FLIPKART s ZOMATO

DATA ANALYST
INTERVIEW QUESTIONS

FLIPKART

Ç Q1 : Find Flipkart customers who purchased products in more than 3 categories

Objective:

Identify customers who have bought products from **more than 3 distinct product categories.**

Sample Dataset — Orders

order_id	customer_id	product_id	category	order_date	amount
1	C001	P001	Electronics	2025-04-10	12000
2	C001	P002	Fashion	2025-05-14	2500
3	C001	P003	Home Appliances	2025-06-02	4500
4	C001	P004	Grocery	2025-07-01	700
5	C002	P005	Electronics	2025-05-18	15000
6	C002	P006	Fashion	2025-06-01	3000

order_id	customer_id	product_id	category	order_date	amount
7	C003	P007	Beauty	2025-07-10	800
8	C003	P008	Grocery	2025-08-12	600
9	C003	P009	Home Appliances	2025-09-01	2500

🪜 Approach:

- Group by customer_id
- Count distinct category
- Filter customers where category count > 3

) SQL Query:

```
SELECT
  customer_id,
  COUNT(DISTINCT category) AS distinct_categories
FROM Orders
GROUP BY customer_id
HAVING COUNT(DISTINCT category) > 3;
```

■ Output:

customer_id	distinct_categories
C001	4

● Interpretation:

Customer **C001** has bought from **4 different categories**, which is more than 3 — hence qualifies.

🧠 Insights:

This type of query checks **customer diversity** and can help Flipkart identify:

- **Cross-category shoppers**
- Potential candidates for **loyalty programs** or **personalized recommendations**

🔍 Find Flipkart customers who returned more than 20% of orders in the last 6 months

🎯 Objective:

Identify customers who have **returned >20% of their total orders** during the **last 6 months**.

🧾 Sample Dataset — Customer_Orders

order_id	customer_id	order_date	order_status	amount
1	C001	2025-04-05	Delivered	1200
2	C001	2025-05-10	Returned	800
3	C001	2025-06-12	Delivered	1500
4	C001	2025-07-20	Returned	900
5	C001	2025-08-25	Delivered	1000
6	C002	2025-05-14	Delivered	2000
7	C002	2025-06-11	Delivered	1800
8	C002	2025-07-10	Returned	900
9	C003	2025-08-05	Delivered	1700

🪜 Approach:

1. Consider orders only from the **last 6 months**.
2. For each customer:
    - Count total orders
    - Count returned orders
3. Compute return percentage = (returned_orders / total_orders) * 100
4. Filter where return% > 20

) **SQL Query:**

```
WITH last_6_months AS (
  SELECT *
  FROM Customer_Orders
  WHERE order_date >= DATEADD(MONTH, -6, GETDATE())  -- last 6 months
)
, customer_summary AS (
  SELECT
    customer_id,
    COUNT(*) AS total_orders,
    SUM(CASE WHEN order_status = 'Returned' THEN 1 ELSE 0 END) AS returned_orders
  FROM last_6_months
  GROUP BY customer_id
)
SELECT
  customer_id,
  total_orders,
  returned_orders,
  ROUND((CAST(returned_orders AS FLOAT) / total_orders) * 100, 2) AS return_percentage
```

FROM customer_summary

WHERE (CAST(returned_orders AS FLOAT) / total_orders) * 100 > 20;

Output:

customer_id	total_orders	returned_orders	return_percentage
C001	5	2	40.00
C002	3	1	33.33

Interpretation:
Both **C001** and **C002** returned more than **20%** of their orders in the past 6 months.

Insights:

Such analysis helps Flipkart:

- Identify **habitual returners**
- Investigate reasons (product quality, delivery issues)
- Reduce logistics losses by **flagging high-return customers**

Q3 — Identify Flipkart customers who purchased *Electronics* and *Fashion* items in the same month

Objective: find (customer_id, year, month) where the customer bought at least one Electronics *and* at least one Fashion item within the same calendar month.

Sample Orders table (schema + sample rows)

order_id	customer_id	product_id	category	order_date	amount
1	C001	P001	Electronics	2025-07-05	12000
2	C001	P002	Fashion	2025-07-15	2500

order_id	customer_id	product_id	category	order_date	amount
3	C002	P003	Electronics	2025-08-01	8000
4	C002	P004	Electronics	2025-08-10	2000
5	C002	P005	Fashion	2025-09-05	1500
6	C003	P006	Fashion	2025-06-15	600
7	C003	P007	Electronics	2025-06-18	1000
8	C004	P008	Electronics	2025-07-01	2000
9	C004	P009	Fashion	2025-08-02	1800

Approach

- Group by customer_id, YEAR(order_date), MONTH(order_date).
- Use conditional aggregation to check presence of Electronics and Fashion in the same group.
- Return the (customer, year, month) pairs (or distinct customers if you prefer).

SQL

```
SELECT
 customer_id,
 YEAR(order_date) AS year,
 MONTH(order_date) AS month,
 SUM(CASE WHEN category = 'Electronics' THEN 1 ELSE 0 END) AS electronics_count,
 SUM(CASE WHEN category = 'Fashion' THEN 1 ELSE 0 END) AS fashion_count
FROM Orders
GROUP BY
 customer_id,
 YEAR(order_date),
 MONTH(order_date)
```

```
HAVING
  SUM(CASE WHEN category = 'Electronics' THEN 1 ELSE 0 END) > 0
  AND SUM(CASE WHEN category = 'Fashion' THEN 1 ELSE 0 END) > 0;
```

Expected output (from sample rows)

customer_id	year	month	electronics_count	fashion_count
C001	2025	7	1	1
C003	2025	6	1	1

Notes / variants

- If you want just distinct customers (ignore which month): wrap the query and SELECT DISTINCT customer_id.
- If you want to **exclude returned orders**, add WHERE order_status <> 'Returned' (or an appropriate filter) before grouping.
- Normalize category names (trim / lower) if data quality is an issue.

Q4 — Identify Flipkart customers who returned more than 10% of their orders *last quarter*

Objective: for the previous quarter (relative to GETDATE()), find customers where (returned_orders / total_orders) > 10%.

Sample Customer_Orders table (schema + rows — dates in last quarter: Jul-Sep 2025 if today is Oct 6, 2025)

order_id	customer_id	order_date	order_status
1	C001	2025-07-05	Delivered
2	C001	2025-07-12	Returned
3	C001	2025-08-20	Returned
4	C001	2025-09-01	Delivered

order_id	customer_id	order_date	order_status
5	C001	2025-09-15	Delivered
6	C002	2025-07-10	Delivered
7	C002	2025-08-15	Delivered
8	C002	2025-09-20	Delivered
	C003	2025-07-02	Returned
10	C003	2025-08-05	Returned
11	C003	2025-09-10	Returned
12	C004	2025-09-28	Delivered
13	C004	2025-09-30	Returned

Approach

- Compute start and end timestamps for *last quarter* dynamically.
- Filter orders into that window.
- For each customer count total orders and returned orders; compute return percentage.
- Filter where > 10%.

T-SQL (robust)

```
-- compute start and end of last quarter, robustly
;WITH quarter_bounds AS (
  SELECT
    DATEADD(QUARTER, DATEDIFF(QUARTER, 0, GETDATE()) - 1, 0) AS q_start,
    DATEADD(QUARTER, DATEDIFF(QUARTER, 0, GETDATE()), 0) AS q_end
),
last_q AS (
```

```
  SELECT co.*
  FROM Customer_Orders co
  CROSS JOIN quarter_bounds qb
  WHERE co.order_date >= qb.q_start
  AND co.order_date <  qb.q_end
)
, customer_summary AS (
  SELECT
    customer_id,
    COUNT(*) AS total_orders,
    SUM(CASE WHEN order_status = 'Returned' THEN 1 ELSE 0 END) AS returned_orders
  FROM last_q
  GROUP BY customer_id
)
SELECT
 customer_id,
 total_orders,
 returned_orders,
 ROUND((CAST(returned_orders AS FLOAT) / NULLIF(total_orders,0)) * 100, 2) AS
return_percentage
FROM customer_summary
WHERE (CAST(returned_orders AS FLOAT) / NULLIF(total_orders,0)) > 0.10;
```

Expected output (from sample rows)

customer_id	total_orders	returned_orders	return_percentage
C001	5	2	40.00

customer_id	total_orders	returned_orders	return_percentage
C003	3	3	100.00
C004	2	1	50.00

Notes / variants

- If you consider only *successful deliveries* when computing returns (i.e., exclude cancelled or pending), adjust the filtering logic.
- Use NULLIF(total_orders,0) to avoid division by zero.
- You can change the threshold (10%) easily in the WHERE clause.
- For performance on large tables, ensure order_date is indexed or use partitioning by date.

Q5 — Find Flipkart customers who frequently bought from Electronics but never purchased Accessories

Interpretation s choices

- The word *frequently* is ambiguous. I'll present two common definitions — pick one in your interview:
    - **Definition A (count-based):** bought Electronics **≥ 5 times** (changeable threshold) and **never** bought Accessories.
    - **Definition B (share-based):** Electronics make up **≥ 50%** of that customer's purchases (and at least 3 electronics orders), and zero Accessories purchases.

Sample Orders table (abbreviated)

order_id	customer_id	category	order_date
1	C005	Electronics	2025-01-05
2	C005	Electronics	2025-02-10
3	C005	Electronics	2025-03-11

order_id	customer_id	category	order_date
4	C005	Electronics	2025-04-09
5	C005	Electronics	2025-06-12
6	C005	Fashion	2025-07-01
7	C006	Electronics	2025-02-05
8	C006	Electronics	2025-03-06
9	C006	Accessories	2025-04-10
10	C007	Electronics	2025-05-01
11	C007	Fashion	2025-05-10
12	C007	Electronics	2025-06-20

- Here **C005**: 5 Electronics, 0 Accessories → qualifies for Definition A.
- **C006**: bought Accessories → disqualify.
- **C007**: only 2 Electronics → may not meet count threshold of 5.

SQL — Definition A (count-based, threshold = 5)

```
SELECT
 customer_id,
 SUM(CASE WHEN category = 'Electronics' THEN 1 ELSE 0 END) AS electronics_count,
SUM(CASE WHEN category = 'Accessories' THEN 1 ELSE 0 END) AS accessories_count
FROM Orders
GROUP BY customer_id
HAVING
 SUM(CASE WHEN category = 'Electronics' THEN 1 ELSE 0 END) >= 5
 AND SUM(CASE WHEN category = 'Accessories' THEN 1 ELSE 0 END) = 0;
```

SQL — Definition B (share-based: Electronics >= 50% and at least 3 electronics orders)

```
SELECT
 customer_id,
 COUNT(*) AS total_orders,
 SUM(CASE WHEN category = 'Electronics' THEN 1 ELSE 0 END) AS electronics_count,
SUM(CASE WHEN category = 'Accessories' THEN 1 ELSE 0 END) AS accessories_count
FROM Orders
GROUP BY customer_id
HAVING
 SUM(CASE WHEN category = 'Electronics' THEN 1 ELSE 0 END) >= 3
 AND (SUM(CASE WHEN category = 'Electronics' THEN 1 ELSE 0 END) * 1.0 / COUNT(*)) >= 0.50
 AND SUM(CASE WHEN category = 'Accessories' THEN 1 ELSE 0 END) = 0;
```

Variant — exclude returned orders
Add WHERE order_status <> 'Returned' (or similar) before the GROUP BY to count only completed purchases.

Notes

- Choose and **state your threshold** (>=5 or >=3 etc.) in the interview — interviewers usually appreciate you defining assumptions explicitly.
- Make sure categories are normalized (e.g., Accessories vs Accessory).
- For very large datasets, prefer NOT EXISTS + a precomputed Electronics-count table for better query plans.

ZOMATO

Problem 1: Find Zomato customers who ordered more than 5 times but rated only once

Objective

Identify customers who:

- Placed **more than 5 orders**, and
- Provided **only one rating** (i.e., rated just one of those orders)

Sample Table — Orders

order_id	customer_id	order_date	restaurant_name	amount	rating
1	C001	2025-04-10	Domino’s	450	4
2	C001	2025-04-15	KFC	350	NULL
3	C001	2025-05-01	Subway	500	NULL
4	C001	2025-05-12	McDonald’s	600	NULL
5	C001	2025-06-10	Burger King	700	NULL
6	C001	2025-06-22	Biryani Blues	550	NULL
7	C002	2025-05-14	Subway	450	5
8	C002	2025-06-11	KFC	500	4
9	C002	2025-07-20	Pizza Hut	600	3
10	C003	2025-06-02	Domino’s	400	2
11	C003	2025-06-25	Burger King	450	NULL
12	C003	2025-07-03	Domino’s	500	NULL
13	C003	2025-07-14	McDonald’s	400	NULL
14	C003	2025-07-30	Subway	650	NULL
15	C003	2025-08-02	KFC	500	NULL

Approach

1. Group the data by customer_id

2. Count total orders per customer
3. Count how many of those orders have a **non-null rating**
4. Filter customers having:
    - total_orders > 5
    - rated_orders = 1

) **SQL Query**

```
SELECT
  customer_id,
  COUNT(*) AS total_orders,
  SUM(CASE WHEN rating IS NOT NULL THEN 1 ELSE 0 END) AS rated_orders
FROM Orders
GROUP BY customer_id
HAVING COUNT(*) > 5
  AND SUM(CASE WHEN rating IS NOT NULL THEN 1 ELSE 0 END) = 1;
```

Output

customer_id	total_orders	rated_orders
C001	6	1
C003	6	1

Business Insight

- These customers are **frequent but low-feedback users.**
- Zomato can **nudge them for reviews** using loyalty points or discount offers.

- Feedback collection helps improve **restaurant ratings** and **food recommendation algorithms**.

Problem 2: Find Zomato customers who ordered only on weekends in the last 3 months

Objective

Identify customers who, in the **last 3 months**, placed orders **exclusively on weekends (Saturday or Sunday)** — no weekday orders.

Sample Table — Orders

order_id	customer_id	order_date	restaurant_name	amount
1	C001	2025-07-06	Domino's	450
2	C001	2025-07-13	Subway	550
3	C001	2025-08-03	KFC	400
4	C002	2025-07-05	Domino's	600
5	C002	2025-07-08	Burger King	500
6	C003	2025-07-07	McDonald's	700
7	C004	2025-08-02	Subway	550
8	C004	2025-08-09	Domino's	600
9	C004	2025-09-07	Burger King	700

Approach

1. Filter only the **last 3 months** of data.
2. For each customer, check:

- How many total orders they placed (total_orders)
- How many of those were on weekends (weekend_orders)

3. Return only those customers where total_orders = weekend_orders.

SQL Query

```
WITH last_3_months AS (
  SELECT *
  FROM Orders
  WHERE order_date >= DATEADD(MONTH, -3, GETDATE())
),
flagged_orders AS (
  SELECT
    customer_id,
    order_date,
    DATENAME(WEEKDAY, order_date) AS day_name,
    CASE WHEN DATENAME(WEEKDAY, order_date) IN ('Saturday', 'Sunday') THEN 1 ELSE 0
END AS is_weekend
  FROM last_3_months
)
SELECT
  customer_id,
  COUNT(*) AS total_orders,
  SUM(is_weekend) AS weekend_orders
FROM flagged_orders
GROUP BY customer_id
HAVING COUNT(*) = SUM(is_weekend);
```

🟩 Output

customer_id	total_orders	weekend_orders
C001	3	3
C004	3	3

💡 Business Insight

- These are **"weekend-only" customers**, possibly office-goers or students.
- Zomato can:
  - Push **weekend combo offers** or **Sunday brunch recommendations**.
  - Study time-based order behavior for **personalized marketing**.

🍽️ Problem 3: Find Zomato customers who ordered from the same restaurant more than 3 times but left low ratings

🎯 Objective

Identify customers who:

- Ordered from **the same restaurant > 3 times**,
- Have an **average rating < 3** for that restaurant.

🧾 Sample Table — Orders

order_id	customer_id	restaurant_name	order_date	amount	rating
1	C001	Domino's	2025-05-10	400	2
2	C001	Domino's	2025-05-25	450	3
3	C001	Domino's	2025-06-10	500	2

order_id	customer_id	restaurant_name	order_date	amount	rating
4	C001	Domino’s	2025-07-01	550	1
5	C002	KFC	2025-06-15	600	4
6	C002	KFC	2025-07-18	550	3
7	C002	KFC	2025-08-10	650	4
8	C003	Subway	2025-05-20	350	2
9	C003	Subway	2025-05-25	300	1
10	C003	Subway	2025-06-01	400	2
11	C003	Subway	2025-06-10	350	3

Approach

- Group by (customer_id, restaurant_name).
- Count total orders and average rating.
- Filter where count > 3 and avg_rating < 3.

SQL Query

```
SELECT
  customer_id,
  restaurant_name,
  COUNT(*) AS total_orders,
  ROUND(AVG(CAST(rating AS FLOAT)), 2) AS avg_rating
FROM Orders
WHERE rating IS NOT NULL
GROUP BY
```

```
  customer_id, restaurant_name
HAVING
  COUNT(*) > 3
  AND AVG(CAST(rating AS FLOAT)) < 3;
```

🟩 Output

customer_id	restaurant_name	total_orders	avg_rating
C001	Domino's	4	2.00
C003	Subway	4	2.00

💡 Business Insight

- These users show **repeat purchase behavior despite low satisfaction.**
- Could indicate **location convenience, lack of alternatives,** or **habit.**
- Zomato can:
  - Prompt "How can we improve your experience?" feedback,
  - Offer **restaurant quality audits** or **discount recovery offers.**

#️⃣Problem 4: Find Zomato restaurants with increasing monthly order counts for 4 consecutive months

🎯 Objective

Identify restaurants showing **consistent growth** in order volume — a key metric for **business performance and investor reports.**

📂Sample Table — Orders

order_id	restaurant_name	order_date
1	Domino’s	2025-05-01
2	Domino’s	2025-05-10
3	Domino’s	2025-06-05
4	Domino’s	2025-06-20
5	Domino’s	2025-07-12
6	Domino’s	2025-08-03
7	Domino’s	2025-08-10
8	KFC	2025-05-15
9	KFC	2025-06-20
10	KFC	2025-07-25
11	KFC	2025-08-05
12	Subway	2025-05-05
13	Subway	2025-06-05
14	Subway	2025-07-05
15	Subway	2025-08-05

Approach

1. Get monthly order counts per restaurant.
2. Use a window function (LAG) to compare order counts of consecutive months.
3. Identify restaurants with *strictly increasing* counts for 4 consecutive months.

SQL Query

```
WITH monthly_orders AS (
  SELECT
    restaurant_name,
    YEAR(order_date) AS yr,
    MONTH(order_date) AS mn,
    COUNT(*) AS order_count
  FROM Orders
  GROUP BY restaurant_name, YEAR(order_date), MONTH(order_date)
),
growth_check AS (
  SELECT
    restaurant_name,
    yr, mn, order_count,
    LAG(order_count, 1) OVER (PARTITION BY restaurant_name ORDER BY yr, mn) AS prev1,
    LAG(order_count, 2) OVER (PARTITION BY restaurant_name ORDER BY yr, mn) AS prev2,
    LAG(order_count, 3) OVER (PARTITION BY restaurant_name ORDER BY yr, mn) AS prev3
  FROM monthly_orders
)
SELECT DISTINCT restaurant_name
FROM growth_check
WHERE order_count > prev1 AND prev1 > prev2 AND prev2 > prev3;
```

Output

restaurant_name
Domino’s

restaurant_name
Subway

🤔 Business Insight

- These restaurants show **steady growth momentum.**
- Zomato can:
    - Offer **premium visibility** or **Zomato Gold partnerships,**
    - Use them as **success stories** for vendor onboarding.

F Problem 5: Find Zomato customers who frequently ordered late night (10 PM - 2 AM) but have low average tip amount

🥣 Objective

Identify customers who order **late at night** (10 PM – 2 AM window) frequently, but **tip below average** — to understand night-hour customer behavior and delivery incentive needs.

🦀 Sample Table — Orders

order_id	customer_id	order_time	tip_amount
1	C001	2025-07-05 22:30	10
2	C001	2025-07-06 23:15	5
3	C001	2025-07-12 00:45	0
4	C001	2025-07-19 21:00	20
5	C002	2025-07-05 23:50	2
6	C002	2025-07-06 01:10	1
7	C002	2025-07-07 23:40	0

order_id	customer_id	order_time	tip_amount
8	C003	2025-07-06 20:00	10
9	C003	2025-07-07 21:30	15

Approach

1. Define **late-night window** as 22:00 → 02:00.
2. Extract hour from order_time.
3. Identify customers having **≥3 late-night orders**.
4. Compute average tip_amount per customer and compare against overall average tip.

SQL Query

```
WITH late_orders AS (
  SELECT
    customer_id,
    tip_amount,
    CASE
      WHEN DATEPART(HOUR, order_time) >= 22 OR DATEPART(HOUR, order_time) < 2
THEN 1
      ELSE 0
    END AS is_late
  FROM Orders
),
summary AS (
  SELECT
    customer_id,
```

```
    COUNT(CASE WHEN is_late = 1 THEN 1 END) AS late_night_orders,
    AVG(tip_amount) AS avg_tip
  FROM late_orders
  GROUP BY customer_id
),
overall AS (
  SELECT AVG(tip_amount) AS global_avg_tip FROM Orders
)
SELECT
  s.customer_id,
  s.late_night_orders,
  ROUND(s.avg_tip,2) AS avg_tip
FROM summary s
CROSS JOIN overall o
WHERE s.late_night_orders >= 3
 AND s.avg_tip < o.global_avg_tip;
```

🟩 Output

customer_id	late_night_orders	avg_tip
C001	3	5.00
C002	3	1.00

👳 Business Insight

- These customers show **habitual late-night ordering** but **low tipping.**
- Useful for:

- Adjusting **delivery charges / surge pricing,**
- Targeting **late-night promotions,**
- Incentivizing **delivery partners** on these time slots.
