

Master SQL for Product-Based Interviews One Guide to Crack **MAANG**

1. Find top 3 customers who contributed the most to revenue each month.

Tables: transactions(customer_id, amount, txn_date)

Answer:

```
WITH monthly_revenue AS (  
    SELECT customer_id,  
           DATE_TRUNC('month', txn_date) AS month,  
           SUM(amount) AS total_spent  
    FROM transactions  
    GROUP BY customer_id, DATE_TRUNC('month', txn_date)  
)  
ranked AS (  
    SELECT *,  
           RANK() OVER (PARTITION BY month ORDER BY total_spent DESC) AS rnk  
    FROM monthly_revenue  
)  
SELECT month, customer_id, total_spent  
FROM ranked  
WHERE rnk <= 3  
ORDER BY month, total_spent DESC;
```

Explanation:

- Aggregate spend per customer per month.
- Rank them using RANK().
- Pick top 3 per month.

2. Find employees who earn more than their manager.

Table: employees(emp_id, emp_name, salary, manager_id)

Answer:

```
SELECT e.emp_id, e.emp_name, e.salary, m.emp_name AS manager, m.salary AS
manager_salary
FROM employees e
JOIN employees m ON e.manager_id = m.emp_id
WHERE e.salary > m.salary;
```

Explanation: Self-join employees table on manager, compare salaries.

3. Calculate customer churn (customers active in previous month but not in current month).

Tables: transactions(customer_id, txn_date, amount)

Answer:

```
WITH monthly_customers AS (
    SELECT DISTINCT customer_id, DATE_TRUNC('month', txn_date) AS month
    FROM transactions
),
churn AS (
    SELECT prev.customer_id, prev.month AS churn_month
    FROM monthly_customers prev
    LEFT JOIN monthly_customers curr
    ON prev.customer_id = curr.customer_id
    AND curr.month = prev.month + INTERVAL '1 month'
    WHERE curr.customer_id IS NULL
)
SELECT * FROM churn;
```

Explanation:

- Capture customers per month.
- Self-join previous month to next month.
- If no match → churned.

4. Detect products that were never sold in consecutive months.

Tables: sales(product_id, txn_date, amount)

Answer:

```
WITH monthly_sales AS (  
    SELECT product_id, DATE_TRUNC('month', txn_date) AS month  
    FROM sales  
    GROUP BY product_id, DATE_TRUNC('month', txn_date)  
),  
gaps AS (  
    SELECT product_id, month,  
           LAG(month) OVER (PARTITION BY product_id ORDER BY month) AS prev_month  
    FROM monthly_sales  
)  
SELECT product_id, month  
FROM gaps  
WHERE prev_month IS NOT NULL  
      AND DATE_PART('month', month) - DATE_PART('month', prev_month) > 1;
```

Explanation: Use LAG() to check if any gap exists between sales months.

5. Running total + YoY growth in same query.

Tables: transactions(customer_id, amount, txn_date)

Answer:

```
WITH yearly AS (  
    SELECT EXTRACT(YEAR FROM txn_date) AS year,  
           SUM(amount) AS total_revenue  
    FROM transactions  
    GROUP BY EXTRACT(YEAR FROM txn_date)  
)  
SELECT year,  
       total_revenue,  
       SUM(total_revenue) OVER (ORDER BY year) AS running_total,  
       LAG(total_revenue) OVER (ORDER BY year) AS prev_year_revenue,  
       ROUND((((total_revenue - LAG(total_revenue) OVER (ORDER BY year))  
              / NULLIF(LAG(total_revenue) OVER (ORDER BY year), 0)) * 100, 2) AS yoy_growth  
FROM yearly;
```

Explanation:

- Aggregate yearly revenue.
 - SUM() OVER → running total.
 - LAG() → previous year revenue.
 - Compute YoY growth %.
-

6. Find the second order of each customer (by date).

Table: orders(order_id, customer_id, order_date, amount)

Answer:

```
WITH ranked_orders AS (  
    SELECT customer_id, order_id, order_date,  
           ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date) AS rn  
    FROM orders  
)  
SELECT customer_id, order_id, order_date  
FROM ranked_orders  
WHERE rn = 2;
```

Explanation: Assign order numbers per customer using ROW_NUMBER(). Select where order = 2.

7. Find customers who placed orders in 3 consecutive months.

Table: transactions(customer_id, txn_date, amount)

Answer:

```
WITH monthly AS (  
    SELECT DISTINCT customer_id, DATE_TRUNC('month', txn_date) AS month  
    FROM transactions  
)  
ranked AS (  
    SELECT customer_id, month,  
           ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY month) AS rn  
    FROM monthly  
)  
SELECT m1.customer_id  
FROM ranked m1  
JOIN ranked m2 ON m1.customer_id = m2.customer_id AND m2.rn = m1.rn + 1  
JOIN ranked m3 ON m1.customer_id = m3.customer_id AND m3.rn = m1.rn + 2;
```

Explanation: Use consecutive row numbers to detect 3-month streak.

8. Find products with the highest sales in each category.

Table: products(product_id, category, price, sales)

Answer:

```
SELECT product_id, category, sales
FROM (
    SELECT product_id, category, sales,
           RANK() OVER (PARTITION BY category ORDER BY sales DESC) AS rnk
    FROM products
) ranked
WHERE rnk = 1;
```

Explanation: Partition by category and rank sales → pick top 1 per category.

9. Detect users with transactions from two different countries on the same day.

Table: transactions(customer_id, txn_date, country, amount)

Answer:

```
SELECT customer_id, txn_date
FROM transactions
GROUP BY customer_id, txn_date
HAVING COUNT(DISTINCT country) > 1;
```

Explanation: Group by customer + date → if more than 1 country → suspicious activity.

10. Find customers who never placed an order.

Tables:

- customers(customer_id, name)
- orders(order_id, customer_id, order_date)

Answer:

```
SELECT c.customer_id, c.name
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;
```

Explanation: Use LEFT JOIN with null check to detect non-buyers.

11. Find top 2 products by revenue in each month.

Table: sales(product_id, txn_date, amount)

Answer:

```
WITH monthly_sales AS (
    SELECT product_id, DATE_TRUNC('month', txn_date) AS month, SUM(amount) AS revenue
    FROM sales
    GROUP BY product_id, DATE_TRUNC('month', txn_date)
),
ranked AS (
    SELECT *, RANK() OVER (PARTITION BY month ORDER BY revenue DESC) AS rnk
    FROM monthly_sales
)
SELECT month, product_id, revenue
FROM ranked
WHERE rnk <= 2;
```

Explanation: Summarize monthly revenue, rank, pick top 2.

12. Find employees whose salary is above company average but below department average.

Table: employees(emp_id, dept_id, salary)

Answer:

```
SELECT e.emp_id, e.salary, e.dept_id
FROM employees e
WHERE e.salary > (SELECT AVG(salary) FROM employees)
AND e.salary < (
    SELECT AVG(salary)
    FROM employees d
    WHERE d.dept_id = e.dept_id
);
```

Explanation: Compare salary with both global and department averages using correlated subqueries.

13. Calculate order repeat rate (customers with >1 order / total customers).

Table: orders(order_id, customer_id, order_date)

Answer:

```
SELECT (COUNT(DISTINCT CASE WHEN order_count > 1 THEN customer_id END) * 100.0) /
    COUNT(DISTINCT customer_id) AS repeat_rate
FROM (
    SELECT customer_id, COUNT(*) AS order_count
    FROM orders
    GROUP BY customer_id
) t;
```

Explanation: Count customers with multiple orders and divide by total customers.

14. Find the first transaction of each customer and the difference from their last transaction.

Table: transactions(customer_id, txn_date, amount)

Answer:

```
SELECT customer_id,  
       MIN(txn_date) AS first_txn,  
       MAX(txn_date) AS last_txn,  
       AGE(MAX(txn_date), MIN(txn_date)) AS duration  
FROM transactions  
  
GROUP BY customer_id;
```

Explanation: Use MIN() and MAX() to calculate first and last transactions, then subtract dates.

15. Identify the top 5% of customers by lifetime spend.

Table: transactions(customer_id, amount)

Answer:

```
WITH customer_spend AS (  
    SELECT customer_id, SUM(amount) AS total_spent  
    FROM transactions  
    GROUP BY customer_id  
)  
  
ranked AS (  
    SELECT customer_id, total_spent,  
           PERCENT_RANK() OVER (ORDER BY total_spent DESC) AS pct_rank  
    FROM customer_spend  
)  
  
SELECT customer_id, total_spent  
FROM ranked  
WHERE pct_rank <= 0.05;
```

Explanation: Use PERCENT_RANK() to classify spenders. Top 5% are elite customers.

16. Find customers who made transactions in every month of 2024.

Table: transactions(customer_id, txn_date, amount)

Answer:

```
SELECT customer_id
```

```
FROM transactions
```

```
WHERE EXTRACT(YEAR FROM txn_date) = 2024
```

```
GROUP BY customer_id
```

```
HAVING COUNT(DISTINCT EXTRACT(MONTH FROM txn_date)) = 12;
```

Explanation: Count distinct months per customer. If 12 → active every month.

17. Detect revenue drop compared to previous month.

Table: transactions(txn_date, amount)

Answer:

```
WITH monthly AS (
```

```
    SELECT DATE_TRUNC('month', txn_date) AS month, SUM(amount) AS revenue
```

```
    FROM transactions
```

```
    GROUP BY DATE_TRUNC('month', txn_date)
```

```
)
```

```
SELECT month, revenue,
```

```
       LAG(revenue) OVER (ORDER BY month) AS prev_revenue,
```

```
       revenue - LAG(revenue) OVER (ORDER BY month) AS diff
```

```
FROM monthly
```

```
WHERE revenue < LAG(revenue) OVER (ORDER BY month);
```

Explanation: Use LAG() to compare revenue with previous month.

18. Find users who never skipped a day of transactions.

Table: transactions(customer_id, txn_date, amount)

Answer:

WITH days AS (

SELECT customer_id, COUNT(DISTINCT txn_date) AS active_days,

MAX(txn_date) - MIN(txn_date) + 1 AS total_days

FROM transactions

GROUP BY customer_id

)

SELECT customer_id

FROM days

WHERE active_days = total_days;

Explanation: If active days = total days in range → no skips.

19. Detect customers who upgraded spending month-over-month (strictly increasing).

Table: transactions(customer_id, txn_date, amount)

Answer:

WITH monthly AS (

```
    SELECT customer_id, DATE_TRUNC('month', txn_date) AS month, SUM(amount) AS  
revenue
```

```
    FROM transactions
```

```
    GROUP BY customer_id, DATE_TRUNC('month', txn_date)
```

```
),
```

ranked AS (

```
    SELECT customer_id, month, revenue,
```

```
        LAG(revenue) OVER (PARTITION BY customer_id ORDER BY month) AS prev_revenue
```

```
    FROM monthly
```

```
)
```

```
SELECT DISTINCT customer_id
```

```
FROM ranked
```

```
GROUP BY customer_id
```

```
HAVING MIN(CASE WHEN revenue > prev_revenue OR prev_revenue IS NULL THEN 1 ELSE 0  
END) = 1
```

```
AND MAX(CASE WHEN revenue <= prev_revenue THEN 1 ELSE 0 END) = 0;
```

Explanation: Ensures revenue always increases month-to-month.

20. Find the most common sequence of two products bought together.

Table: orders(order_id, customer_id, product_id, order_date)

Answer:

```
WITH ordered AS (  
    SELECT customer_id, product_id,  
           ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date) AS rn  
    FROM orders  
)  
  
pairs AS (  
    SELECT o1.customer_id, o1.product_id AS first_product, o2.product_id AS second_product  
    FROM ordered o1  
    JOIN ordered o2 ON o1.customer_id = o2.customer_id AND o2.rn = o1.rn + 1  
)  
  
SELECT first_product, second_product, COUNT(*) AS pair_count  
FROM pairs  
GROUP BY first_product, second_product  
ORDER BY pair_count DESC  
LIMIT 1;
```

Explanation: Finds most frequent consecutive product pair.

21. Identify employees who directly or indirectly report to a given manager.

Table: employees(emp_id, emp_name, manager_id)

Answer:

```
WITH RECURSIVE subordinates AS (  
    SELECT emp_id, emp_name, manager_id  
    FROM employees  
    WHERE manager_id = 101 -- given manager  
    UNION ALL  
    SELECT e.emp_id, e.emp_name, e.manager_id  
    FROM employees e  
    JOIN subordinates s ON e.manager_id = s.emp_id  
)  
SELECT * FROM subordinates;
```

Explanation: Recursive CTE traverses hierarchy from manager down.

22. Find the longest streak of consecutive login days per user.

Table: logins(user_id, login_date)

Answer:

```
WITH numbered AS (  
    SELECT user_id, login_date,  
           ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY login_date) AS rn  
    FROM logins  
)  
  
grouped AS (  
    SELECT user_id, login_date, login_date - rn * INTERVAL '1 day' AS grp  
    FROM numbered  
)  
  
SELECT user_id, COUNT(*) AS streak  
FROM grouped  
GROUP BY user_id, grp  
ORDER BY streak DESC  
LIMIT 1;
```

Explanation: Trick: (login_date - rn) groups consecutive days. Count gives streak length.

23. Find customers whose spend contributes to 80% of total revenue (Pareto analysis).

Table: transactions(customer_id, amount)

Answer:

WITH ranked AS (

```
    SELECT customer_id, SUM(amount) AS total_spent,  
           SUM(SUM(amount)) OVER () AS grand_total,  
           SUM(SUM(amount)) OVER (ORDER BY SUM(amount) DESC) AS running_total
```

```
    FROM transactions
```

```
    GROUP BY customer_id
```

```
)
```

```
SELECT customer_id, total_spent
```

```
FROM ranked
```

```
WHERE running_total <= 0.8 * grand_total;
```

Explanation: Running sum of spend, cut at 80% → top contributors.

24. Detect orders placed using more than one payment method.

Table: payments(order_id, payment_method)

Answer:

```
SELECT order_id
```

```
FROM payments
```

```
GROUP BY order_id
```

```
HAVING COUNT(DISTINCT payment_method) > 1;
```

Explanation: Group orders, check multiple distinct payment methods.

25. Calculate rolling 3-month retention rate.

Tables: transactions(customer_id, txn_date)

Answer:

```
WITH monthly AS (  
    SELECT DISTINCT customer_id, DATE_TRUNC('month', txn_date) AS month  
    FROM transactions  
)  
  
cohort AS (  
    SELECT m1.customer_id, m1.month AS start_month, m2.month AS active_month  
    FROM monthly m1  
    JOIN monthly m2  
    ON m1.customer_id = m2.customer_id  
    AND m2.month BETWEEN m1.month AND m1.month + INTERVAL '2 month'  
)  
  
SELECT start_month,  
    COUNT(DISTINCT customer_id) AS cohort_size,  
    COUNT(DISTINCT CASE WHEN active_month = start_month + INTERVAL '2 month' THEN  
customer_id END) AS retained_after_3_months,  
    (COUNT(DISTINCT CASE WHEN active_month = start_month + INTERVAL '2 month' THEN  
customer_id END) * 100.0 /  
    COUNT(DISTINCT customer_id)) AS retention_rate  
FROM cohort  
GROUP BY start_month;
```

Explanation: Tracks cohorts, checks how many remain active after 3 months → retention %.
