Day 2 – SQL & Snowflake Foundations (Part 2)

---

1. SQL Essentials Continued: GROUP BY, ORDER BY

Concept Explanation

- **ORDER BY** sorts query results (ascending or descending).
- **GROUP BY** is used with aggregation functions (COUNT, SUM, AVG) to summarize data by groups.
- Commonly used functions: `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`.

Query: Sort customers by signup date

```
SELECT CUSTOMER_ID, CUSTOMER_NAME, SIGNUP_DATE
FROM CUSTOMERS
ORDER BY SIGNUP_DATE ASC;
```

Explanation: Returns customers sorted from earliest to latest signup date.

Query: Average order amount by customer

```
SELECT CUSTOMER_ID, AVG(ORDER_AMOUNT) AS AVG_SPEND
FROM ORDERS
GROUP BY CUSTOMER_ID
ORDER BY AVG_SPEND DESC;
```

Explanation: Groups orders by customer and shows which customers spend the most.

Query: Total sales per product

```
SELECT PRODUCT_ID, SUM(SALES_AMOUNT) AS TOTAL_SALES
FROM SALES
GROUP BY PRODUCT_ID
ORDER BY TOTAL_SALES DESC;
```

Explanation: Summarizes revenue by product and sorts highest-selling items first.

---

2. Joins and Aggregations (INNER, LEFT, RIGHT)

Concept Explanation

JOIN Types

- **INNER JOIN** → Returns only matching rows from both tables
- **LEFT JOIN** → Returns all rows from left table + matches from right
- **RIGHT JOIN** → Returns all rows from right table + matches from left

Used to combine customer, order, and product tables.

Query: INNER JOIN – Customers with their orders

```
SELECT c.CUSTOMER_ID, c.CUSTOMER_NAME, o.ORDER_ID, o.ORDER_AMOUNT
FROM CUSTOMERS c
INNER JOIN ORDERS o
    ON c.CUSTOMER_ID = o.CUSTOMER_ID;
```

Explanation: Returns only customers who have placed at least one order.

Query: LEFT JOIN – All customers even if they have no orders

```
SELECT c.CUSTOMER_ID, c.CUSTOMER_NAME, o.ORDER_ID
FROM CUSTOMERS c
LEFT JOIN ORDERS o
    ON c.CUSTOMER_ID = o.CUSTOMER_ID;
```

Explanation: Includes customers with **no orders**, showing `NULL` for missing order data.

Query: RIGHT JOIN – All orders even if customer info missing

```
SELECT c.CUSTOMER_NAME, o.ORDER_ID, o.ORDER_AMOUNT
FROM CUSTOMERS c
RIGHT JOIN ORDERS o
    ON c.CUSTOMER_ID = o.CUSTOMER_ID;
```

Explanation: Returns all orders, including those with missing/incomplete customer records.

Query: Join with aggregation – Total sales per customer

```
SELECT c.CUSTOMER_NAME, SUM(o.ORDER_AMOUNT) AS TOTAL_SPENT
```

```
FROM CUSTOMERS c
INNER JOIN ORDERS o
    ON c.CUSTOMER_ID = o.CUSTOMER_ID
GROUP BY c.CUSTOMER_NAME
ORDER BY TOTAL_SPENT DESC;
```

Explanation: Combines JOIN + GROUP BY to calculate each customer's total purchase value.

---

3. Hands-on: Customer/Product Dataset Queries

Concept Explanation

Hands-on practice typically involves combining customer, product, and order tables to produce insights such as:

- Best-selling products
- Top customers
- Order counts
- Revenue summaries

Query: Top 5 products by revenue

```
SELECT p.PRODUCT_NAME, SUM(s.SALES_AMOUNT) AS TOTAL_REVENUE
FROM SALES s
JOIN PRODUCTS p
    ON s.PRODUCT_ID = p.PRODUCT_ID
GROUP BY p.PRODUCT_NAME
ORDER BY TOTAL_REVENUE DESC
LIMIT 5;
```

Explanation: Finds the highest revenue-generating products.

Query: Count orders per customer

```
SELECT c.CUSTOMER_NAME, COUNT(o.ORDER_ID) AS TOTAL_ORDERS
FROM CUSTOMERS c
LEFT JOIN ORDERS o
    ON c.CUSTOMER_ID = o.CUSTOMER_ID
GROUP BY c.CUSTOMER_NAME
ORDER BY TOTAL_ORDERS DESC;
```

Explanation: Shows which customers order most frequently.

Query: Identify customers with no purchases

```
SELECT c.CUSTOMER_ID, c.CUSTOMER_NAME
FROM CUSTOMERS c
LEFT JOIN ORDERS o
    ON c.CUSTOMER_ID = o.CUSTOMER_ID
WHERE o.ORDER_ID IS NULL;
```

Explanation: Uses LEFT JOIN + `NULL` filter to find customers who never ordered.

---

4. Subqueries & Nesting Queries

Concept Explanation

Subqueries (nested queries) are queries inside other queries.
Used for:

- Filtering based on aggregated values
- Creating temporary result sets
- Pulling the "top" or "latest" record

Types include:

- **Scalar subqueries** (return one value)
- **IN subqueries**
- **Nested FROM subqueries**

Query: Customers with above-average order amounts

```
SELECT CUSTOMER_ID, ORDER_AMOUNT
FROM ORDERS
WHERE ORDER_AMOUNT >
      (SELECT AVG(ORDER_AMOUNT) FROM ORDERS);
```

Explanation: The inner query calculates the overall average; the outer query finds orders above that level.

Query: Products in the top 10% revenue bracket

```
SELECT PRODUCT_ID, TOTAL_SALES
FROM (
    SELECT PRODUCT_ID,
           SUM(SALES_AMOUNT) AS TOTAL_SALES
    FROM SALES
```

```
    GROUP BY PRODUCT_ID
) AS PRODUCT_SUMMARY
WHERE TOTAL_SALES >
      (SELECT AVG(TOTAL_SALES) FROM (
            SELECT PRODUCT_ID, SUM(SALES_AMOUNT) AS TOTAL_SALES
            FROM SALES
            GROUP BY PRODUCT_ID
      ));
```

Explanation: Nested subqueries summarize product revenue and filter those performing above average.

Query: Latest order from each customer (correlated subquery)

```
SELECT *
FROM ORDERS o
WHERE ORDER_DATE = (
    SELECT MAX(ORDER_DATE)
    FROM ORDERS
    WHERE CUSTOMER_ID = o.CUSTOMER_ID
);
```

Explanation: For each customer, the subquery finds their latest order date.