





# Query Performance in Snowflake

INTRODUCTION TO DATA MODELING IN SNOWFLAKE







**Nuno Rocha**  
Director of Engineering

# Data storage

	Name	Age	Address	Points
	George	21	Art street 1	999-328-232
	Mary	32	Mia street 32	234-111-281
	Paul	44	Uma street 4	309-221-444
	Laura	39	York 9	082-028-042

  
**Row-Based storage**





# Data storage (1)

	Name	Age	Address	Points
	George	21	Art street 1	999-328-232
	Mary	32	Mia street 32	234-111-281
	Paul	44	Uma street 4	309-221-444
	Laura	39	York 9	082-028-042

→  
**Row-Based storage**







# Data storage (2)

	Name	Age	Address	Points
	George	21	Art street 1	999-328-232
	Mary	32	Mia street 32	234-111-281
	Paul	44	Uma street 4	309-221-444
	Laura	39	York 9	082-028-042

Row-Based storage

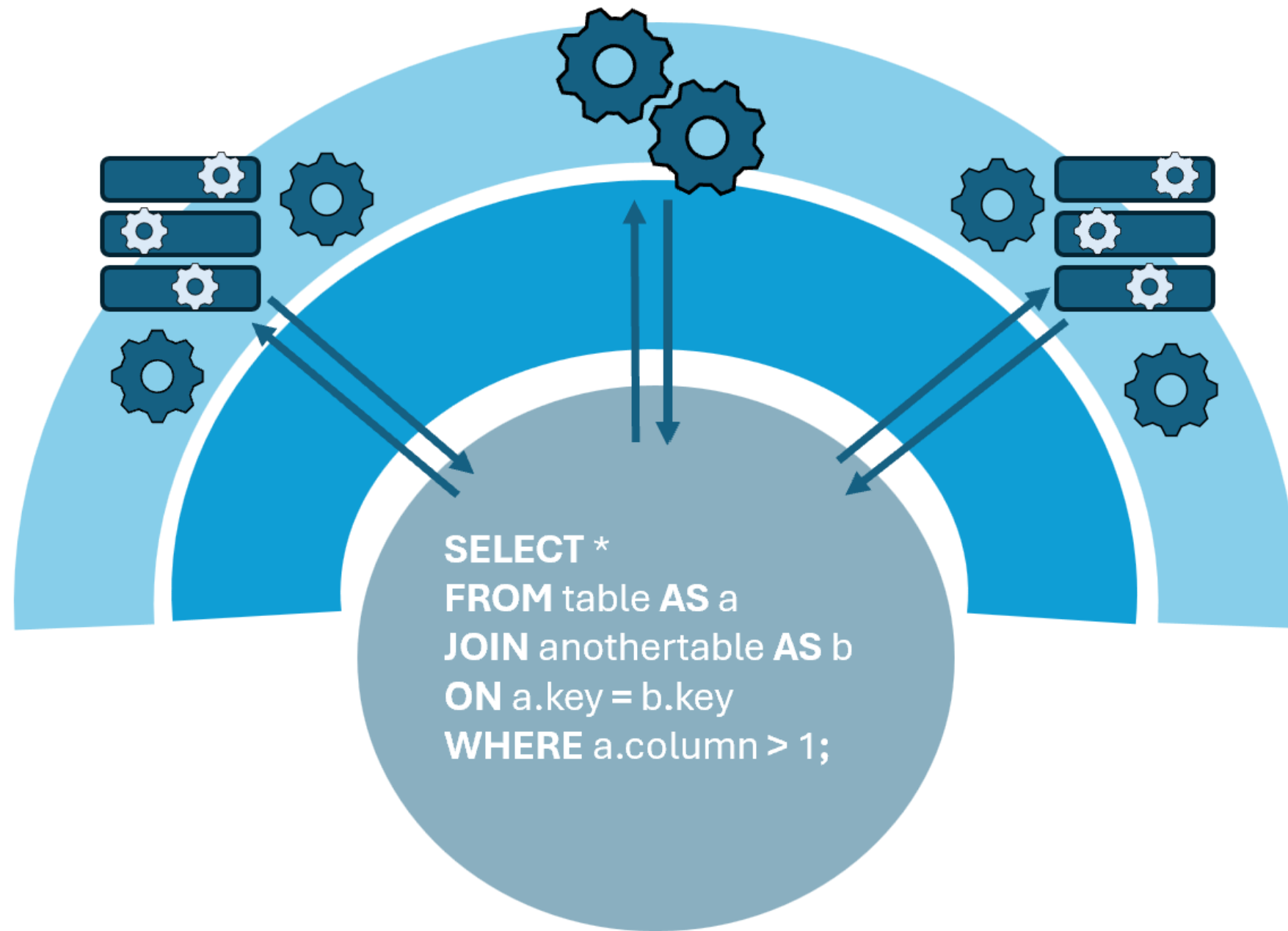


				
Name	George	Mary	Paul	Laura
Age	21	32	44	39
Address	Art street 1	Mia street 32	Uma street 4	York 9
Phone	999-328-232	234-111-281	309-221-444	082-028-042

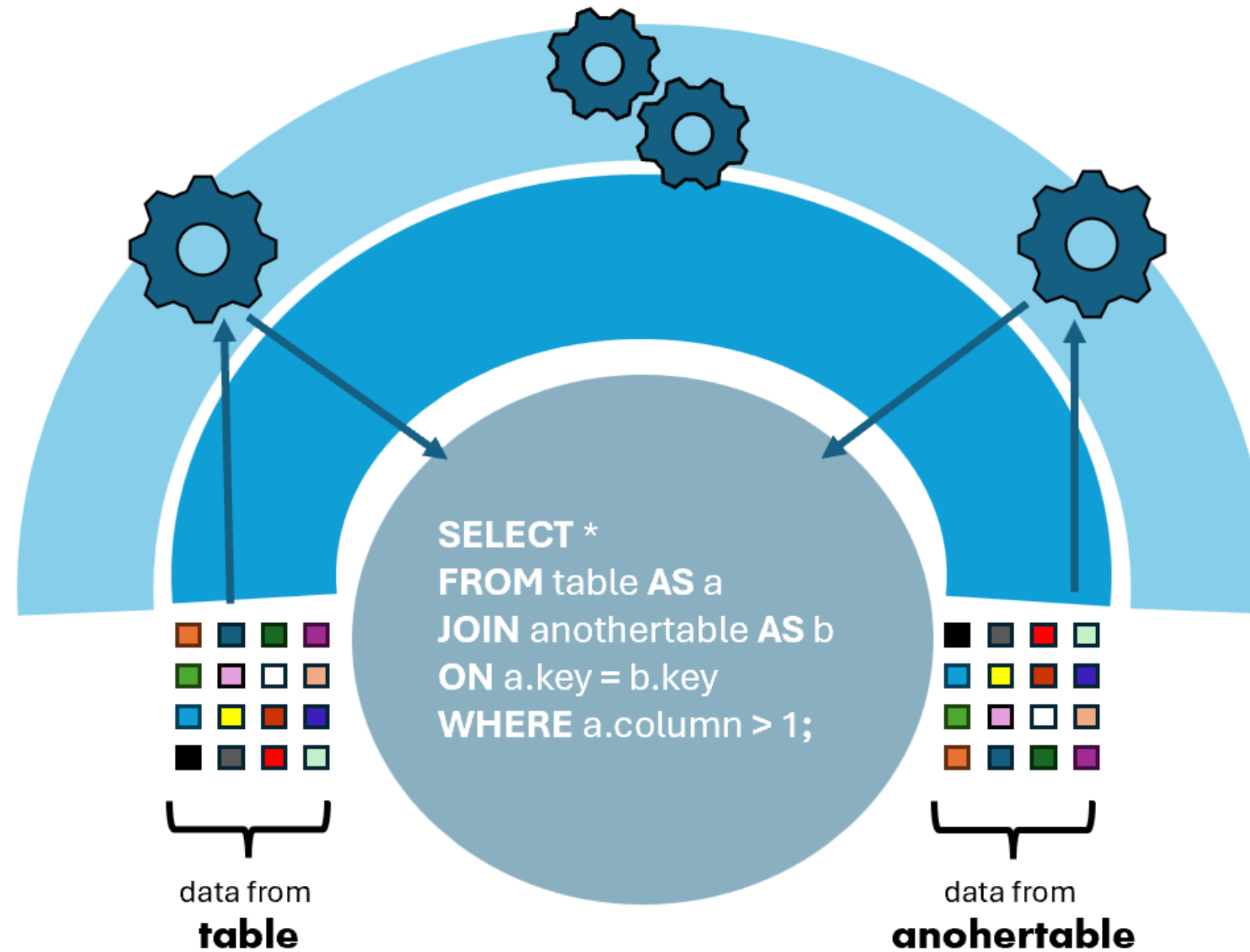
Column-Based storage



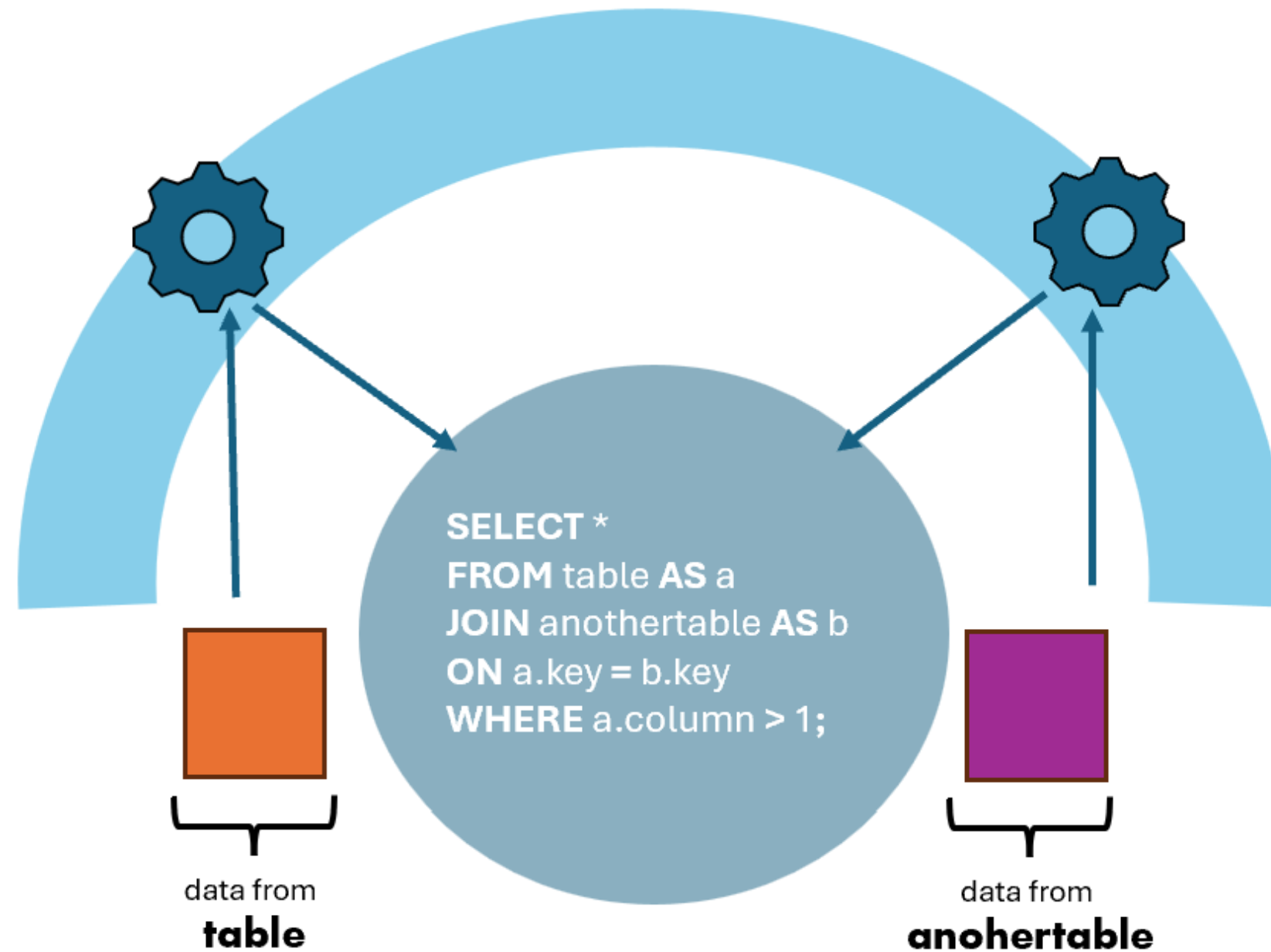
# Massively parallel processing



# MPP's role in data storage



# MPP's role in data storage (1)



# Visualizing query execution times

The screenshot displays the Snowflake Data Studio interface. On the left, a sidebar shows the database structure under 'SNOWFLAKE\_SAMPLE\_DATA', including schemas like 'TPCH\_SF1' and tables like 'CUSTOMER', 'ORDERS', etc. The main panel shows a SQL query executed against 'SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF1'. The query selects customer names, order counts, and total prices for urgent orders, grouped by customer name and ordered by total price descending.

```
1 SELECT c.c_name AS customer_name,
2       COUNT(DISTINCT o_orderkey) AS order_count,
3       SUM(o_totalprice) AS total_price
4 FROM customer AS c
5      JOIN orders AS o
6      ON c.c_custkey = o.o_custkey
7 WHERE o.o_orderpriority = '1-URGENT'
8 GROUP BY c.c_name
9 ORDER BY total_price DESC
```

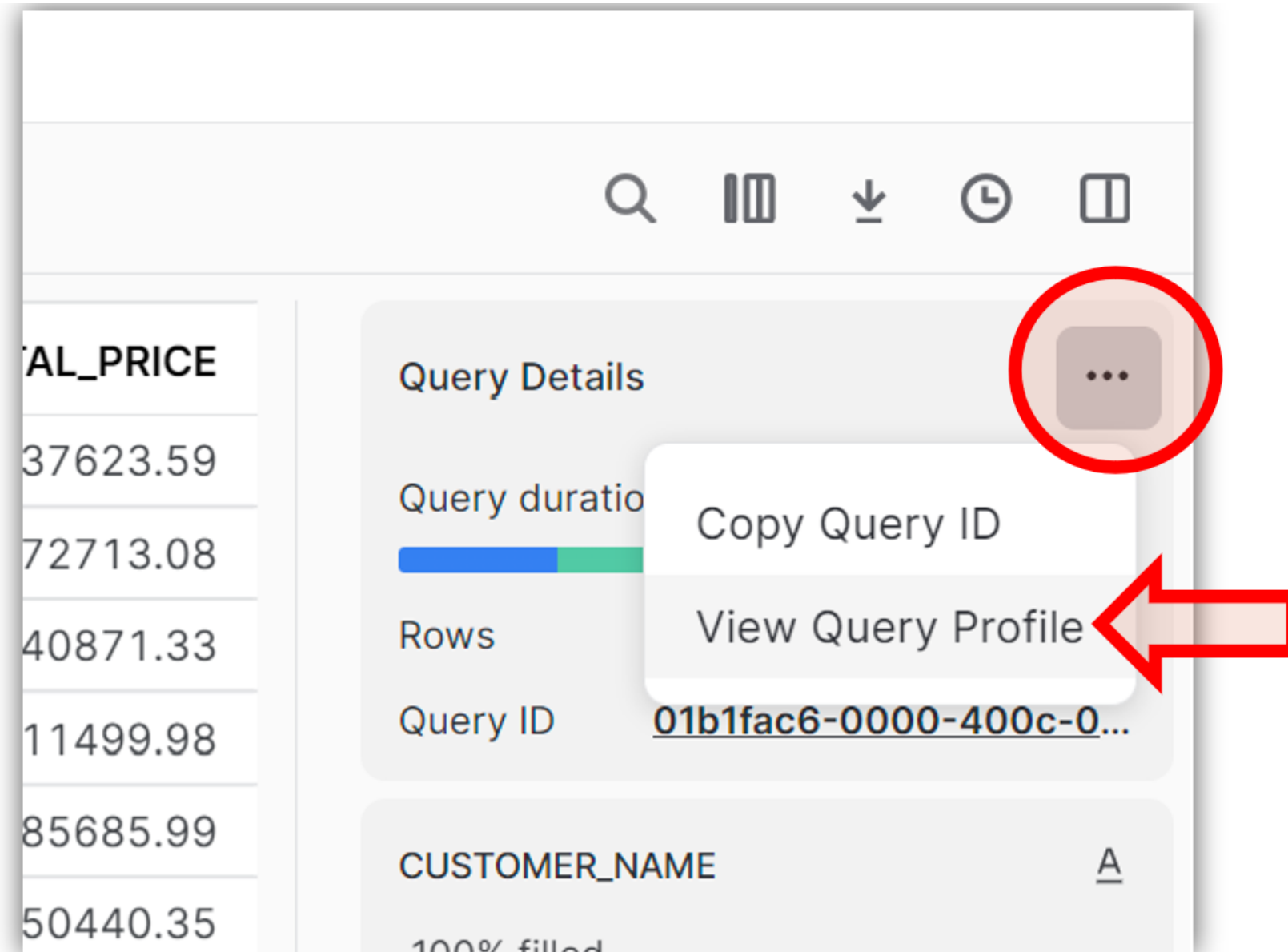
Below the query, the 'Results' tab shows a table with 14 rows of data:

	CUSTOMER_NAME	ORDER_COUNT	TOTAL_PRICE
1	Customer#000057058	13	2737623.59
2	Customer#000048778	13	2572713.08
3	Customer#000036970	12	2540871.33
4	Customer#000061813	12	2511499.98
5	Customer#000013195	12	2485685.99
6	Customer#000041449	11	2450440.35
7	Customer#000035041	12	2432854.00
8	Customer#000020668	10	2413272.52
9	Customer#000077374	10	2361455.40
10	Customer#000066073	12	2360901.38
11	Customer#000067273	11	2344260.61
12	Customer#000035473	13	2339517.04
13	Customer#000040786	12	2297743.91
14	Customer#000031039	11	2278870.78

On the right, the 'Query Details' panel shows the query duration as 712ms, 92.3K rows, and the query ID '01b1fac6-0000-400c-0...'. Below this, there are three visualizations: a 'CUSTOMER\_NAME' bar chart (100% filled), an 'ORDER\_COUNT' bar chart (range 1 to 14), and a 'TOTAL\_PRICE' bar chart (range 965.09 to 2,737,623.59).




# Visualizing query execution times (1)



The screenshot shows a Snowflake interface. On the left, a table displays a column of prices. On the right, a 'Query Details' panel is visible. A context menu is open over the 'Query ID' field in the details panel, with a red arrow pointing to the 'View Query Profile' option.

TOTAL_PRICE
37623.59
72713.08
40871.33
11499.98
85685.99
50440.35

**Query Details**

Query duration 

Rows

Query ID [01b1fac6-0000-400c-0...](#)

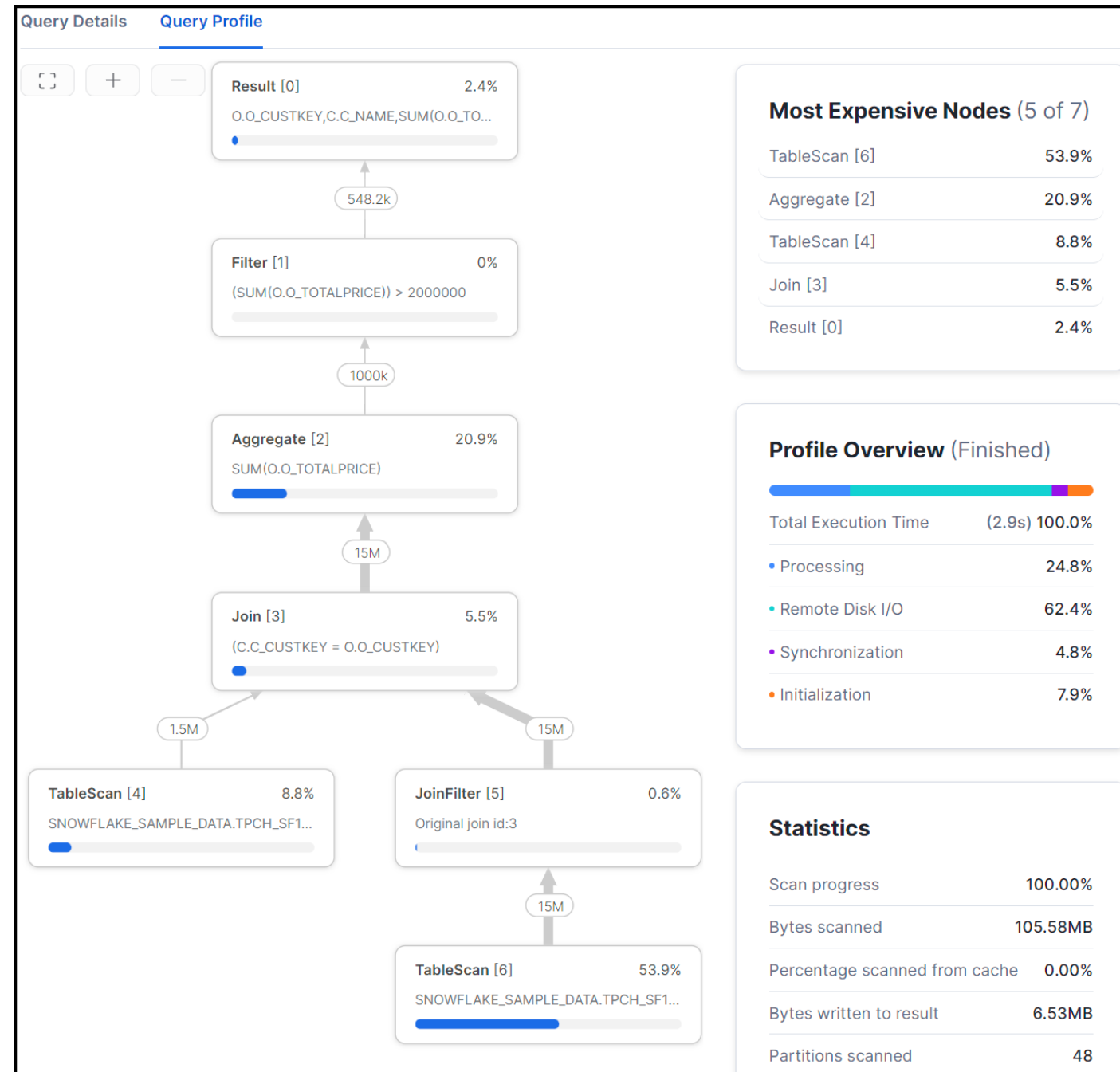
**CUSTOMER\_NAME** [A](#)

100% filled

**Context Menu:**

- Copy Query ID
- View Query Profile

# Visualizing query execution times (2)



# Terminology overview

- **Massively Parallel Processing (MPP):** Snowflake's engine that processes data using multiple servers simultaneously.
- **Micro-partitions:** Small data storage segments in Snowflake that enhance retrieval speed.
- **Columnar Storage:** Data storage format that stores each column of data separately.
- **Row-based Storage:** Traditional data storage format where each row of data is stored sequentially.
- **PostgreSQL:** Open-source relational database system that uses row-based storage, commonly compared to Snowflake for performance benchmarking.
- **Query Profile:** Snowflake feature to visualize the steps and resource usage of query execution.

# Let's practice!

INTRODUCTION TO DATA MODELING IN SNOWFLAKE

# Snowflake Data Objects

INTRODUCTION TO DATA MODELING IN SNOWFLAKE

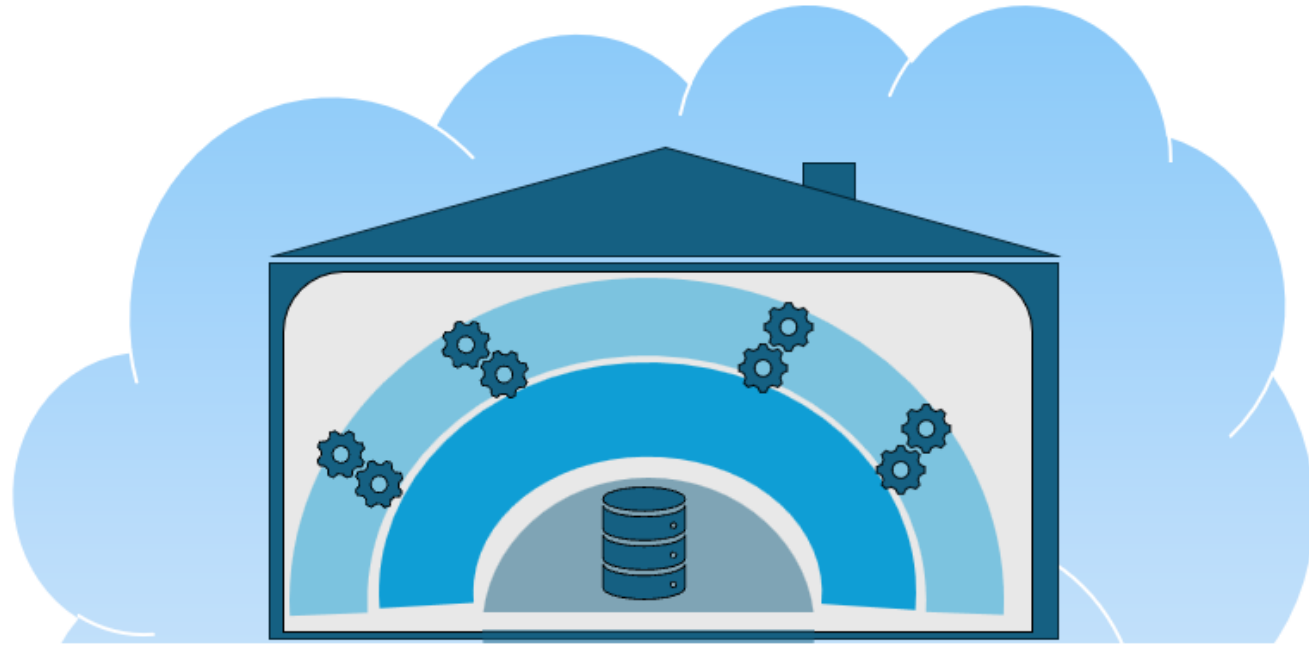


**Nuno Rocha**  
Director of Engineering

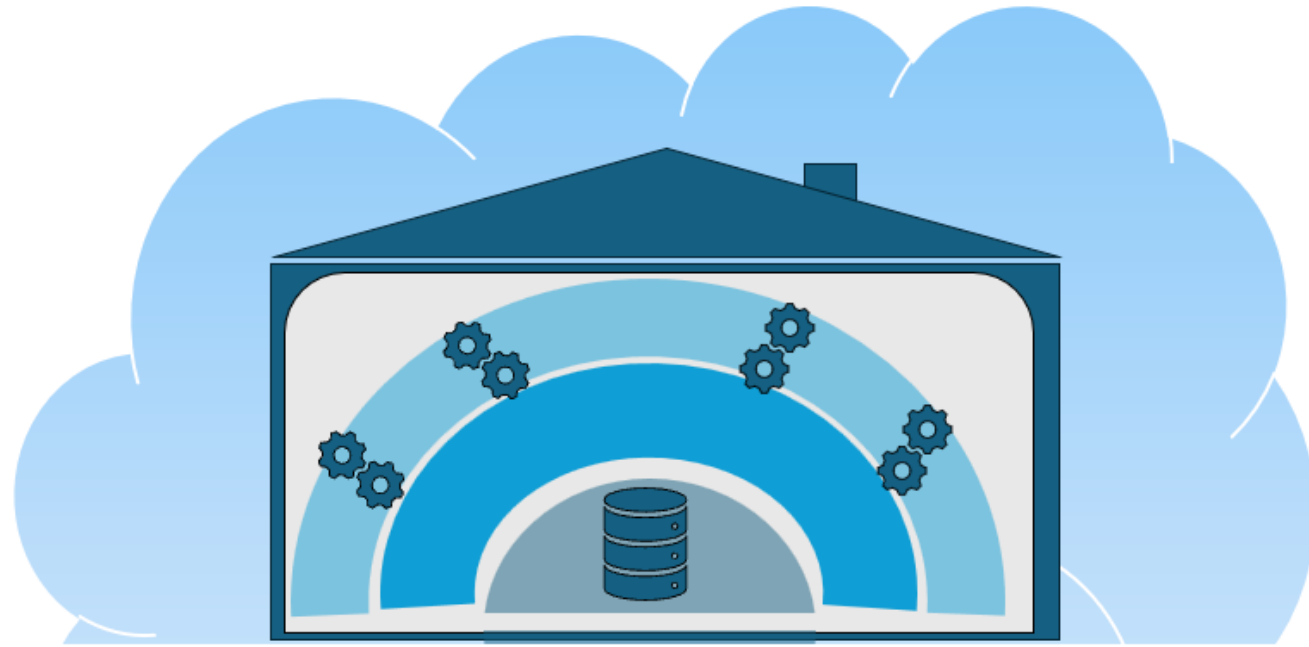
# Data warehouse



# Traditional vs. virtual warehouse



# Traditional vs. virtual warehouse (1)



**vs.**



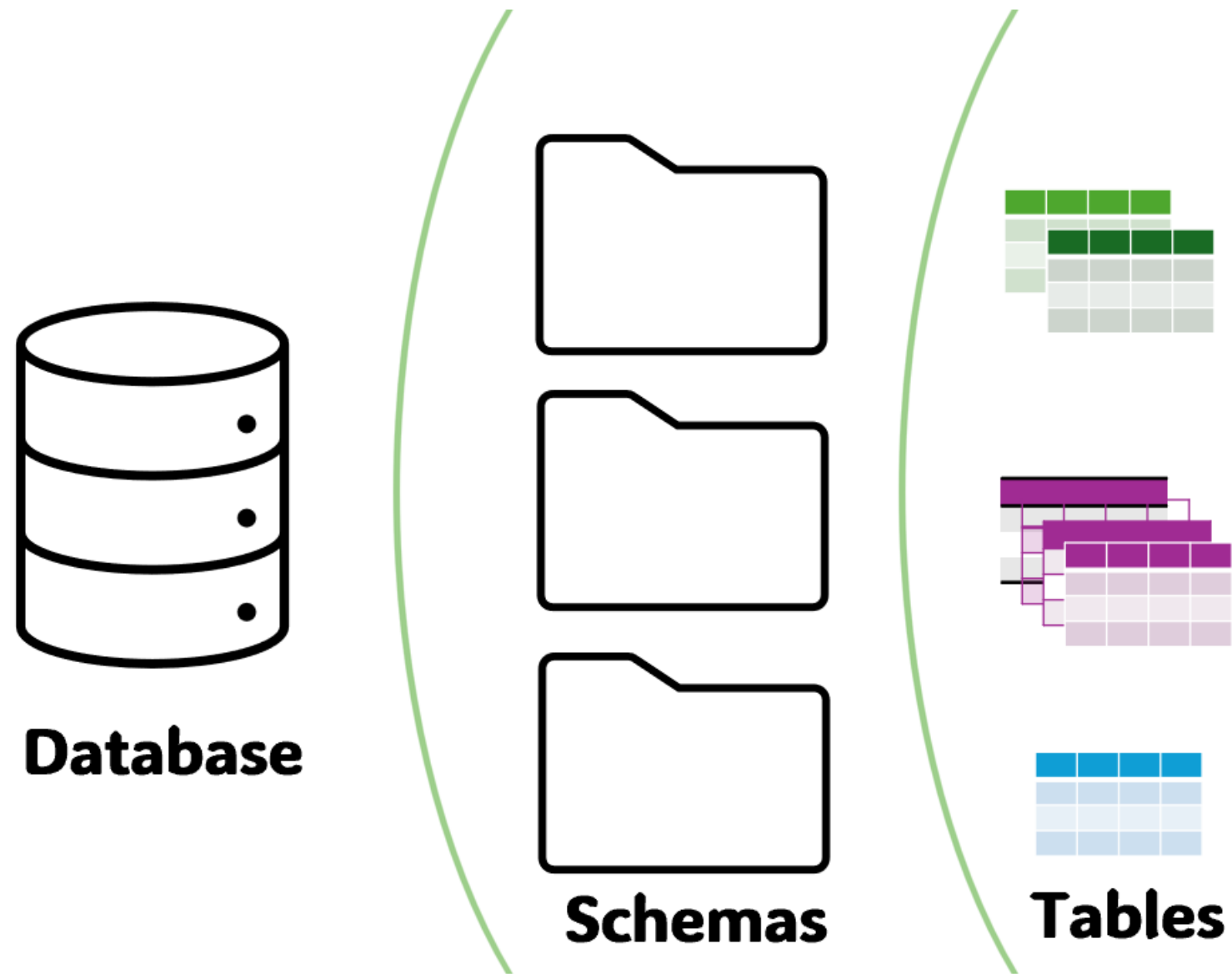


# Traditional vs. virtual warehouse (2)

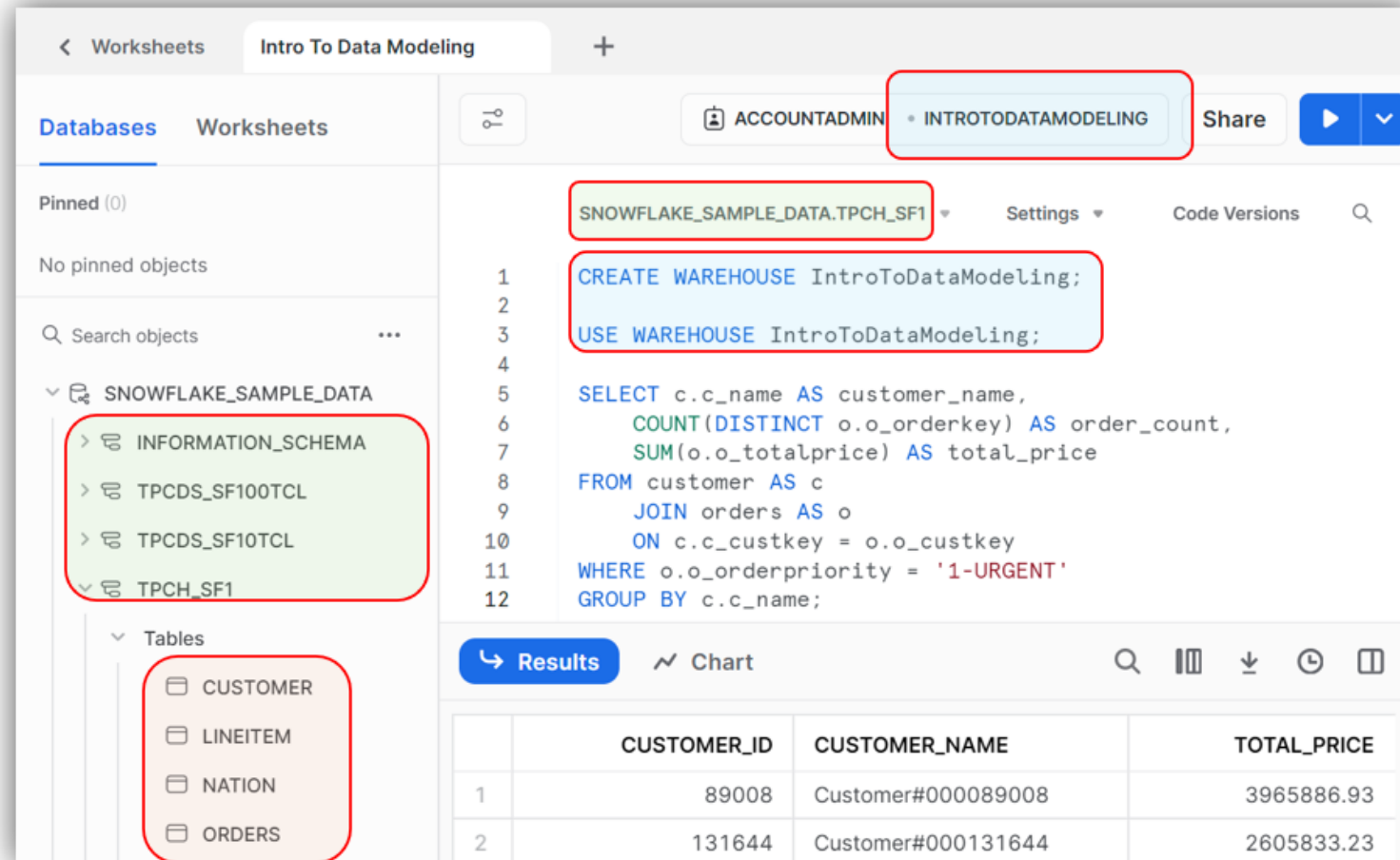
- **CREATE WAREHOUSE** : Snowflake clause that establishes a virtual computing resource for data processing tasks.
- **USE WAREHOUSE** : Snowflake clause that designates the active data warehouse for the current session.



# Schemas and tables



# Schemas and tables (1)



The screenshot displays the Snowflake Data Studio interface. The top navigation bar shows 'Worksheets' and 'Intro To Data Modeling'. The left sidebar shows the database structure under 'Databases' and 'Worksheets'. The 'Databases' tab is active, showing 'Pinned (0)' and 'No pinned objects'. The 'Search objects' field is empty. The 'SNOWFLAKE\_SAMPLE\_DATA' database is expanded, showing 'INFORMATION\_SCHEMA', 'TPCDS\_SF100TCL', 'TPCDS\_SF10TCL', and 'TPCH\_SF1'. The 'Tables' section is expanded, showing 'CUSTOMER', 'LINEITEM', 'NATION', and 'ORDERS'. The main editor shows a SQL query with the following code:

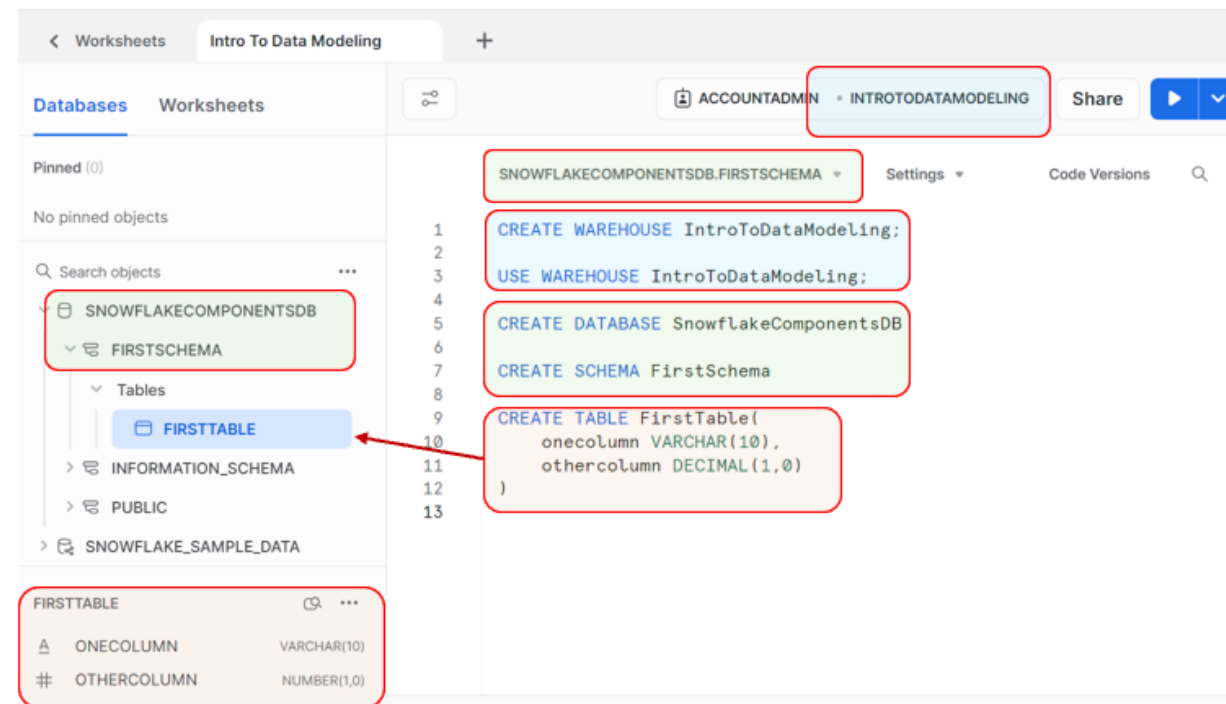
```
1 CREATE WAREHOUSE IntroToDataModeling;
2
3 USE WAREHOUSE IntroToDataModeling;
4
5 SELECT c.c_name AS customer_name,
6        COUNT(DISTINCT o.o_orderkey) AS order_count,
7        SUM(o.o_totalprice) AS total_price
8 FROM customer AS c
9      JOIN orders AS o
10     ON c.c_custkey = o.o_custkey
11 WHERE o.o_orderpriority = '1-URGENT'
12 GROUP BY c.c_name;
```

The query results are displayed in a table with the following columns: CUSTOMER\_ID, CUSTOMER\_NAME, and TOTAL\_PRICE. The results show two rows of data:

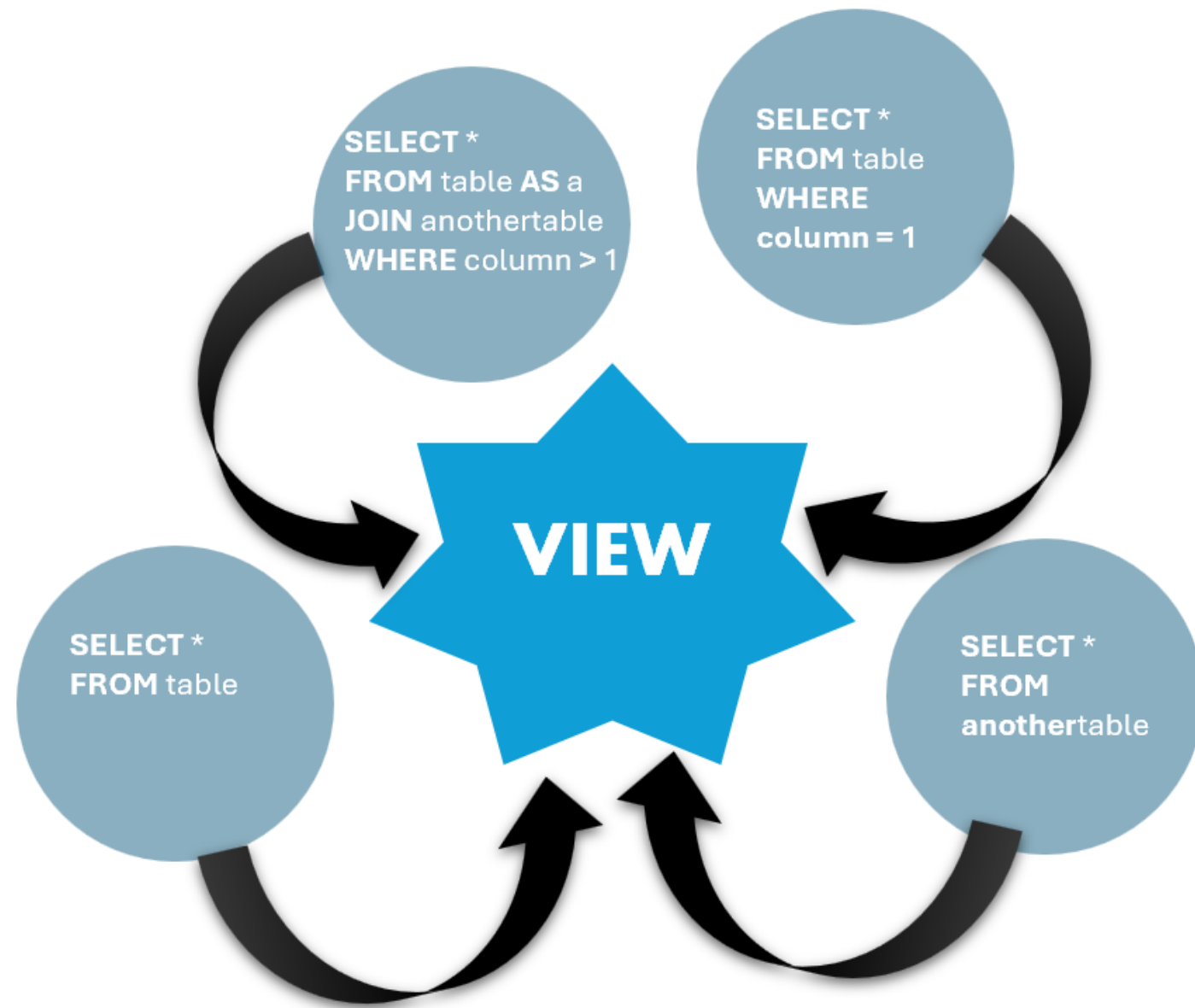
	CUSTOMER_ID	CUSTOMER_NAME	TOTAL_PRICE
1	89008	Customer#000089008	3965886.93
2	131644	Customer#000131644	2605833.23

# Schemas and tables (2)

- **CREATE DATABASE** : Snowflake clause to create a new database for organizing data objects.
- **CREATE SCHEMA** : Snowflake clause to define a container for grouping tables and views.
- **CREATE TABLE** : Snowflake command to create a new table structure and data columns within a specified schema.



# Views



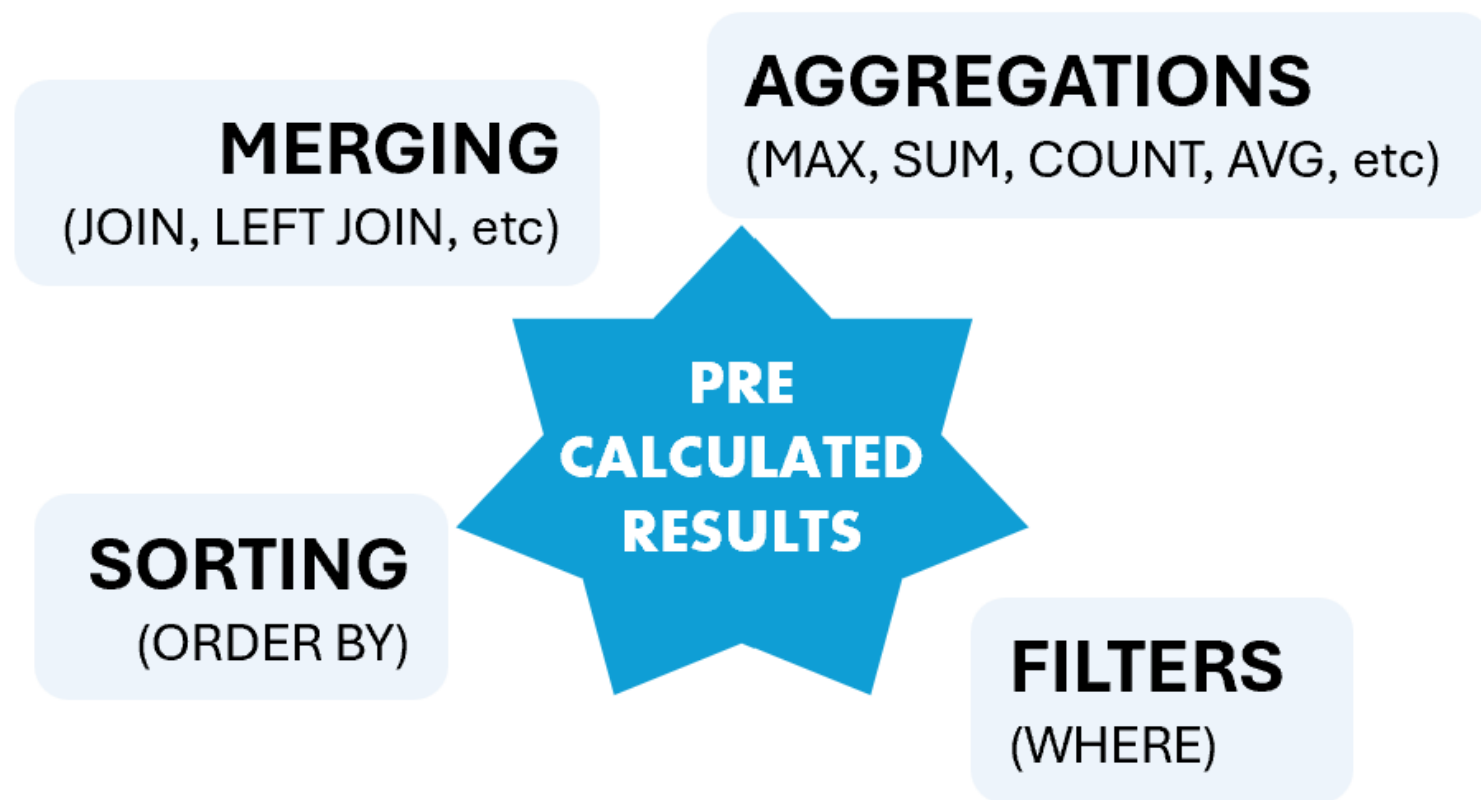
- **VIEWS:**
  - Acts like a virtual entity.
  - Avoid data duplicates.
  - Saves storage space and ensures data consistency.
  - Dynamically present data based on query logic.

# Views (1)

- Example VIEW summarizing customers' urgent order

```
CREATE OR REPLACE VIEW customer_urgent_orders AS
SELECT c.c_name AS customer_name,
       COUNT(DISTINCT o_orderkey) AS order_count,
       SUM(o_totalprice) AS total_price
FROM customer AS c
      JOIN orders AS o
      ON c.c_custkey = o.o_custkey
WHERE o.o_orderpriority = '1-URGENT'
GROUP BY c.c_name;
ORDER BY total_price DESC;
```

# Materialized views



- **MATERIALIZED VIEWS:**
  - Store pre-calculated results.
  - Improve query performance by storing computed operations.
  - Reduces query processing time.
  - Maintain a refreshed data snapshot for retrieval at any time.
  - Encapsulate complex operations for streamlined data modeling.

# Materialized views (1)

```
CREATE OR REPLACE MATERIALIZED VIEW top_customers AS
SELECT o.o_custkey AS customer_id,
       c.c_name AS customer_name,
       CASE
         WHEN SUM(o.o_totalprice) > 5000000 THEN 'Over Price'
         WHEN SUM(o.o_totalprice) > 3000000 THEN 'Top Price'
         WHEN SUM(o.o_totalprice) > 2000000 THEN 'Average Price'
         ELSE 'Review Price'
       END AS total_price
FROM customer AS c
      JOIN orders AS o ON c.c_custkey = o.o_custkey
GROUP BY o.o_custkey, c.c_name
HAVING total_price > 2000000;
```

```
CREATE OR REPLACE VIEW customer_financial_summary AS
SELECT c.customerid,
       -- Create a new conditional attribute
       CASE
         WHEN AVG(c.estimatedsalary) > 150000 THEN 'Top Income'
         WHEN AVG(c.estimatedsalary) > 90000 THEN 'High Income'
         THEN AVG(c.estimatedsalary) > 20000 THEN 'Average Income'
         ELSE 'Low Income'
       END AS customer_category,
       -- Add aggregation to the attributes
       COUNT(DISTINCT cp.productid) AS product_count
FROM customers AS c
      LEFT JOIN customerproducts AS cp
        ON c.customerid = cp.customerid
-- Group the results
GROUP BY c.customerid;
```



# Terminology overview

- **Data warehouse (traditional):** A centralized system for storing and analyzing large volumes of data using a single server setup.
- **Virtual data warehouse (in Snowflake):** A flexible, scalable set of cloud-based computing resources used explicitly for processing and analyzing data.
- **Database:** In Snowflake, a database is the primary container for data, holding schemas.
- **Schema:** Collection of logical structures, that contain tables, views, etc.
- **Table:** Representation of an entity in the database; fundamental data storage structure, organized in rows and columns within schemas.
- **View:** Saved query that presents data as a virtual entity without storing the data separately.
- **Materialized View:** A stored version of a view that physically saves the query result for faster access.

# Functions overview

```
CREATE OR REPLACE WAREHOUSE data_warehouse_name;
```

```
USE WAREHOUSE data_warehouse_name;
```

```
CREATE OR REPLACE DATABASE database_name;
```

```
CREATE OR REPLACE SCHEMA schema_name;
```

```
CREATE OR REPLACE TABLE table_name(  
    column_name datatype,  
    other_columns datatype);
```

# Exemplary (materialized) view template

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name,
       SUM(column_name2) AS sum_alias,
       CASE WHEN column_name5    condition    value THEN assigned_value
            ELSE another_assigned_value
       END AS case_alias
FROM table_name AS table_alias
   JOIN other_table AS other_alias ON table_alias.FK = other_alias.PK
   LEFT JOIN another_table AS another_alias ON table_alias.FK = other_alias.PK
GROUP BY column_name;
```

# Let's practice!

INTRODUCTION TO DATA MODELING IN SNOWFLAKE

# Query Optimization

INTRODUCTION TO DATA MODELING IN SNOWFLAKE



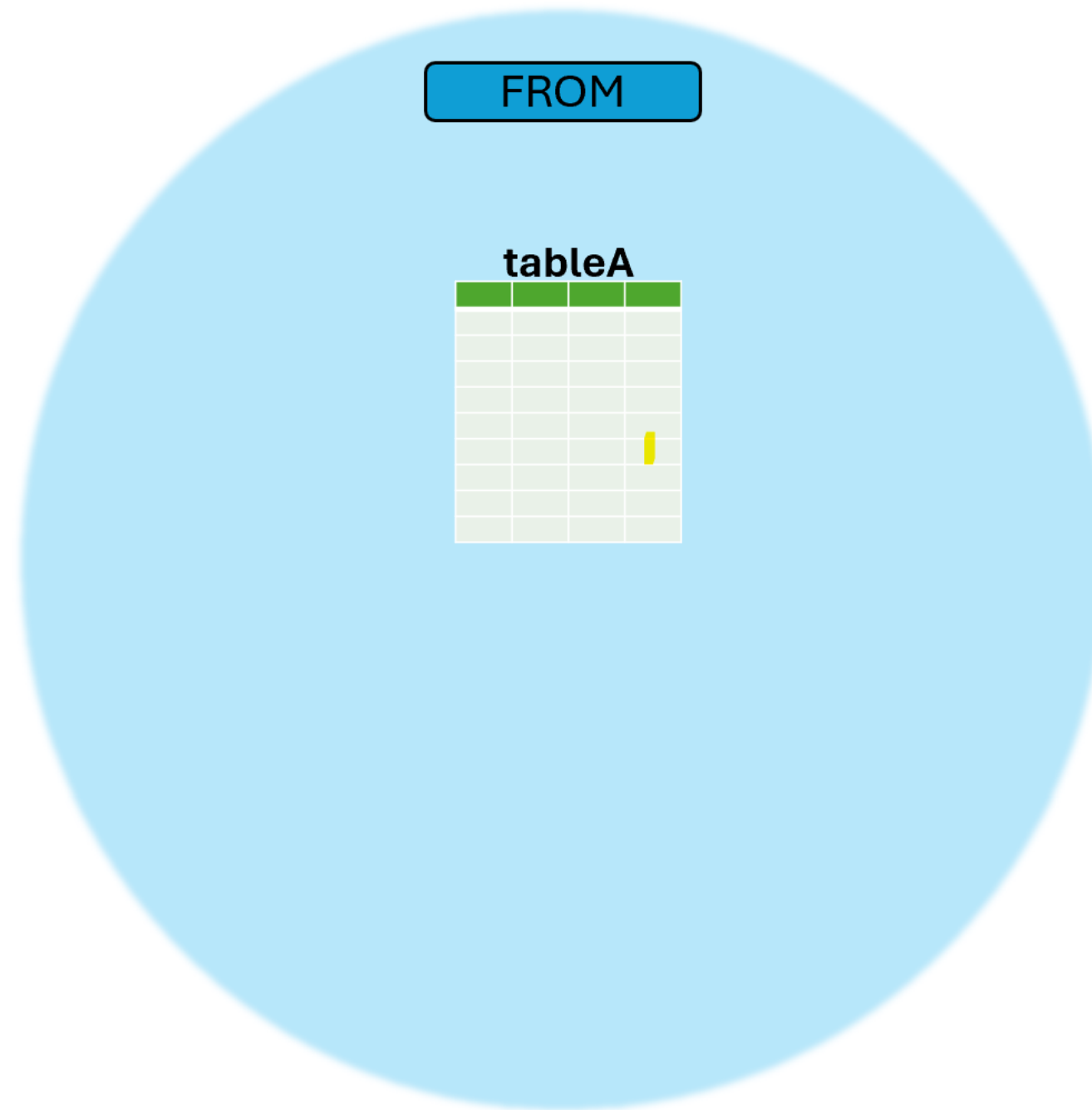
**Nuno Rocha**

Director of Engineering

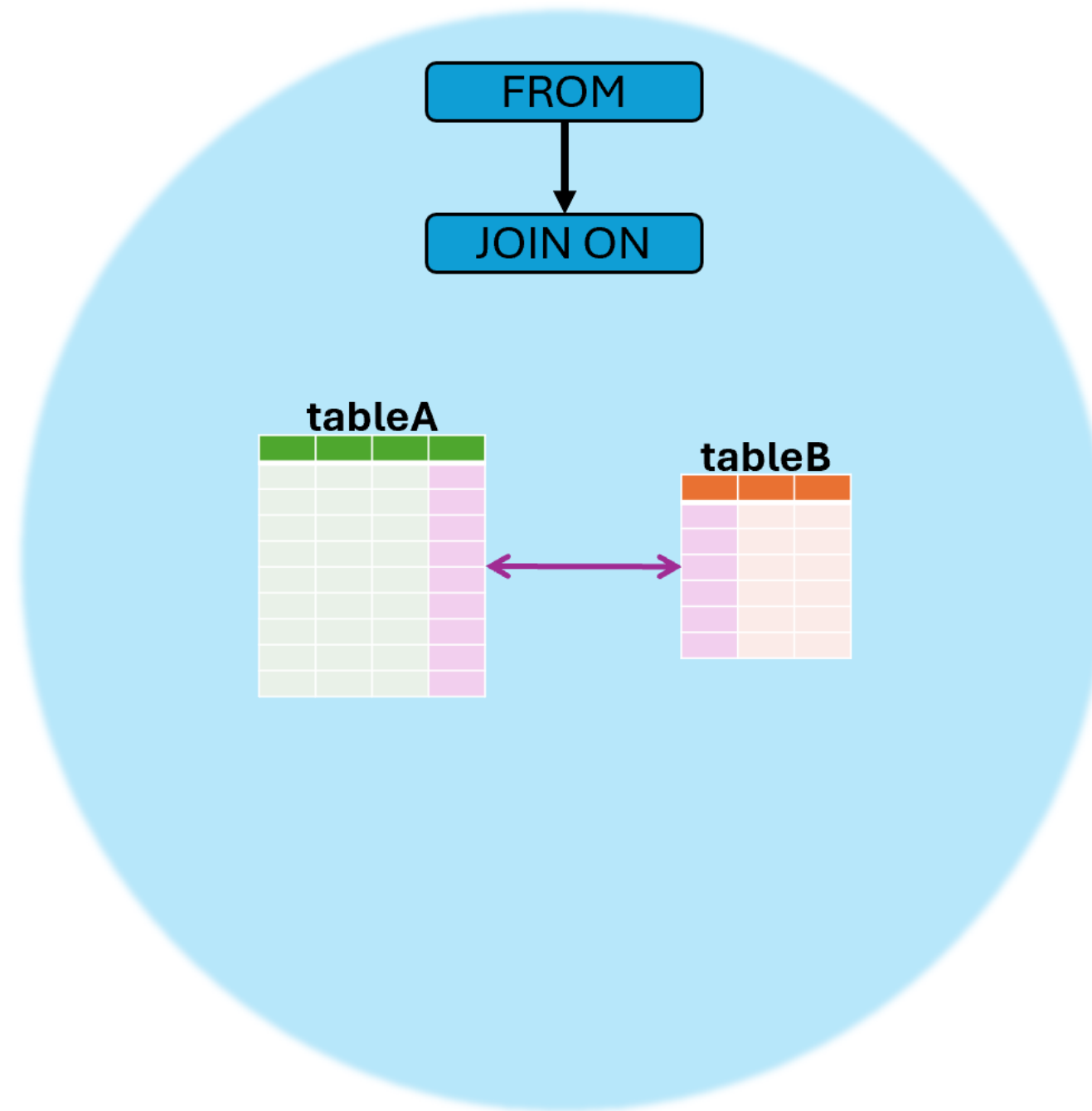
# Query execution order

```
SELECT a.*,  
       AVG(b. column)  
FROM tableA AS a  
     JOIN tableB AS b  
       ON a.key=b.key  
WHERE a.column = value  
GROUP BY a.column  
HAVING AVG(b.column) > 0  
ORDER BY a.column  
LIMIT 100;
```

# Query execution order (1)

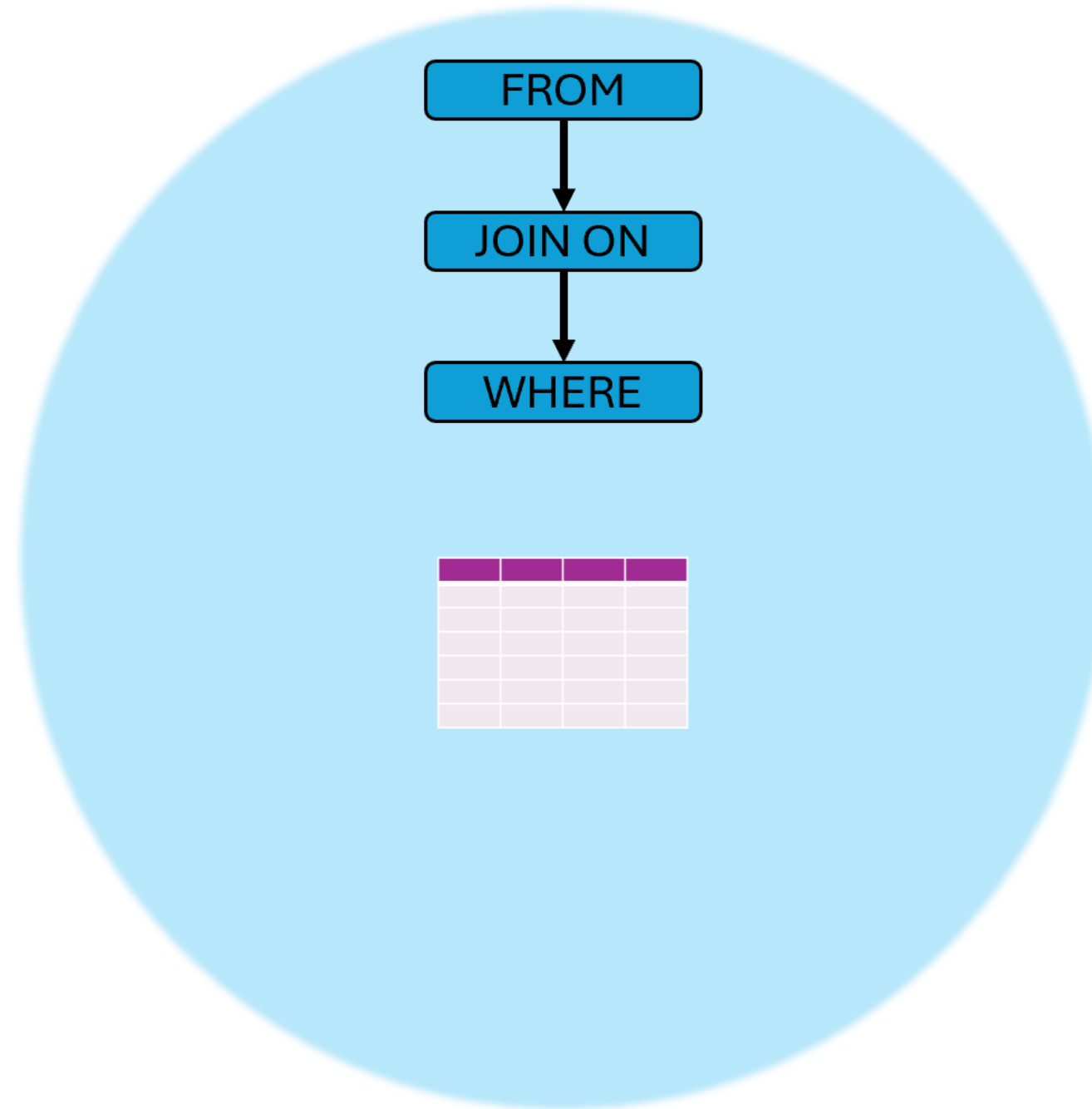


# Query execution order (2)

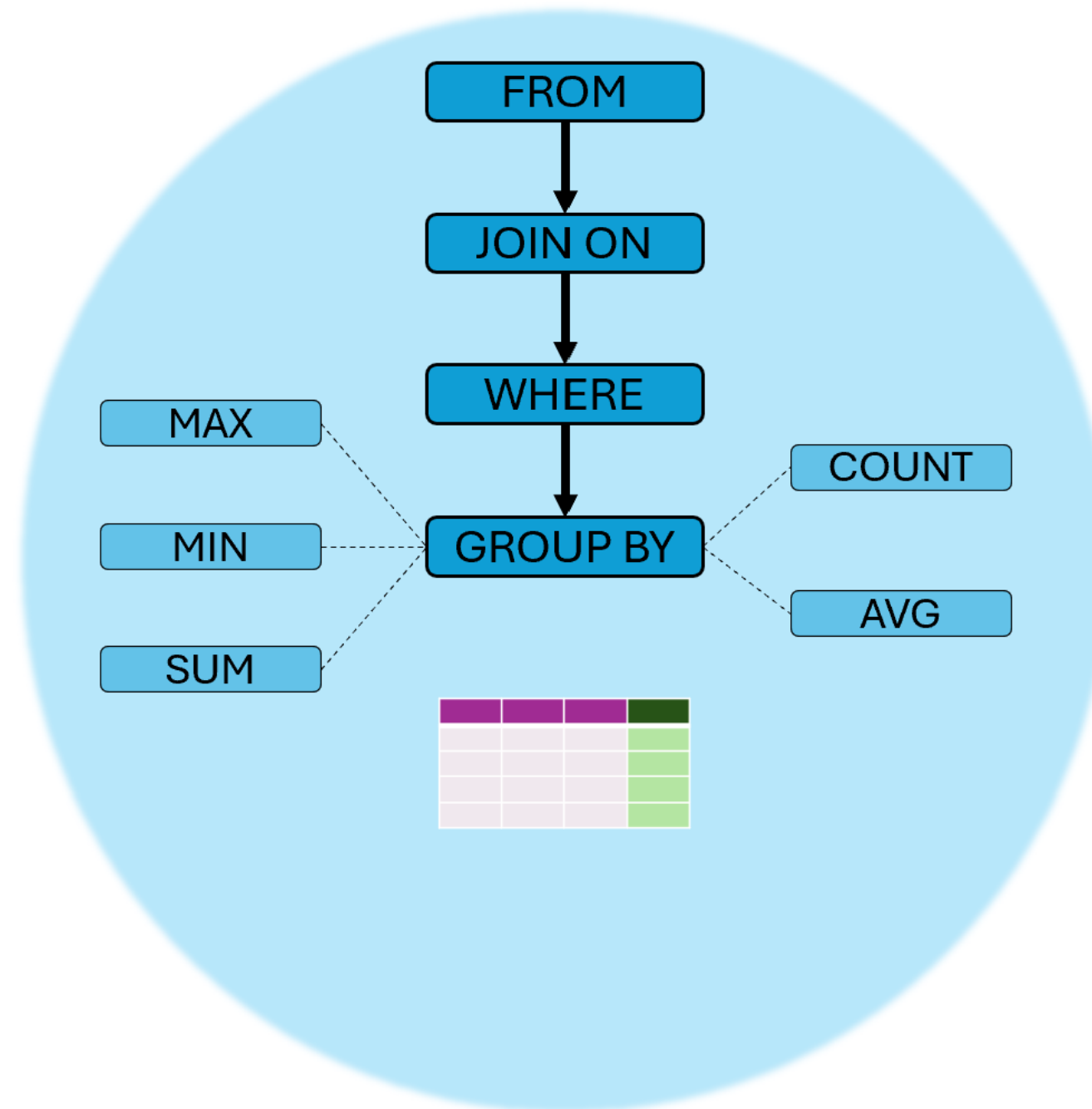




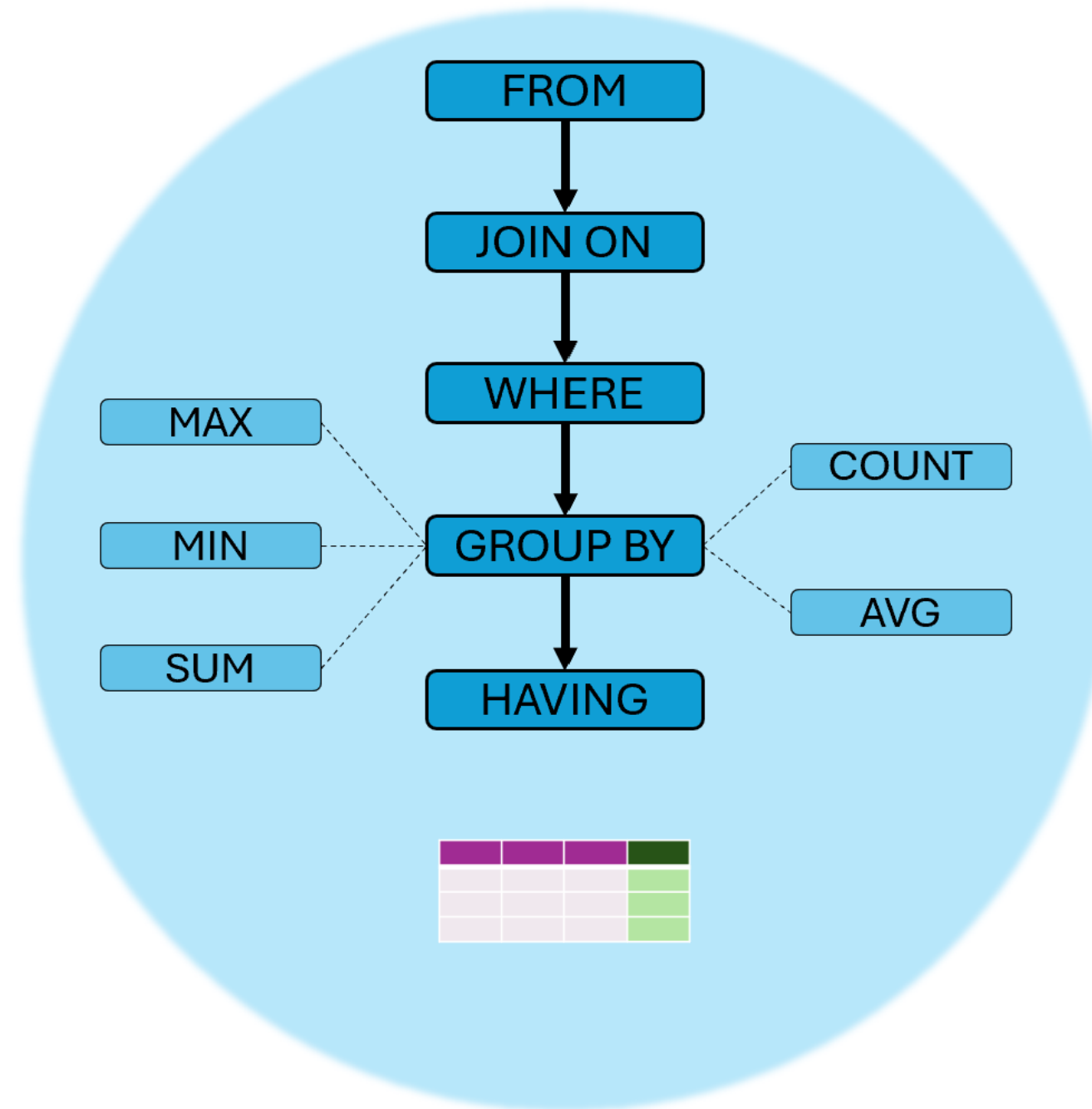
# Query execution order (3)



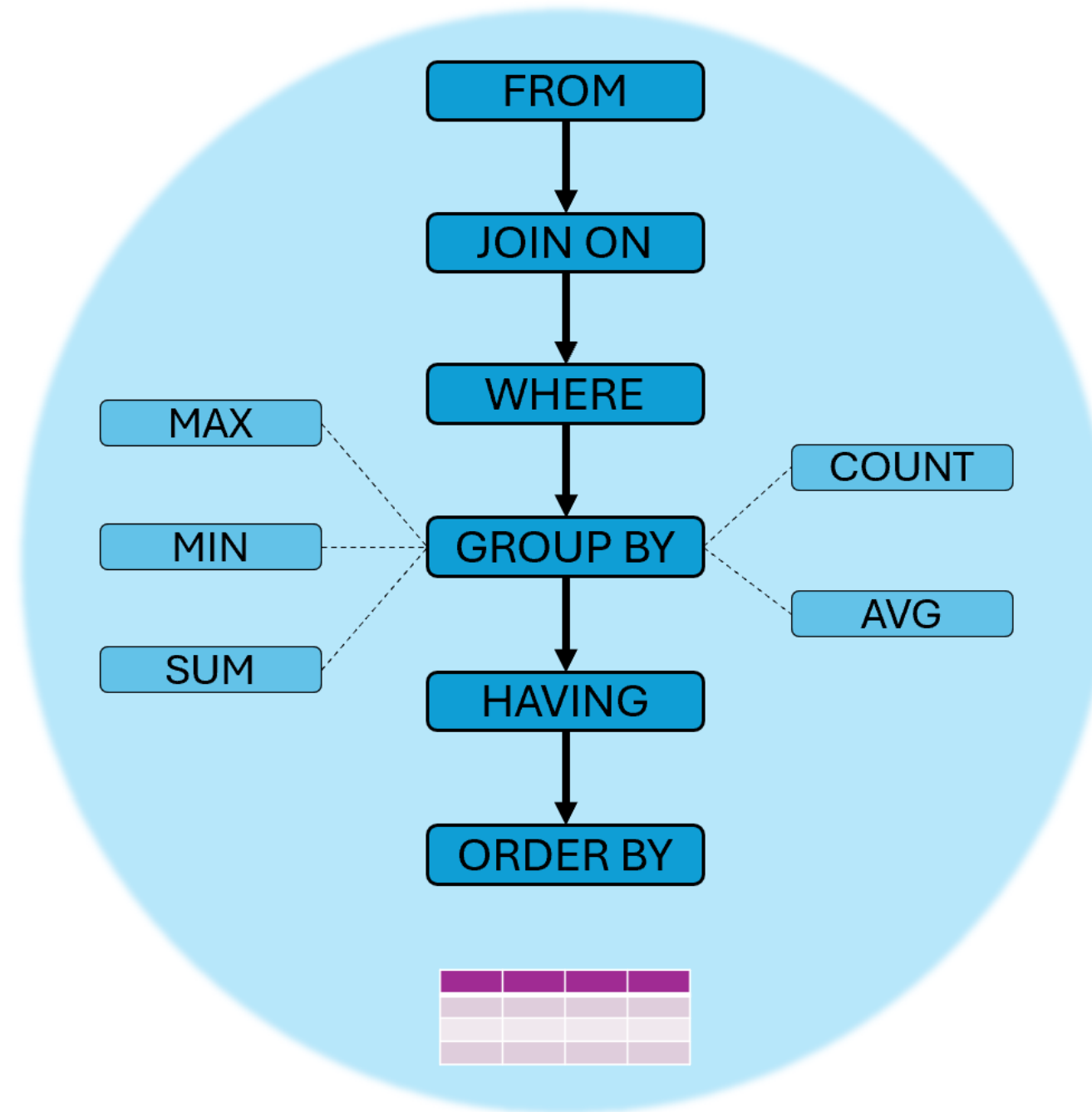
# Query execution order (4)



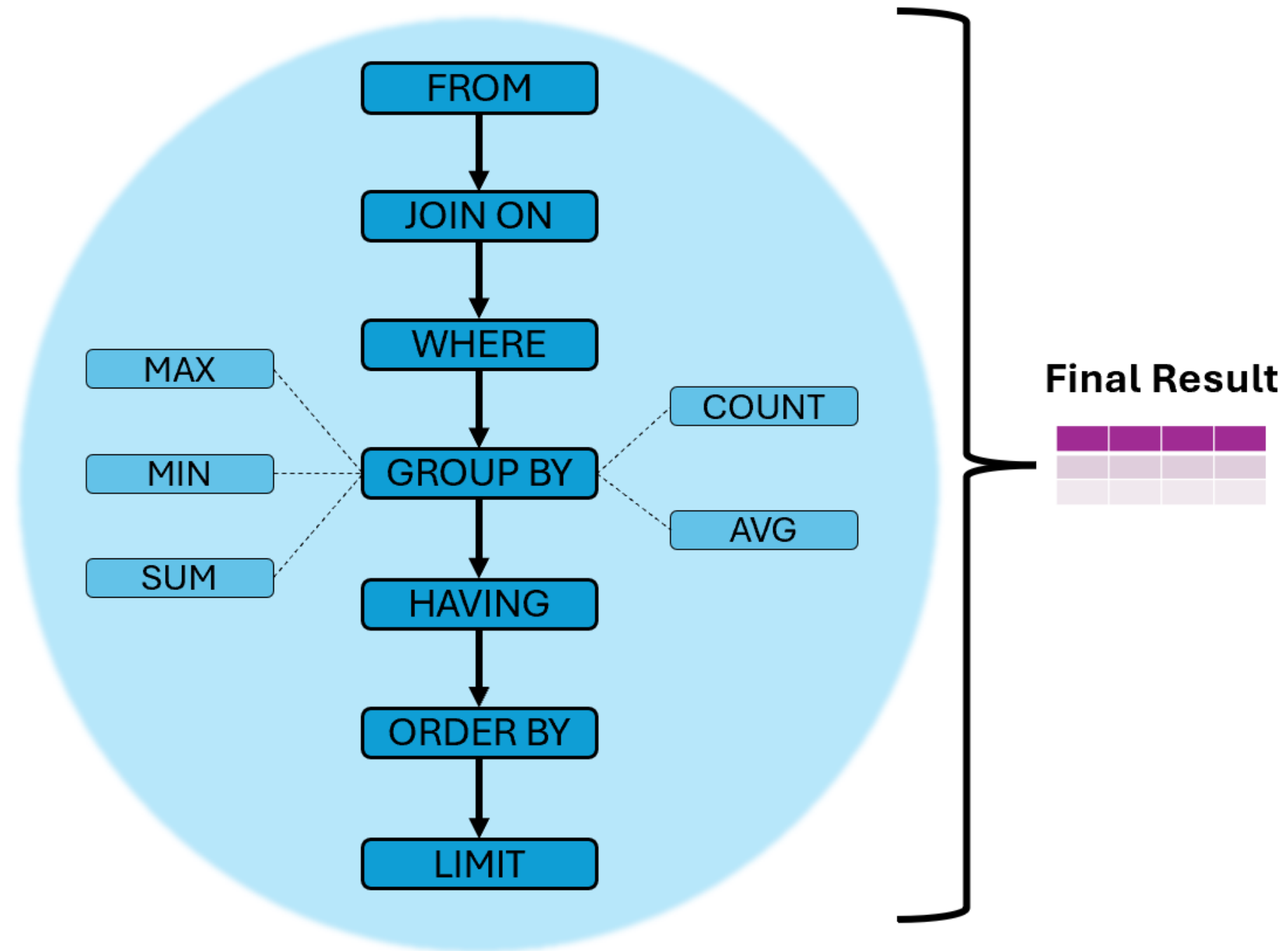
# Query execution order (5)



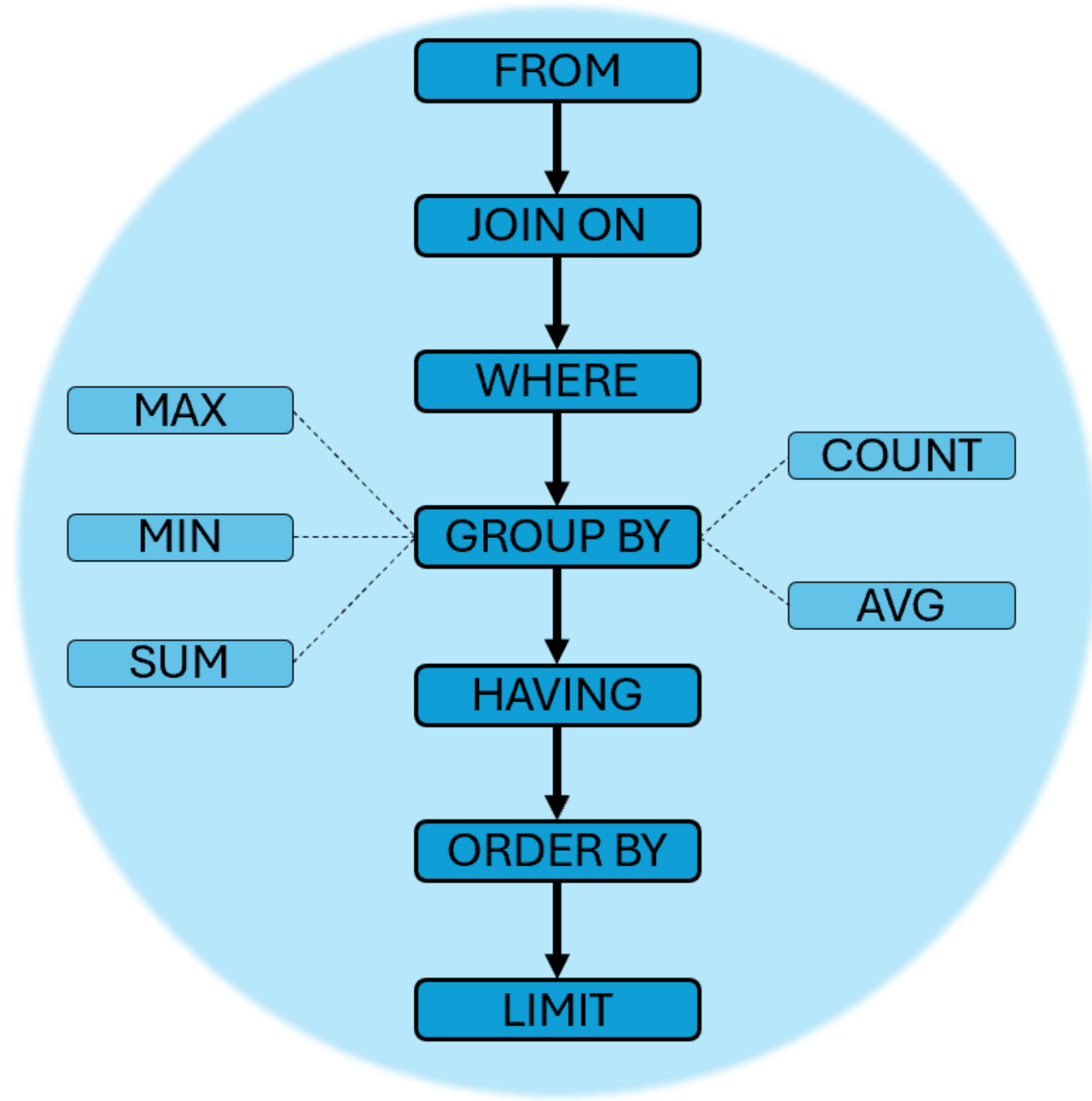
# Query execution order (6)



# Query execution order (7)



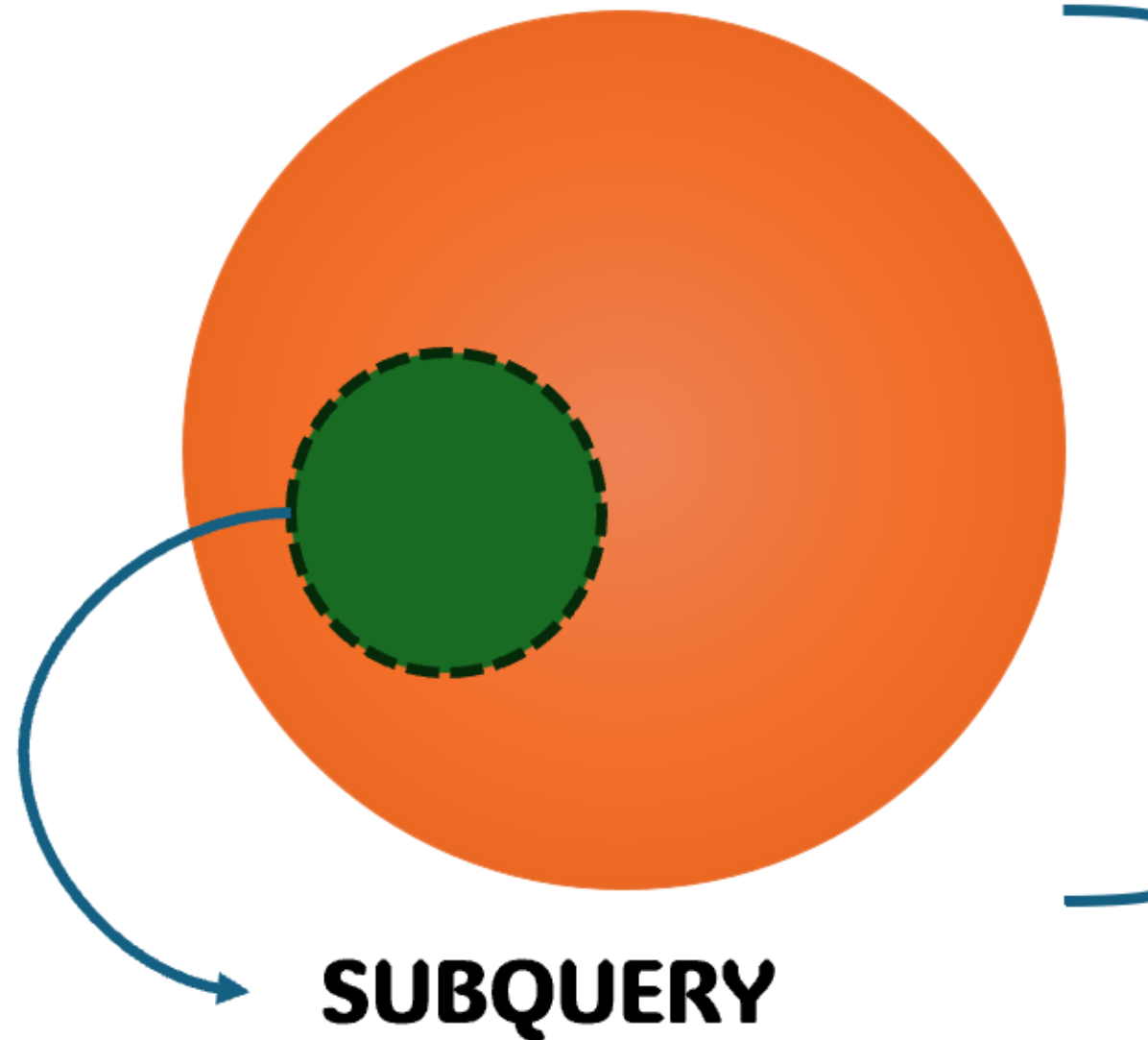
# Query execution order (8)



- **BEST PRACTICES:**
  - Avoid using SELECT \*; specify only necessary columns
  - Implement LIMIT filtering to reduce data volume
  - Use the WHERE clause early for row filtering and memory conservation
  - Employ GROUP BY with aggregations on narrowed datasets to optimize processing

# Subqueries

```
WITH customer_status AS (  
  SELECT c.customerid,  
         c.age,  
         c.tenure,  
         CASE  
           WHEN ch.customerid IS NOT NULL THEN 'Churned'  
           ELSE 'Active'  
         END AS status  
  FROM customers AS c  
  LEFT JOIN churn AS ch  
  ON c.customerid = ch.customerid  
  GROUP BY c.customerid, c.age, c.tenure, status  
)  
SELECT status,  
       COUNT(customerid) AS unique_customers,  
       -- Calculate averages  
       AVG(age) AS average_age,  
       AVG(tenure) AS average_tenure  
FROM customer_status  
WHERE customerid IN (SELECT customerid  
                     FROM customers  
                     WHERE estimatedsalary > 175000)  
  
GROUP BY status  
-- Filter data  
HAVING average_tenure > 2;
```

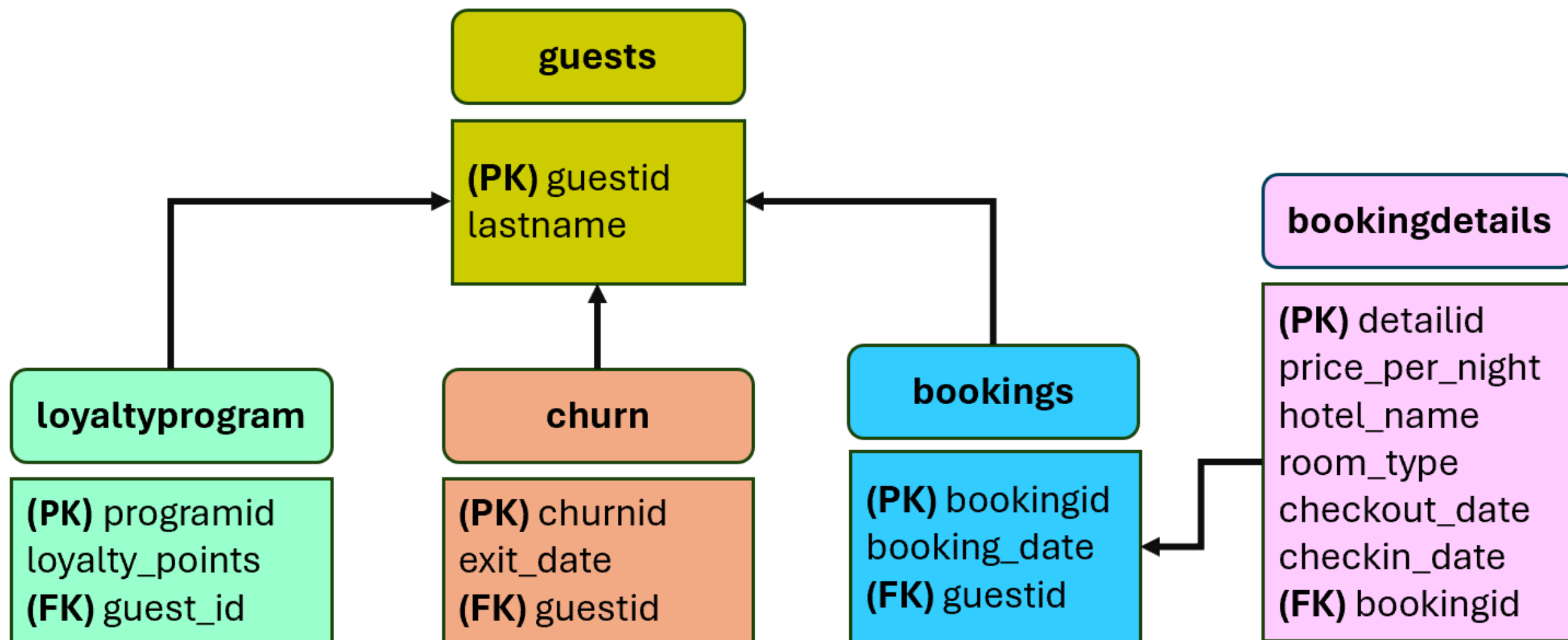


```
WITH customer_status AS (  
  SELECT c.customerid,  
         c.age,  
         c.tenure,  
         CASE  
           WHEN ch.customerid IS NOT NULL THEN 'Churned'  
           ELSE 'Active'  
         END AS status  
  FROM customers AS c  
  LEFT JOIN churn AS ch  
  ON c.customerid = ch.customerid  
  GROUP BY c.customerid, c.age, c.tenure, status  
)  
SELECT status,  
       -- Count customers  
       COUNT(customerid) AS unique_customers  
FROM customer_status  
WHERE customerid IN (SELECT customerid  
                     FROM customers  
                     WHERE estimatedsalary > 175000)  
  
-- Aggregate values  
GROUP BY status
```

**QUERY**

```
WITH customer_status AS (  
  SELECT c.customerid,  
         c.age,  
         c.tenure,  
         CASE  
           WHEN ch.customerid IS NOT NULL THEN 'Churned'  
           ELSE 'Active'  
         END AS status  
  FROM customers AS c  
  LEFT JOIN churn AS ch  
  ON c.customerid = ch.customerid  
  GROUP BY c.customerid, c.age, c.tenure, status  
)  
-- Extract attribute from CTE  
SELECT status  
FROM customer_status  
-- Filter results  
WHERE customerid IN (SELECT customerid  
                     FROM customers  
                     WHERE estimatedsalary > 175000);
```

# Subqueries



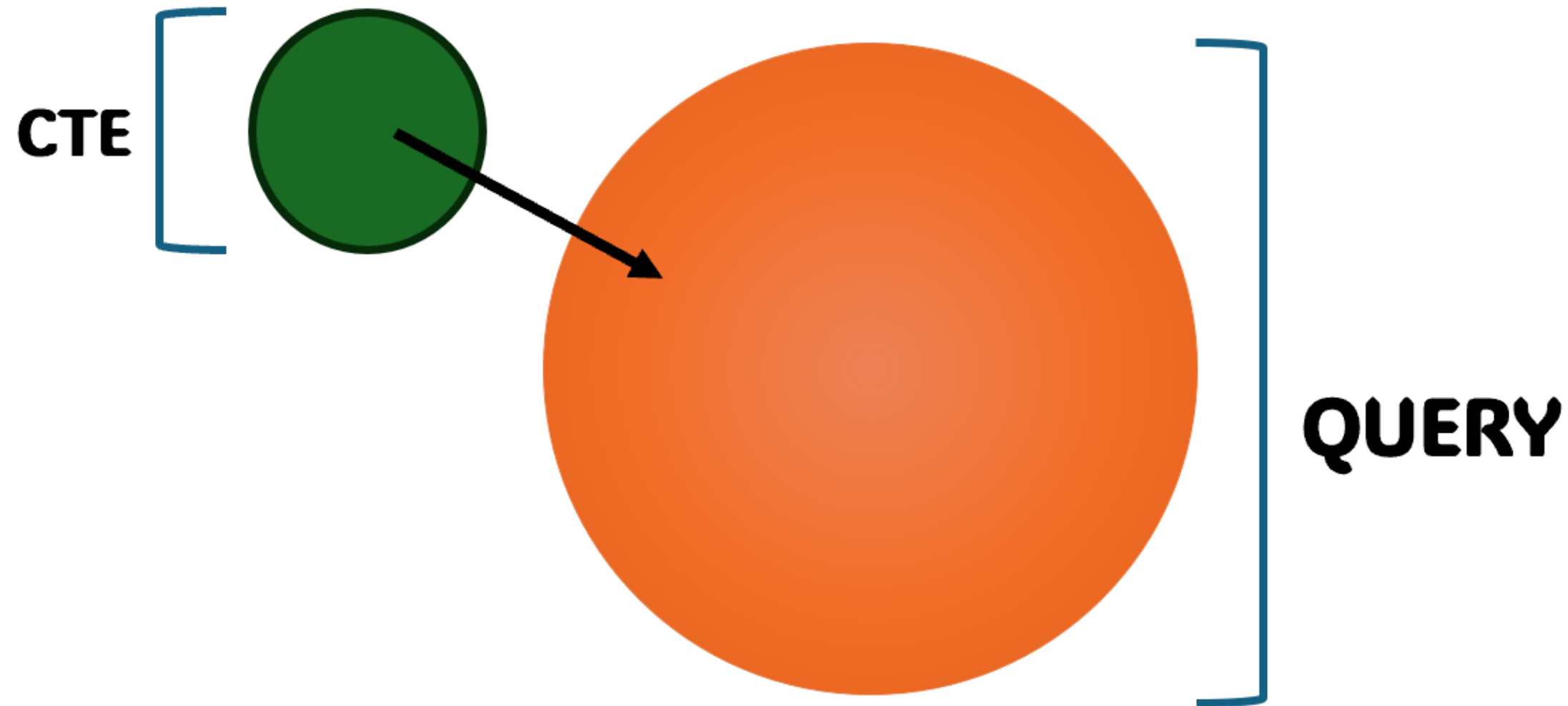


# Subqueries

- Query all guests that have more than 1000 loyalty points

```
SELECT *  
FROM guests  
WHERE id IN (SELECT guest_id  
              FROM loyalty_program  
              WHERE loyalty_points > 1000);
```

# Common table expressions



# Common table expressions

- Query the latest booking details

```
WITH latest_booking AS (  
    SELECT guest_id,  
           MAX(checkout_date) AS latest_checkout  
    FROM booking_details  
    GROUP BY guest_id  
)  
SELECT bd.*,  
       bd.checkout_date AS latest_booking_date  
FROM booking_details bd  
JOIN latest_booking lb  
    ON bd.guest_id = lb.guest_id  
    AND bd.checkout_date = lb.latest_checkout;
```

# CTEs and subqueries

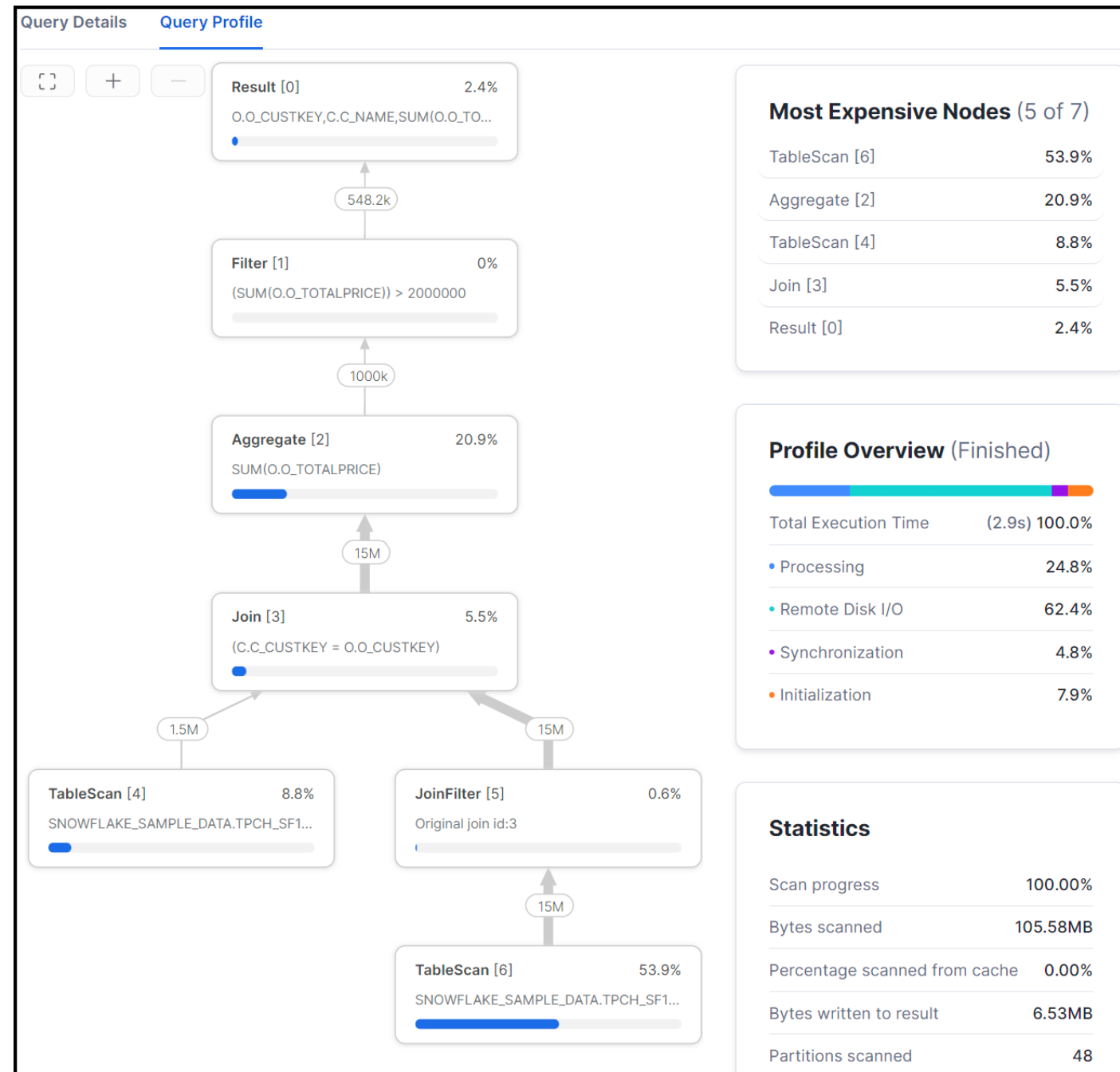
## CTEs

- **Pros**
  - Enhances readability for complex queries
  - Enables reusability in the same query
  - Improves organization of SQL queries
- **Cons**
  - Can introduce performance overhead
  - Limited to the scope of a single query

## Subqueries

- **Pros**
  - Simple and direct for single-use cases
  - Flexible in various parts of a SQL statement
- **Cons**
  - It can reduce readability with complexity
  - Potential performance issues with nested instances

# Visualizing query execution times



# Terminology and functions overview

- **Query Optimization**: Fine-tuning queries to maximize efficiency and performance
- **Subquery**: A smaller query inside a main query that helps focus on specific data
- **Common Table Expressions (CTEs)**: Temporary virtual table during a query
- **WITH .. AS** : SQL command to define a CTE
- **LIMIT** : SQL clause constraining the number of rows in query results
- **HAVING** : SQL clause used to filter data that aggregated with functions like SUM, MAX, etc
- **WHERE** : SQL clause used to filter rows before grouping, enhancing query efficiency

# Exemplary CTE and subquery template

- Subquery

```
SELECT *  
FROM table_name  
WHERE column_name IN (SELECT column_name  
                       FROM table_name  
                       WHERE column_name condition value);
```

- CTE

```
WITH latest_booking_dates AS (  
    SELECT column_name  
    FROM table_name)  
SELECT *  
FROM table_name a  
    JOIN other_table_name b  
    ON a.key_column = b.key_column;
```

# Let's practice!

INTRODUCTION TO DATA MODELING IN SNOWFLAKE



# Wrap-up

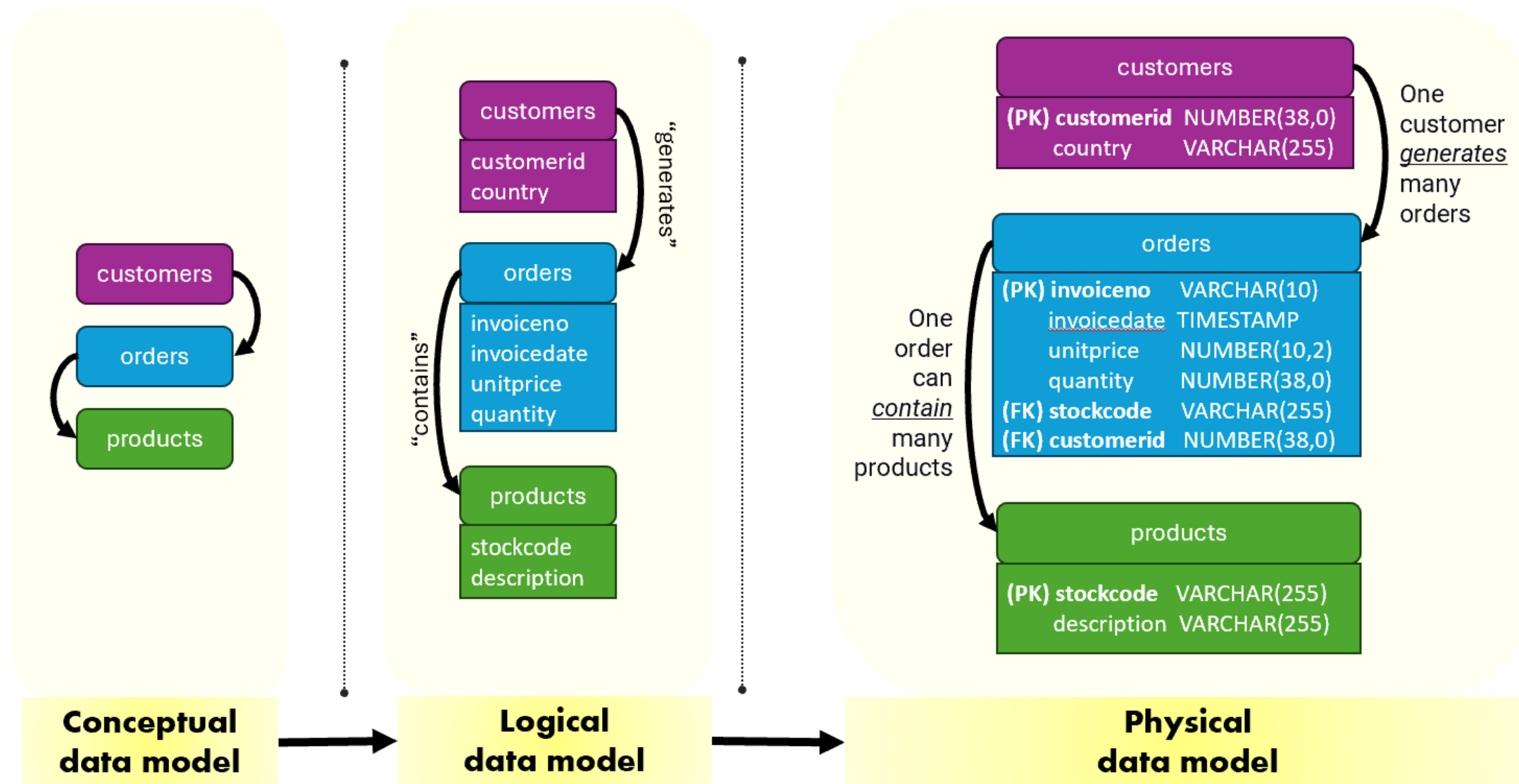
INTRODUCTION TO DATA MODELING IN SNOWFLAKE



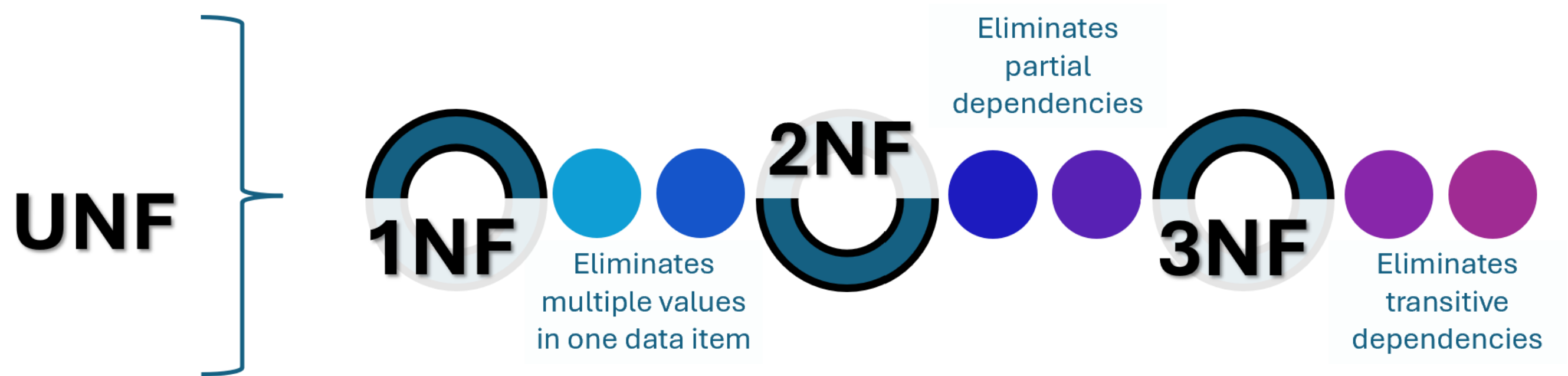
**Nuno Rocha**

Director of Engineering

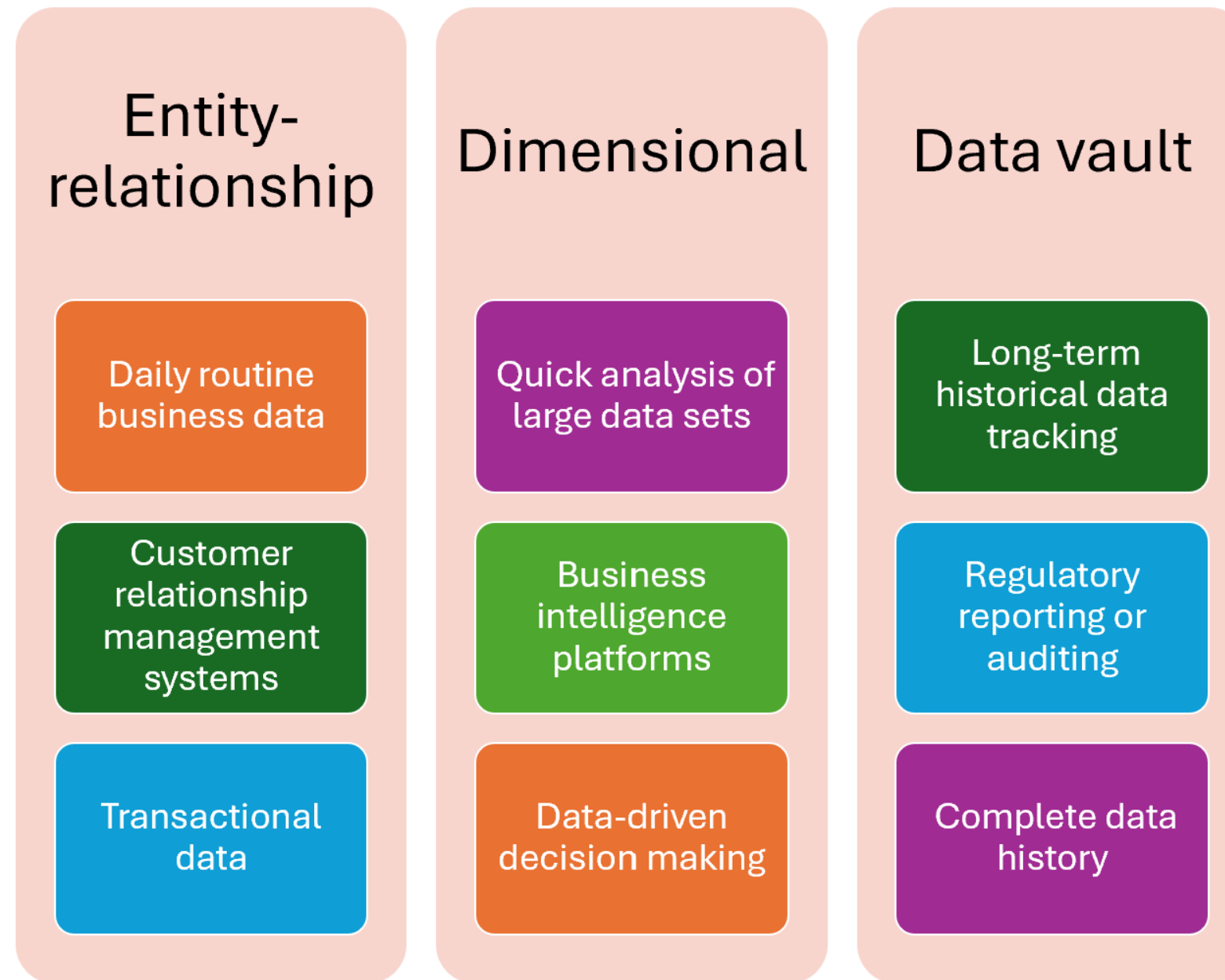
# Chapter 1 - Fundamentals of Data Modeling



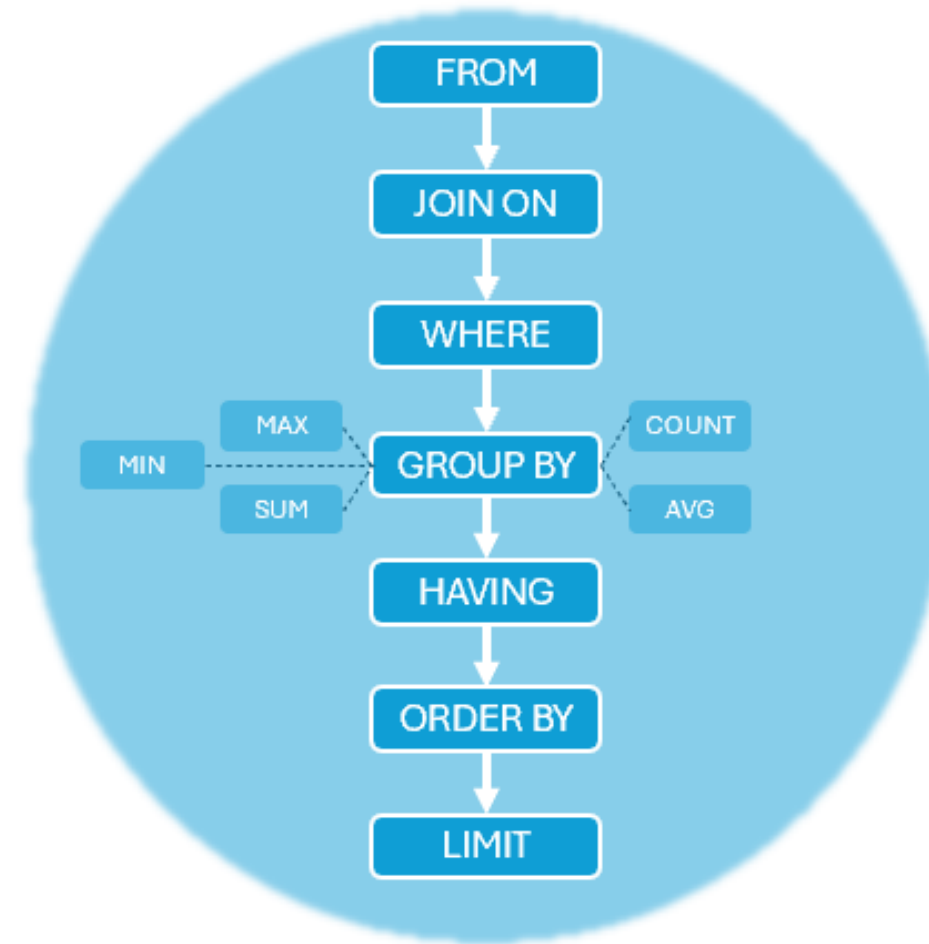
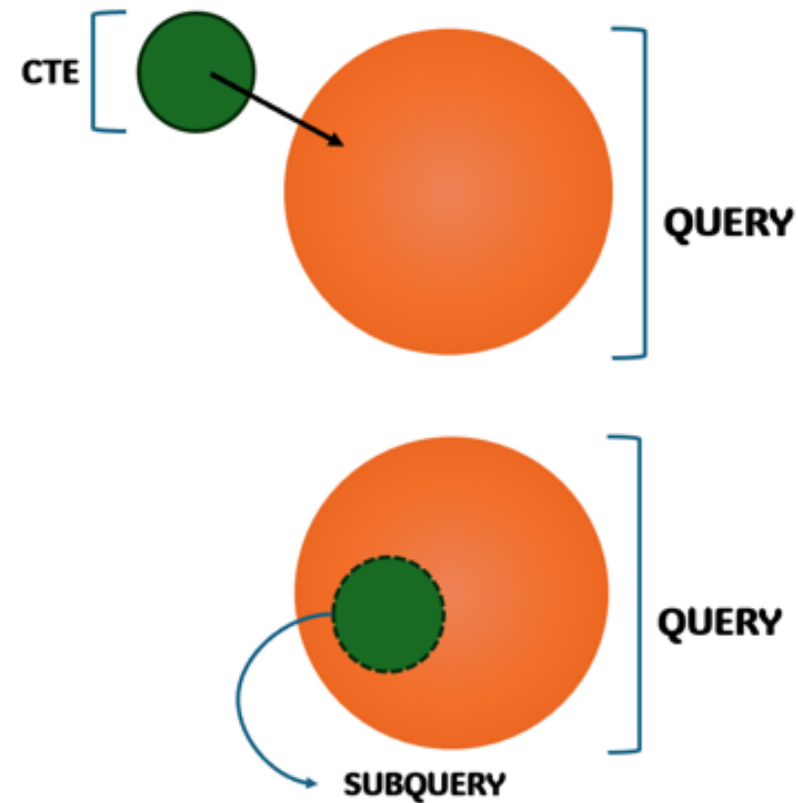
# Chapter 2 - Beginning of Relationships and Normalization



# Chapter 3 - Data Modeling Techniques



# Chapter 4 - Snowflake Components



# Congratulations!!

INTRODUCTION TO DATA MODELING IN SNOWFLAKE