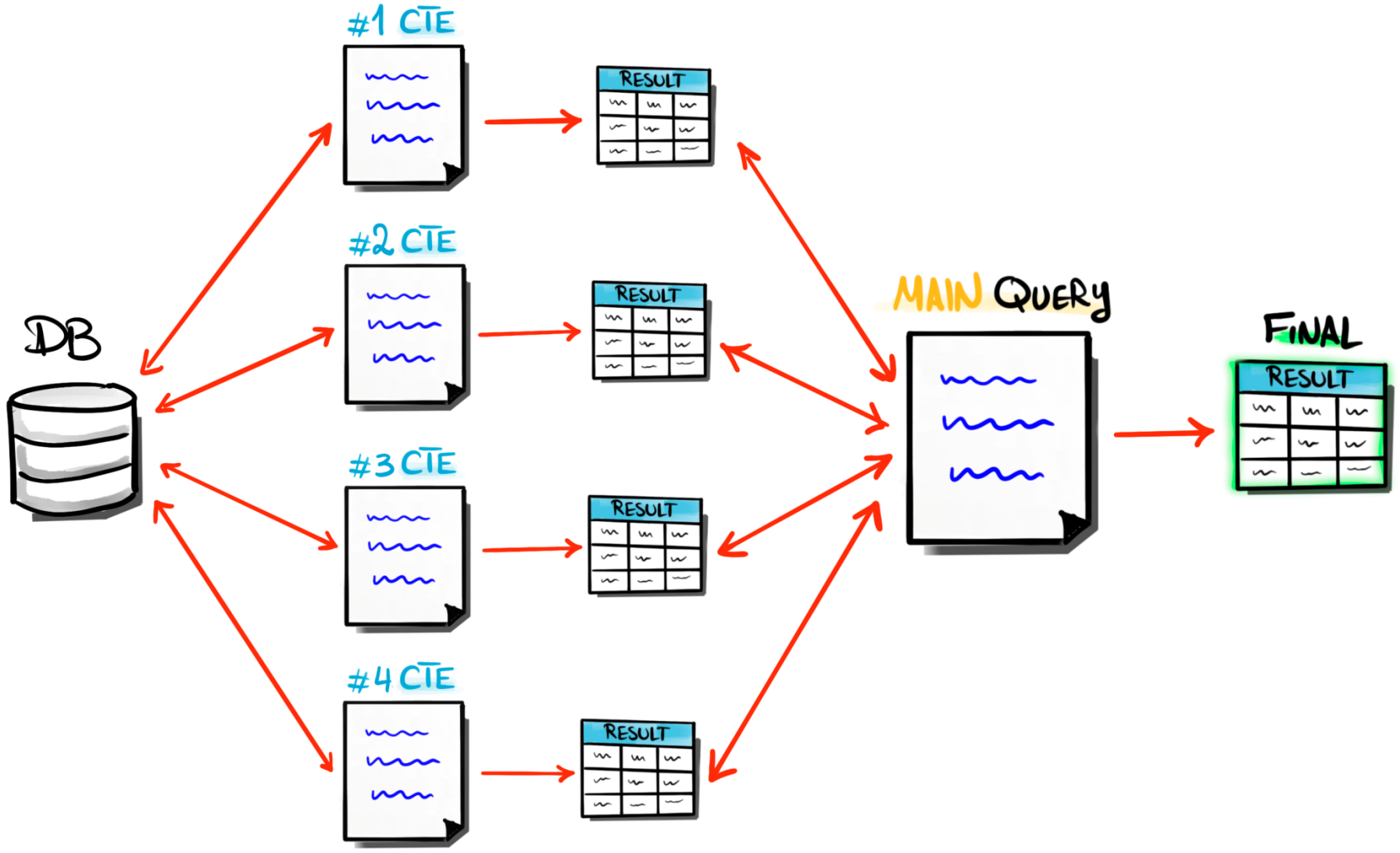
Standalone CTE



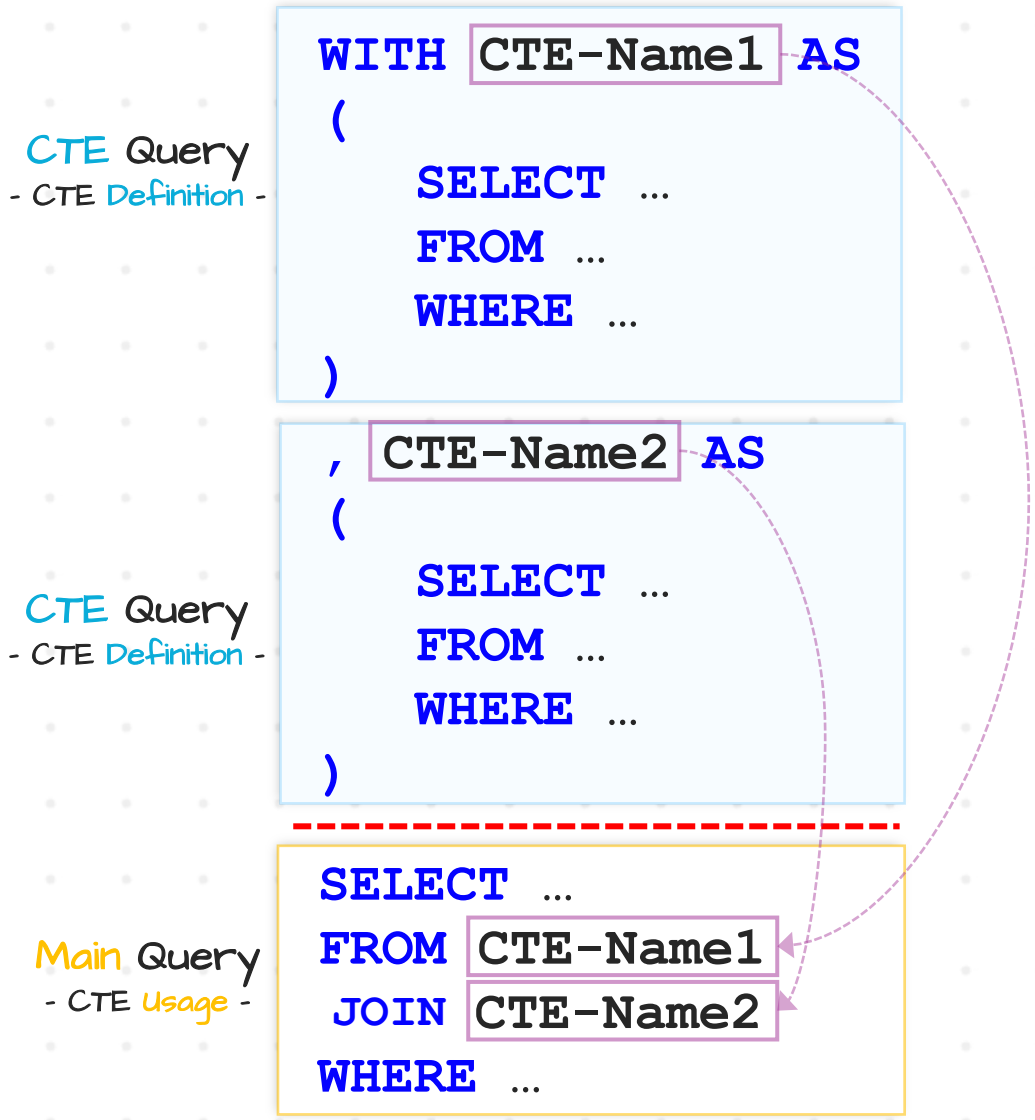
Standalone CTE



Multiple CTEs



Multiple CTEs



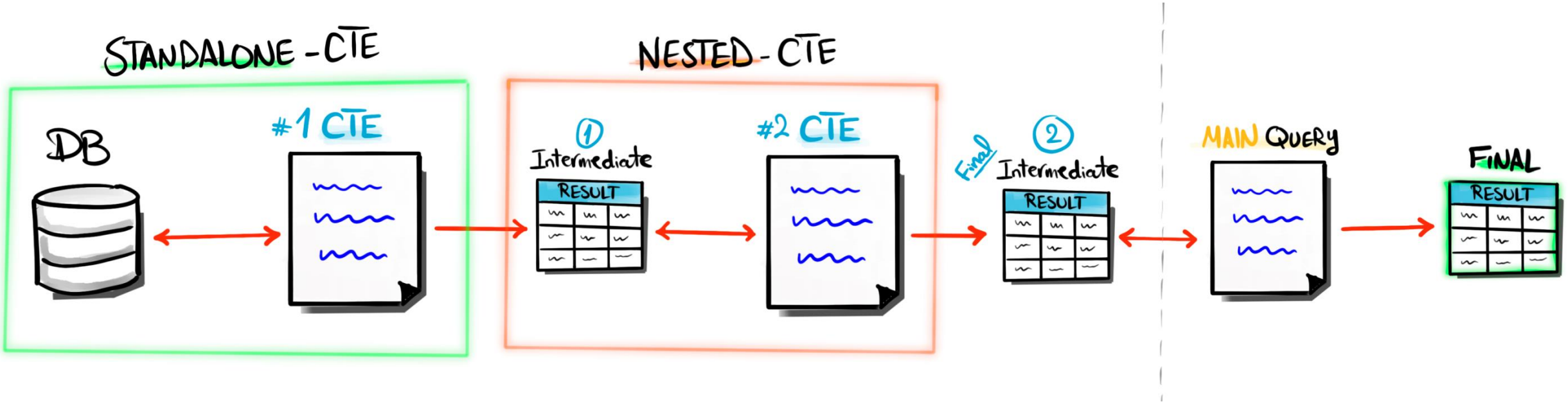
Nested CTE

CTE inside another CTE

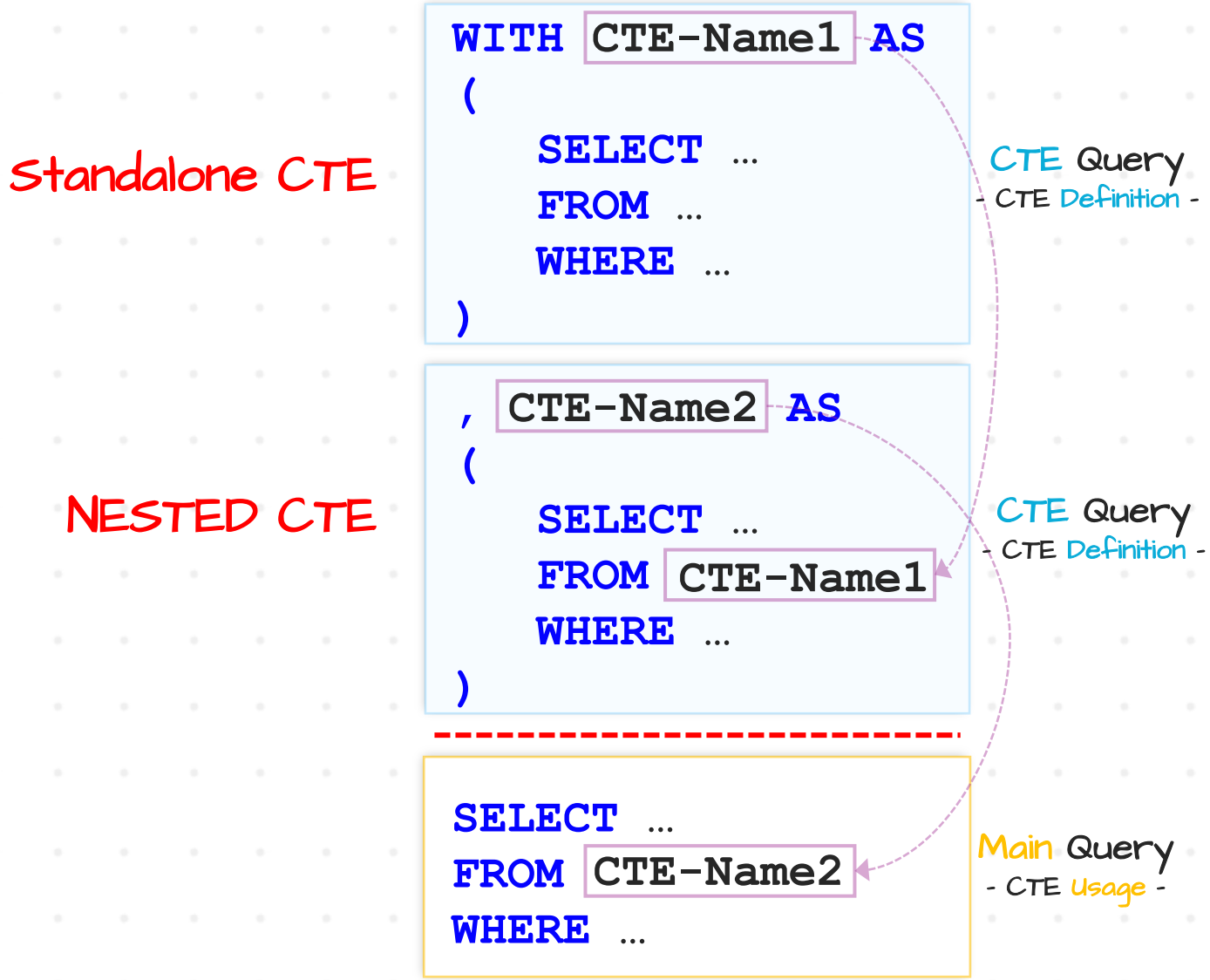
A nested CTE uses the result of another CTE,

so it can't run independently.

Nested CTEs


STANDALONE - CTE
DB
#1 CTE
NESTED - CTE
1
Intermediate
RESULT
#2 CTE
Final
2
Intermediate
RESULT
MAIN QUERY
FINAL
RESULT

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.

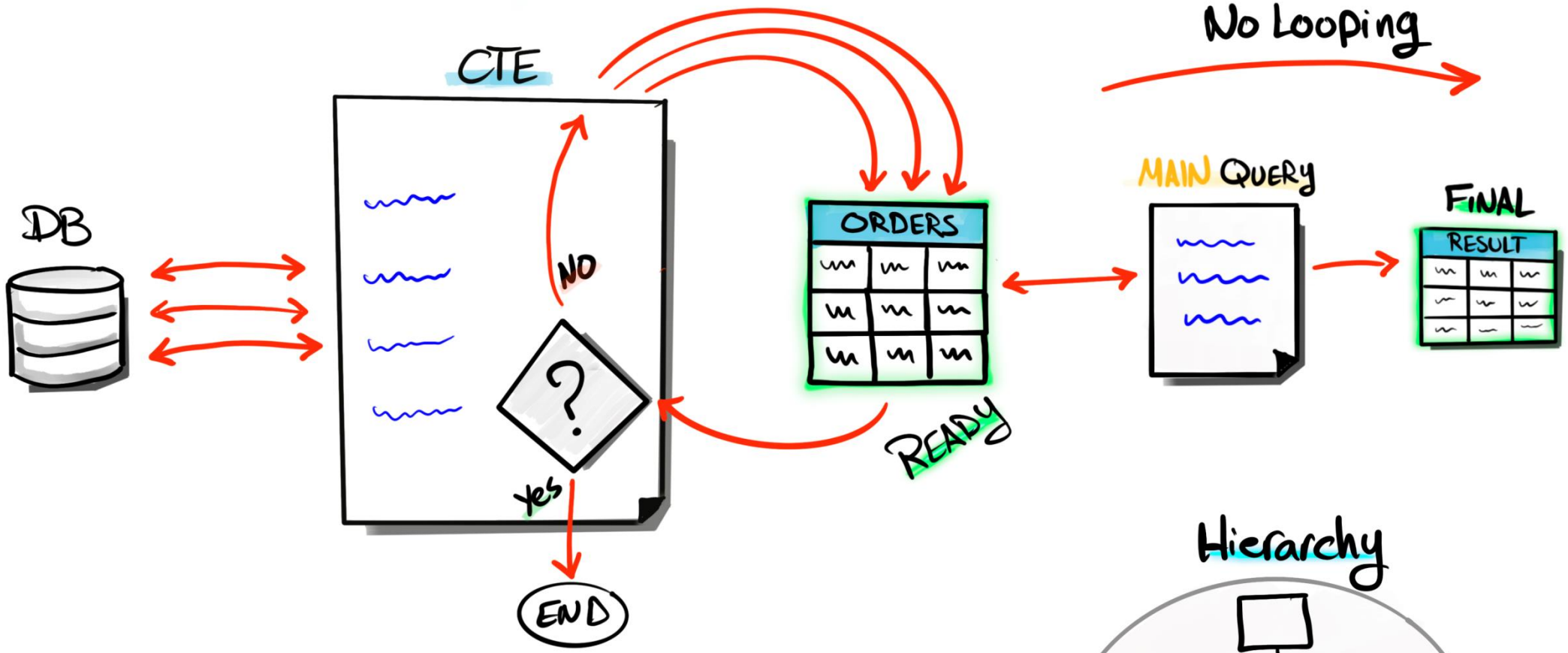
Non-Recursive CTE

is executed only once without any repetition.

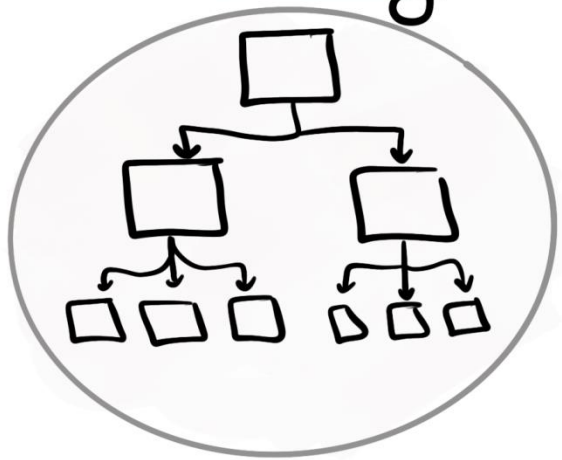
Recursive CTE

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

Recursive CTE



Hierarchy



Recursive CTE

DATA WITH BARAA

CTE Query
- CTE Definition -

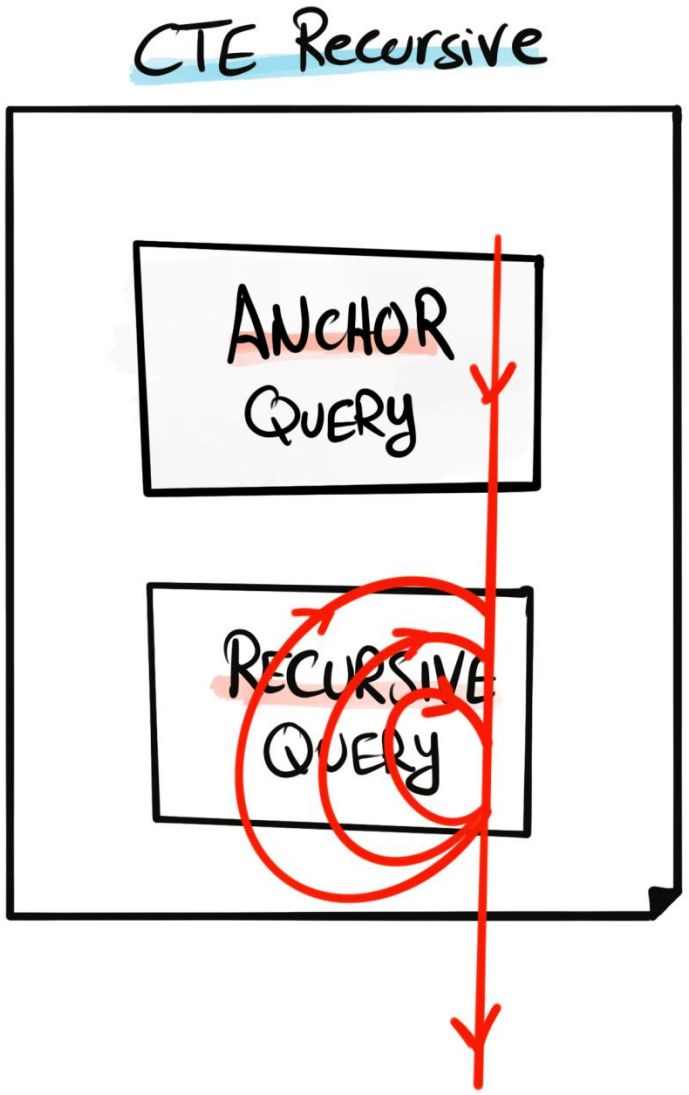
```
WITH CTE-Name AS
(
    SELECT ...
    FROM ...
    WHERE ...
    ----------
    UNION ALL
    ----------
    SELECT ...
    FROM CTE-Name
    WHERE [Break Condition]
)
```

Anchor Query

Recursive Query

Main Query
- CTE Usage -

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



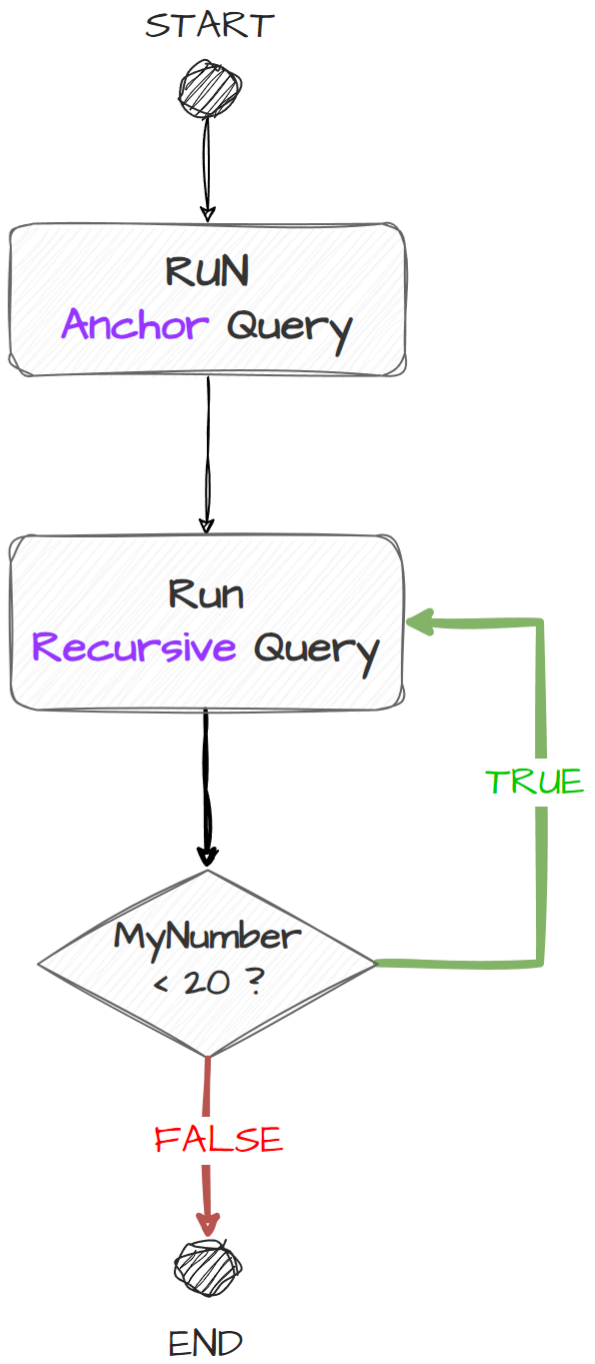
Data with Baraa

```sql
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



```sql
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

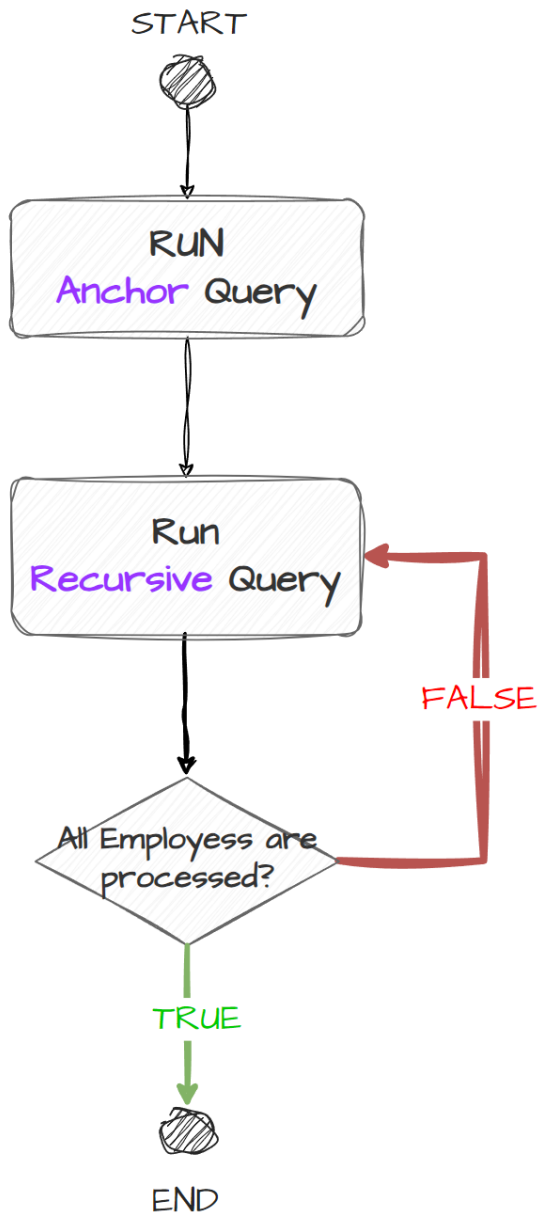
Recursive Query

Employees

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

CTE RESULT

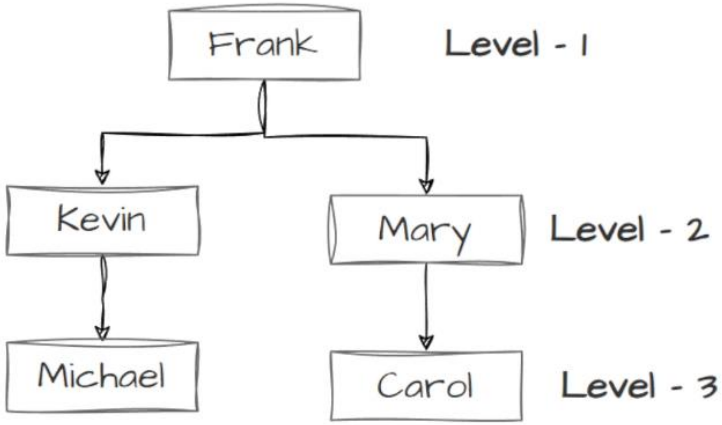
EmployeeID	FirstName	ManagerID	Level
1	Frank	NULL	1
2	Kevin	1	2
3	Mary	1	2
4	Michael	2	3
5	Carol	3	3



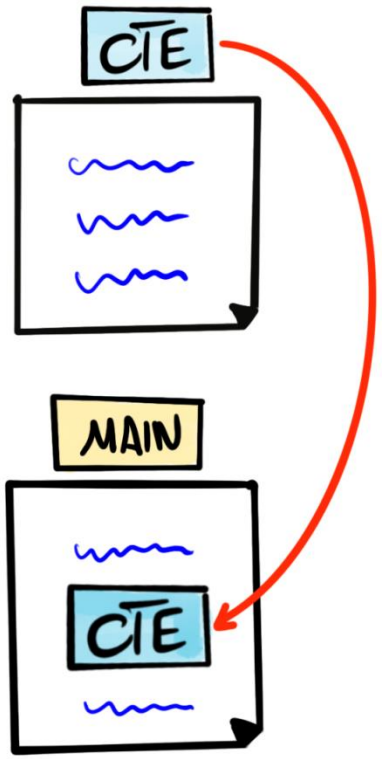
```sql
WITH CTE_Emp_Hierarchy AS
(
    SELECT
        EmployeeID,
        FirstName,
        ManagerID,
        1 AS Level
    FROM Sales.Employees
    WHERE ManagerID IS NULL

    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

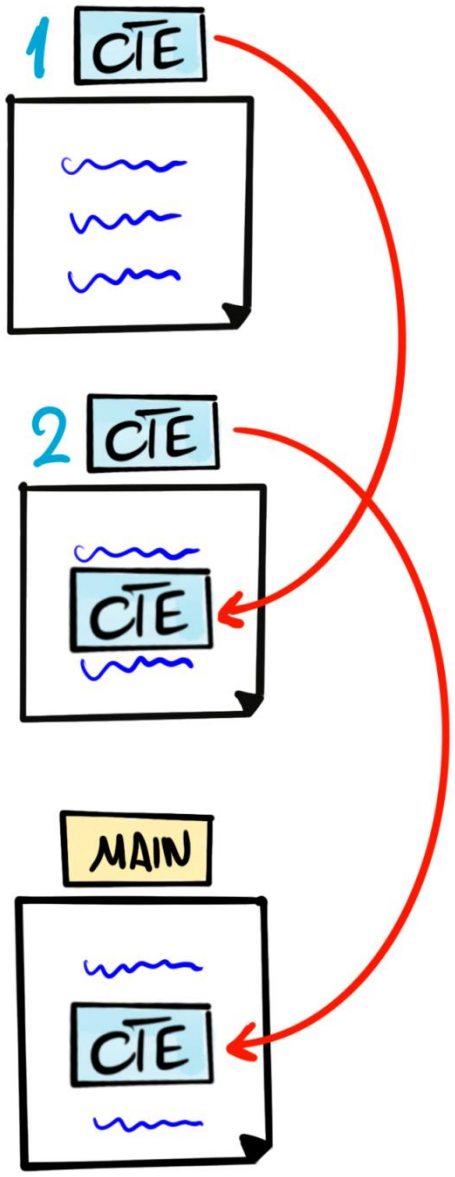
EmployeeID	FirstName	ManagerID	Level
1	Frank	NULL	1
2	Kevin	1	2
3	Mary	1	2
4	Michael	2	3
5	Carol	3	3



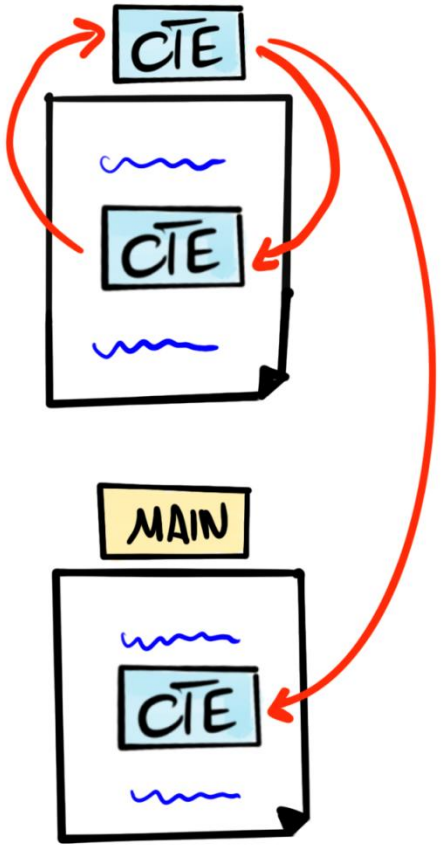
STANDALONE



NESTED



RECURSIVE

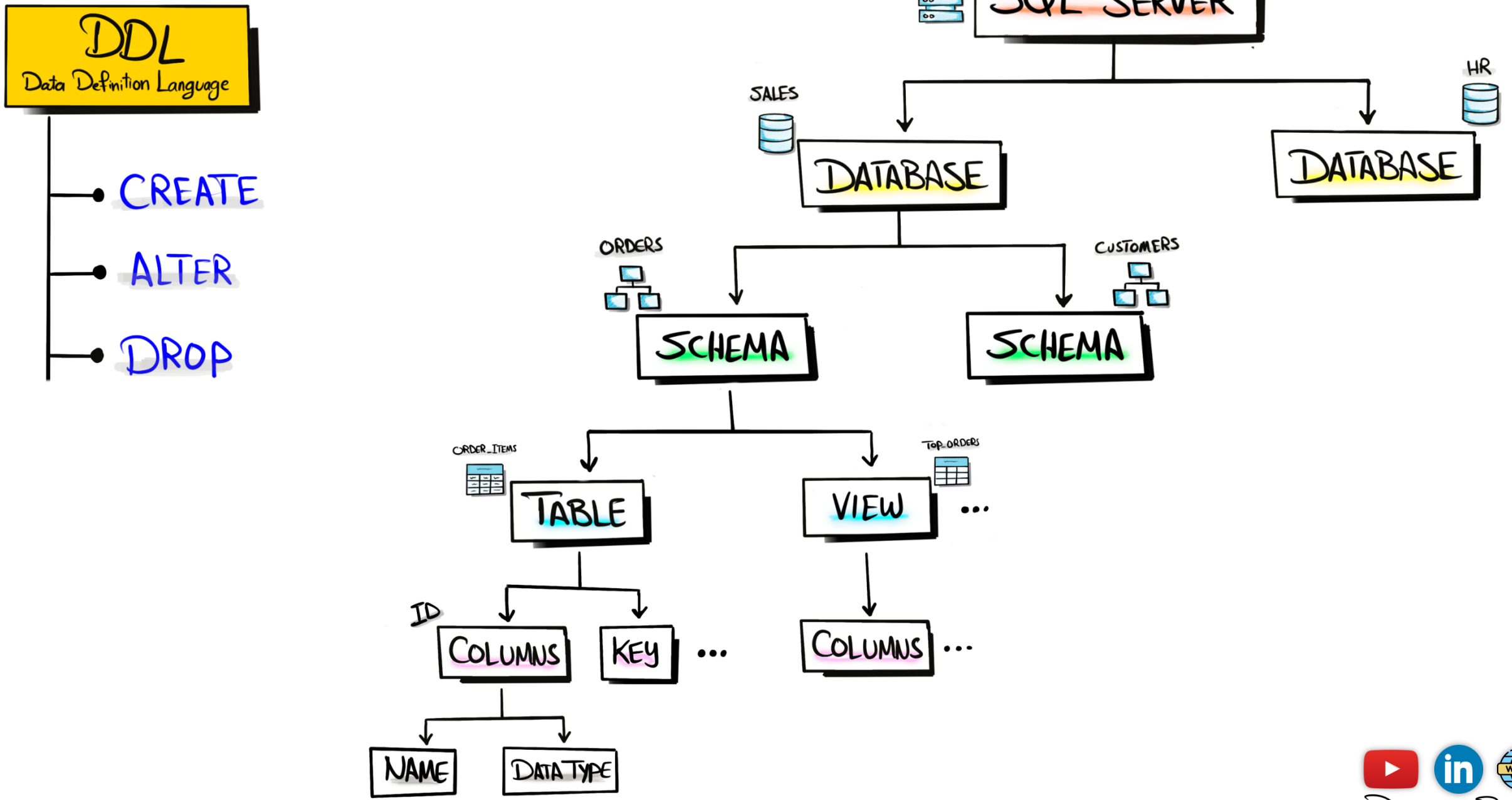


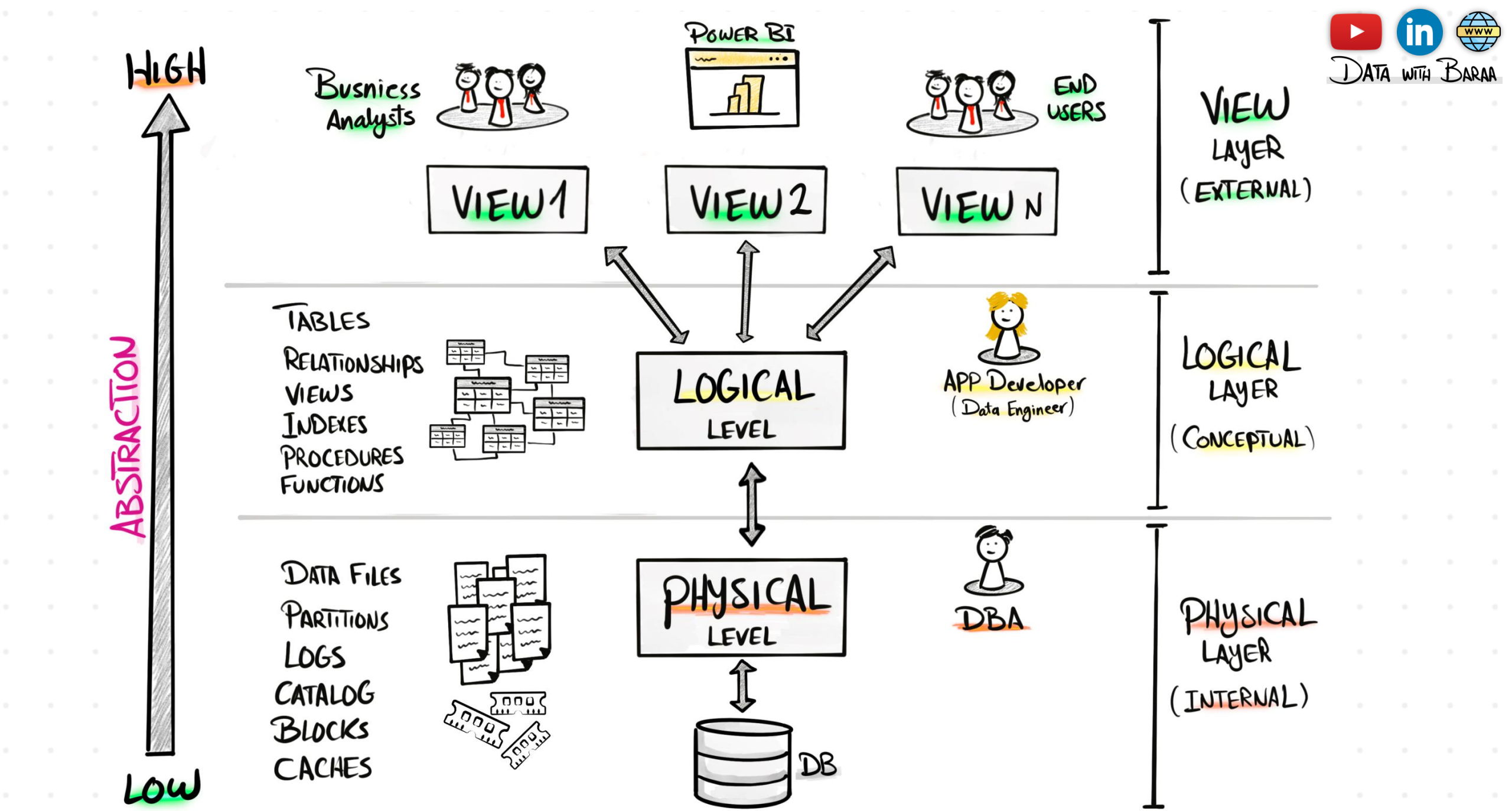
DATA WITH BARAA

Views

Database Object

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



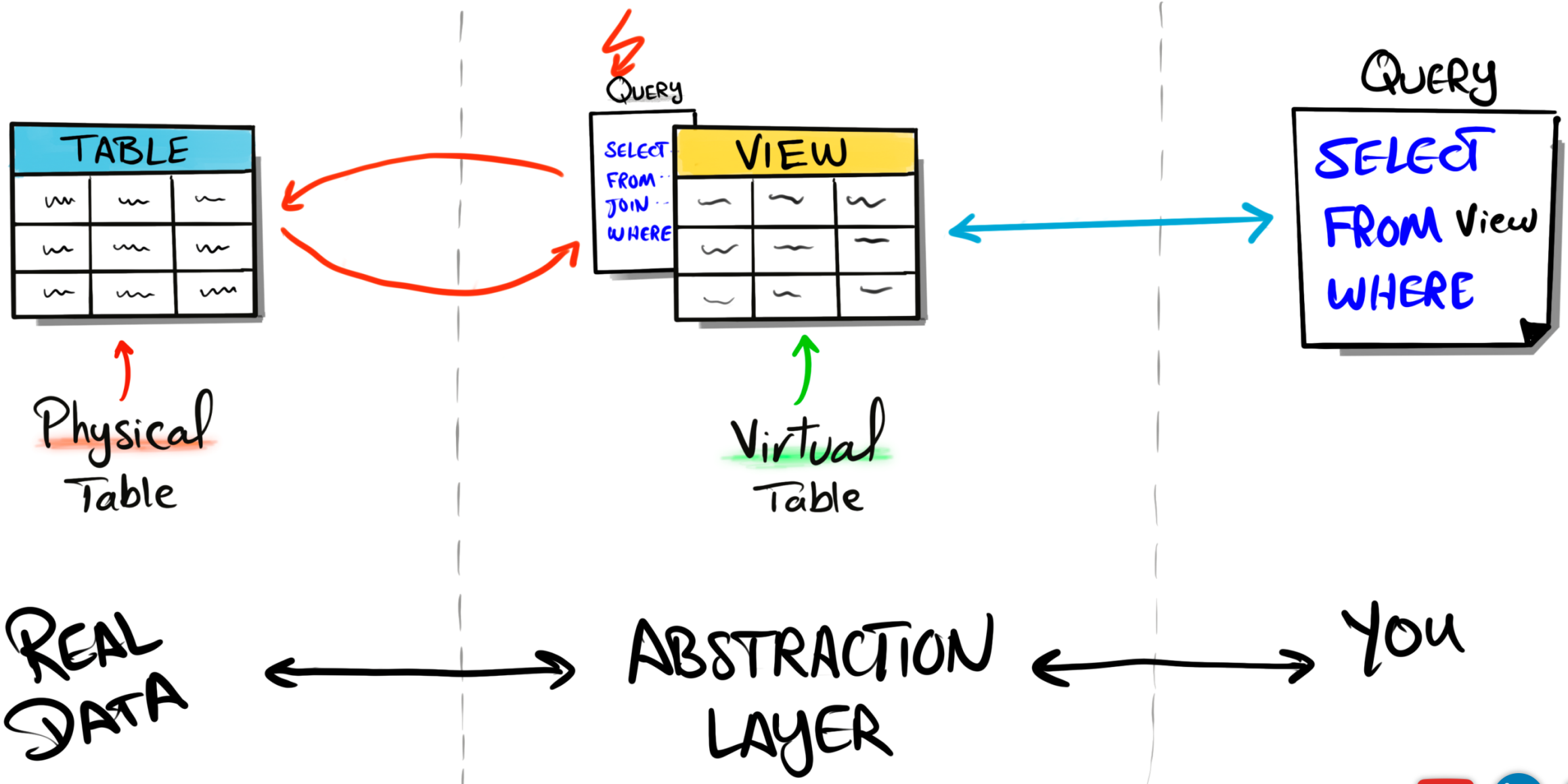

HIGH
ABSTRACTION
LOW
Busniess Analysts
POWER BI
END USERS
VIEW1
VIEW2
VIEW N
VIEW LAYER (EXTERNAL)
TABLES
RELATIONSHIPS
VIEWS
INDEXES
PROCEDURES
FUNCTIONS
LOGICAL LEVEL
APP Developer (Data Engineer)
LOGICAL LAYER (CONCEPTUAL)
DATA FILES
PARTITIONS
LOGS
CATALOG
BLOCKS
CACHES
PHYSICAL LEVEL
DBA
DB
PHYSICAL LAYER (INTERNAL)
DATA WITH BARAA

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.



DATA WITH BARAA

VIEW

No Persistance

Easy to Maintain

Slow Response

Read

Flexible

TABLE

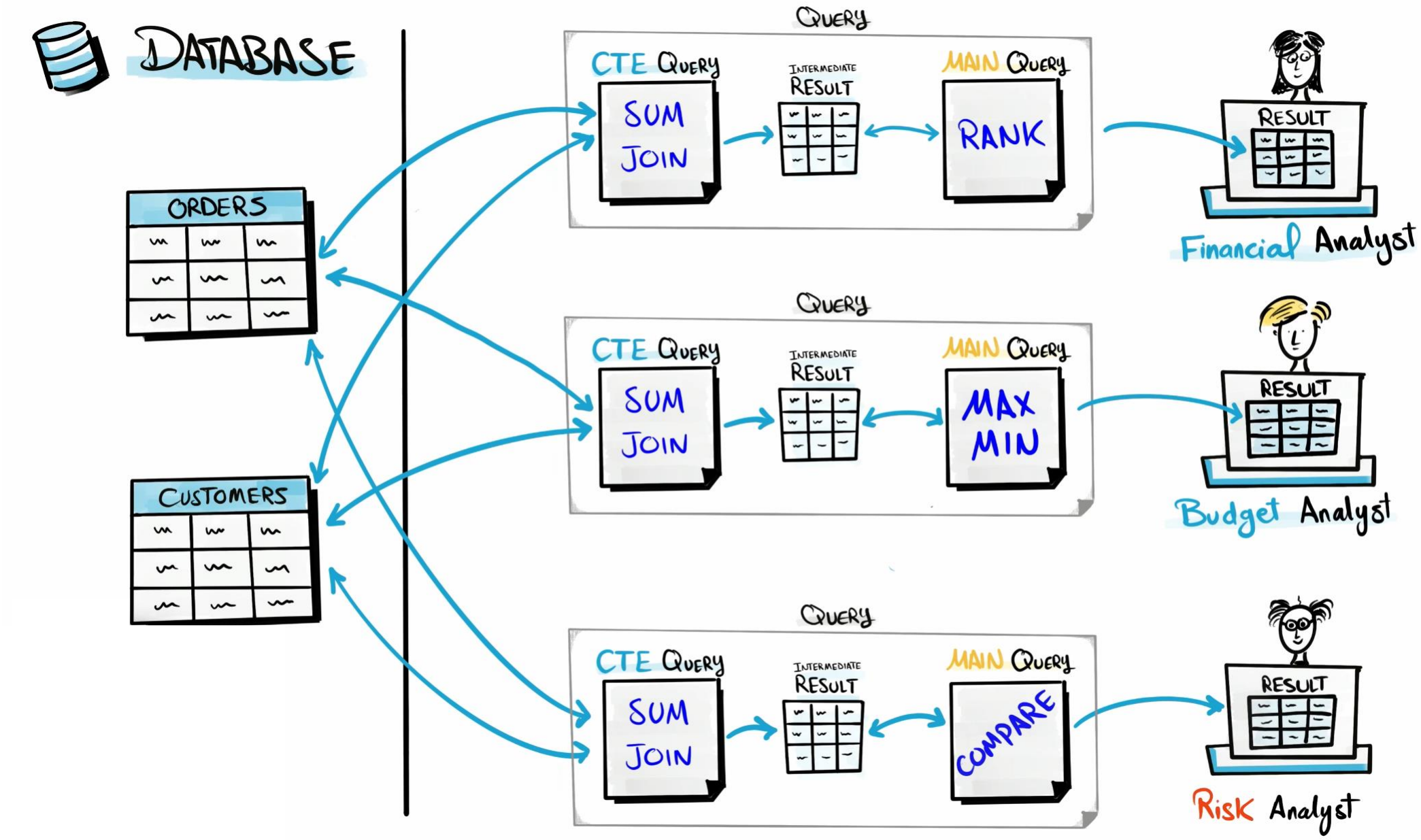
Persisted Data

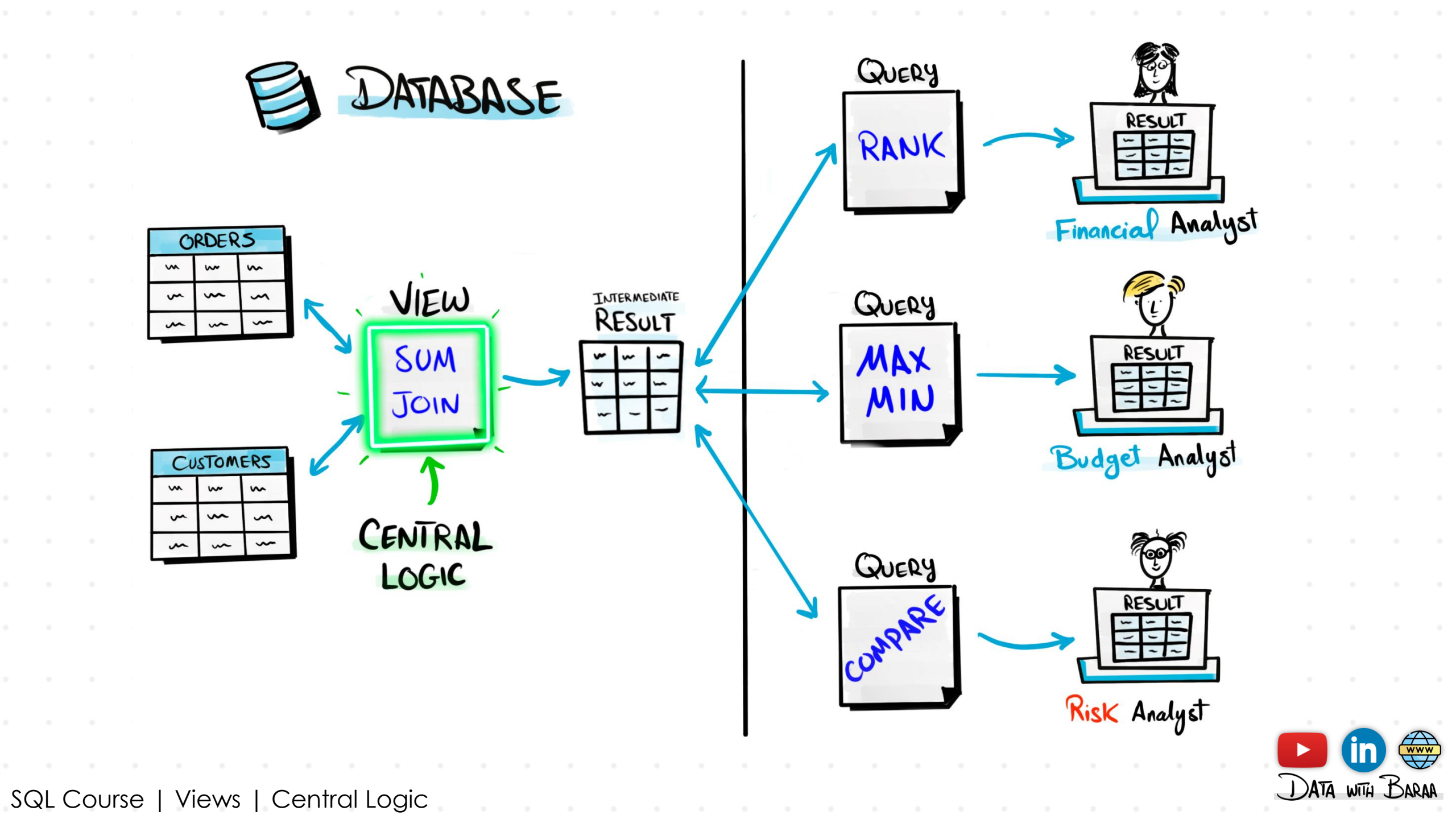
Hard to Maintain

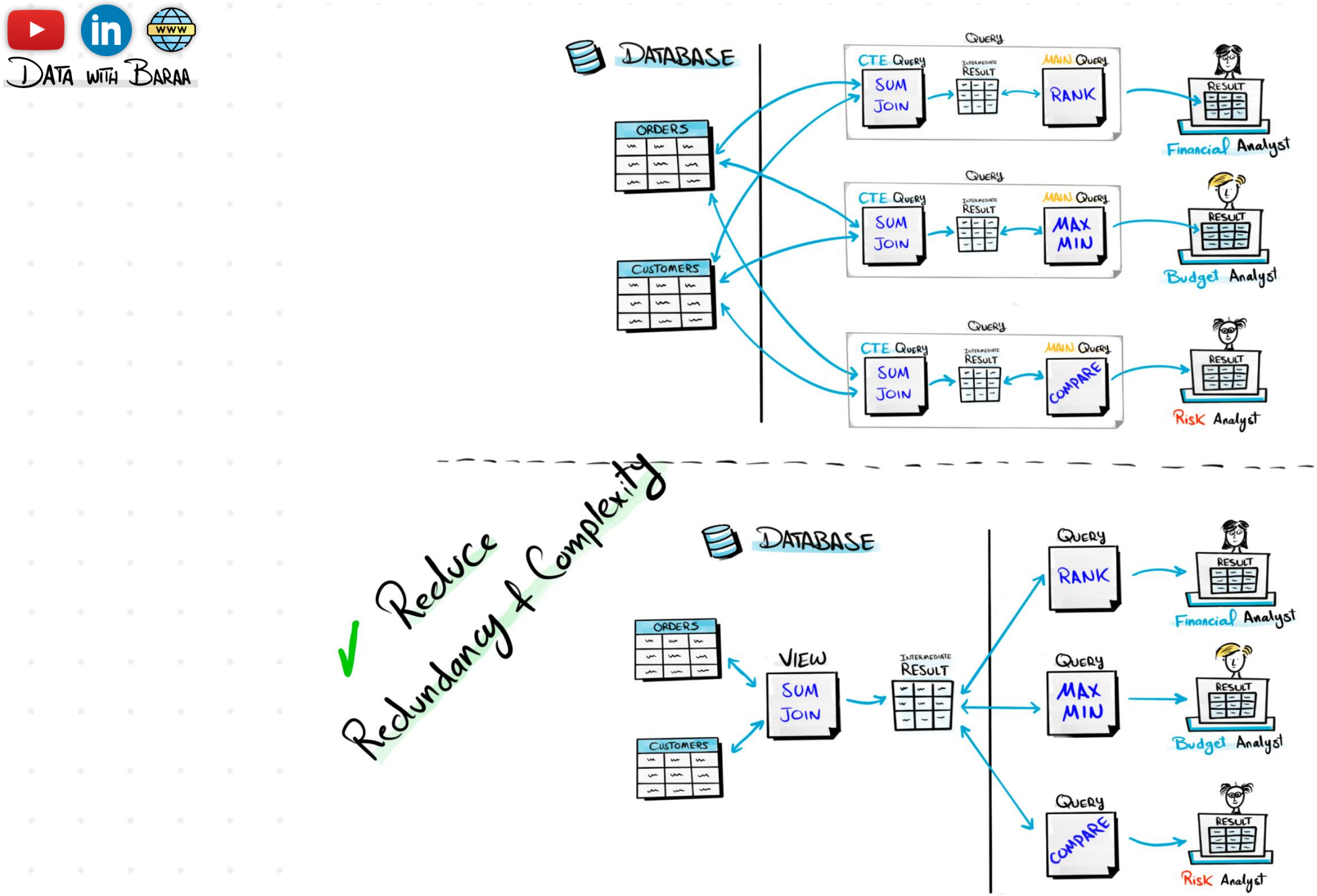
Fast Response

Read / Write

Hard to Change






DATA WITH BARAA
DATABASE
ORDERS
CUSTOMERS
QUERY
CTE QUERY
SUM
JOIN
INTERMEDIATE
RESULT
MAIN QUERY
RANK
MAX
MIN
COMPARE
RESULT
Financial Analyst
Budget Analyst
Risk Analyst
✓ Reduce Redundancy & Complexity
DATABASE
ORDERS
CUSTOMERS
VIEW
SUM
JOIN
INTERMEDIATE
RESULT
QUERY
RANK
MAX
MIN
COMPARE
RESULT
Financial Analyst
Budget Analyst
Risk Analyst

DATA WITH BARAA

VIEWS	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 - on the Fly -
Need to Maintain - CREATE/DROP -	No Maintenance - Auto cleanup -

VIEWS

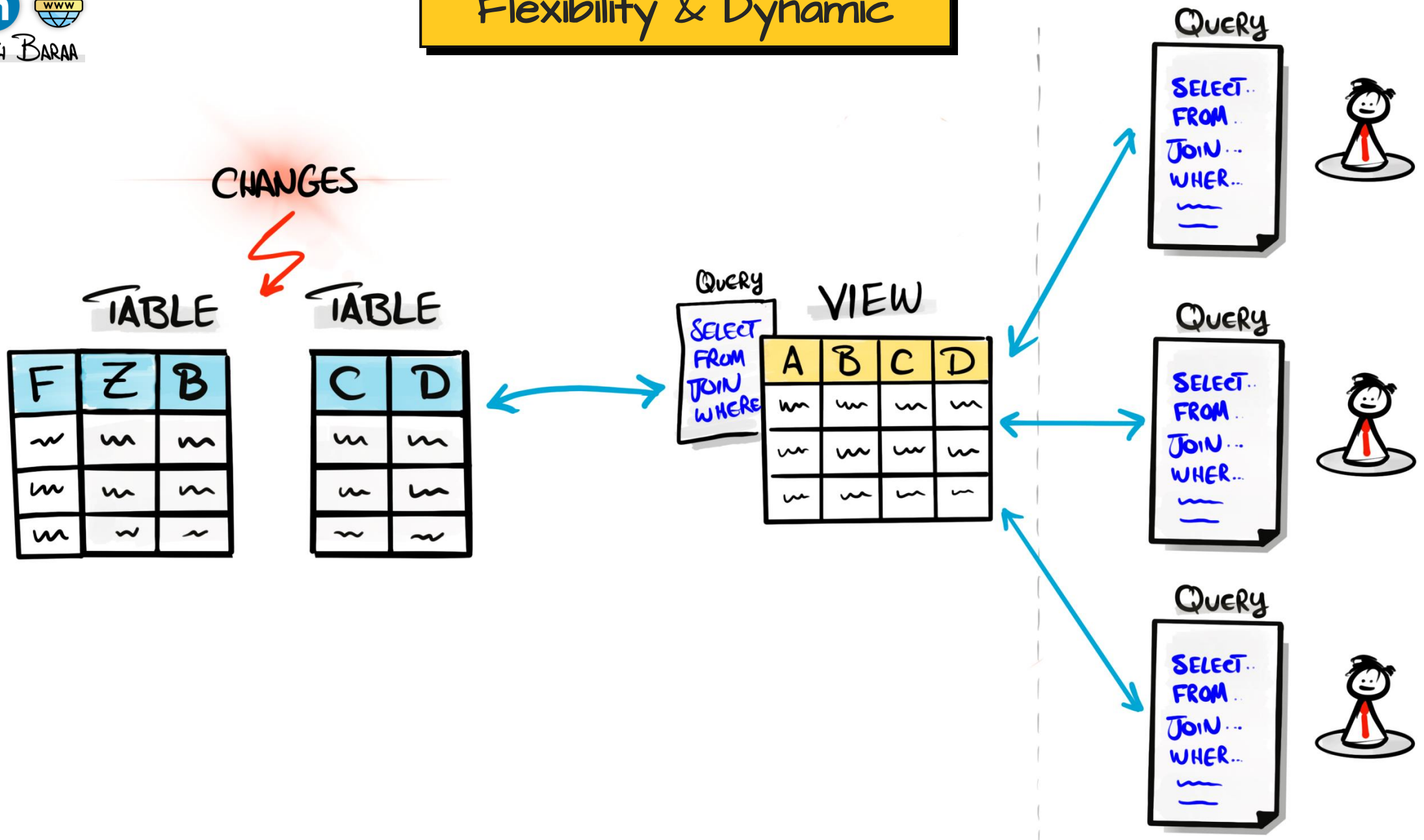
DDL Statement

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

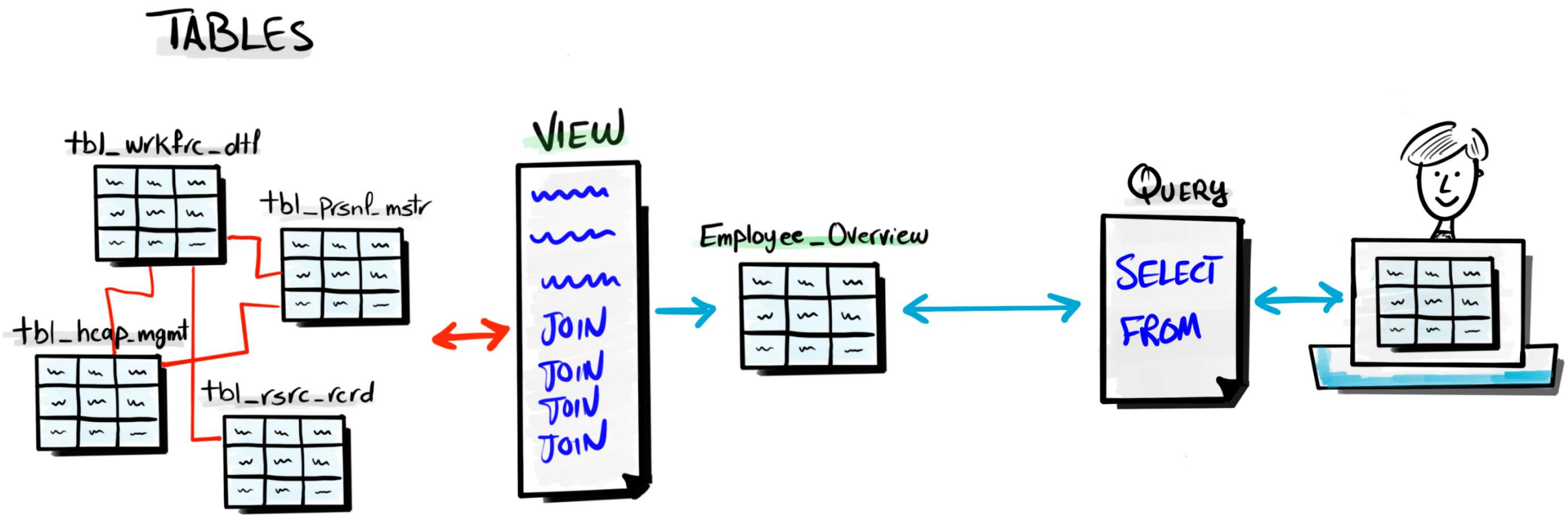
Query

DATA WITH BARAA

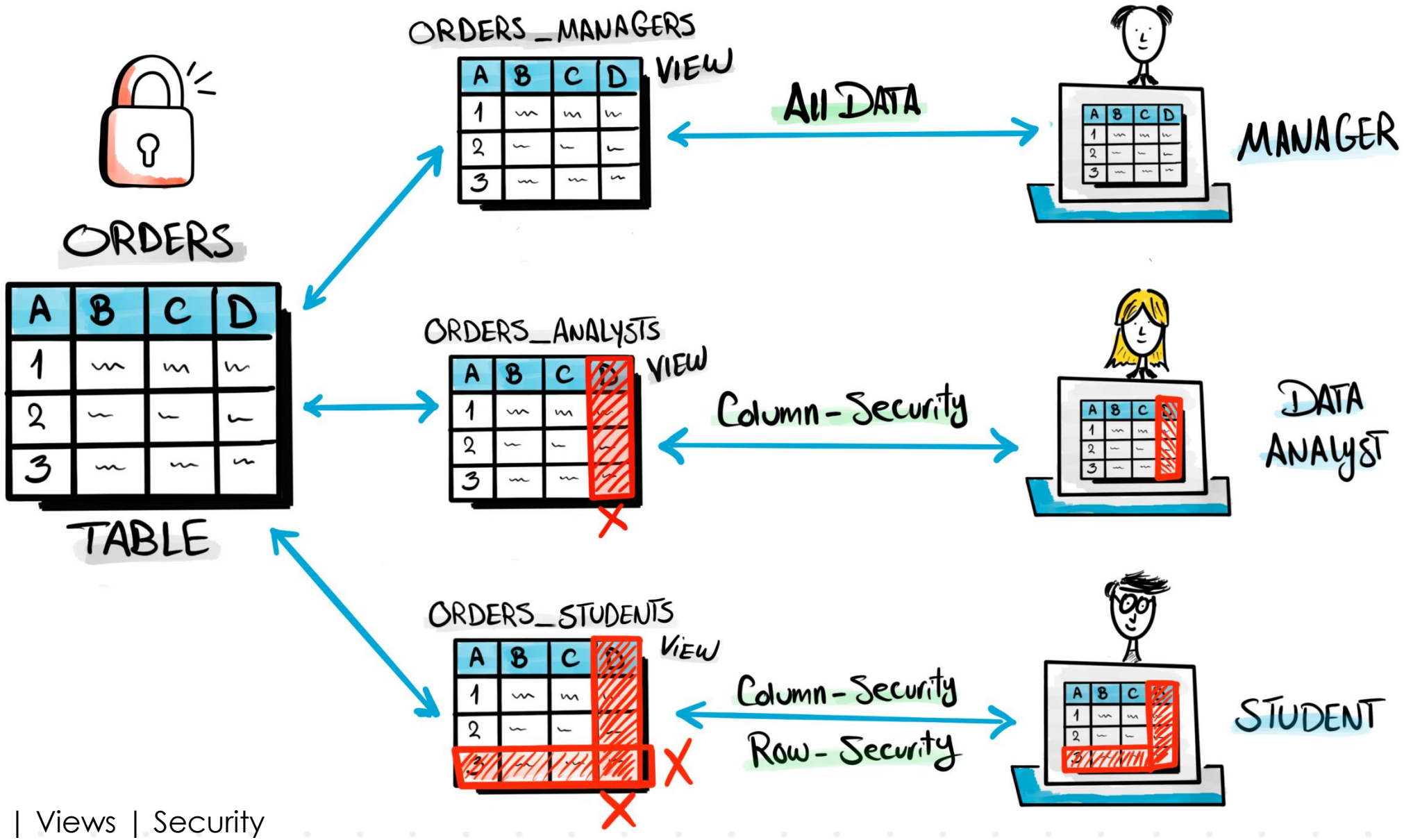
Flexibility & Dynamic



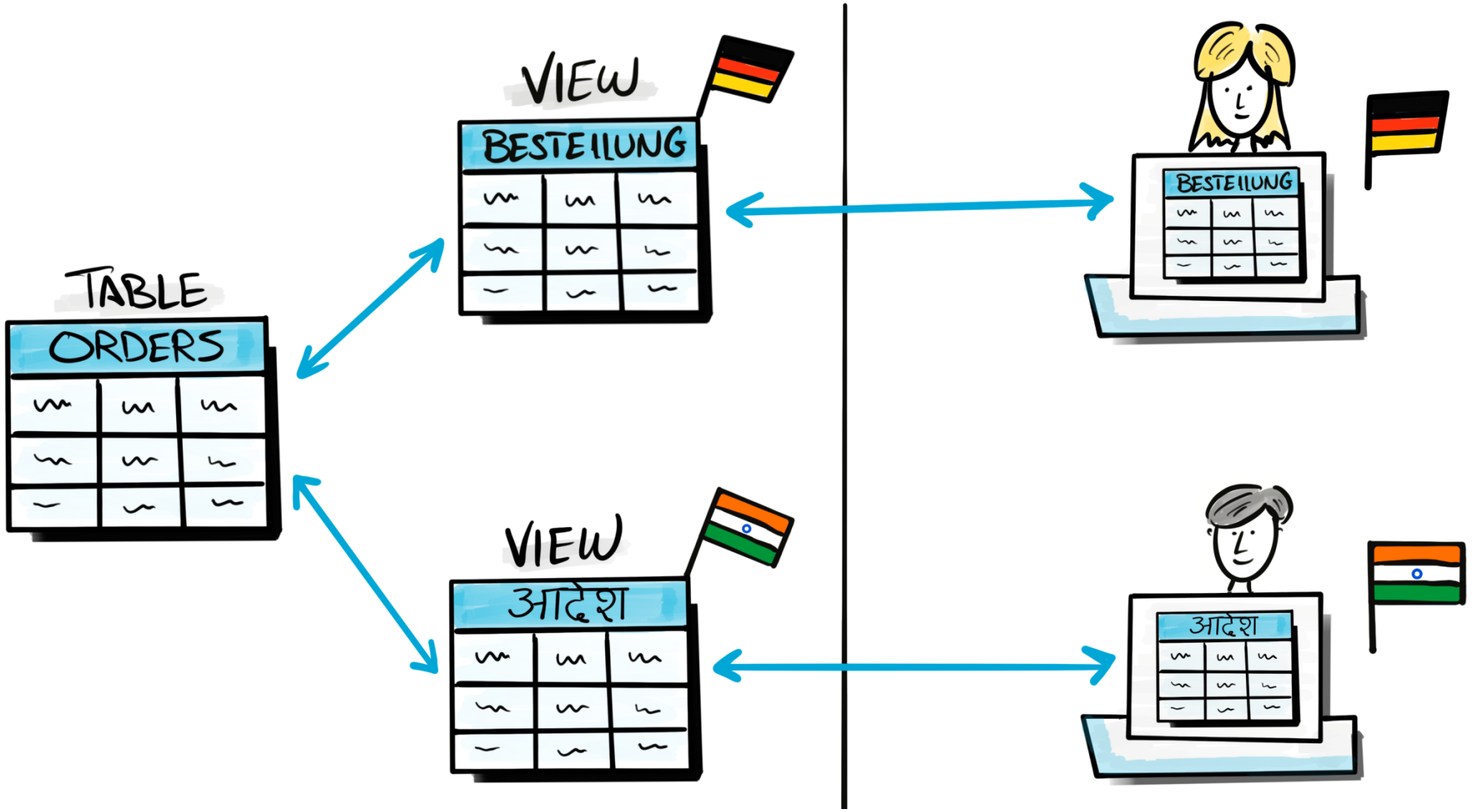
Hide Complexity



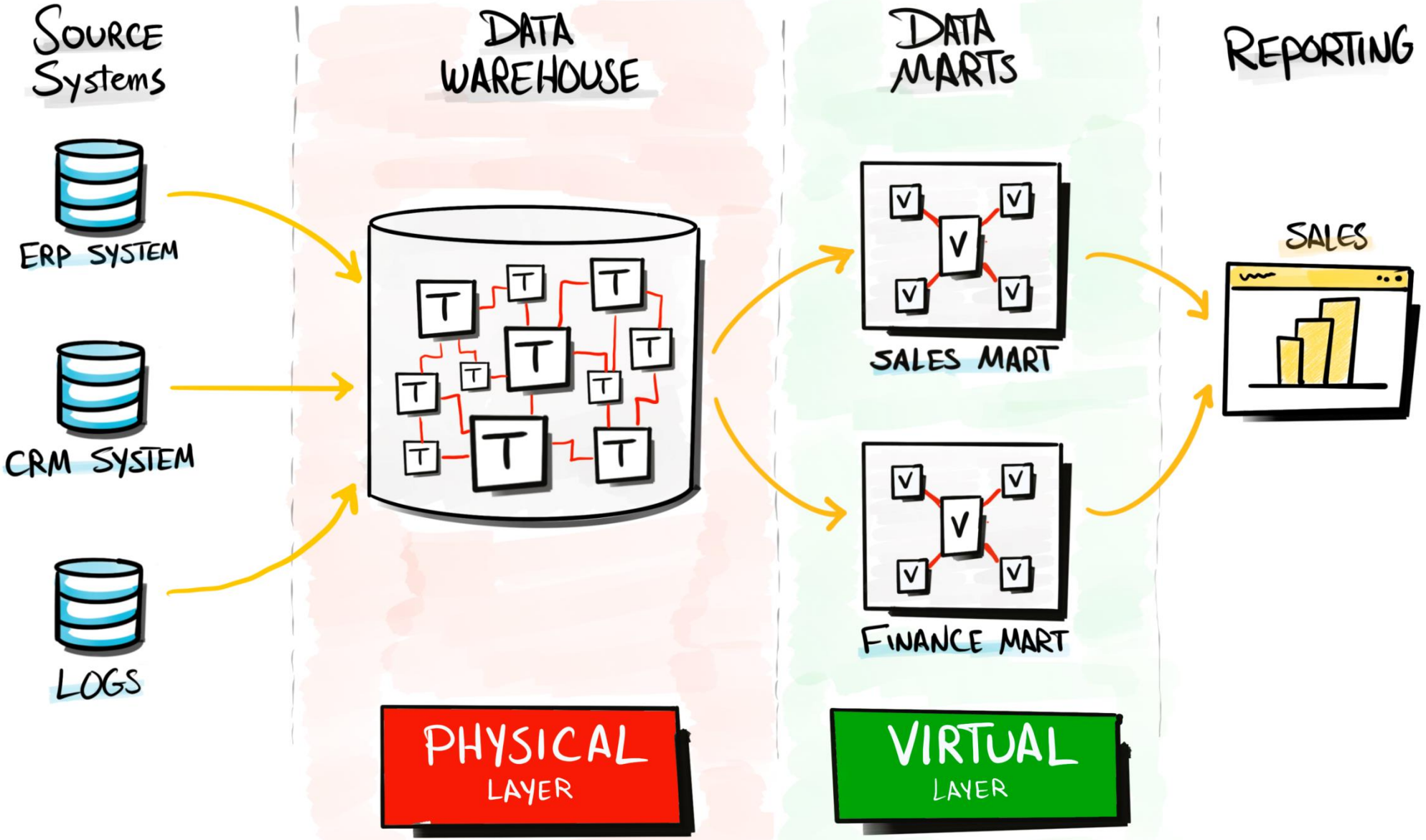
Security


ORDERS_MANAGERS
VIEW
All DATA
MANAGER
ORDERS
TABLE
ORDERS_ANALYSTS
VIEW
Column-Security
DATA ANALYST
ORDERS_STUDENTS
VIEW
Column-Security
Row-Security
STUDENT

Multiple Languages



Virtual Data Marts



VIEWS

- Virtual Tabel 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 maintaine.

USE CASES

- Store Central Complex Business Logic to be reused.
- Hide Complexitity 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

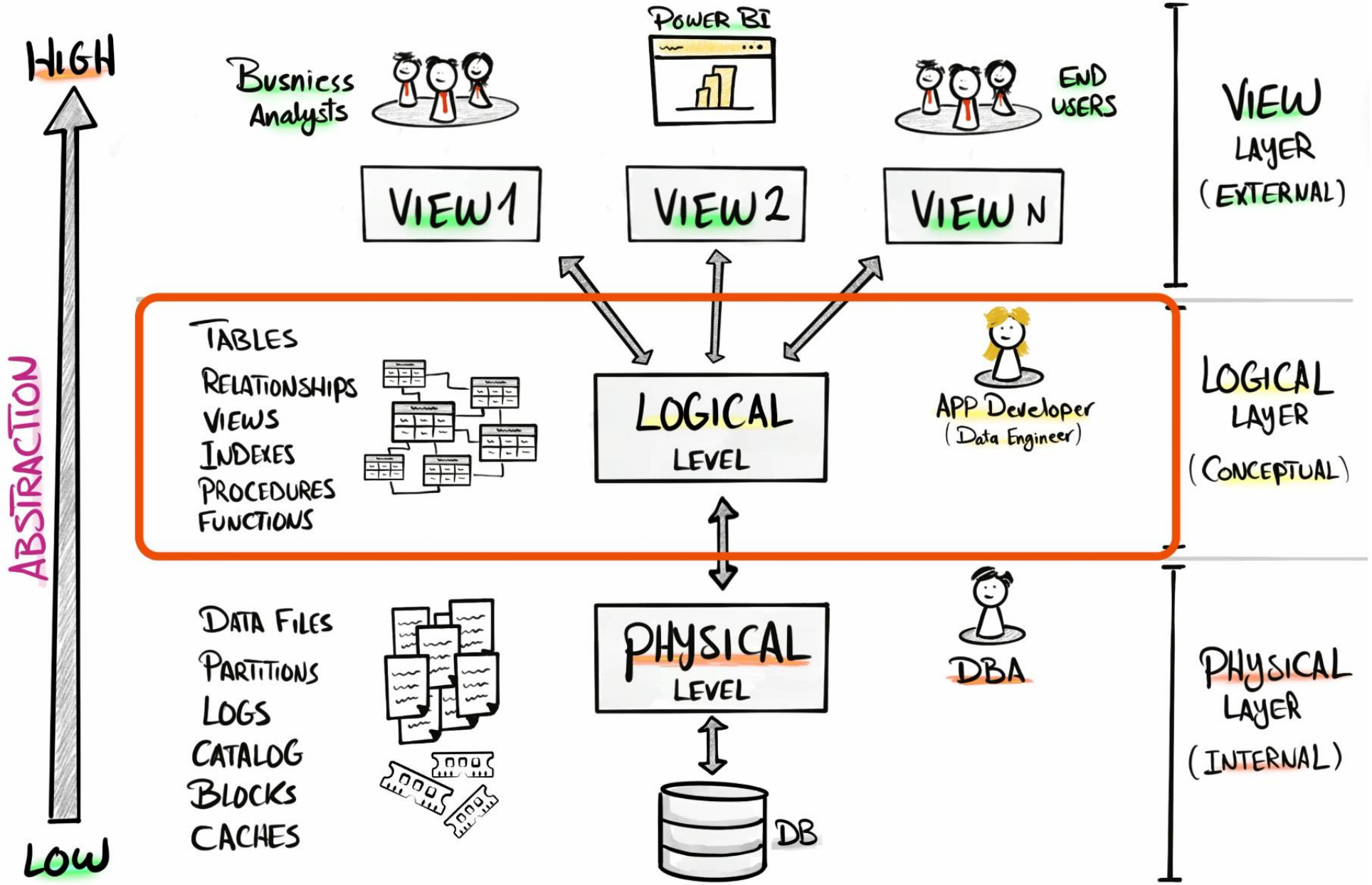
Create Table As SELECT

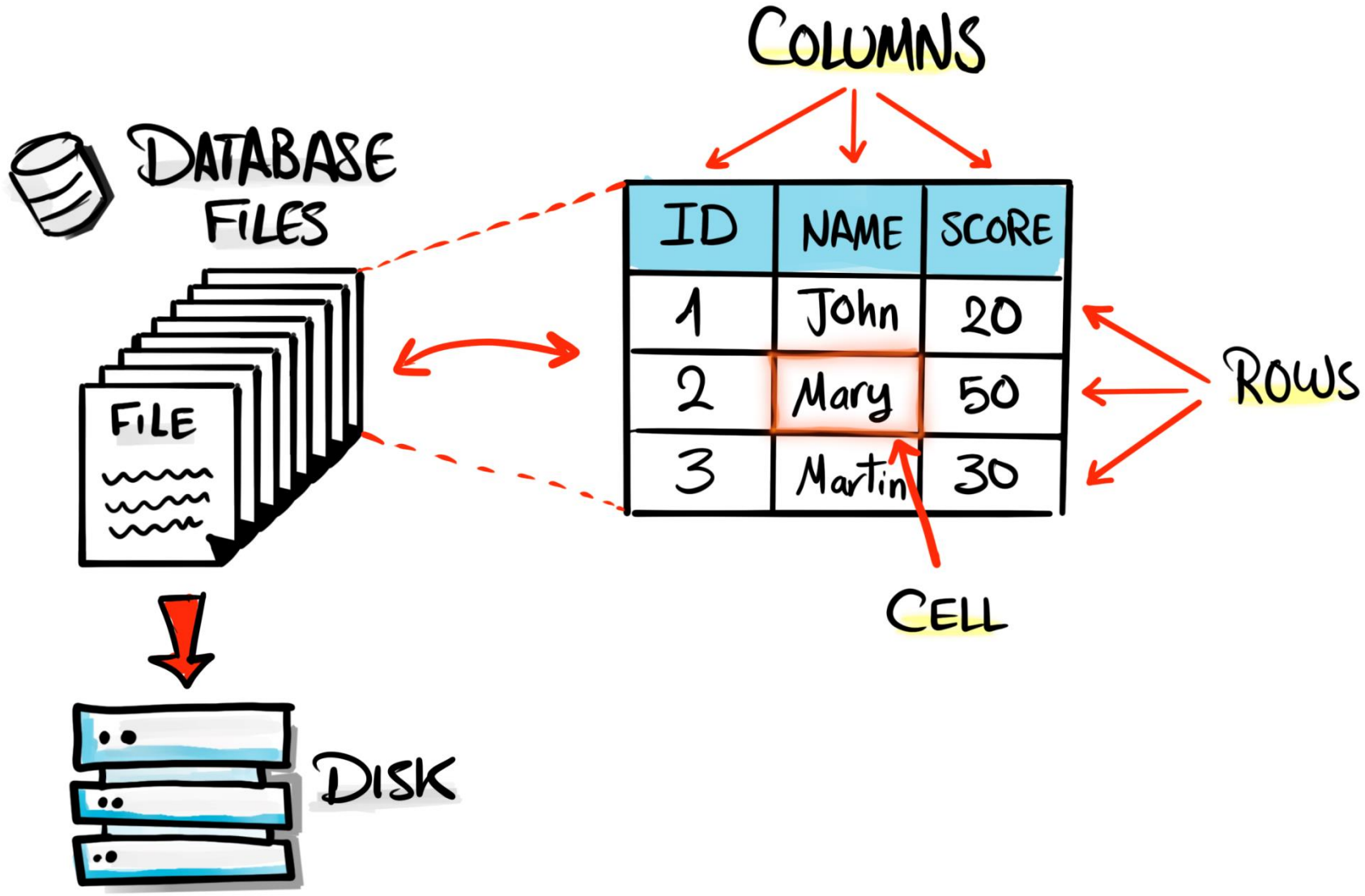
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | CTAS

DB TABLE

A table is a structured collection of data,

similar to a spreadsheet or grid (Excel)


HIGH
ABSTRACTION
LOW
Busniess Analysts
POWER BI
END USERS
VIEW1
VIEW2
VIEW N
VIEW LAYER (EXTERNAL)
TABLES
RELATIONSHIPS
VIEWS
INDEXES
PROCEDURES
FUNCTIONS
LOGICAL LEVEL
APP Developer (Data Engineer)
LOGICAL LAYER (CONCEPTUAL)
DATA FILES
PARTITIONS
LOGS
CATALOG
BLOCKS
CACHES
PHYSICAL LEVEL
DBA
DB
PHYSICAL LAYER (INTERNAL)


COLUMNS
DATABASE FILES
FILE
ID
NAME
SCORE
1
John
20
2
Mary
50
3
Martin
30
ROWS
CELL
DISK

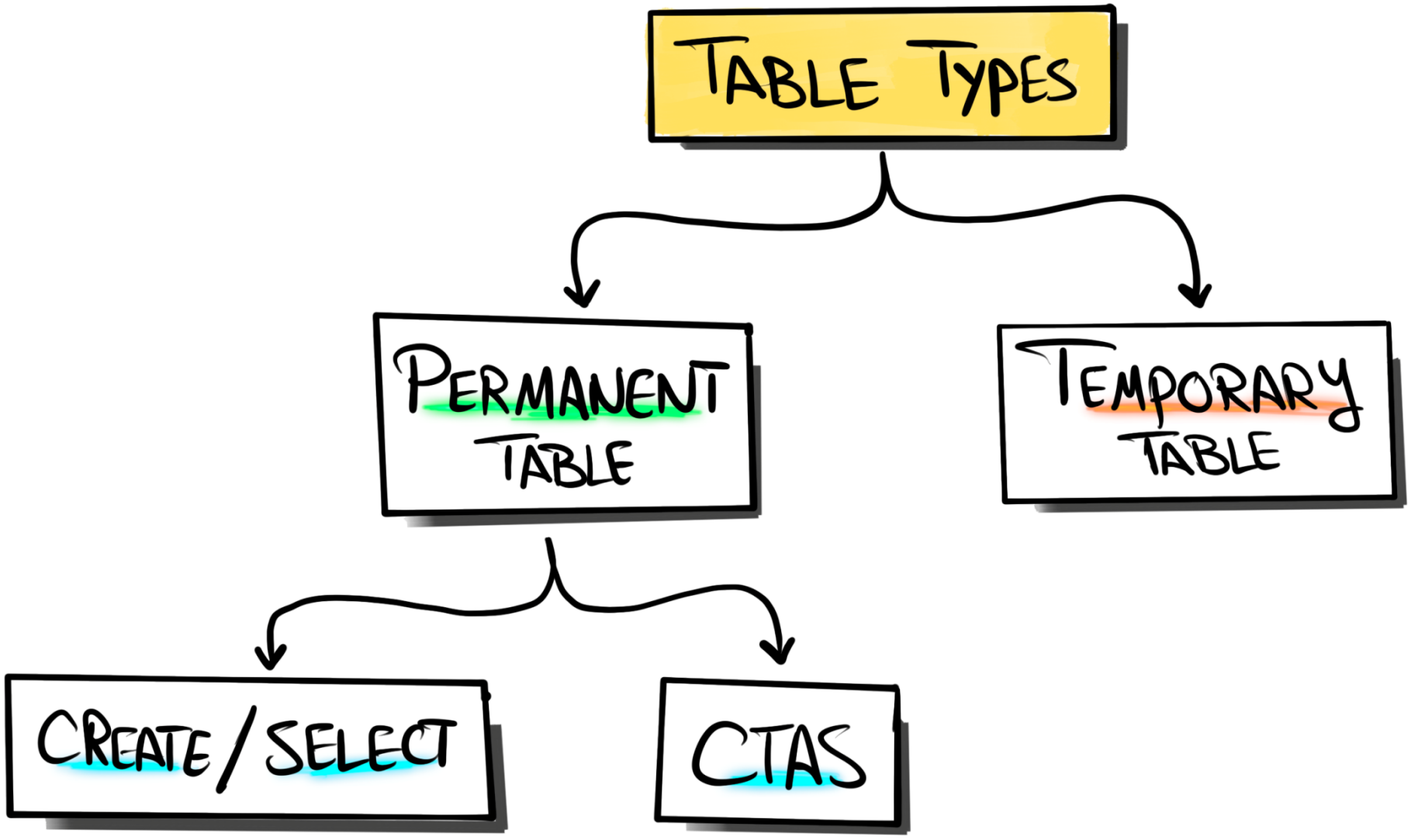
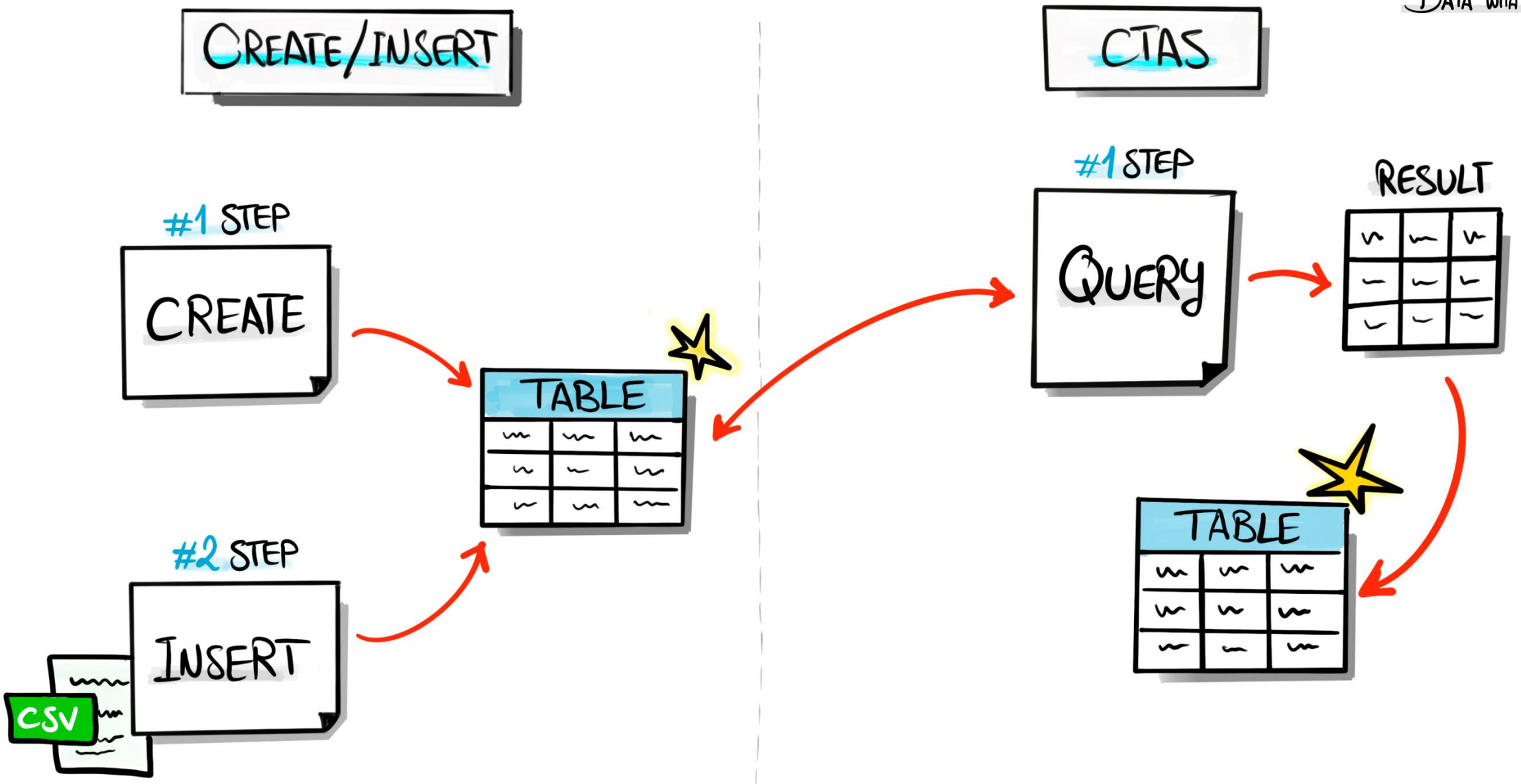
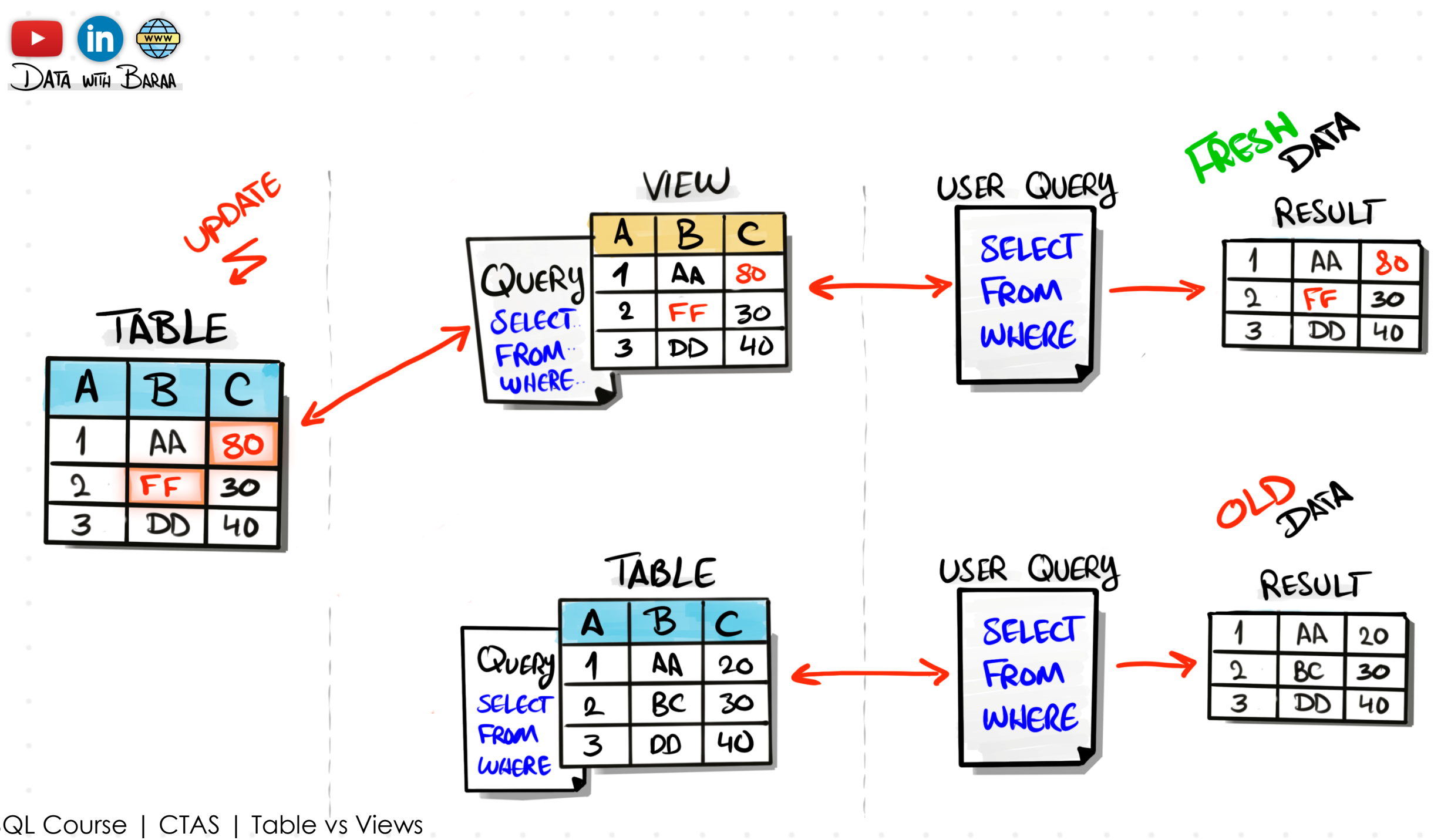


TABLE TYPES
PERMANENT TABLE
TEMPORARY TABLE
CREATE / SELECT
CTAS


CREATE/INSERT
#1 STEP
CREATE
TABLE
#2 STEP
INSERT
CSV
CTAS
#1 STEP
QUERY
RESULT
TABLE


DATA WITH BARAA
UPDATE
TABLE
A B C
1 AA 80
2 FF 30
3 DD 40
VIEW
QUERY
SELECT
FROM
WHERE
A B C
1 AA 80
2 FF 30
3 DD 40
USER QUERY
SELECT
FROM
WHERE
FRESH DATA
RESULT
1 AA 80
2 FF 30
3 DD 40
TABLE
QUERY
SELECT
FROM
WHERE
A B C
1 AA 20
2 BC 30
3 DD 40
USER QUERY
SELECT
FROM
WHERE
OLD DATA
RESULT
1 AA 20
2 BC 30
3 DD 40

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

Query

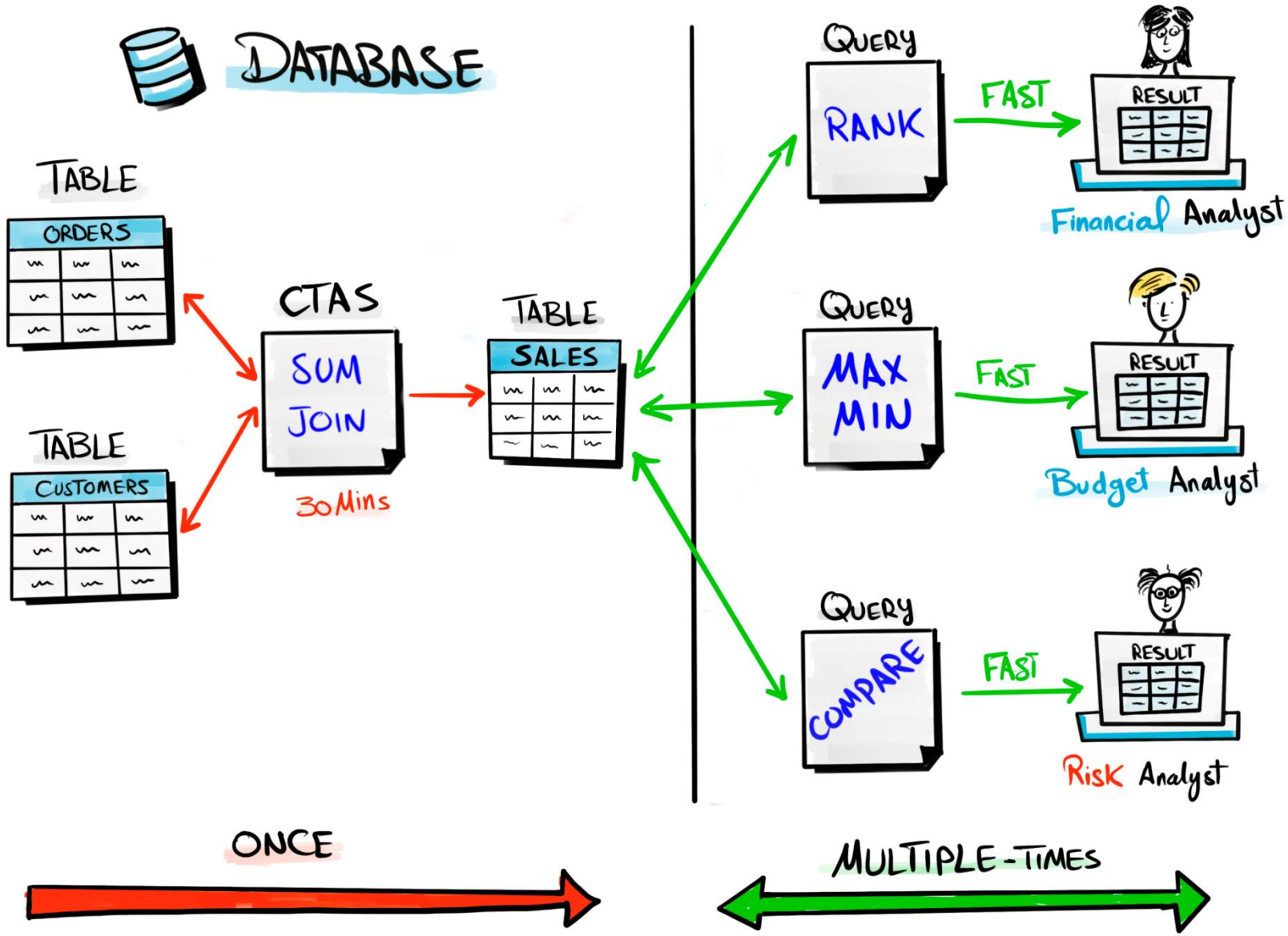
```
CREATE TABLE NAME AS
(
    SELECT ...
    FROM ...
    WHERE ...
)
```

MySQL | Postgres | Oracle

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

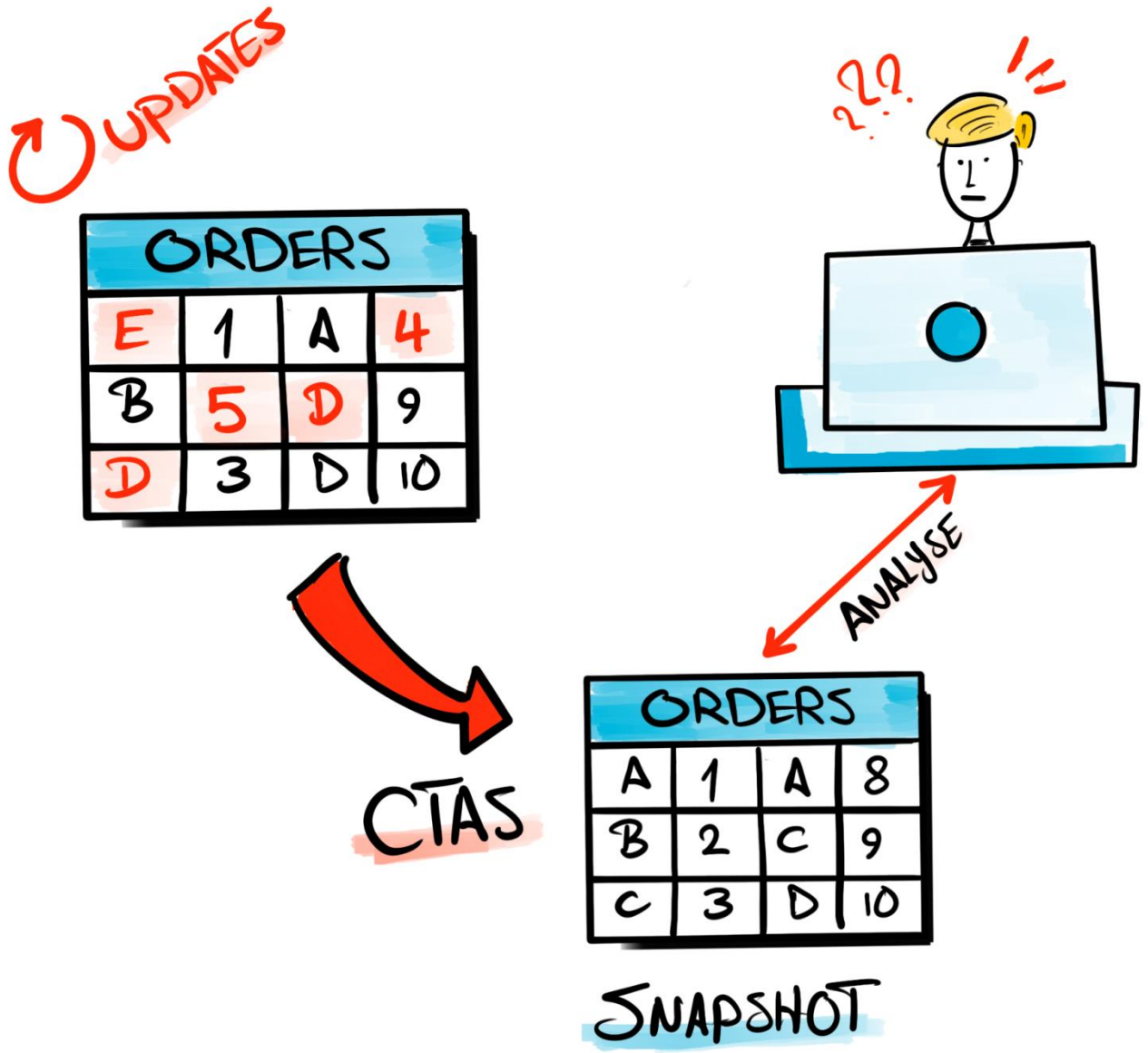
Sql Server

Optimize Performance

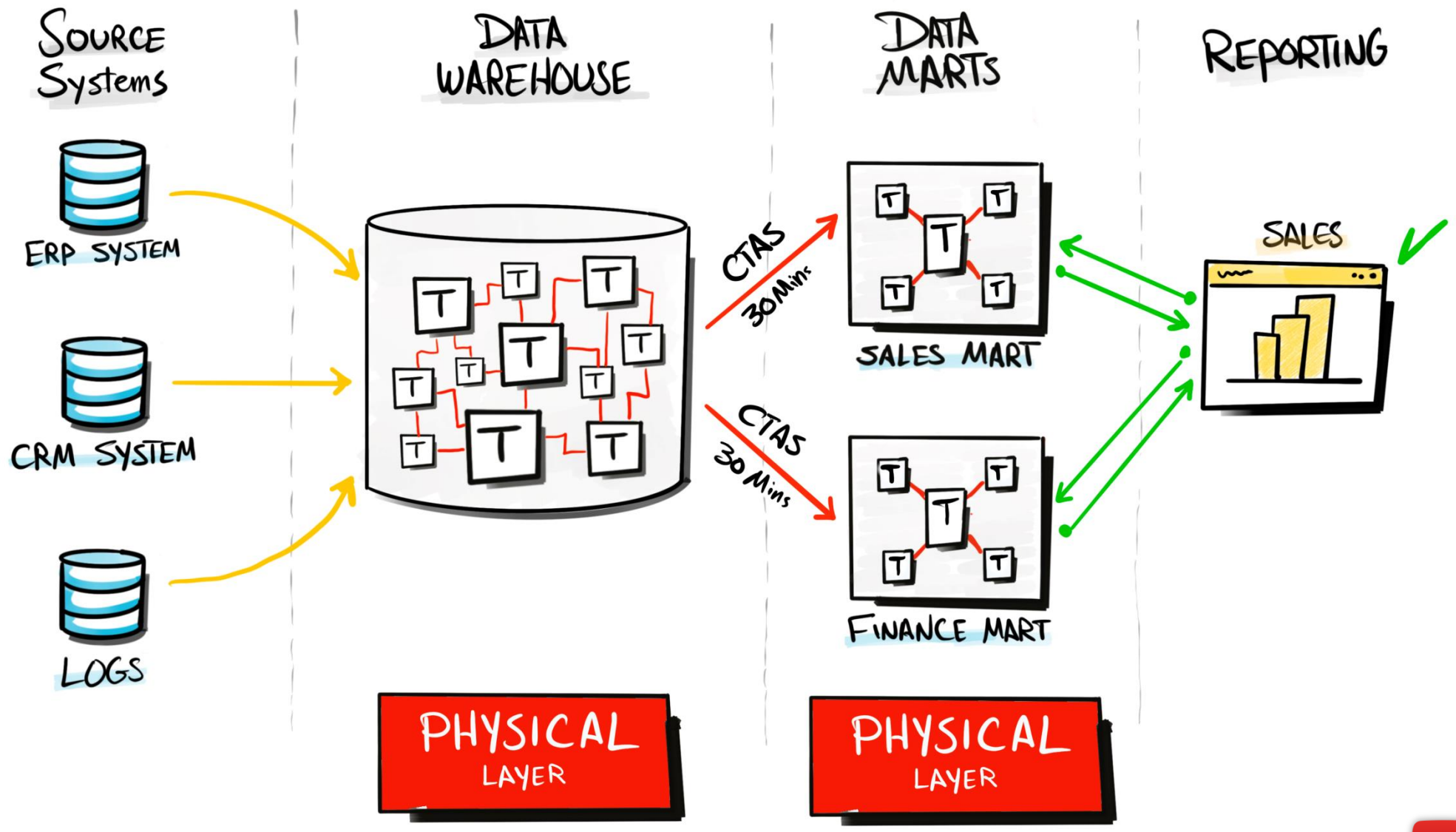


Create Snapshot

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

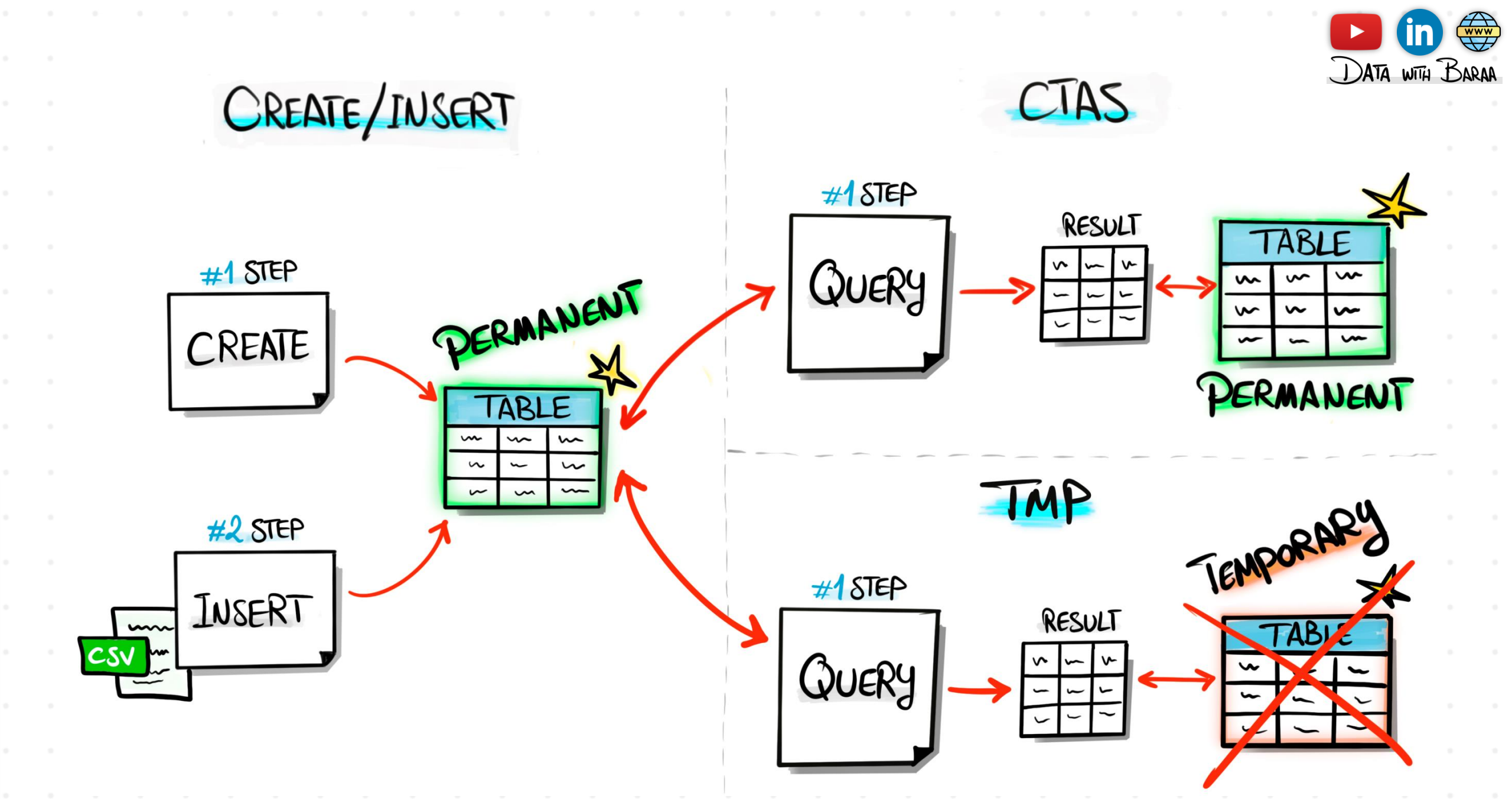
PERMENANT
CREATE TABLE

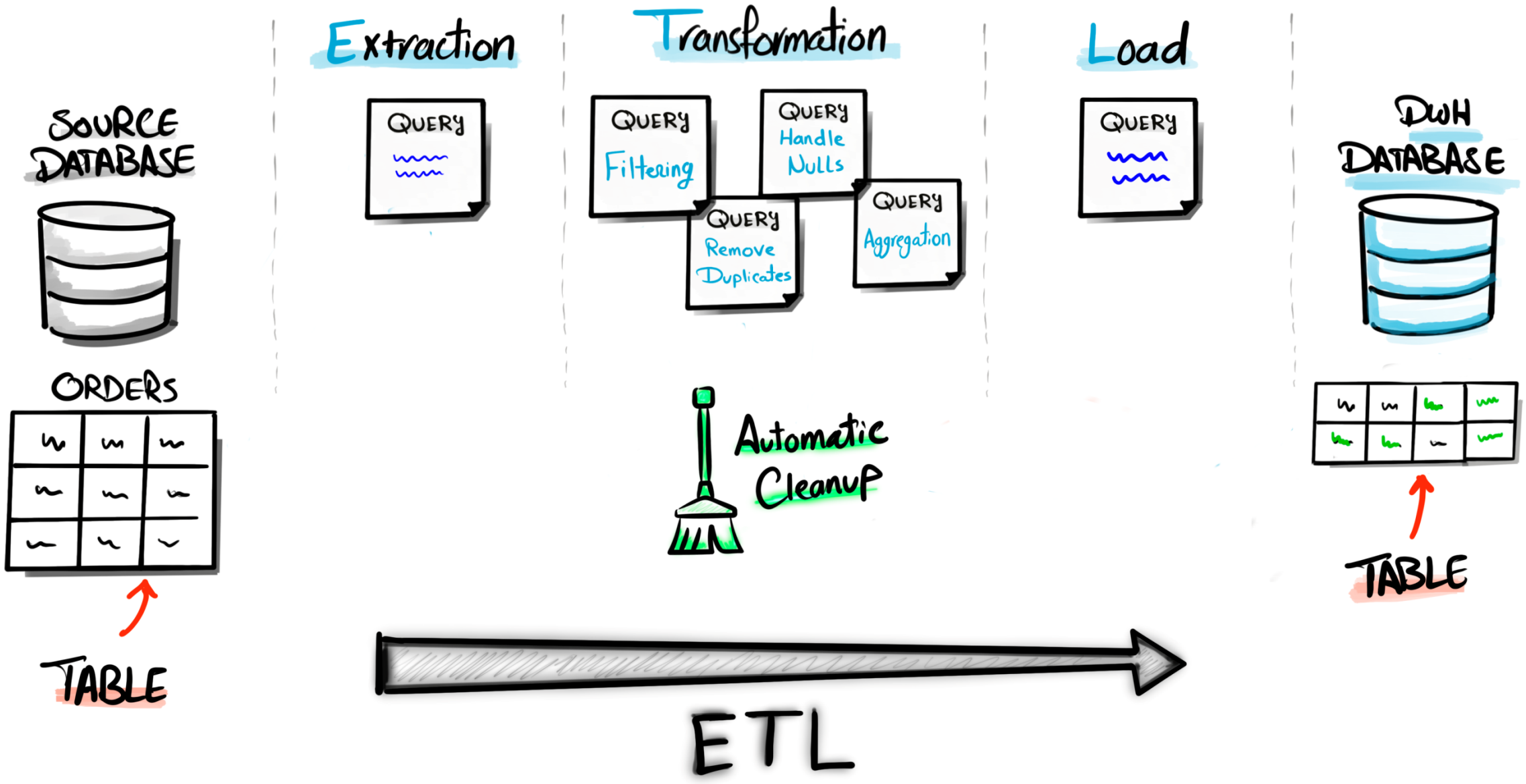
```
CREATE TABLE TABLE-NAME AS
(
    SELECT ...
    FROM ...
    WHERE ...
)
```

TEMPORARY
TABLE

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

MySQL | Postgres | Oracle


DATA WITH BARAA
CREATE/INSERT
#1 STEP
CREATE
PERMANENT
TABLE
#2 STEP
INSERT
CSV
CTAS
#1 STEP
QUERY
RESULT
TABLE
PERMANENT
TMP
#1 STEP
QUERY
RESULT
TEMPORARY
TABLE


Extraction
Transformation
Load
SOURCE DATABASE
QUERY
QUERY Filtering
QUERY Handle Nulls
QUERY Remove Duplicates
QUERY Aggregation
QUERY
DWH DATABASE
ORDERS
Automatic Cleanup
TABLE
TABLE
ETL

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)

Types

- **PERMANENT**: Data Live for ever in Database.
  - CREATE - INSERT
  - CTAS : Create Table from a Query.
- **TEMPORARY**: Data Live only during the session.

CTAS USE CASES

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

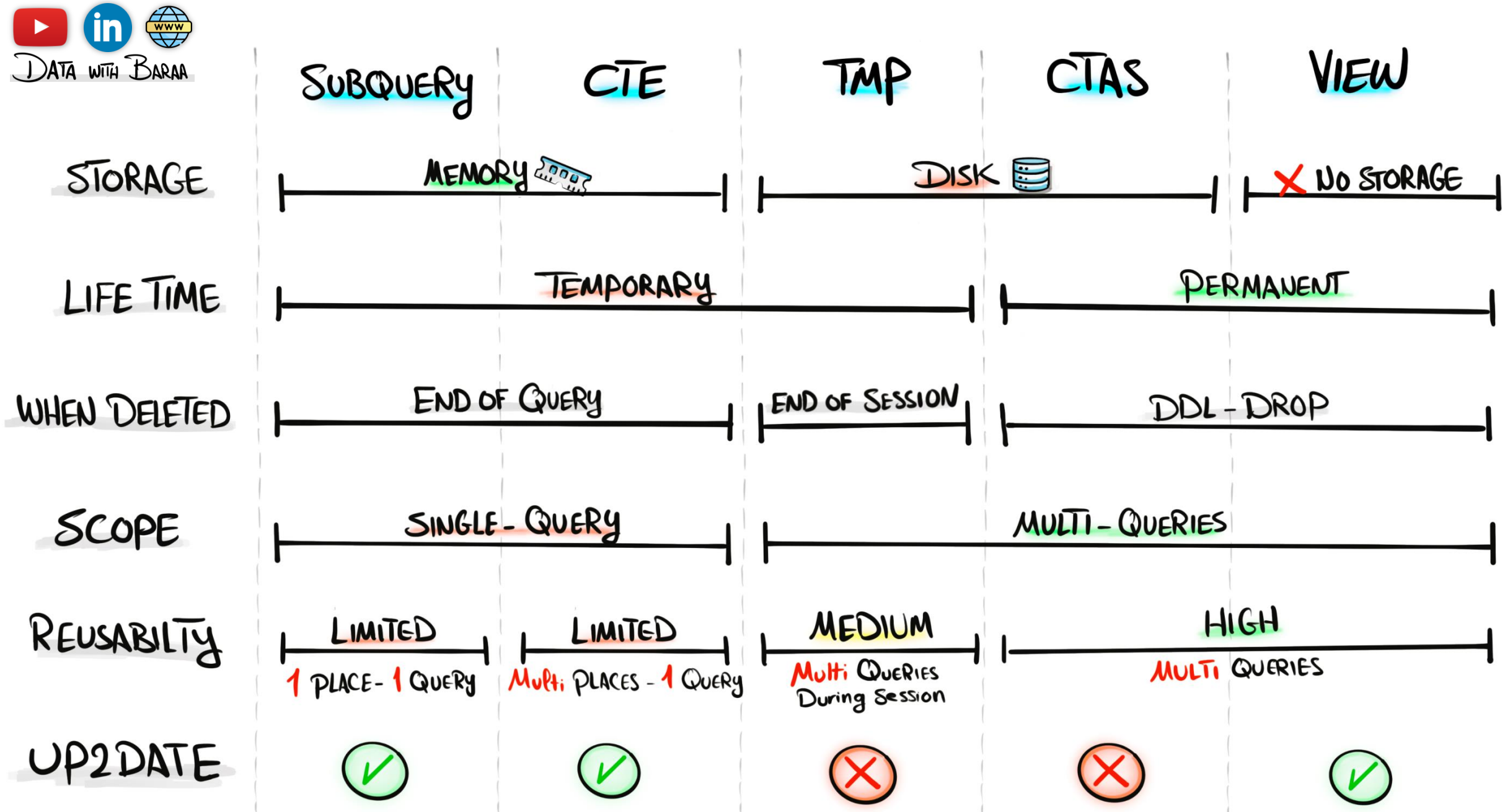
Advantage of TEMP Tables Automatic cleanup of data after session ends.

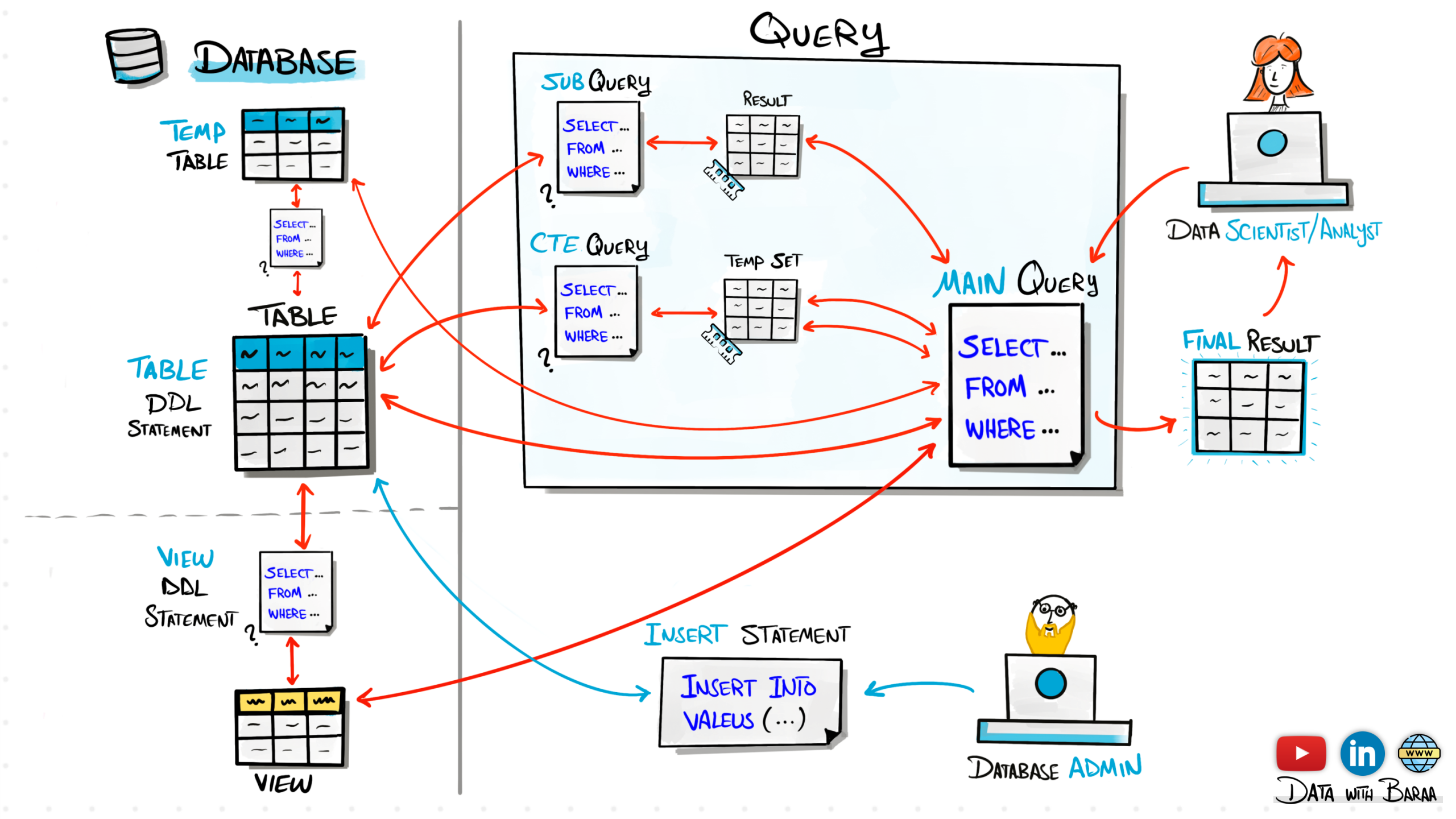
DATA WITH BARAA

Big Picture All Techniques

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

DATA WITH BARAA



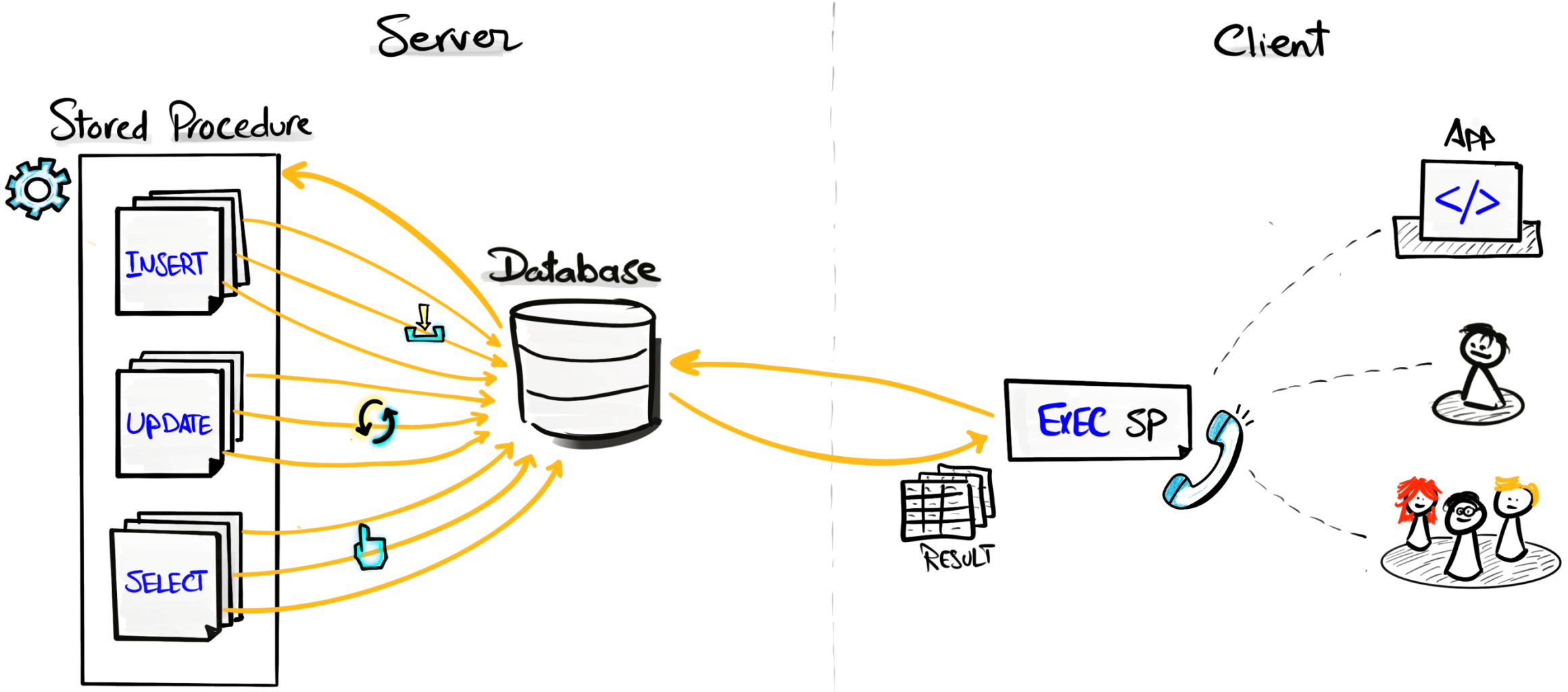

DATABASE
TEMP TABLE
SELECT ...
FROM ...
WHERE ...
TABLE
TABLE
DDL
STATEMENT
VIEW
DDL
STATEMENT
SELECT ...
FROM ...
WHERE ...
VIEW
QUERY
SUB QUERY
SELECT ...
FROM ...
WHERE ...
RESULT
CTE QUERY
SELECT ...
FROM ...
WHERE ...
TEMP SET
MAIN QUERY
SELECT ...
FROM ...
WHERE ...
DATA SCIENTIST/ANALYST
FINAL RESULT
INSERT STATEMENT
INSERT INTO
VALUES (...)
DATABASE ADMIN
DATA WITH BARAA

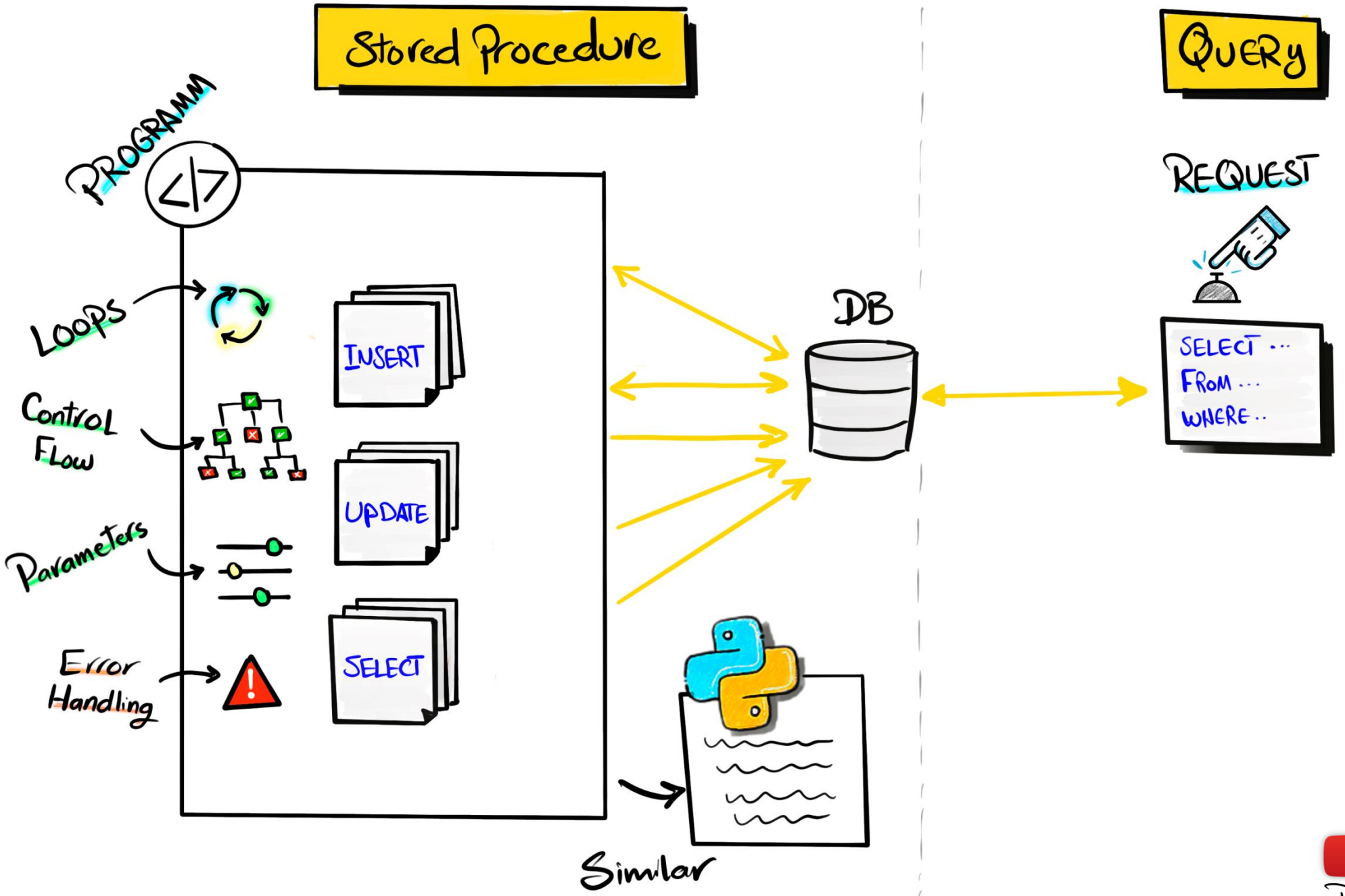
DATA WITH BARAA

Stored Procedure

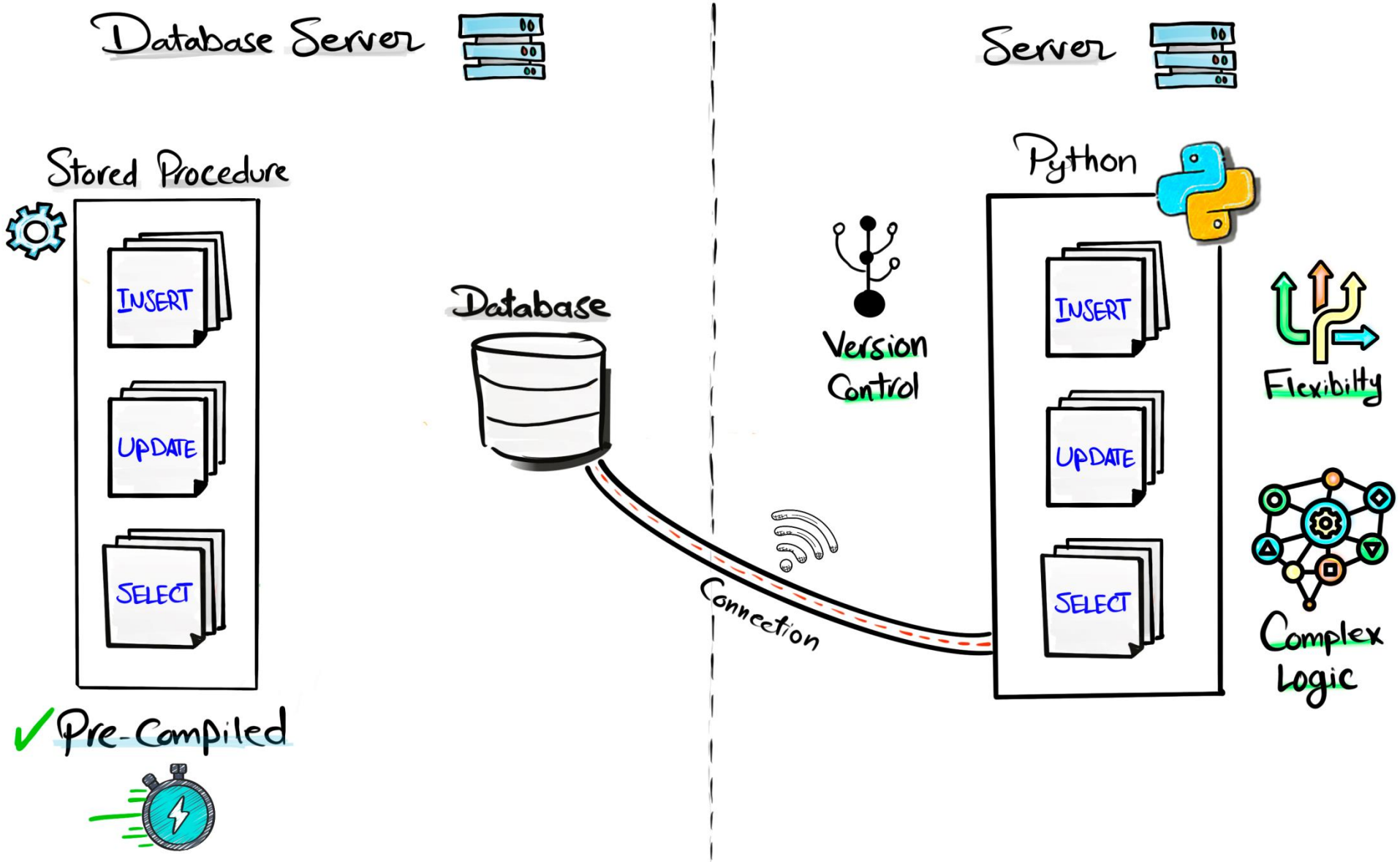
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Stored Procedure

Stored Procedure





Stored Procedure vs Python



Stored Procedure

Stored Procedure
Definition ----->

```
CREATE PROCEDURE ProcedureName AS
BEGIN

  -- SQL STATEMENTS GO HERE

END
```

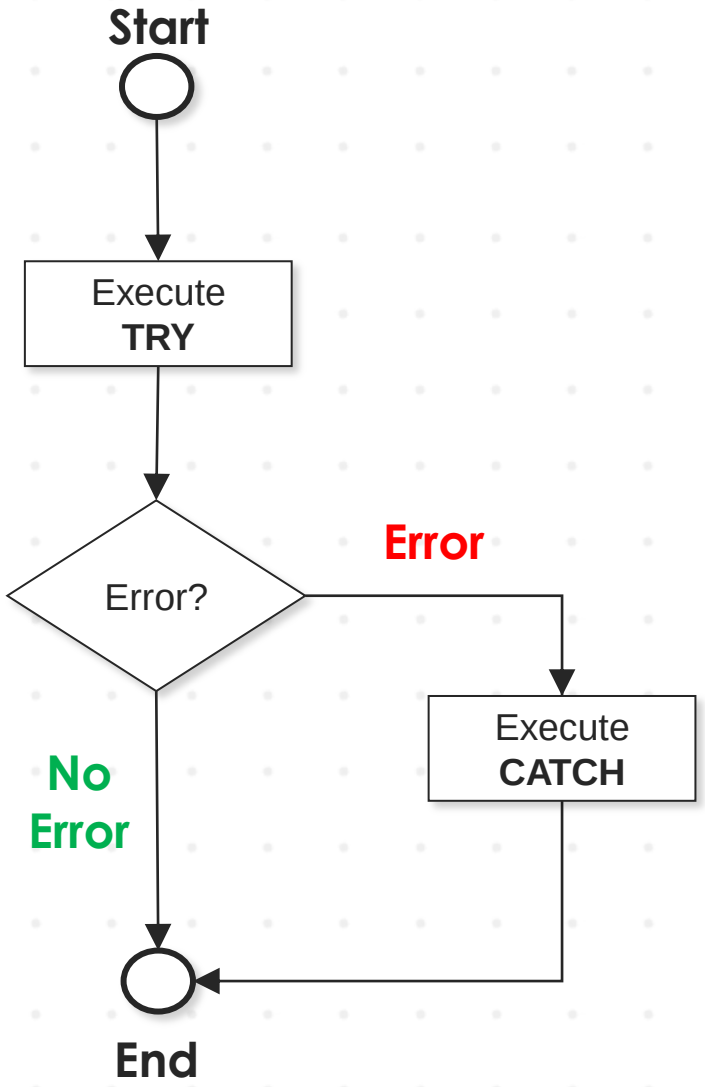
Stored Procedure
Execution (Call) ----->

```
EXEC ProcedureName
```

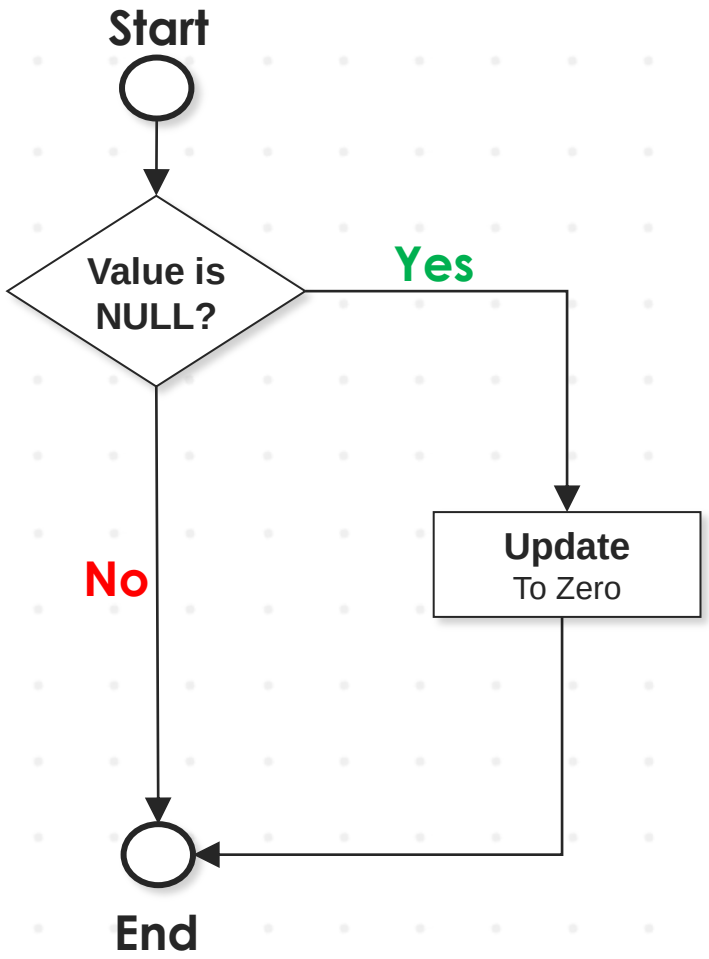
Error Handling

```
BEGIN TRY

-- SQL statements that might cause an error

END TRY
```

```
BEGIN CATCH

-- SQL statements To Handle The Error

END CATCH
```



Flow Control

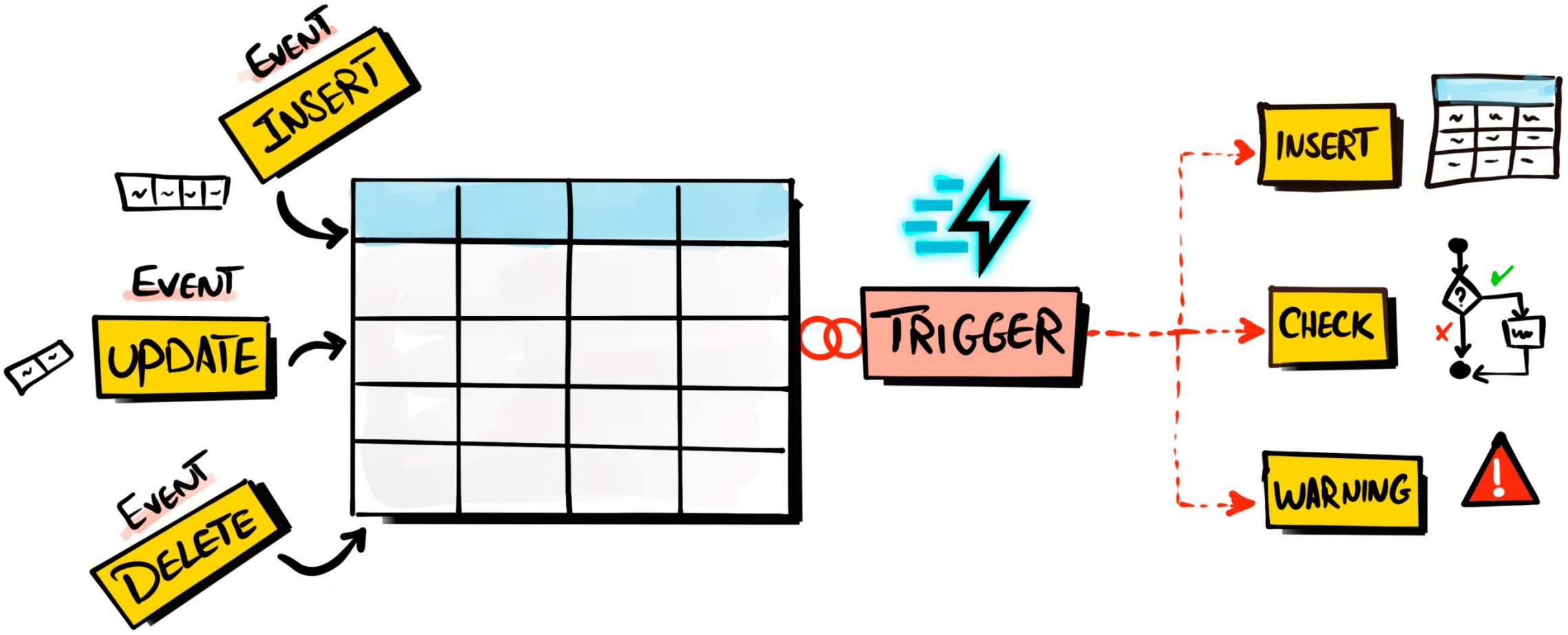


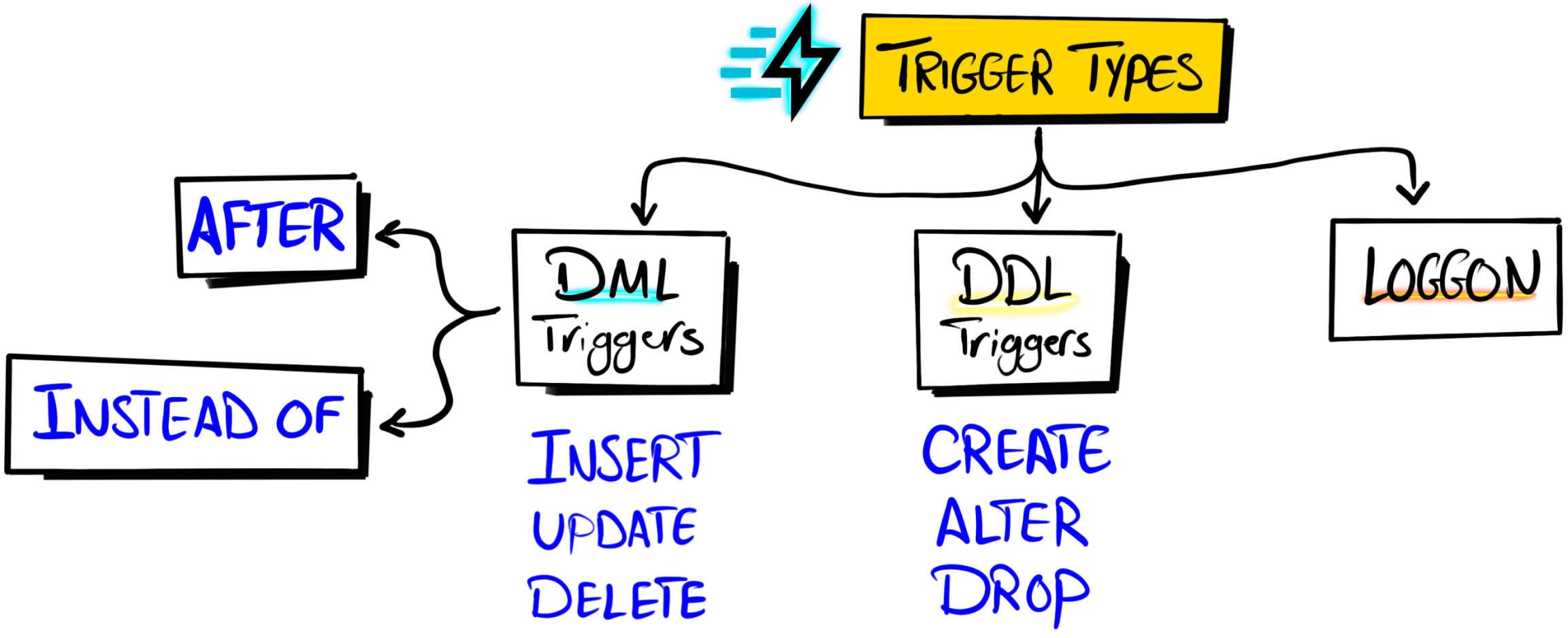
DATA WITH BARAA

Triggers

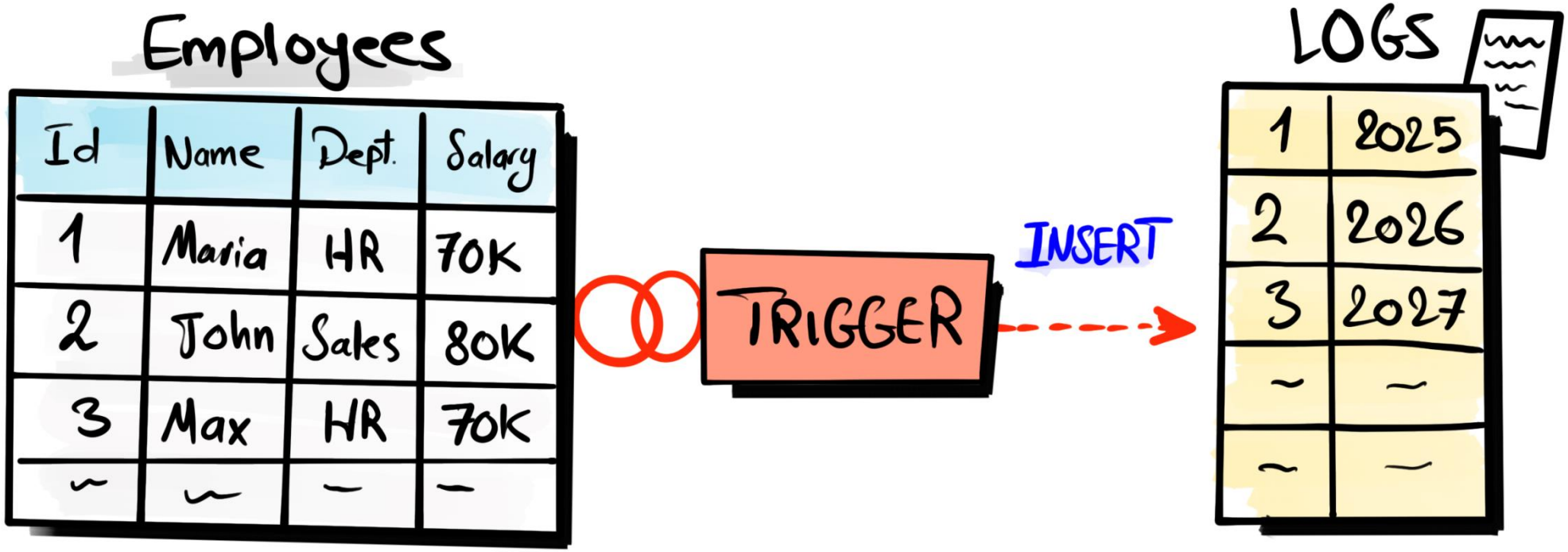
Baraa Khatib Salkini
YouTube | **DATA WITH BARAA**
SQL Course | Triggers

Triggers




TRIGGER TYPES
DML Triggers
DDL Triggers
LOGGON
AFTER
INSTEAD OF
INSERT
UPDATE
DELETE
CREATE
ALTER
DROP

Maintaining Logs



Triggers

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

WHEN ----->

WHAT ----->