TechStore Analytics Hub

Data Analysis Case Study Using PostgreSQL and Snowflake

1. Write a query to calculate the running sum of sales quantity for each product.

```sql
WITH running_sales AS (
    SELECT
        sale_id,
        product_id,
        sale_date,
        quantity,
        SUM(quantity) OVER (PARTITION BY product_id ORDER BY sale_date) AS running_sum
    FROM factsales
)
SELECT
    sale_id,
    product_id,
    sale_date,
    quantity,
    running_sum
FROM running_sales
ORDER BY sale_date
```

- The output shows the **cumulative quantity sold** for each product, calculated using a **running sum**. For example: Product 2 (on 2024-03-27) had a quantity of 3 sold, giving a running sum of 3.On the next sale (2024-03-29) for Product 2, only 1 unit was sold, making the running sum 4.

Table | Chart — 100 rows — 18ms

	SALE_ID	PRODUCT_ID	SALE_DATE	QUANTITY	RUNNING_SUM
1	59	2	2024-03-27	3	3
2	6	2	2024-03-29	1	4
3	33	12	2024-04-01	1	1
4	22	5	2024-04-01	3	3
5	72	2	2024-04-02	2	6
6	45	13	2024-04-07	3	3
7	100	13	2024-04-09	1	4

2. Write a query to calculate the Month-on-Month (MoM) percentage change in sales quantity for each product.

- This output shows **the MoM percentage change** in product sales, highlighting fluctuations in sales from month to month. For example, Product 1 saw a 50% increase from June to July, while Product 2 experienced a 200% increase from April to May.

```
WITH monthly_sales AS (
    SELECT
        product_id,
        TO_VARCHAR(sale_date, 'YYYY-MM') AS year_month,
        SUM(quantity) AS monthly_sales
    FROM factsales
    GROUP BY product_id, year_month
)
SELECT
    product_id,
    year_month,
    monthly_sales,
    (monthly_sales - LAG(monthly_sales) OVER (PARTITION BY product_id ORDER BY year_month)) * 100.0 /
    LAG(monthly_sales) OVER (PARTITION BY product_id ORDER BY year_month) AS mom_percentage_change
FROM monthly_sales
ORDER BY product_id, year_month
```

	PRODUCT_ID	YEAR_MONTH	MONTHLY_SALES	MOM_PERCENTAGE_CHANGE
1	1	2024-06	2	null
2	1	2024-07	3	50.000000
3	1	2024-10	1	-66.666667
4	2	2024-03	4	null
5	2	2024-04	2	-50.000000
6	2	2024-05	6	200.000000
7	2	2024-10	3	-50.000000

3. Create a query that calculates the rolling sum of sales revenue over the past 3 months.

```sql
WITH monthly_revenue AS (
    SELECT
        TO_VARCHAR(s.sale_date, 'YYYY-MM') AS year_month,
        SUM(s.quantity * p.price * (1 - s.discount)) AS monthly_revenue
    FROM factsales s
    JOIN products p ON s.product_id = p.product_id
    GROUP BY year_month
)
SELECT
    year_month,
    monthly_revenue,
    SUM(monthly_revenue) OVER (
        ORDER BY year_month
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS rolling_3_month_revenue
FROM monthly_revenue
ORDER BY year_month
```

	YEAR_MONTH	MONTHLY_REVENUE	ROLLING_3_MONTH_REVENUE
1	2024-03	3083.57	3083.57
2	2024-04	10035.48	13119.05
3	2024-05	14828.00	27947.05
4	2024-06	9989.68	34853.16
5	2024-07	15678.15	40495.83
6	2024-08	5750.73	31418.56
7	2024-09	9431.98	30860.86

- This output shows the **monthly revenue** along with the **rolling 3-month revenue** for each month. For example, the **3-month rolling revenue** for **June 2024** is **34,853.16**, which is the sum of revenues from April to June 2024.

4. Create a query to calculate the total sales for each product by month.

```sql
WITH monthly_sales AS (
    SELECT
        s.product_id,
        TO_VARCHAR(s.sale_date, 'YYYY-MM') AS year_month,
        SUM(s.quantity * p.price * (1 - s.discount)) AS total_sales
    FROM factsales s
    JOIN products p ON s.product_id = p.product_id  -- Correct join between factsales and products
    GROUP BY s.product_id, year_month
)
SELECT
    monthly_sales.product_id,
    year_month,
    total_sales,
    SUM(total_sales) OVER (PARTITION BY monthly_sales.product_id ORDER BY year_month) AS cumulative_sales
FROM monthly_sales
ORDER BY monthly_sales.product_id, year_month
```

- This output shows **total sales** and **cumulative sales** for each product across different months. The **cumulative sales** column represents the running total of **total sales** for each product. For example, the **cumulative sales** for **Product 1** in **July 2024** is **4235.76**, which is the sum of sales from June and July.

	PRODUCT_ID	YEAR_MONTH	TOTAL_SALES	CUMULATIVE_SALES
1	1	2024-06	1598.40	1598.40
2	1	2024-07	2637.36	4235.76
3	1	2024-10	839.16	5074.92
4	2	2024-03	3083.57	3083.57
5	2	2024-04	1798.00	4881.57
6	2	2024-05	5070.36	9951.93
7	2	2024-10	2589.12	12541.05

5. Write a query that shows the change in sales quantity compared to the previous sale for each customer.

```sql
WITH sales_changes AS (
    SELECT
        customer_id,
        sale_date,
        quantity,
        LAG(quantity) OVER (PARTITION BY customer_id ORDER BY sale_date) AS previous_quantity
    FROM factsales
)
SELECT
    customer_id,
    sale_date,
    quantity,
    quantity - previous_quantity AS quantity_change
FROM sales_changes
ORDER BY customer_id, sale_date
```

	CUSTOMER_ID	SALE_DATE	QUANTITY	QUANTITY_CHANGE
1	1	2024-04-25	2	null
2	1	2024-06-24	3	1
3	1	2024-07-27	2	-1
4	1	2024-12-16	3	1
5	2	2024-05-09	3	null
6	2	2024-12-28	3	0
7	2	2025-03-14	1	-2

- This output shows the **quantity change** between consecutive sales for each customer. The **Quantity Change** is calculated by comparing the current sale's quantity to the previous sale. For example, **Customer 1** increased their quantity from **2** to **3** between April and June 2024, resulting in a **change of +1**, while the next sale in July shows a **decrease of -1**

6. Calculate Discount Percentage Change for Each Product.

- This output shows the **percentage change in discount** for each product between consecutive sales. For example, **Product 1** had a **100% decrease** in discount from **0.20** to **0.00** between June and July 2024, while **Product 2** also shows a **100% decrease** in discount from **0.19** to **0.00** between March 2024 and April 2024.

```sql
WITH discount_changes AS (
    SELECT
        product_id,
        sale_date,
        discount,
        LAG(discount) OVER (PARTITION BY product_id ORDER BY sale_date) AS previous_discount
    FROM factsales
)
SELECT
    product_id,
    sale_date,
    discount,
    CASE
        WHEN previous_discount = 0 THEN NULL
        ELSE ROUND((discount - previous_discount) * 100.0 / previous_discount, 2)
    END AS discount_percentage_change
FROM discount_changes
ORDER BY product_id, sale_date;
```

	PRODUCT_ID	SALE_DATE	DISCOUNT	DISCOUNT_PERCENTAGE_CHANGE
1	1	2024-06-15	0.20	null
2	1	2024-07-06	0.00	-100.00
3	1	2024-07-07	0.18	null
4	1	2024-10-31	0.16	-11.11
5	2	2024-03-27	0.19	null
6	2	2024-03-29	0.00	-100.00
7	2	2024-04-02	0.00	null

7. Calculate Cumulative Sales Revenue for Top 5 Products

```sql
WITH product_sales AS (
    SELECT
        s.product_id,
        SUM(quantity * p.price * (1 - s.discount)) AS total_revenue
    FROM factsales s
    JOIN products p ON s.product_id = p.product_id
    GROUP BY s.product_id
),
ranked_sales AS (
    SELECT
        product_id,
        total_revenue,
        RANK() OVER (ORDER BY total_revenue DESC) AS revenue_rank
    FROM product_sales
)
SELECT
    ranked_sales.product_id,
    total_revenue
FROM ranked_sales
WHERE revenue_rank <= 5
ORDER BY total_revenue DESC
```

	PRODUCT_ID	TOTAL_REVENUE
1	2	19769.01
2	15	18332.77
3	17	15381.45
4	11	11068.92
5	8	9625.12

- This output shows the **total revenue** generated for each product. For example, **Product 2** has generated a total revenue of **19769.01**, while **Product 8** has generated **9625.12** in total revenue. This allows for comparison of revenue across different products.

8. Calculate Previous and Next Sale Price for Each Product.

- This output displays the **previous and next prices** for each product based on the sale date. For example, **Product 1** had a price of **999** on **2024-06-15**, with no previous price available (as it's the first sale), and the next price is **999** on **2024-07-06**.

```sql
WITH price_changes AS (
    SELECT
        s.product_id,
        s.sale_date,
        p.price,
        LAG(p.price) OVER (PARTITION BY s.product_id ORDER BY s.sale_date) AS previous_price,
        LEAD(p.price) OVER (PARTITION BY s.product_id ORDER BY s.sale_date) AS next_price
    FROM factsales s
    JOIN products p ON s.product_id = p.product_id
)
SELECT
    product_id,
    sale_date,
    price,
    previous_price,
    next_price
FROM price_changes
ORDER BY product_id, sale_date;
```

	PRODUCT_ID	SALE_DATE	PRICE	PREVIOUS_PRICE	NEXT_PRICE
1	1	2024-06-15	999	null	999
2	1	2024-07-06	999	999	999
3	1	2024-07-07	999	999	999
4	1	2024-10-31	999	999	null
5	2	2024-03-27	899	null	899
6	2	2024-03-29	899	899	899
7	2	2024-04-02	899	899	899

9. Calculate Total Quantity Sold for Each Product in a Rolling Window.

```sql
WITH product_sales AS (
    SELECT
        product_id,
        sale_date,
        quantity,
        ROW_NUMBER() OVER (PARTITION BY product_id ORDER BY sale_date) AS sale_rank
    FROM factsales
)
SELECT
    product_id,
    sale_date,
    quantity,
    SUM(quantity) OVER (
        PARTITION BY product_id
        ORDER BY sale_date
        ROWS BETWEEN 5 PRECEDING AND CURRENT ROW
    ) AS rolling_sales_quantity
FROM product_sales
ORDER BY product_id, sale_date;
```

- This output shows the **rolling sales quantity** for each product. The **rolling sales quantity** is the cumulative total of quantities sold over a specific window. For example, **Product 1** had a rolling sales quantity of **6** by **2024-10-31**, which includes sales from previous dates. Similarly, **Product 2**'s rolling sales quantity reaches **6** by **2024-04-02**.

	PRODUCT_ID	SALE_DATE	QUANTITY	ROLLING_SALES_QUANTITY
1	1	2024-06-15	2	2
2	1	2024-07-06	1	3
3	1	2024-07-07	2	5
4	1	2024-10-31	1	6
5	2	2024-03-27	3	3
6	2	2024-03-29	1	4
7	2	2024-04-02	2	6

10. Calculate Sales Growth for Each Product Over 6 Months.

This output shows the **monthly sales** for each product and provides a framework to calculate **sales growth** over 6 months. The **Sales Growth** field is currently empty (null) but would typically show the percentage change in **monthly sales** compared to the previous month. For example:
iPhone 12 shows varying sales figures for **June, July, and October 2024**, and **Galaxy S21** shows sales data for **March to October 2024**.

```
WITH monthly_sales AS (
    SELECT
        s.product_id,
        TO_VARCHAR(s.sale_date, 'YYYY-MM') AS year_month,
        SUM(s.quantity * p.price * (1 - s.discount)) AS monthly_sales
    FROM factsales s
    JOIN products p ON s.product_id = p.product_id
    GROUP BY s.product_id, year_month
)
SELECT
    ms.product_id,
    p.product_name,
    ms.year_month,
    ms.monthly_sales,
    (ms.monthly_sales - LAG(ms.monthly_sales, 6)
    OVER (PARTITION BY ms.product_id ORDER BY ms.year_month)) * 100.0 / LAG(ms.monthly_sales, 6)
    OVER (PARTITION BY ms.product_id ORDER BY ms.year_month) AS sales_growth
FROM monthly_sales ms
JOIN products p ON ms.product_id = p.product_id
ORDER BY ms.product_id, ms.year_month;
```

	PRODUCT_ID	PRODUCT_NAME	YEAR_MONTH	MONTHLY_SALES	SALES_GROWTH
1	1	iPhone 12	2024-06	1598.40	null
2	1	iPhone 12	2024-07	2637.36	null
3	1	iPhone 12	2024-10	839.16	null
4	2	Galaxy S21	2024-03	3083.57	null
5	2	Galaxy S21	2024-04	1798.00	null
6	2	Galaxy S21	2024-05	5070.36	null
7	2	Galaxy S21	2024-10	2589.12	null

Business Recommendations :

To improve sales, focus on optimizing **discount strategies** for products with significant fluctuations in sales. Utilize **rolling sales data** for better **inventory management** and **sales forecasting**, ensuring high-demand products are well-stocked. Target marketing efforts towards **top-performing products** and analyze **sales growth trends** to identify opportunities for promotions. Consider **dynamic pricing** to adjust for market demand, and explore **customer retention** programs based on **sales behavior** insights.