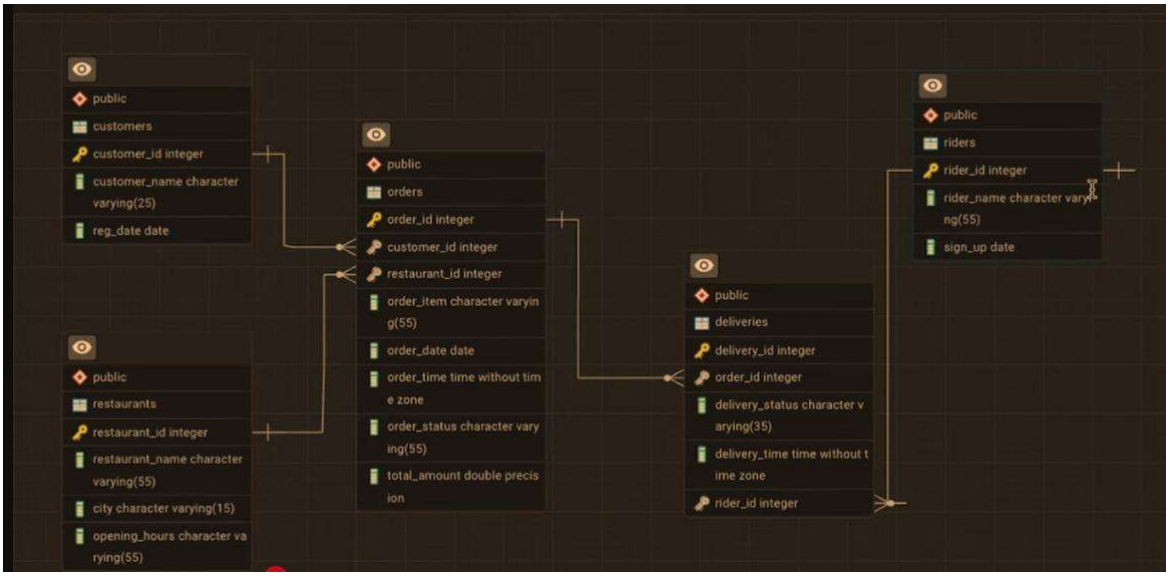


1. **WAQ to find top 5 most frequently ordered dishes by customer called "Vanshika Mishra" in past 1 year**

```
WITH dish_frequency AS (
    SELECT o.order_item,
        COUNT(*) AS order_count,
        DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
    FROM orders o
    JOIN customers c
        ON o.customer_id = c.customer_id
    WHERE c.customer_name = 'Vanshika Mishra'
      AND o.order_date >= CURRENT_DATE - INTERVAL '1 year'
    GROUP BY o.order_item
)
SELECT order_item, order_count
FROM dish_frequency
WHERE rnk <=5
ORDER BY order_count DESC;
```

2. **Identify time slots during which most orders are placed. In 2 hour intervals**

```
SELECT
    FLOOR(HOUR(order_time) / 2) * 2 AS slot_start_hour,
    FLOOR(HOUR(order_time) / 2) * 2 + 2 AS slot_end_hour,
    COUNT(*) AS total_orders
FROM orders
GROUP BY slot_start_hour, slot_end_hour
ORDER BY total_orders DESC;
```

3. **Find AOV per customer who has placed more than 750 orders**

```
SELECT  c.customer_id, c.customer_name,
    COUNT(o.order_id) AS total_orders,
    AVG(o.total_amount) AS aov
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
```

```
GROUP BY c.customer_id, c.customer_name
HAVING COUNT(o.order_id) > 750
ORDER BY aov DESC;
```

4. **List the customers name and id who spent more than 40000 in total on food orders**

```
SELECT  c.customer_id,  c.customer_name,
   Sum(o.total_amount) AS order_total
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.customer_name
HAVING   Sum(o.total_amount) > 40000
ORDER BY  order_total DESC;
```

5. **WAQ to find orders that were placed but not delivered**
   Return each restaurant name, city and number of not delivered orders

```
SELECT  r.restaurant_name, r.city,
   COUNT(o.order_id) AS not_delivered_orders
FROM restaurants r
JOIN orders o ON r.restaurant_id = o.restaurant_id
LEFT JOIN deliveries d ON o.order_id = d.order_id
WHERE d.delivery_id IS NULL
  OR d.delivery_status <> 'Delivered'
GROUP BY r.restaurant_name, r.city
ORDER BY not_delivered_orders DESC;
```

6. **Rank restaurants by total revenue of last year – name,revenue, rank in each city**

```
WITH restaurant_revenue AS (
   SELECT  r.restaurant_name, r.city,
      SUM(o.total_amount) AS total_revenue
   FROM restaurants r
   JOIN orders o  ON r.restaurant_id = o.restaurant_id
   WHERE o.order_date >= CURRENT_DATE - INTERVAL '1 year'
   GROUP BY r.restaurant_name, r.city
)
SELECT restaurant_name, city,total_revenue,
   DENSE_RANK() OVER (
      PARTITION BY city
      ORDER BY total_revenue DESC
   ) AS revenue_rank
FROM restaurant_revenue
ORDER BY city, revenue_rank;
```

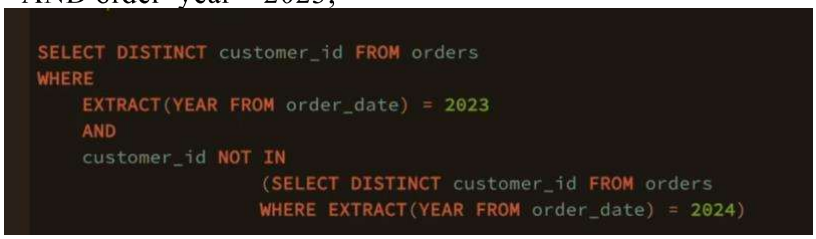
7. **Identify most popular dish by number of orders for each city**

```
WITH dish_counts AS (
   SELECT r.city, o.order_item,
      COUNT(*) AS order_count,
      DENSE_RANK() OVER (
         PARTITION BY r.city
         ORDER BY COUNT(*) DESC
      ) AS rn
   FROM orders o
```

```
    JOIN restaurants r ON o.restaurant_id = r.restaurant_id
    GROUP BY r.city, o.order_item
)
SELECT city, order_item AS most_popular_dish, order_count
FROM dish_counts
WHERE rn = 1;
```

8. **Find customers who placed order in 2023 but not in 2024**

```
WITH cte AS (
    SELECT customer_id, order_date,
        YEAR(order_date) AS order_year,
        ROW_NUMBER() OVER (
            PARTITION BY customer_id
            ORDER BY order_date DESC
        ) AS rn
    FROM orders
)
SELECT customer_id
FROM cte
WHERE rn = 1
  AND order_year = 2023;
```



```
SELECT DISTINCT customer_id FROM orders
WHERE
    EXTRACT(YEAR FROM order_date) = 2023
    AND
    customer_id NOT IN
                (SELECT DISTINCT customer_id FROM orders
                WHERE EXTRACT(YEAR FROM order_date) = 2024)
```

9. **Compare order cancellation rate for each restaurant between the current year and previous year**

```
WITH current_year AS (
    SELECT restaurant_id,
        SUM(order_status = 'cancelled') * 100.0 / COUNT(order_id) AS current_cancellation_rate
    FROM orders
    WHERE YEAR(order_date) = YEAR(CURDATE())
    GROUP BY restaurant_id
),
prev_year AS (
    SELECT restaurant_id,
        SUM(order_status = 'cancelled') * 100.0 / COUNT(order_id) AS prev_cancellation_rate
    FROM orders
    WHERE YEAR(order_date) = YEAR(CURDATE()) - 1
    GROUP BY restaurant_id
)
SELECT c.restaurant_id, c.current_cancellation_rate, p.prev_cancellation_rate,
    c.current_cancellation_rate - p.prev_cancellation_rate AS rate_change
FROM current_year c
JOIN prev_year p
    ON c.restaurant_id = p.restaurant_id;
```

Not good as table is scanned twice

```
SELECTrestaurant_id,

   -- current year cancellation rate
   SUM(order_status = 'cancelled' AND order_date >= DATE_FORMAT(CURDATE(), '%Y-01-
01') ) * 100.0
   / NULLIF( SUM(order_date >= DATE_FORMAT(CURDATE(), '%Y-01-01')), 0 ) AS
current_year_cancel_rate,

   -- previous year cancellation rate
   SUM(order_status = 'cancelled'
      AND order_date >= DATE_FORMAT(CURDATE() - INTERVAL 1 YEAR, '%Y-01-01')
       AND order_date <  DATE_FORMAT(CURDATE(), '%Y-01-01')
   ) * 100.0
   / NULLIF(
      SUM(order_date >= DATE_FORMAT(CURDATE() - INTERVAL 1 YEAR, '%Y-01-01')
          AND order_date <  DATE_FORMAT(CURDATE(), '%Y-01-01')),
       0
   ) AS prev_year_cancel_rate
FROM orders
GROUP BY restaurant_id;
```

10. Determine each rider's average delivery time

```
SELECT r.rider_id, r.rider_name,
   SEC_TO_TIME( AVG(TIME_TO_SEC(TIMEDIFF(d.delivery_time,o.order_time))))
AS avg_delivery_time
FROM deliveries d
JOIN orders o
   ON d.order_id = o.order_id
JOIN riders r
   ON d.rider_id = r.rider_id
WHERE d.delivery_status = 'Delivered'
GROUP BY r.rider_id, r.rider_name;
```

11. Calculate each restaurant's growth ration by number of orders placed since its joining

```
WITH cte AS (
   SELECT o.restaurant_id,
      MONTH(o.order_date) AS order_month,
      COUNT(o.order_id) AS orders_count
   FROM orders o
   JOIN deliveries d ON o.order_id = d.order_id
   WHERE d.delivery_status = 'Delivered'
   GROUP BY o.restaurant_id, order_month
)
SELECT restaurant_id, order_month,orders_count,
   LAG(orders_count) OVER (
      PARTITION BY restaurant_id
      ORDER BY order_month
   ) AS prev_month_orders,
   ROUND(
      (orders_count - LAG(orders_count) OVER (
```

```
        PARTITION BY restaurant_id
        ORDER BY order_month
    )) * 100.0
    / LAG(orders_count) OVER (
        PARTITION BY restaurant_id
        ORDER BY order_month
    ),2) AS mom_growth_pct
FROM cte;
```

12. **Customer segmentation :- segment customers into gold or silver bracket based on total spending**
    If total spending > aov then gold else silver
    For each segment find the total revenue and orders placed

```
WITH aov_cte AS (
    SELECT  AVG(total_amount) AS aov
    FROM orders
),
total_spending_cte AS (
    SELECT customer_id,
        SUM(total_amount) AS total_spending,
        COUNT(order_id) AS total_orders
    FROM orders
    GROUP BY customer_id
),
bracket_cte AS (
    SELECT ts.customer_id, ts.total_spending,ts.total_orders, a.aov,
        CASE WHEN ts.total_spending > a.aov THEN 'Gold'
            ELSE 'Silver'
        END AS bracket
    FROM total_spending_cte ts
    CROSS JOIN aov_cte a
)
SELECT bracket,
    SUM(total_spending) AS total_revenue,
    SUM(total_orders) AS total_orders
FROM bracket_cte
GROUP BY bracket;
```

13. **Calculate a rider's monthly income assuming they earn 8% of order amount**

```
SELECT d.rider_id, r.rider_name,
    MONTH(o.order_date) AS order_month,
    ROUND(SUM(o.total_amount * 0.08), 2) AS monthly_income
FROM deliveries d
JOIN orders o ON d.order_id = o.order_id
JOIN riders r ON d.rider_id = r.rider_id
WHERE d.delivery_status = 'Delivered'
GROUP BY d.rider_id, r.rider_name, order_month
ORDER BY d.rider_id, order_month;
```

14. **Rider Rating --> find number of 5-4-3 star rating each driver has**
    delivery_time<15 min 5 star rating

15-20 min 4 star rating
20 min 3 star rating

```
WITH cte AS (
   SELECT o.order_id, d.rider_id,o.order_time,d.delivery_time,
      CASE WHEN TIMESTAMPDIFF(MINUTE, o.order_time, d.delivery_time) < 15
 THEN '5 star'
         WHEN TIMESTAMPDIFF(MINUTE, o.order_time, d.delivery_time) BETWEEN 15
AND 20 THEN '4 star'
        WHEN TIMESTAMPDIFF(MINUTE, o.order_time, d.delivery_time) > 20 THEN '3 star'
         ELSE 'bad performance'
      END AS rating
   FROM orders o
   JOIN deliveries ON o.order_id = d.order_id
   WHERE d.delivery_status = 'Delivered'
)
SELECT rider_id,
   SUM(rating = '5 star') AS five_star_count,
   SUM(rating = '4 star') AS four_star_count,
   SUM(rating = '3 star') AS three_star_count
FROM cte
GROUP BY rider_id;
```

15. **Identify the order frequency per day of the week and identify the peak day for each restaurant**

```
WITH daily_orders AS (
   SELECT o.restaurant_id,
      DAYNAME(o.order_date) AS day_of_week,
      COUNT(o.order_id) AS order_count
   FROM orders o
   JOIN deliveries d ON o.order_id = d.order_id
   WHERE d.delivery_status = 'Delivered'
   GROUP BY o.restaurant_id, DAYOFWEEK(o.order_date),DAYNAME(o.order_date)
),
ranked_days AS (
   SELECT restaurant_id, day_of_week,order_count,
      DENSE_RANK() OVER (
         PARTITION BY restaurant_id
         ORDER BY order_count DESC
      ) AS rnk
   FROM daily_orders
)
SELECT rd.restaurant_id, r.restaurant_name,rd.day_of_week AS peak_day,
   rd.order_count AS peak_orders
FROM ranked_days rd
JOIN restaurants r ON rd.restaurant_id = r.restaurant_id
WHERE rnk = 1
ORDER BY rd.restaurant_id;
```

16. **Calculate month-on-month growth in total orders and identify the top 3 cities contributing to the growth or decline.**

```
WITH monthly_city_orders AS (
```

```
    SELECT o.city,
        MONTH(o.order_date) AS order_month,
        COUNT(o.order_id) AS total_orders
    FROM orders o
    JOIN deliveries d ON o.order_id = d.order_id
    WHERE d.delivery_status = 'Delivered'
    GROUP BY o.city, order_month
),
mom_change AS (
    SELECT city,order_month, total_orders,
        total_orders - LAG(total_orders) OVER (
            PARTITION BY city
            ORDER BY order_month
        ) AS order_change
    FROM monthly_city_orders
)
SELECT order_month,city,order_change,
    RANK() OVER (
        PARTITION BY order_month
        ORDER BY order_change DESC
    ) AS growth_rank,
    RANK() OVER (
        PARTITION BY order_month
        ORDER BY order_change ASC
    ) AS decline_rank
FROM mom_change
WHERE order_change IS NOT NULL
  AND (
      RANK() OVER (
          PARTITION BY order_month
          ORDER BY order_change DESC
      ) <= 3
   OR RANK() OVER (
          PARTITION BY order_month
          ORDER BY order_change ASC
      ) <= 3
  )
ORDER BY order_month, order_change DESC;
```

17. **For each restaurant, find the peak ordering day of the week and the percentage of weekly orders that occur on that day.**

```
WITH daily_orders AS (
    SELECT o.restaurant_id,
        DAYNAME(o.order_date) AS day_of_week,
        COUNT(o.order_id) AS order_count
    FROM orders o
    JOIN deliveries  ON o.order_id = d.order_id
    WHERE d.delivery_status = 'Delivered'
    GROUP BY o.restaurant_id, DAYOFWEEK(o.order_date), DAYNAME(o.order_date)
),
weekly_totals AS (
```

```
    SELECT  restaurant_id,
        SUM(order_count) AS weekly_orders
    FROM daily_orders
    GROUP BY restaurant_id
),
ranked_days AS (
    SELECT d.restaurant_id,d.day_of_week, d.order_count,w.weekly_orders,
        RANK() OVER (
            PARTITION BY d.restaurant_id
            ORDER BY d.order_count DESC
        ) AS rnk
    FROM daily_orders d
    JOIN weekly_totals w ON d.restaurant_id = w.restaurant_id
)
SELECT rd.restaurant_id, r.restaurant_name rd.day_of_week AS peak_day,
rd.order_count AS peak_day_orders,
    ROUND(rd.order_count * 100.0 / rd.weekly_orders, 2) AS peak_day_percentage
FROM ranked_days rd
JOIN restaurants r ON rd.restaurant_id = r.restaurant_id
WHERE rd.rnk = 1
ORDER BY rd.restaurant_id;
```

18. Compute monthly active customers (MAC) and identify MoM churned customers.

```
WITH monthly_customers AS (
    SELECT DISTINCT o.customer_id,
        MONTH(o.order_date) AS order_month
    FROM orders o
    JOIN deliveries d ON o.order_id = d.order_id
    WHERE d.delivery_status = 'Delivered'
)
SELECT m1.customer_id,
    DATE_ADD(m1.order_month, INTERVAL 1 MONTH) AS churn_month
FROM monthly_customers m1
LEFT JOIN monthly_customers m2 ON m1.customer_id = m2.customer_id
   AND m2.order_month = DATE_ADD(m1.order_month, INTERVAL 1 MONTH)
WHERE m2.customer_id IS NULL
ORDER BY churn_month, m1.customer_id;
```

19. Identify new vs repeat customers per month and calculate their revenue contribution.

```
WITH first_order AS (
    SELECT customer_id,
        MIN(MONTH(order_Date)) AS first_order_month
    FROM orders
    GROUP BY customer_id
),
monthly_revenue AS (
    SELECT DATE_FORMAT(order_date, '%Y-%m-01') AS order_month,
        customer_id,
        SUM(total_amount) AS revenue
    FROM orders
    GROUP BY order_month, customer_id
```

```
),
classified_revenue AS (
   SELECT mr.order_month,
      CASE WHEN mr.order_month = fo.first_order_month THEN 'New'
         ELSE 'Repeat'
      END AS customer_type,
      mr.revenue
   FROM monthly_revenue mr
   JOIN first_order fo ON mr.customer_id = fo.customer_id
)
SELECT order_month, customer_type,
   ROUND(
      SUM(revenue) * 100.0 /
      SUM(SUM(revenue)) OVER (PARTITION BY order_month), 2 ) AS revenue_percentage
FROM classified_revenue
GROUP BY order_month, customer_type
ORDER BY order_month, customer_type;
```

20. Perform customer cohort analysis showing retention for Month-0, Month-1, Month-2.

```
WITH first_order AS (
   SELECT customer_id,
      MONTH(MIN(order_date)) AS cohort_month
   FROM orders
   GROUP BY customer_id
),
monthly_activity AS (
   SELECT DISTINCT o.customer_id,
     MONTH(o.order_date) AS activity_month
   FROM orders o
),
cohort_activity AS (
   SELECT f.cohort_month, f.customer_id,
      TIMESTAMPDIFF( MONTH, f.cohort_month, m.activity_month) AS month_number
   FROM first_order f
   JOIN monthly_activity m ON f.customer_id = m.customer_id
   WHERE TIMESTAMPDIFF(MONTH,f.cohort_month,m.activity_month) BETWEEN 0 AND 2
)
SELECT cohort_month,
   COUNT(DISTINCT CASE WHEN month_number = 0 THEN customer_id END) AS month_0,
   COUNT(DISTINCT CASE WHEN month_number = 1 THEN customer_id END) AS month_1,
   COUNT(DISTINCT CASE WHEN month_number = 2 THEN customer_id END) AS month_2
FROM cohort_activity
GROUP BY cohort_month
ORDER BY cohort_month;
```

21. Calculate restaurant churn: restaurants that had orders in one month but none in the following month.

```
WITH monthly_restaurants AS (
   SELECT DISTINCT restaurant_id,
     MONTH(order_date) AS order_month
   FROM orders
```

```
),
restaurant_churn AS (
   SELECT m1.restaurant_id, m1.order_month AS active_month,
      DATE_ADD(m1.order_month, INTERVAL 1 MONTH) AS churn_month
   FROM monthly_restaurants m1
   LEFT JOIN monthly_restaurants m2 ON m1.restaurant_id = m2.restaurant_id
     AND m2.order_month = DATE_ADD(m1.order_month, INTERVAL 1 MONTH)
   WHERE m2.restaurant_id IS NULL
)
SELECT churn_month,
   COUNT(DISTINCT restaurant_id) AS churned_restaurants
FROM restaurant_churn
GROUP BY churn_month
ORDER BY churn_month;
```

22. Determine cancellation rate per restaurant per month and identify restaurants with a consistent increase in cancellations.

```
WITH monthly_orders AS (
   SELECT restaurant_id,
      MONTH(order_date),
      COUNT(*) AS total_orders,
      SUM( order_status = 'Cancelled' ) AS cancelled_orders
   FROM orders
   GROUP BY restaurant_id, order_month
)
SELECT restaurant_id,order_month,cancelled_orders,total_orders,
   ROUND(cancelled_orders * 1.0 / total_orders, 3) AS cancellation_rate
FROM monthly_orders
ORDER BY restaurant_id, order_month;
```

23. For each city, identify the peak ordering hour per month.

```
WITH hourly_orders AS (
   SELECT city,
      MONTH(order_time) as month,
      HOUR(order_time) AS order_hour,
      COUNT(*) AS order_count
   FROM orders
   GROUP BY city, order_month, order_hour
),
ranked_hours AS (
   SELECT city,order_month,order_hour, order_count,
      DENSE_RANK() OVER (
         PARTITION BY city, order_month
         ORDER BY order_count DESC
      ) AS rnk
   FROM hourly_orders
)
SELECT City,   order_month,order_hour AS peak_hour,order_count
FROM ranked_hours
WHERE rnk = 1
ORDER BY city, order_month;
```

24. Calculate rider utilization metrics: deliveries per active day per rider per month.

```
WITH daily_rider_activity AS (
   SELECT d.rider_id,
      DATE(d.delivery_time) AS delivery_date,
      COUNT(*) AS deliveries_per_day
   FROM deliveries d
   WHERE d.delivery_status = 'Delivered'
   GROUP BY d.rider_id, DATE(d.delivery_time)
),
monthly_rider_metrics AS (
   SELECT rider_id,
      MONTH(delivery_date) AS delivery_month,
      SUM(deliveries_per_day) AS total_deliveries,
      COUNT(*) AS active_days
   FROM daily_rider_activity
   GROUP BY rider_id, delivery_month
)
SELECT rider_id, delivery_month, total_deliveries,active_days,
   ROUND(total_deliveries * 1.0 / active_days, 2) AS deliveries_per_active_day
FROM monthly_rider_metrics
ORDER BY rider_id, delivery_month;
```

25. Compare weekend vs weekday performance for restaurants and identify those heavily weekend-dependent (>60%).

```
With count as(
Select orders_id, restaurant_id, dayname(order_date)  as day, count(order_id) as orders_count
From orders
Group by restaurant_id , day,  orders_id
)
Select *,
Sum(Case when day=”sunday” or day=”saturday” then amount else 0) as weekend_orders,
 Sum(Case when day not in (“sunday”,”saturday”)  then amount else 0) as weekday_orders
Group by restaurant_id
```

26. Identify customers who were in the top 10% spenders in one month but dropped out of top 30% in the next month.

```
WITH monthly_spend AS (
   SELECT customer_id, MONTH( order_date) as month,
      SUM(order_amount) AS total_spend
   FROM orders
   GROUP BY customer_id, month
),
ranked AS (
   SELECT customer_id, month,total_spend,
      NTILE(10) OVER (
         PARTITION BY month
         ORDER BY total_spend DESC
      ) AS top_10_bucket,
      NTILE(3) OVER (
         PARTITION BY month
```

```
        ORDER BY total_spend DESC
      ) AS top_30_bucket
  FROM monthly_spend
)
SELECT r1.customer_id, r1.month AS high_spend_month, r2.month AS drop_month
FROM ranked r1
JOIN ranked r ON r1.customer_id = r2.customer_id
AND r2.month = DATE_FORMAT(
      DATE_ADD(CONCAT(r1.month, '-01'), INTERVAL 1 MONTH),
      '%Y-%m'
   )
WHERE r1.top_10_bucket = 1
  AND r2.top_30_bucket <> 1;
```

27. Identify restaurants with declining order trends over the last 3 months.

```
WITH monthly_orders AS (
  SELECT restaurant_id,
    MONTH(order_date) as month,
    COUNT(order_id) AS order_cnt
  FROM orders
  GROUP BY restaurant_id, month
),
trend_check AS (
  SELECT restaurant_id, month,order_cnt,
    LAG(order_cnt, 1) OVER (
      PARTITION BY restaurant_id
      ORDER BY month
    ) AS prev_1_month,
    LAG(order_cnt, 2) OVER (
      PARTITION BY restaurant_id
      ORDER BY month
    ) AS prev_2_month
  FROM monthly_orders
)
SELECT DISTINCT
  restaurant_id
FROM trend_check
WHERE prev_2_month > prev_1_month
  AND prev_1_month > order_cnt;
```

28. For each rider, compute monthly earnings and identify riders with unstable income patterns.

```
WITH monthly_earnings AS (
  SELECT rider_id,
    DATE_FORMAT(delivery_date, '%Y-%m') AS month,
    SUM(earning_amount) AS monthly_income
  FROM deliveries
  GROUP BY rider_id, month
),
mom_change AS (
  SELECT rider_id, month, monthly_income,
```

```
        LAG(monthly_income) OVER (
            PARTITION BY rider_id
            ORDER BY month
        ) AS prev_income
    FROM monthly_earnings
)
SELECT DISTINCTrider_id
FROM mom_change
WHERE prev_income IS NOT NULL
  AND (monthly_income - prev_income) / prev_income <= -0.25;
```

29. Customers who ordered every month in the last 6 months

```
WITH last_6_months AS (
    SELECT customer_id,
        MONTH(order_date) AS month
    FROM orders
    WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
),
monthly_presence AS (
    SELECTcustomer_id,
        COUNT(DISTINCT month) AS active_months
    FROM last_6_months
    GROUP BY customer_id
)
SELECT customer_id
FROM monthly_presence
WHERE active_months = 6;
```

30. Customers whose avg order value increased for 3 consecutive months

```
WITH monthly_aov AS (
    SELECT customer_id, MONTH(order_date) as month,
        AVG(order_amount) AS aov
    FROM orders
    GROUP BY customer_id, month
),
trend_check AS (
    SELECT customer_id, month, aov,
        LAG(aov, 1) OVER (
            PARTITION BY customer_id
            ORDER BY month
        ) AS prev_1_aov,
        LAG(aov, 2) OVER (
            PARTITION BY customer_id
            ORDER BY month
        ) AS prev_2_aov
```

```
    FROM monthly_aov
)
SELECT DISTINCT  customer_id
FROM trend_check
WHERE prev_2_aov < prev_1_aov
  AND prev_1_aov < aov;
```

31. Customers contributing top 20% of revenue but bottom 50% of order count

```
WITH customer_metrics AS (
    SELECT customer_id,
        SUM(total_amount) AS total_spend,
        COUNT(order_id) AS total_orders
    FROM orders
    GROUP BY customer_id
),
ranked AS (
    SELECT customer_id,total_spend,total_orders,
        NTILE(5) OVER (ORDER BY total_spend DESC) AS revenue_bucket,
        NTILE(2) OVER (ORDER BY total_orders ASC) AS order_bucket
    FROM customer_metrics
)
SELECT customer_id
FROM ranked
WHERE revenue_bucket = 1    -- top 20% revenue
  AND order_bucket = 1;     -- bottom 50% order count
```

32. Restaurants that lost top customers MoM

```
WITH monthly_spend AS (
    SELECT
        restaurant_id,
        customer_id,
        DATE_FORMAT(order_date, '%Y-%m-01') AS month_start,
        SUM(order_amount) AS total_spend
    FROM orders
    GROUP BY restaurant_id, customer_id, month_start
),
ranked AS (
    SELECT
        restaurant_id,
        customer_id,
        month_start,
        total_spend,
```

```
        NTILE(10) OVER (
            PARTITION BY restaurant_id, month_start
            ORDER BY total_spend DESC
        ) AS spend_bucket
    FROM monthly_spend
),
top_customers AS (
    SELECT
        restaurant_id,
        customer_id,
        month_start
    FROM ranked
    WHERE spend_bucket = 1
)
SELECT DISTINCT
    t.restaurant_id
FROM top_customers t
LEFT JOIN top_customers t_next
  ON t.restaurant_id = t_next.restaurant_id
AND t.customer_id = t_next.customer_id
AND t_next.month_start = DATE_ADD(t.month_start, INTERVAL 1 MONTH)
WHERE t_next.customer_id IS NULL;
```

33. Riders with increasing delivery time trend

```
WITH monthly_delivery_time AS (
    SELECT rider_id,
        DATE_FORMAT(order_date, '%Y-%m-01') AS month_start,
        AVG(delivery_time_minutes) AS avg_delivery_time
    FROM deliveries
    GROUP BY rider_id, month_start
),
trend_check AS (
    SELECT rider_id, month_start, avg_delivery_time,
        LAG(avg_delivery_time, 1) OVER (
            PARTITION BY rider_id
            ORDER BY month_start
        ) AS prev_1,
        LAG(avg_delivery_time, 2) OVER (
            PARTITION BY rider_id
            ORDER BY month_start
        ) AS prev_2
    FROM monthly_delivery_time
)
```

```
SELECT DISTINCT  rider_id
FROM trend_check
WHERE prev_2 < prev_1
  AND prev_1 < avg_delivery_time;
```

34. Longest ordering streak per customer

```
WITH daily_orders AS (
   SELECT DISTINCT customer_id,
      DATE(order_date) AS order_day
   FROM orders
),
grp AS (
   SELECT customer_id,order_day,
      DATE_SUB( order_day,
         INTERVAL ROW_NUMBER() OVER (
            PARTITION BY customer_id
            ORDER BY order_day
         ) DAY
      ) AS grp_key
   FROM daily_orders
)
SELECT customer_id,
   MAX(streak_len) AS longest_streak
FROM (
   SELECT customer_id,grp_key,
      COUNT(*) AS streak_len
   FROM grp
   GROUP BY customer_id, grp_key
) t
GROUP BY customer_id;
```

35. Customers who **never reordered from the same restaurant**

```
WITH customer_restaurant_counts AS (
   SELECT
      customer_id,
      restaurant_id,
      COUNT(order_id) AS order_count
   FROM orders
   GROUP BY customer_id, restaurant_id
)
SELECT DISTINCT
   customer_id
FROM customer_restaurant_counts
```

```
GROUP BY customer_id
HAVING MAX(order_count) = 1;
```

36. Customers who switched **primary restaurant** MoM

```
WITH monthly_primary AS (
   SELECT
      customer_id,
      DATE_FORMAT(order_date, '%Y-%m-01') AS month_start,
      restaurant_id,
      RANK() OVER (
         PARTITION BY customer_id, DATE_FORMAT(order_date, '%Y-%m-01')
         ORDER BY COUNT(order_id) DESC
      ) AS rk
   FROM orders
   GROUP BY customer_id, month_start, restaurant_id
),
primary_restaurant AS (
   SELECT customer_id, month_start, restaurant_id
   FROM monthly_primary
   WHERE rk = 1
)
SELECT
   curr.customer_id,
   curr.month_start AS current_month,
   curr.restaurant_id AS current_restaurant,
   prev.restaurant_id AS prev_restaurant
FROM primary_restaurant curr
JOIN primary_restaurant prev
  ON curr.customer_id = prev.customer_id
AND curr.month_start = DATE_ADD(prev.month_start, INTERVAL 1 MONTH)
WHERE curr.restaurant_id <> prev.restaurant_id;
```

37. Restaurants that **lost >30% of their top customers MoM**

```
WITH monthly_spend AS (
   SELECT
      restaurant_id,
      customer_id,
      DATE_FORMAT(order_date, '%Y-%m-01') AS month_start,
      SUM(order_amount) AS total_spend
   FROM orders
   GROUP BY restaurant_id, customer_id, month_start
),
ranked AS (
   SELECT
```

```
        restaurant_id,
        customer_id,
        month_start,
        total_spend,
       NTILE(10) OVER (PARTITION BY restaurant_id, month_start ORDER BY total_spend DESC)
AS spend_bucket
    FROM monthly_spend
),
top_customers AS (
    SELECT restaurant_id, customer_id, month_start
    FROM ranked
    WHERE spend_bucket = 1
),
next_month_check AS (
    SELECT
        t.restaurant_id,
        t.month_start,
        COUNT(*) AS top_customers_count,
        SUM(CASE WHEN t_next.customer_id IS NULL THEN 1 ELSE 0 END) AS lost_count
    FROM top_customers t
    LEFT JOIN top_customers t_next
      ON t.restaurant_id = t_next.restaurant_id
     AND t.customer_id = t_next.customer_id
     AND t_next.month_start = DATE_ADD(t.month_start, INTERVAL 1 MONTH)
    GROUP BY t.restaurant_id, t.month_start
)
SELECT restaurant_id, month_start, lost_count, top_customers_count
FROM next_month_check
WHERE lost_count / top_customers_count > 0.3;
```

38. Riders who churned after consistent activity

```
WITH monthly_activity AS (
    SELECT rider_id,
        DATE_FORMAT(order_date, '%Y-%m-01') AS month_start,
        COUNT(order_id) AS orders_in_month
    FROM deliveries
    GROUP BY rider_id, month_start
),
activity_lag AS (
    SELECT rider_id, month_start, orders_in_month,
        LAG(orders_in_month, 1) OVER (PARTITION BY rider_id ORDER BY month_start) AS
prev_month,
        LAG(orders_in_month, 2) OVER (PARTITION BY rider_id ORDER BY month_start) AS
prev_2_month
```

```
    FROM monthly_activity
)
SELECT DISTINCT rider_id
FROM activity_lag
WHERE prev_2_month > 0
  AND prev_month > 0
  AND orders_in_month = 0;
```