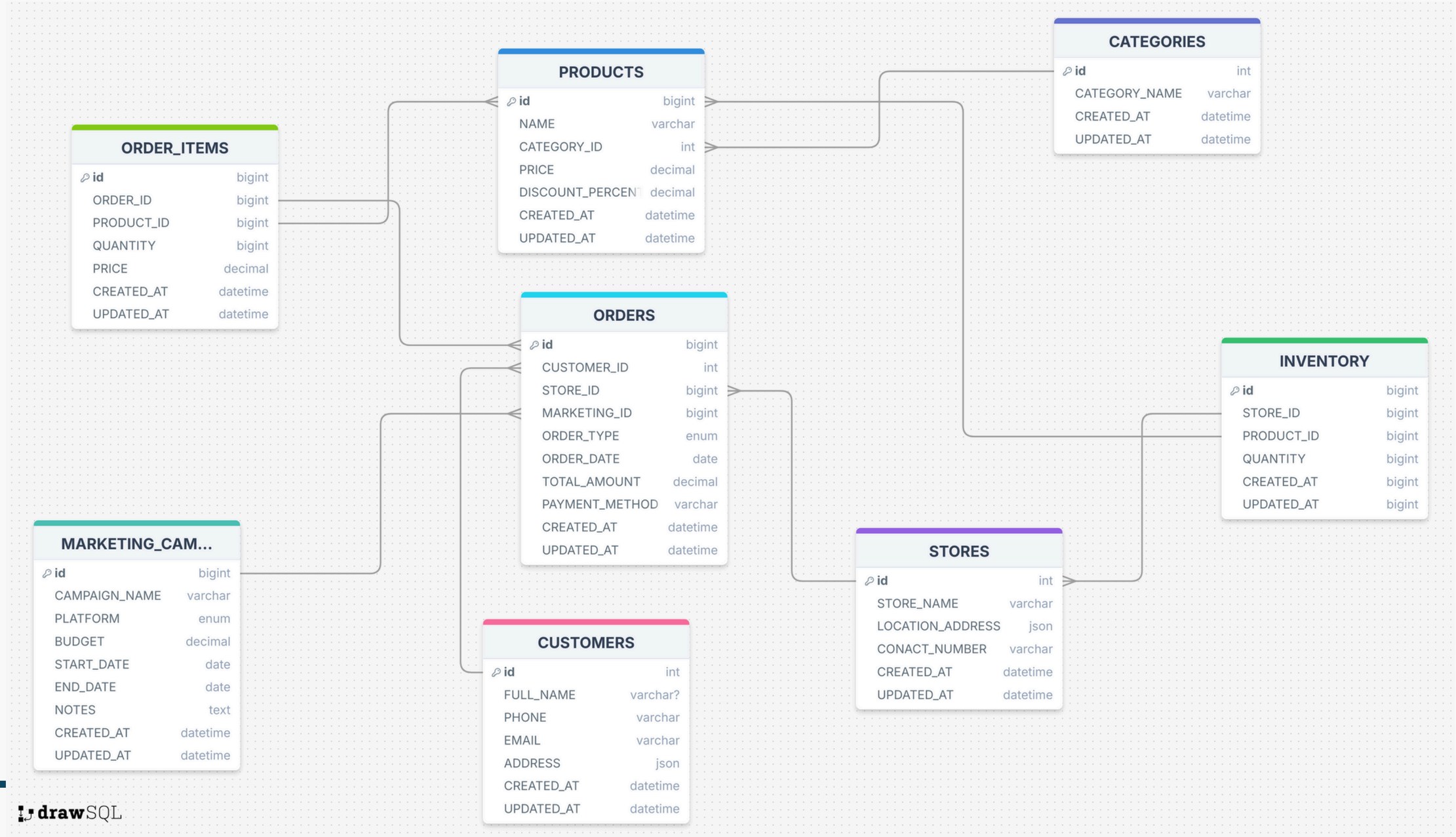
PRESENTATION ON

SK MART RETAIL CASE STUDY

USING SQL

Prepared by MD.KAMRUL ISLAM

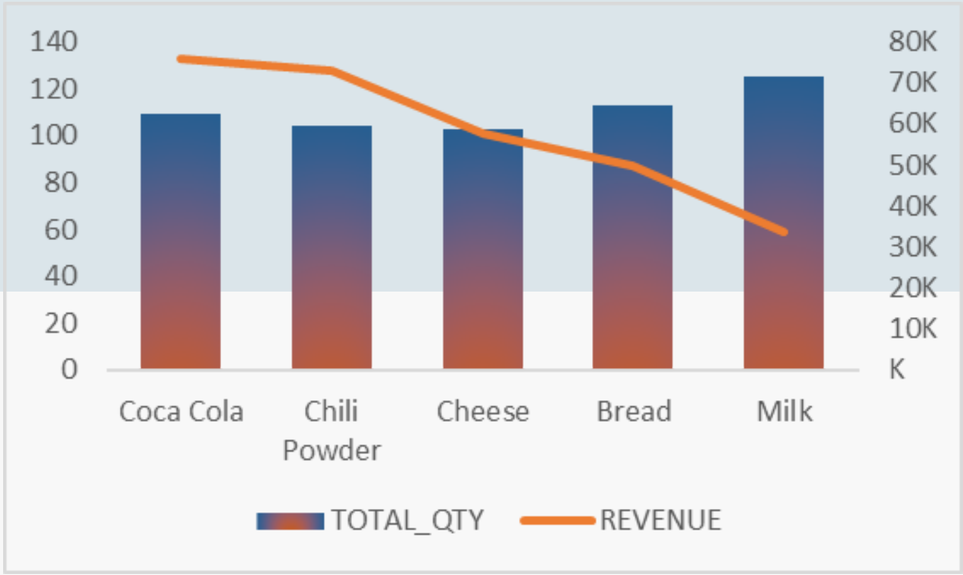
Er Diagram:


CATEGORIES
id int
CATEGORY_NAME varchar
CREATED_AT datetime
UPDATED_AT datetime
PRODUCTS
id bigint
NAME varchar
CATEGORY_ID int
PRICE decimal
DISCOUNT_PERCEN decimal
CREATED_AT datetime
UPDATED_AT datetime
ORDER_ITEMS
id bigint
ORDER_ID bigint
PRODUCT_ID bigint
QUANTITY bigint
PRICE decimal
CREATED_AT datetime
UPDATED_AT datetime
ORDERS
id bigint
CUSTOMER_ID int
STORE_ID bigint
MARKETING_ID bigint
ORDER_TYPE enum
ORDER_DATE date
TOTAL_AMOUNT decimal
PAYMENT_METHOD varchar
CREATED_AT datetime
UPDATED_AT datetime
INVENTORY
id bigint
STORE_ID bigint
PRODUCT_ID bigint
QUANTITY bigint
CREATED_AT bigint
UPDATED_AT bigint
MARKETING_CAM...
id bigint
CAMPAIGN_NAME varchar
PLATFORM enum
BUDGET decimal
START_DATE date
END_DATE date
NOTES text
CREATED_AT datetime
UPDATED_AT datetime
STORES
id int
STORE_NAME varchar
LOCATION_ADDRESS json
CONACT_NUMBER varchar
CREATED_AT datetime
UPDATED_AT datetime
CUSTOMERS
id int
FULL_NAME varchar?
PHONE varchar
EMAIL varchar
ADDRESS json
CREATED_AT datetime
UPDATED_AT datetime
drawSQL

1. What are the top 5 best-selling products by quantity and revenue?

Query:

```sql
SELECT
    P.NAME,
    SUM(OI.QUANTITY) AS TOTAL_QTY,
    SUM(OI.PRICE*OI.QUANTITY) AS REVENUE
FROM PRODUCTS AS P
LEFT JOIN ORDER_ITEMS AS OI
ON P.ID = OI.PRODUCT_ID
GROUP BY 1
ORDER BY TOTAL_QTY DESC, REVENUE DESC
LIMIT 5;
```



Output:

NAME	TOTAL_QTY	REVENUE
Milk	125	33610.44
Bread	113	50019.13
Coca Cola	109	75729.20
Chili Powder	104	73182.72
Cheese	103	57705.00

- Milk ranks 1st in quantity sold but has lowest revenue among the top 5 — likely a low-priced product.
- Coca Cola ranks 3rd in quantity but 1st in revenue — high price or profit margin.
- Chili Powder and Cheese have lower quantities but generate high revenue, suggesting higher unit prices.
- Bread strikes a balance — decent quantity and good revenue.

2.Which customers placed the most orders?

Query:

```sql
SELECT
    C.FULL_NAME,
    COUNT(O.ID) AS TOTAL_ORDERS,
    RANK() OVER( ORDER BY COUNT(O.ID) DESC) AS RANK_ORDERS
FROM CUSTOMERS AS C
LEFT JOIN ORDERS AS O
ON C.ID = O.CUSTOMER_ID
GROUP BY 1
LIMIT 2;
```

Output:

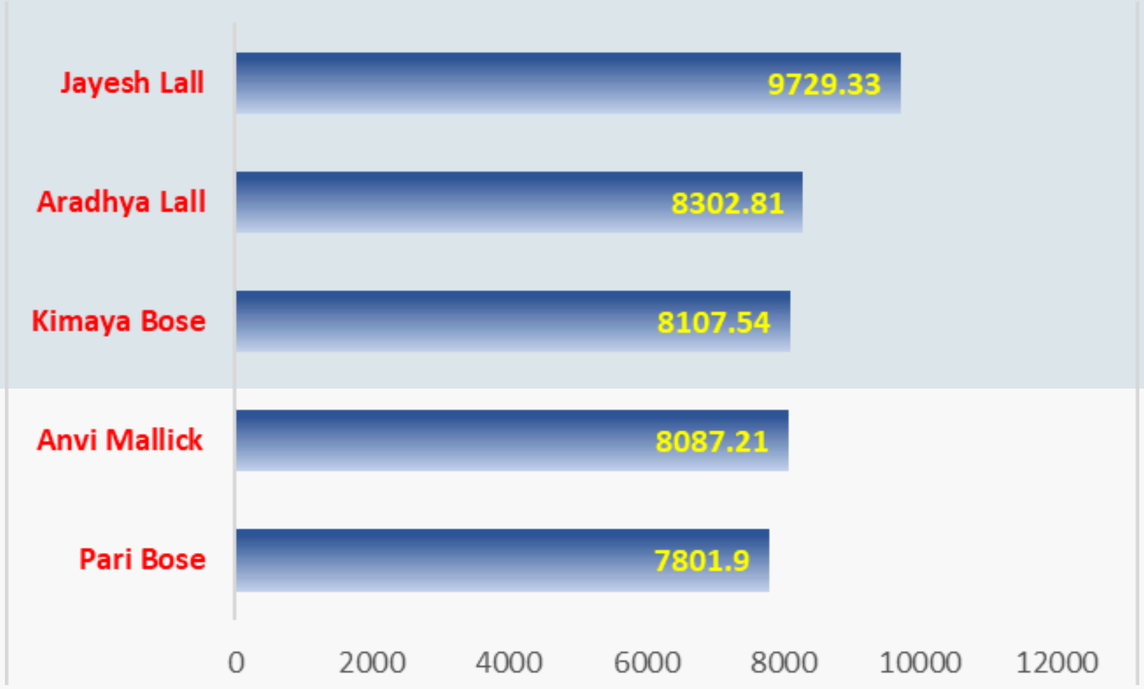
FULL_NAME	TOTAL_ORDERS	RANK_ORDERS
Jayesh Lall	9	1
Kimaya Bose	9	1

- Jayesh Lall - Placed 9 orders (Rank: 1)
- Kimaya Bose - Also placed 9 orders (Rank: 1)

3. Who are the top customers based on total spending?

Query:

```sql
SELECT
    C.FULL_NAME,
    SUM(TOTAL_AMOUNT) AS TOTAL_SPEND
FROM CUSTOMERS AS C
LEFT JOIN ORDERS AS O
ON C.ID = O.CUSTOMER_ID
GROUP BY 1
ORDER BY 2 DESC
LIMIT 5;
```



Output:

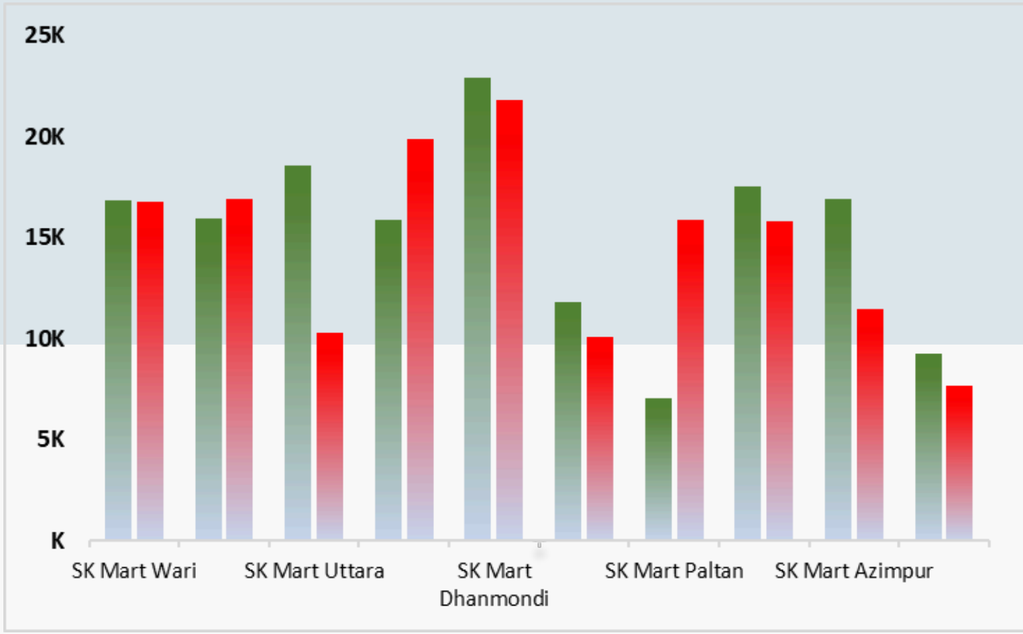
FULL_NAME	TOTAL_SPEND
Jayesh Lall	9729.33
Aradhya Lall	8302.81
Kimaya Bose	8107.54
Anvi Mallick	8087.21
Pari Bose	7801.90

- Jayesh Lall is the highest-spending customer, with nearly $1,400 more than the second-highest (Aradhya Lall).
- The top 5 are relatively close in spending (difference between #2 and #5 is just $500).
- Kimaya Bose appears again (from previous analysis), confirming they are both a frequent buyer (9 orders) and a high-value customer.

4. Compare online vs. offline sales for each store.

Query:

```sql
SELECT
    s.id AS store_id,
    s.store_name,
    SUM(CASE WHEN o.order_type = 'online' THEN o.total_amount ELSE 0 END) A
    SUM(CASE WHEN o.order_type = 'offline' THEN o.total_amount ELSE 0 END)
FROM stores s
JOIN orders o
ON s.id = o.store_id
GROUP BY s.id, s.store_name
ORDER BY S.ID
```



- Offline Dominance:
- 4/6 stores (Tejgaon, Bardhara, Dhammond, Wari) have higher offline sales.
- Bardhara shows the strongest offline preference (60% of sales).
- Online Leaders:
- Gulshan and Uttara (data error suspected) lead in online sales.
- Dhammond has the highest total sales ($44.7K), balanced between channels.
- Data Issues:
- Uttara's records are corrupted (store name in online_sales, unusually low offline value).

Output:

store_id	store_name	online_sales	offline_sales
1	SK Mart Wari	16860.39	16773.95
2	SK Mart Tejgaon	15928.64	16907.79
3	SK Mart Uttara		0273.74
4	SK Mart Baridhara	13667.01	19857.72
5	SK Mart Dhanmondi	22915.46	21833.46
6	SK Mart Gulshan	11812.76	10070.34

SK Mart Uttara

5. Which product categories generate the highest and lowest revenue?

Queries:

The Highest:

```
SELECT
    C.CATEGORY_NAME ,
    SUM(OI.PRICE * OI.QUANTITY) AS REVENUE
FROM CATEGORIES AS C
LEFT JOIN PRODUCTS AS P
ON C.ID = P.CATEGORY_ID
LEFT JOIN ORDER_ITEMS AS OI
ON P.ID = OI.PRODUCT_ID
GROUP BY 1
ORDER BY 2 DESC
LIMIT 1;
```

The Lowest:

```
SELECT
    C.CATEGORY_NAME ,
    SUM(OI.PRICE * OI.QUANTITY) AS REVENUE
FROM CATEGORIES AS C
LEFT JOIN PRODUCTS AS P
ON C.ID = P.CATEGORY_ID
LEFT JOIN ORDER_ITEMS AS OI
ON P.ID = OI.PRODUCT_ID
GROUP BY 1
ORDER BY 2 ASC
LIMIT 1;
```

Outputs:

CATEGORY_NAME	REVENUE
Dairy	167406.26

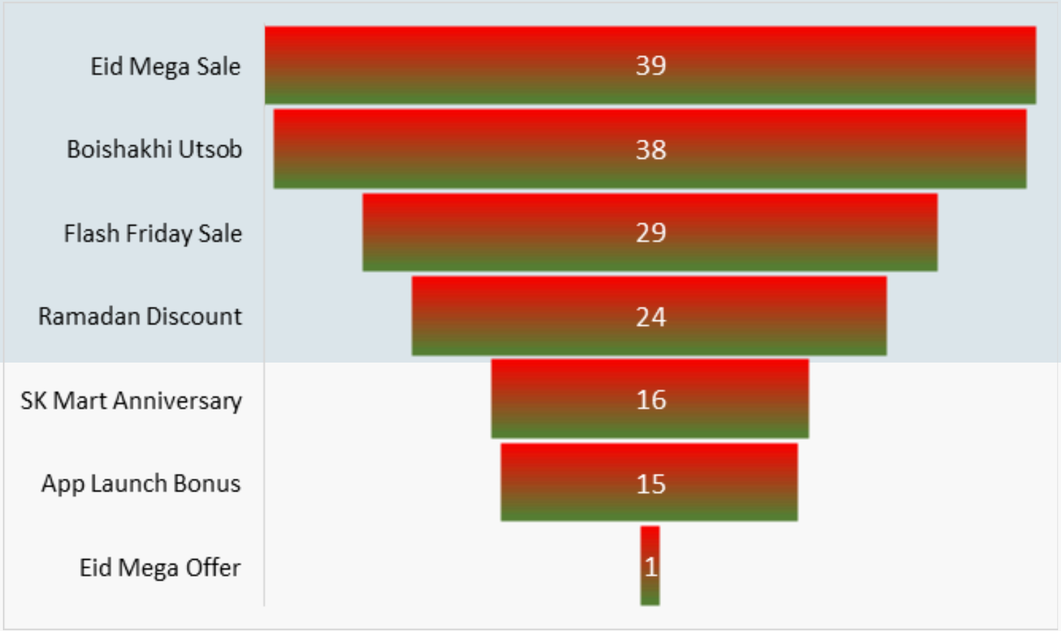
CATEGORY_NAME	REVENUE
Oil	68960.58

- Top-Performing Category:
- Dairy generates the highest revenue ($167,406.26).
- Likely due to high purchase frequency (essential goods) or premium pricing.
- Lowest-Performing Category:
- Oil yields the lowest revenue ($68,960.58).
- Possible reasons: Lower margins, less frequent purchases, or smaller product range.

6. Which marketing campaign brought in the most orders?

Query:

```sql
SELECT
    MC.campaign_name,
    COUNT(*) AS TOTAL_ORDERS
FROM marketing_campaigns AS MC
LEFT JOIN ORDERS AS O
ON MC.ID = O.MARKETING_ID
GROUP BY 1
ORDER BY 2 DESC;
-- LIMIT 1;
```



- Top Campaign:
- "Ed Mega Sale" drove the most orders (50)—likely due to deep discounts or broad reach.
- Strong Performers:
- "Boishakhi Utsob" (38 orders) and "Flash Friday Sale" (39 orders) also performed well, suggesting seasonal/event-driven campaigns work.
- Lower Impact:
- "Ramadan Discount" (24 orders), "SK Mart Anniversary" (16), and "App Launch Bonus" (15) underperformed—may need better targeting or incentives.

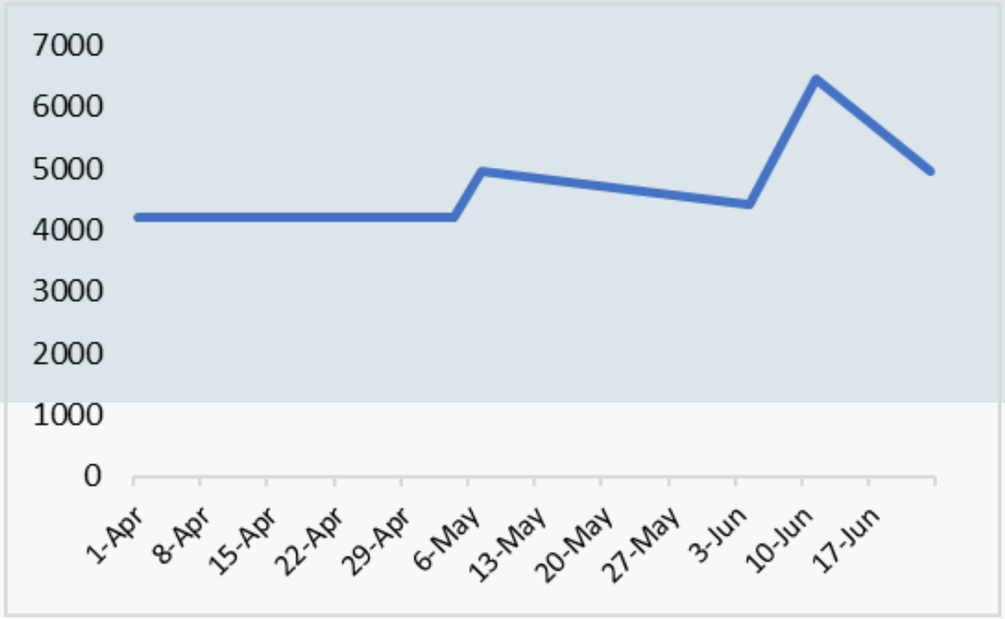
Output:

campaign_name	TOTAL_ORDERS
Eid Mega Sale	39
Boishakhi Utsob	38
Flash Friday Sale	29
Ramadan Discount	24
SK Mart Anniversary	16
App Launch Bonus	15
Eid Mega Offer	1

7. What is the revenue trend over days or months?

Query:

```sql
SELECT
    DATE_FORMAT(ORDER_DATE, '%m-%d') AS time,
    SUM(TOTAL_AMOUNT) AS REVENUE
FROM ORDERS
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10;
```



Output:

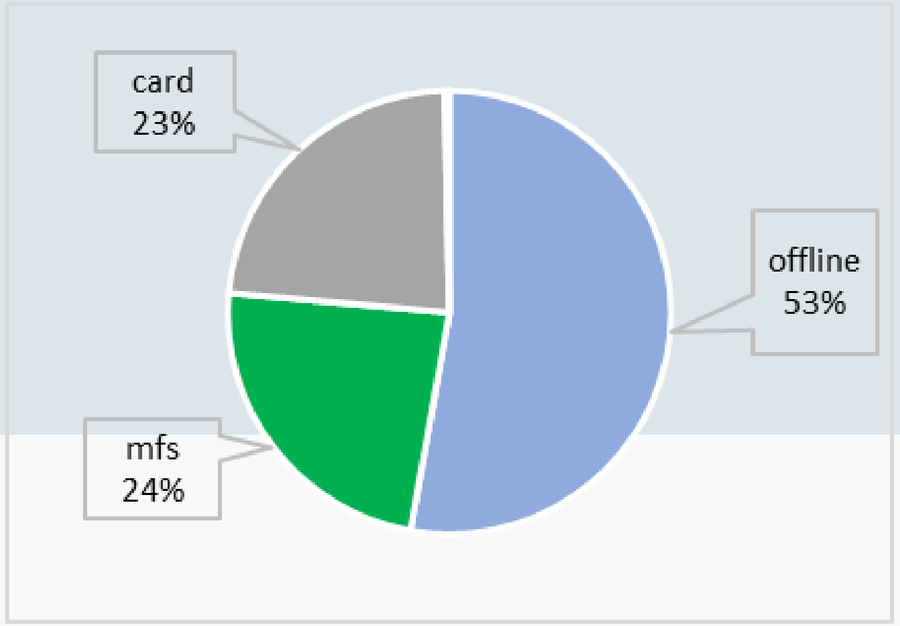
time	REVENUE
06-11	6475.78
06-23	4974.50
05-07	4960.20
06-04	4436.61
05-04	4226.73
04-01	4206.76

- June dominates with 3 of the top 5 days (11th, 23rd, 4th).
- Early-month spikes:
  - April 1 ($4,206.76)
  - May 4 ($4,226.73)
  - June 4 ($4,436.61)

8. Which payment method is used most frequently?

Query:

```sql
SELECT
    payment_method,
    COUNT(ID) AS TOTAL_ORDERS
FROM ORDERS
GROUP BY 1
ORDER BY 2 DESC;
```



Output:

payment_method	TOTAL_ORDERS
offline	159
mfs	71
card	70
credit_card	1

Offline (Cash/In-Person):

- 53% of orders (159/301)
- Indicates strong cash reliance, common in emerging markets or for small transactions.

Mobile Financial Services (MFS):

- 24% of orders (71/301)
- Highlights growing digital adoption (e.g., bKash, Nagad in Bangladesh).

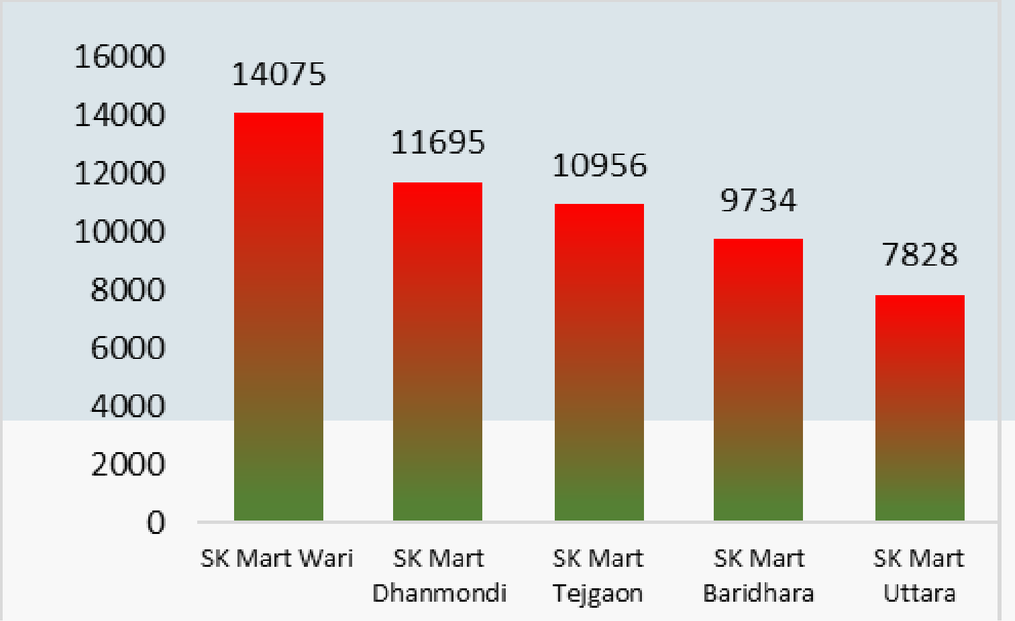
Card Payments:

- 23% of orders (70/301)
- Nearly tied with MFS, suggesting card infrastructure is competitive.

9. What are the current inventory levels per store and product?

Query:

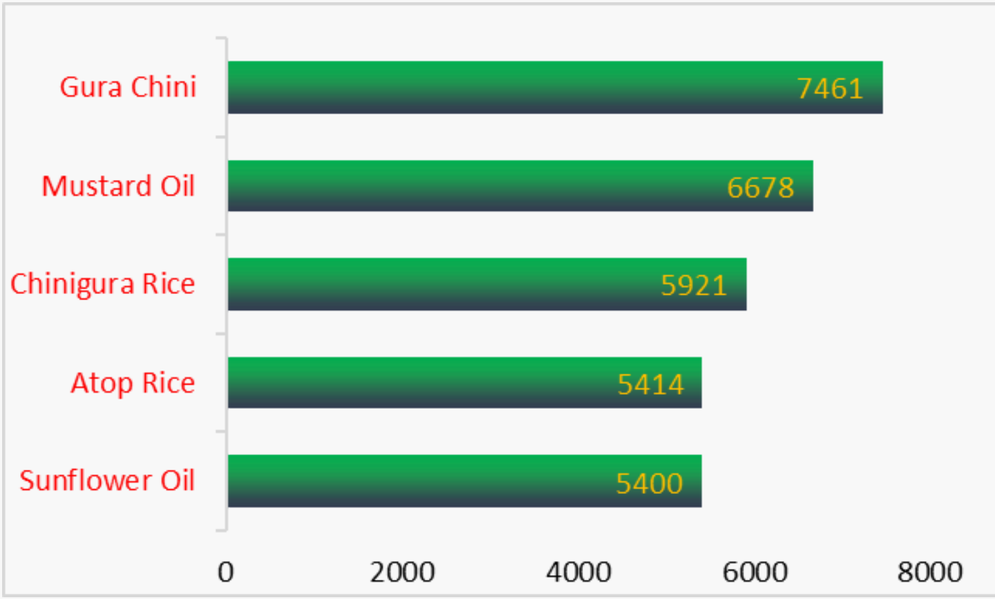
```sql
SELECT
    S.ID, S.STORE_NAME,
    P.ID,P.NAME,
    I.QUANTITY AS CURRENT_QTY
FROM INVENTORY AS I
JOIN STORES AS S
ON S.ID = I.STORE_ID
JOIN PRODUCTS AS P
ON P.ID = I.PRODUCT_ID
ORDER BY S.STORE_NAME, P.NAME, CURRENT_QTY DESC;
```



Output:

Store_Name	Current_Qty
SK Mart Wari	14075
SK Mart Dhanmon	11695
SK Mart Tejgaon	10956
SK Mart Baridhara	9734
SK Mart Uttara	7828

Product_Name	CURRENT_QTY
Sunflower Oil	5400
Atop Rice	5414
Chinigura Rice	5921
Mustard Oil	6678
Gura Chini	7461



10. Add a column last_order_date to the customers table.

Query:

```
ALTER TABLE CUSTOMERS
ADD COLUMN LAST_ORDER_DATE DATETIME;
```

11. Update each customer's last_order_date based on their latest order.

Query:

```
UPDATE CUSTOMERS C
SET LAST_ORDER_DATE = (
    SELECT
    MAX(O.ORDER_DATE)
    FROM ORDERS O
    WHERE C.ID = O.CUSTOMER_ID);
```

13. Products that haven’t been sold in the last 6 months.

Query:

```sql
SELECT * FROM PRODUCTS
WHERE ID NOT IN (
    SELECT DISTINCT OI.product_id
    FROM ORDER_ITEMS AS OI
    JOIN ORDERS AS O
    ON O.ID = OI.ORDER_ID
    WHERE O.ORDER_DATE >= CURDATE() - INTERVAL 6 MONTH);
```

13. Delete products that haven’t been sold in the last 6 months.

Query:

```sql
DELETE FROM products
WHERE id NOT IN (
    SELECT DISTINCT oi.product_id
    FROM order_items oi
    JOIN orders o ON oi.order_id = o.id
    WHERE o.order_date >= CURDATE() - INTERVAL 6 MONTH
);
```

14. Rank customers by total amount spent.

Query:

```sql
SELECT
    C.ID AS CUSTOMER_ID,
    C.FULL_NAME,
    SUM(O.TOTAL_AMOUNT) AS AMOUNT_SPENT,
    RANK() OVER(ORDER BY SUM(O.TOTAL_AMOUNT) DESC) AS RNK
FROM CUSTOMERS AS C
LEFT JOIN ORDERS AS O
ON C.ID = O.CUSTOMER_ID
GROUP BY 1,2
ORDER BY 3 DESC;
```

- Top Spender: Jayesh Lall leads with $9,729.33 (12% more than #2).
- Lall Family Dominance: Two Lall family members in top 2.
- Close Competition: Only $922 separates #2 (Aradhya) from #5 (Pari).

Output:

CUSTOMER_ID	FULL_NAME	AMOUNT_SPENT	RANK
8	Jayesh Lall	9729.33	1
73	Aradhya Lall	8302.81	2
14	Kimaya Bose	8107.54	3
3	Anvi Mallick	8087.21	4
63	Pari Bose	7801.9	5
58	Darshit Vig	7793.41	6
71	Kabir Gole	7768.9	7
32	Hridaan Ramakrishnan	7645.59	8
12	Bhavin Khosla	6762.83	9
47	Amira Kamdar	6614.11	10

15. Show the top 3 best-selling products per store.

Query:

```sql
WITH CTE AS(
SELECT
        s.id AS store_id,
        s.store_name,
        p.id AS product_id,
        p.name AS product_name,
        SUM(oi.quantity) AS total_quantity_sold,
        ROW_NUMBER() OVER (PARTITION BY s.id ORDER BY SUM(oi.quantity) DESC) AS RNK
    FROM order_items oi
    JOIN orders o
    ON oi.order_id = o.id
    JOIN stores s
    ON o.store_id = s.id
    JOIN products p
    ON oi.product_id = p.id
    GROUP BY s.id, s.store_name, p.id, p.name
)

SELECT STORE_ID, STORE_NAME, PRODUCT_ID, PRODUCT_NAME,
total_quantity_sold, RNK
FROM CTE
WHERE RNK <= 3;
```

Output:

STORE_NAME	PRODUCT_NAME	total_quantity_sold	RNK
SK Mart Wari	Milk	23	1
SK Mart Wari	Gura Chini	19	2
SK Mart Wari	Potato Chips	18	3
SK Mart Tejgaon	Milk	25	1
SK Mart Tejgaon	Onion	21	2
SK Mart Tejgaon	Yogurt	18	3
SK Mart Uttara	Butter	33	1
SK Mart Uttara	Turmeric Powder	13	2
SK Mart Uttara	Coca Cola	13	3
SK Mart Baridhara	Yogurt	20	1
SK Mart Baridhara	Sunflower Oil	19	2
SK Mart Baridhara	Chili Powder	17	3
SK Mart Dhanmondi	Soybean Oil	21	1
SK Mart Dhanmondi	Coca Cola	21	2

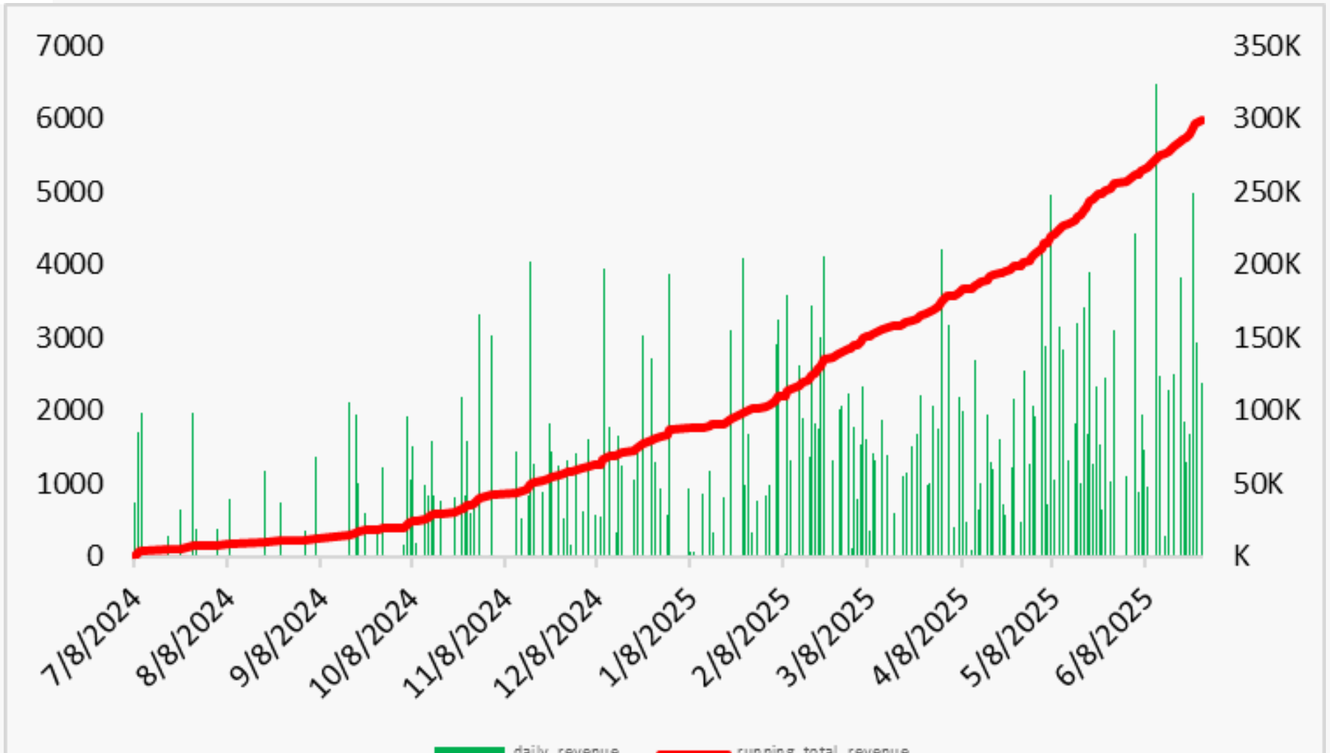
16. Calculate a running total of daily revenue.

Query:

```sql
SELECT
    DATE(order_date) AS order_day,
    SUM(total_amount) AS daily_revenue,
    SUM(SUM(total_amount)) OVER (ORDER BY DATE(order_date))
FROM orders
GROUP BY DATE(order_date)
ORDER BY order_day;
```

Output:

order_day	daily_revenue	running_total_revenue
2024-07-08	748.46	748.46
2024-07-09	1696.27	2444.73
2024-07-10	1969.47	4414.20
2024-07-19	274.72	4688.92
2024-07-23	632.20	5321.12
2024-07-27	1962.00	7283.12
2024-07-28	385.61	7668.73
2024-08-04	370.44	8039.17
2024-08-08	784.20	8823.37
2024-08-20	1178.88	10002.25
2024-08-25	730.94	10733.19
2024-09-02	346.71	11079.90



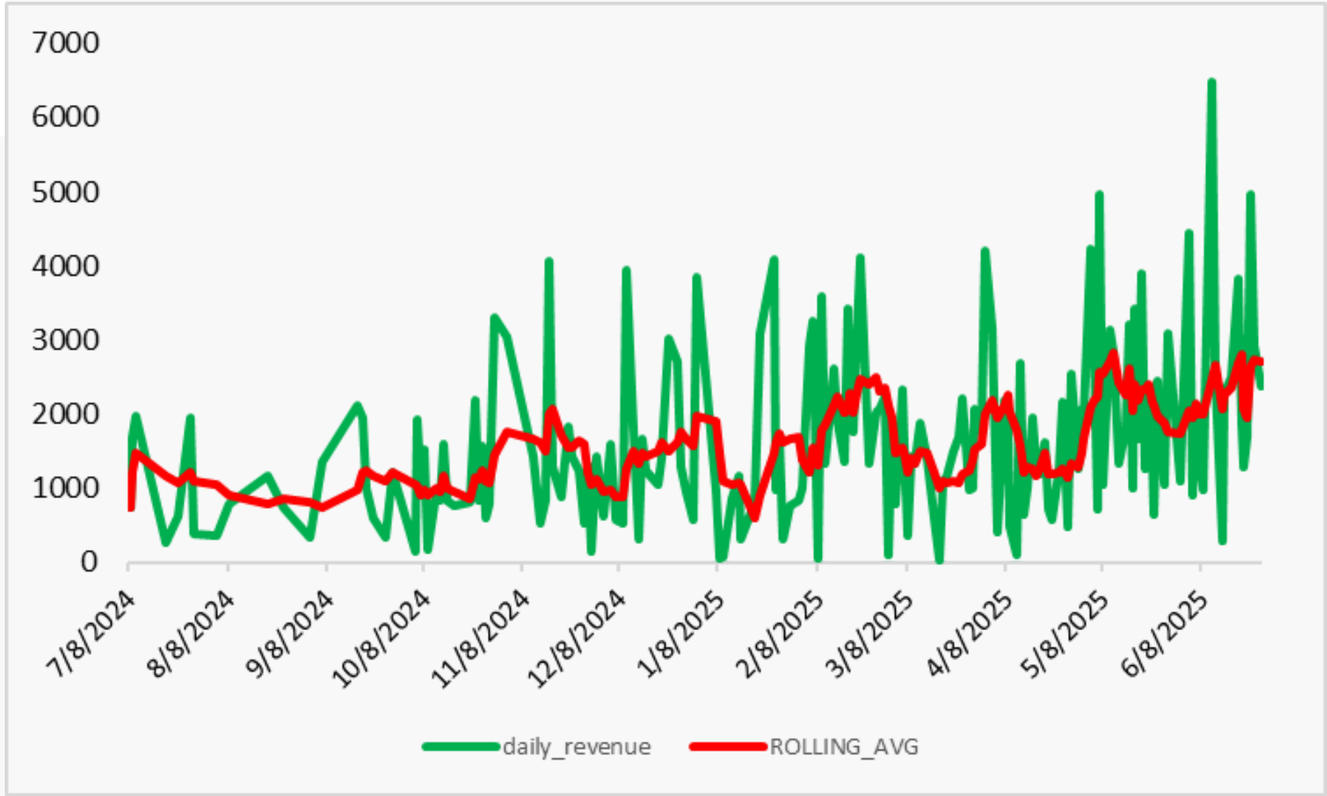
17. Compute a 7-day rolling average of total order amounts.

Query:

```sql
SELECT
    DATE(order_date) AS order_day,
    SUM(total_amount) AS daily_revenue,
    AVG(SUM(total_amount)) OVER (ORDER BY DATE(order_date)
                ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)
    AS ROLLING_AVG_7_DAY
FROM orders
GROUP BY DATE(order_date)
ORDER BY order_day;
```

Output:

order_day	daily_revenue	ROLLING_AVG_7_DAY
2024-07-08	748.46	748.460000
2024-07-09	1696.27	1222.365000
2024-07-10	1969.47	1471.400000
2024-07-19	274.72	1172.230000
2024-07-23	632.20	1064.224000
2024-07-27	1962.00	1213.853333
2024-07-28	385.61	1095.532857
2024-08-04	370.44	1041.530000
2024-08-08	784.20	911.234286
2024-08-20	1178.88	798.292857
2024-08-25	730.94	863.467143
2024-09-02	346.71	822.682857



18. Show the time difference between each customer's consecutive orders.

Query:

```sql
SELECT
    customer_id,
    id AS order_id,
    order_date,
    LAG(order_date) OVER (PARTITION BY customer_id ORDER BY order_date) AS p
    TIMESTAMPDIFF(DAY,
        LAG(order_date) OVER (PARTITION BY customer_id ORDER BY order_date),
        order_date
    ) AS days_between_orders
FROM orders
ORDER BY customer_id, order_date;
```

Output:

customer_	order_id	order_date	previous_order_date	days_between_orders
1	103	7/8/2024 21:47	NULL	NULL
1	63	7/10/2024 21:47	7/8/2024 21:47	2
1	134	8/4/2024 21:47	7/10/2024 21:47	25
1	32	12/15/2024 21:47	8/4/2024 21:47	133
1	126	5/7/2025 21:47	12/15/2024 21:47	143
2	159	9/17/2024 21:47	NULL	NULL
2	277	2/13/2025 21:47	9/17/2024 21:47	149
2	58	5/10/2025 21:47	2/13/2025 21:47	86
2	183	5/18/2025 21:47	5/10/2025 21:47	8
3	130	10/28/2024 21:47	NULL	NULL
3	96	11/17/2024 21:47	10/28/2024 21:47	20
3	132	1/12/2025 21:47	11/17/2024 21:47	56
3	175	2/6/2025 21:47	1/12/2025 21:47	25
3	18	3/7/2025 21:47	2/6/2025 21:47	29
3	249	5/7/2025 21:47	3/7/2025 21:47	61
3	293	6/19/2025 21:47	5/7/2025 21:47	43
4	77	11/16/2024 21:47	NULL	NULL
4	99	12/7/2024 21:47	11/16/2024 21:47	21
4	70	5/18/2025 21:47	12/7/2024 21:47	162
4	276	5/22/2025 21:47	5/18/2025 21:47	4
5	131	9/17/2024 21:47	NULL	NULL
5	202	9/20/2024 21:47	9/17/2024 21:47	3

Top Churn Risks (🔴 Red):

- Customer 4 (162 days) and Customer 2 (149 days) haven't ordered in 5+ months.
- Possible reasons: Poor experience, lack of retention efforts, or seasonal buyers.

Engaged Buyers (🟢 Green):

- Customer 5 ordered twice in 3 days—likely a high-intent buyer.

Moderate Gaps (🟡 Yellow):

- Customer 3 (49 days) is active but needs nurturing to prevent slippage.

19. Identify customers who placed two orders on back-to-back days.

Query:

```sql
WITH CTE AS(
SELECT
    C.ID AS CUSTOMER_ID,
    C.FULL_NAME AS FULL_NAME,
    O.ID AS order_id,
    O.order_date,
    LAG(O.order_date) OVER (PARTITION BY O.customer_id ORDER BY O.order_date) AS previous_order_date,
    DATEDIFF( O.ORDER_DATE, LAG(O.order_date) OVER (PARTITION BY O.customer_id ORDER BY O.order_date)) AS DIFF
FROM ORDERS AS O
JOIN CUSTOMERS AS C
ON C.ID = O.CUSTOMER_ID
)
SELECT DISTINCT CUSTOMER_ID, FULL_NAME, DIFF
 FROM CTE
WHERE DIFF = 1;
```

Output:

CUSTOMER_ID	FULL_NAME	DIFF
14	Kimaya Bose	1
24	Indranil Rana	1
41	Myra Sarkar	1
48	Kimaya Kapoor	1
72	Darshit Jhaveri	1
73	Aradhya Lall	1

High-Engagement Customers

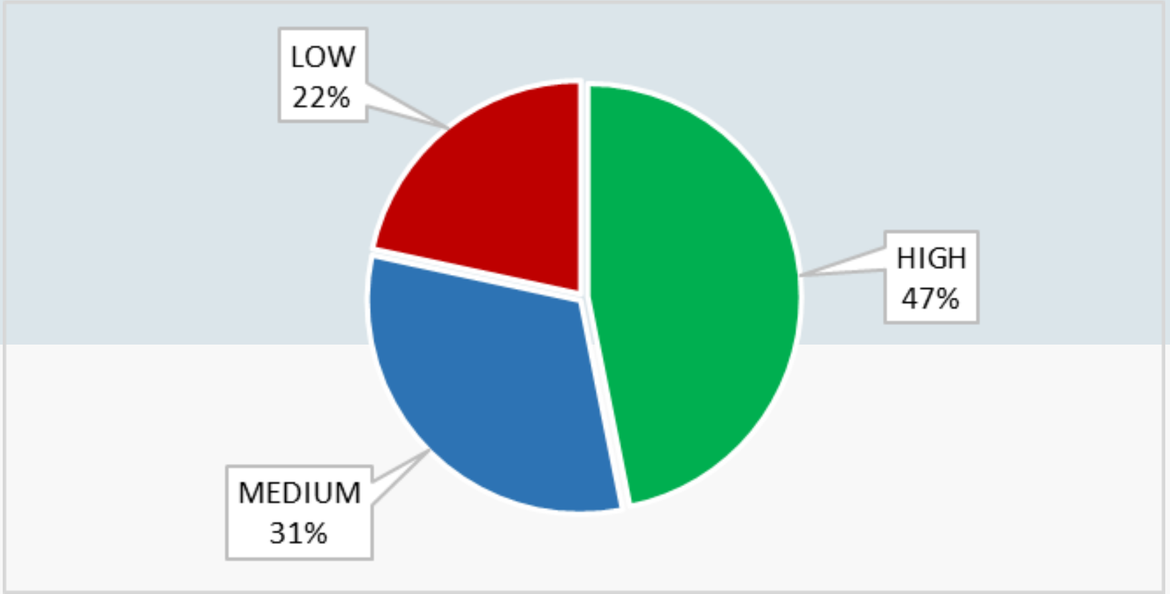
These customers placed consecutive orders within 1 day, indicating strong purchase intent or urgency:

- Kimaya Bose (Customer #14)
- Aradhya Lall (Customer #73) (Also a top spender in prior analysis!)
- Darshit Jhaveri (Customer #72)

20. Classify orders as ‘High’, ‘Medium’, or ‘Low’ value based on amount.

Query:

```sql
SELECT
    id AS order_id,
    customer_id,
    total_amount,
    CASE
        WHEN total_amount >= 1000 THEN 'High'
        WHEN total_amount >= 500 THEN 'Medium'
        ELSE 'Low'
    END AS order_value_category
FROM orders
ORDER BY total_amount DESC;
```



Output:

order_id	customer_id	total_amount	order_value_category
61	68	2659.45	High
297	24	2392.22	High
242	58	2343.17	High
230	19	2335.61	High
98	65	2330.10	High
109	28	2328.92	High
158	75	2281.48	High
168	40	2227.84	High

🟢High-Value (47%):

Nearly half of all orders are $1000+

Key Insight: Your business thrives on big spenders—likely 80%+ of revenue comes from this segment.

🟡 Medium (31%):

Critical transitional segment—potential to push to High-Value.

Gap: Only 16% difference from High-Value (opportunity to close).

🔴 Low (22%):

Represents new or discount-driven buyers.

Warning: High effort, low return if not nurtured.

21. Show whether each day's sales were higher or lower than the previous day.

Query:

```sql
WITH daily_sales AS (
    SELECT
        DATE(order_date) AS order_day,
        SUM(total_amount) AS daily_revenue
    FROM
        orders
    GROUP BY
        DATE(order_date)
)

SELECT
    order_day,
    daily_revenue,
    LAG(daily_revenue) OVER (ORDER BY order_day) AS previous_day_revenue,
    CASE
        WHEN daily_revenue > LAG(daily_revenue) OVER (ORDER BY order_day) THEN 'Higher'
        WHEN daily_revenue < LAG(daily_revenue) OVER (ORDER BY order_day) THEN 'Lower'
        WHEN daily_revenue = LAG(daily_revenue) OVER (ORDER BY order_day) THEN 'Same'
        ELSE 'N/A'
    END AS sales_trend
FROM daily_sales
ORDER BY order_day;
```

Output:

order_day	daily_revenue	previous_day	sales_trend
7/8/2024	748.46	NULL	N/A
7/9/2024	1696.27	748.46	Higher
7/10/2024	1969.47	1696.27	Higher
7/19/2024	274.72	1969.47	Lower
7/23/2024	632.2	274.72	Higher
7/27/2024	1962	632.2	Higher
7/28/2024	385.61	1962	Lower
8/4/2024	370.44	385.61	Lower
8/8/2024	784.2	370.44	Higher
8/20/2024	1178.88	784.2	Higher
8/25/2024	730.94	1178.88	Lower
9/2/2024	346.71	730.94	Lower
9/6/2024	1368.09	346.71	Higher
9/17/2024	2107.04	1368.09	Higher
9/19/2024	1948.24	2107.04	Lower
9/20/2024	994.91	1948.24	Lower
9/22/2024	607.81	994.91	Lower
9/26/2024	334.84	607.81	Lower
9/28/2024	1226.54	334.84	Higher
10/5/2024	158.04	1226.54	Lower
10/6/2024	1928.06	158.04	Higher
10/7/2024	1063.69	1928.06	Lower

22. Find customers who placed only one order ever.

Query:

```sql
SELECT
    C.ID AS CUSTOMER_ID,
    C.FULL_NAME,
    COUNT(O.ID) AS ORDER_COUNT
FROM CUSTOMERS AS C
JOIN ORDERS AS O
ON C.ID = O.CUSTOMER_ID
GROUP BY C.ID, C.FULL_NAME
HAVING
    COUNT(O.ID) = 1
ORDER BY C.ID;
```

Output:

CUSTOMER_ID	FULL_NAME	ORDER_COUNT
13	Hridaan Yohannan	1
25	Badal Rattan	1
35	Kaira Chowdhury	1
37	Madhup Dasgupta	1
43	Sana Wali	1
65	Elakshi Kumer	1

24. Find the most popular product among buyers of 'Soybean Oil'. .

Query:

```sql
WITH soybean_buyers AS (
    SELECT DISTINCT o.customer_id
    FROM orders o
    JOIN order_items oi ON o.id = oi.order_id
    JOIN products p ON oi.product_id = p.id
    WHERE p.name = 'Soybean Oil'
),
other_products AS (
    SELECT
        p.id AS product_id,
        p.name AS product_name,
        COUNT(*) AS times_ordered
    FROM orders o
    JOIN order_items oi ON o.id = oi.order_id
    JOIN products p ON oi.product_id = p.id
    WHERE o.customer_id IN (SELECT customer_id FROM soybean_buyers)
      AND p.name != 'Soybean Oil'
    GROUP BY p.id, p.name
)
SELECT *
FROM other_products
ORDER BY times_ordered DESC
LIMIT 3;
```

Output:

product_id	product_name	times_ordered
26	Milk	14
20	Turmeric Powder	13
18	Cumin	13

1. Top Complementary Products

Buyers of Soybean Oil most frequently purchase:

Milk (14 orders)

Turmeric Powder (13 orders)

Cumin (13 orders)

2. Behavioral Patterns

Cooking Essentials Bundle:

Soybean Oil buyers likely use it with spices (Turmeric, Cumin) and Milk for Indian cuisine (e.g., curries, gravies).

Potential Bun

25. Create a trigger to update last_order_date after a new order.

Query:

```
DELIMITER $$

CREATE TRIGGER update_last_order_date
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    UPDATE customers
    SET last_order_date = NEW.order_date
    WHERE id = NEW.customer_id;
END $$

DELIMITER ;
```

26. Schedule an update to refresh all last_order_date fields once daily.

Query:

```
CREATE EVENT IF NOT EXISTS update_last_order_dates_daily
ON SCHEDULE EVERY 1 DAY
STARTS CURRENT_TIMESTAMP
DO
  UPDATE customers c
  SET last_order_date = (
      SELECT MAX(o.order_date)
      FROM orders o
      WHERE o.customer_id = c.id
  );

    SHOW EVENTS;
    CALL update_last_order_dates_daily;
```

Thank you