SQL Query Interview Examples

# EXAMPLE 1

Given two tables:

* Transactions - transaction\_id, customer\_id, date, product\_id, quantity
* Product - product\_id, product\_category, unit\_price

## Q1 - Top 3 product categories by revenue last month

Clarify: What defines "last month"? Calendar month or rolling 30 days?

**Approach:**

* JOIN Transactions → Product on product\_id
* Filter to last month
* Calculate revenue: quantity \* unit\_price
* GROUP BY category, SUM revenue, ORDER DESC, LIMIT 3

SELECT  
 p.product\_category,  
 SUM(t.quantity \* p.unit\_price) AS total\_revenue  
FROM Transactions t  
JOIN Product p ON t.product\_id = p.product\_id  
WHERE t.date >= DATE\_FORMAT(CURDATE() - INTERVAL 1 MONTH, '%Y-%m-01')  
 AND t.date < DATE\_FORMAT(CURDATE(), '%Y-%m-01')  
GROUP BY p.product\_category  
ORDER BY total\_revenue DESC  
LIMIT 3;

## Q2 - Customer spend % by category

Clarify: Percentages should sum to 100% per customer, correct?

**Approach:**

* JOIN tables to get customer + category + spend
* Use window function to get customer total
* Calculate percentage

WITH category\_spend AS (  
 SELECT  
 t.customer\_id,  
 p.product\_category,  
 SUM(t.quantity \* p.unit\_price) AS spend  
 FROM Transactions t  
 JOIN Product p ON t.product\_id = p.product\_id  
 GROUP BY t.customer\_id, p.product\_category  
)  
SELECT  
 customer\_id,  
 product\_category,  
 spend,  
 ROUND(100.0 \* spend / SUM(spend) OVER (PARTITION BY customer\_id), 2) AS pct\_of\_total  
FROM category\_spend  
ORDER BY customer\_id, pct\_of\_total DESC;

## Q3 - Month-over-month growth by customer

Clarify: Growth as percentage change? How to handle first month (no prior)?

**Approach:**

* Aggregate revenue by customer + month
* Use LAG() to get prior month
* Calculate (current - prior) / prior \* 100

WITH monthly\_revenue AS (  
 SELECT  
 t.customer\_id,  
 DATE\_FORMAT(t.date, '%Y-%m-01') AS month,  
 SUM(t.quantity \* p.unit\_price) AS revenue  
 FROM Transactions t  
 JOIN Product p ON t.product\_id = p.product\_id  
 GROUP BY t.customer\_id, DATE\_FORMAT(t.date, '%Y-%m-01')  
)  
SELECT  
 customer\_id,  
 month,  
 revenue,  
 LAG(revenue) OVER (PARTITION BY customer\_id ORDER BY month) AS prev\_revenue,  
 ROUND(  
 100.0 \* (revenue - LAG(revenue) OVER (PARTITION BY customer\_id ORDER BY month)) /  
 NULLIF(LAG(revenue) OVER (PARTITION BY customer\_id ORDER BY month), 0),  
 2  
 ) AS mom\_growth\_pct  
FROM monthly\_revenue  
ORDER BY customer\_id, month;

# EXAMPLE 2

Given tables:

* Product - product\_category, product\_id, price
* Transactions - customer\_id, date, product\_id, quantity
* Customer - customer\_id, name, address, region, account\_type

## Q1 - Top 3 categories by quantity sold in 2019

Approach: Similar to Example 1 Q1, but aggregate quantity instead of revenue.

SELECT  
 p.product\_category,  
 SUM(t.quantity) AS total\_quantity  
FROM Transactions t  
JOIN Product p ON t.product\_id = p.product\_id  
WHERE t.date >= '2019-01-01' AND t.date < '2020-01-01'  
GROUP BY p.product\_category  
ORDER BY total\_quantity DESC  
LIMIT 3;

## Q2 - Top 5 customers per region in top 3 categories

Clarify: Top 5 per region or top 5 overall filtered by region?

**Approach:**

* Use Q1 result as subquery/CTE
* Filter transactions to those categories
* JOIN Customer for region
* RANK within each region, filter to top 5

WITH top\_categories AS (  
 SELECT p.product\_category  
 FROM Transactions t  
 JOIN Product p ON t.product\_id = p.product\_id  
 WHERE t.date >= '2019-01-01' AND t.date < '2020-01-01'  
 GROUP BY p.product\_category  
 ORDER BY SUM(t.quantity) DESC  
 LIMIT 3  
),  
customer\_sales AS (  
 SELECT  
 c.customer\_id,  
 c.region,  
 SUM(t.quantity) AS total\_qty  
 FROM Transactions t  
 JOIN Product p ON t.product\_id = p.product\_id  
 JOIN Customer c ON t.customer\_id = c.customer\_id  
 WHERE p.product\_category IN (SELECT product\_category FROM top\_categories)  
 AND c.region IN ('US', 'EMEA', 'APAC')  
 GROUP BY c.customer\_id, c.region  
),  
ranked AS (  
 SELECT  
 customer\_id,  
 region,  
 total\_qty,  
 ROW\_NUMBER() OVER (PARTITION BY region ORDER BY total\_qty DESC) AS rn  
 FROM customer\_sales  
)  
SELECT customer\_id, region, total\_qty  
FROM ranked  
WHERE rn <= 5  
ORDER BY region, rn;

# EXAMPLE 3

Given table:

* Orders - marketplace\_id, order\_id, customer\_id, item, units, order\_date

## Q1 - % of 2021 orders that were in Q1 2021

Approach: Count Q1 orders, divide by total 2021 orders.

SELECT  
 ROUND(  
 100.0 \* SUM(CASE WHEN order\_date >= '2021-01-01' AND order\_date < '2021-04-01' THEN 1 ELSE 0 END) /  
 COUNT(\*),  
 2  
 ) AS q1\_pct\_of\_total  
FROM Orders  
WHERE order\_date >= '2021-01-01' AND order\_date < '2022-01-01';

## Q2 - Top 10 items from first purchases in 2021 (US)

Clarify: "First purchase" = customer's first order ever, or first in 2021?

**Approach:**

* Find each customer's first order date in 2021 (US)
* Get items from those orders
* Rank by frequency

WITH first\_orders AS (  
 SELECT customer\_id, MIN(order\_date) AS first\_date  
 FROM Orders  
 WHERE order\_date >= '2021-01-01' AND order\_date < '2022-01-01'  
 AND marketplace\_id = 'US'  
 GROUP BY customer\_id  
)  
SELECT  
 o.item,  
 COUNT(\*) AS times\_sold  
FROM Orders o  
JOIN first\_orders fo  
 ON o.customer\_id = fo.customer\_id  
 AND o.order\_date = fo.first\_date  
WHERE o.marketplace\_id = 'US'  
GROUP BY o.item  
ORDER BY times\_sold DESC  
LIMIT 10;

# Other Examples

## Unique customers & average sales (Nov 2021)

SELECT  
 COUNT(DISTINCT customer\_id) AS unique\_customers,  
 AVG(sales\_amount) AS avg\_sales  
FROM Orders  
WHERE order\_date >= '2021-11-01' AND order\_date < '2021-12-01';

## Average of customer averages (Nov 2021)

WITH customer\_avg AS (  
 SELECT customer\_id, AVG(sales\_amount) AS avg\_sales  
 FROM Orders  
 WHERE order\_date >= '2021-11-01' AND order\_date < '2021-12-01'  
 GROUP BY customer\_id  
)  
SELECT AVG(avg\_sales) AS avg\_of\_avg\_sales  
FROM customer\_avg;

## Why are they different? (Conceptual)

The simple average weights each transaction equally. The average of averages weights each customer equally.

If $50 < $60, it means high-volume customers have lower average order values. A few customers placing many small orders pull down the simple average, but in the average-of-averages, they only count once.

**Key insight: This reveals customer behavior skew—some customers order frequently with smaller amounts.**

## Histogram of order counts (including zeros)

Approach: Need all customers, LEFT JOIN to Nov orders, count orders, then group by count.

WITH nov\_orders AS (  
 SELECT customer\_id, COUNT(\*) AS order\_count  
 FROM Orders  
 WHERE order\_date >= '2021-11-01' AND order\_date < '2021-12-01'  
 GROUP BY customer\_id  
),  
all\_customers AS (  
 SELECT  
 c.customer\_id,  
 COALESCE(n.order\_count, 0) AS order\_count  
 FROM Customers c  
 LEFT JOIN nov\_orders n ON c.customer\_id = n.customer\_id  
)  
SELECT  
 order\_count,  
 COUNT(\*) AS customer\_count  
FROM all\_customers  
GROUP BY order\_count  
ORDER BY order\_count;

## Week with largest WoW customer change

**Approach:**

* Count distinct customers per week
* Use LAG() for prior week
* Calculate absolute change, find max

WITH weekly\_customers AS (  
 SELECT  
 DATE(order\_date - INTERVAL WEEKDAY(order\_date) DAY + INTERVAL 6 DAY) AS week\_ending,  
 COUNT(DISTINCT customer\_id) AS customer\_count  
 FROM Orders  
 WHERE order\_date >= CURDATE() - INTERVAL 1 YEAR  
 GROUP BY week\_ending  
),  
with\_change AS (  
 SELECT  
 week\_ending,  
 customer\_count,  
 LAG(customer\_count) OVER (ORDER BY week\_ending) AS prev\_count,  
 ABS(customer\_count - LAG(customer\_count) OVER (ORDER BY week\_ending)) AS abs\_change  
 FROM weekly\_customers  
)  
SELECT week\_ending, customer\_count, prev\_count, abs\_change  
FROM with\_change  
ORDER BY abs\_change DESC  
LIMIT 1;

# Interview Framework Summary

|  |  |
| --- | --- |
| Step | What to Do |
| Clarify | Restate the problem, ask about edge cases |
| Identify | Tables, joins, filters needed |
| Decompose | Break into CTEs or logical steps |
| Write | Start simple, add complexity |
| Validate | Talk through edge cases (NULLs, zeros, ties) |
| Optimize | Mention indexes, alternatives if asked |