SQL Interview Questions: A Complete Guide

# Part 1: SQL Concept Questions

The key: Be concise, give a real example, and mention a gotcha or edge case.

## Q1 - Explain the Various Types of Joins

Framework: Define → Visual mental model → Use case

"Joins combine rows from two tables based on a related column."

* **INNER JOIN:** Only matching rows from both tables
* **LEFT JOIN:** All rows from left table + matches from right (NULLs where no match)
* **RIGHT JOIN:** Opposite of LEFT
* **FULL OUTER JOIN:** All rows from both, NULLs where no match
* **CROSS JOIN:** Cartesian product - every row paired with every row

**Gotcha:** LEFT JOIN with a WHERE clause on the right table often behaves like an INNER JOIN unless you check for NULL.

## Q2 - What is the Difference Between UNION and UNION ALL?

"Both combine result sets vertically."

* **UNION:** Removes duplicates (slower - requires sorting/hashing)
* **UNION ALL:** Keeps all rows (faster)

**Best practice:** Use UNION ALL when you know there are no duplicates or duplicates are acceptable.

## Q3 - What is the Difference Between WHERE and HAVING?

**WHERE** filters rows *before* aggregation. **HAVING** filters *after* aggregation.

Example: "Find categories with total sales > $1000" needs HAVING because you filter on SUM().

## Q4 - What is the Difference Between DIST Key and SORT Key?

"Both optimize query performance in columnar databases like Redshift."

* **DIST Key:** Controls how data is distributed across nodes (optimize for JOINs)
* **SORT Key:** Controls row ordering within each node (optimize for range filters and ORDER BY)

## Q5 - What is the Difference Between LAG and LEAD Functions?

"Window functions that access other rows relative to current row."

* **LAG:** Looks backward (previous row)
* **LEAD:** Looks forward (next row)

Common use: Month-over-month comparisons.

# Part 2: SQL Query Questions - Framework

Framework for any SQL interview question:

**1. Clarify -** Ask about edge cases, date ranges, tie-breaking

**2. Break it down -** Identify the steps (often maps to CTEs)

**3. Write incrementally -** Build and verify each piece

**4. Optimize -** Mention indexes, explain choices

# Example 1: Transactions & Products

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

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;

**Talk through it:** "I join to get price, filter to last month, aggregate by category, sort descending, limit 3."

## Q2: Customer Spend % by Category

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

## Q3: Month-over-Month Growth by Customer

WITH monthly\_sales 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\_month,  
 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\_sales  
ORDER BY customer\_id, month;

Key points to mention:

* Use NULLIF to avoid divide-by-zero
* LAG with PARTITION BY for per-customer comparison

# Example 2: Products, Transactions & Customers

Given three 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 in 2019

SELECT  
 p.product\_category,  
 SUM(t.quantity) AS total\_quantity  
FROM Transactions t  
JOIN Product p ON t.product\_id = p.product\_id  
WHERE YEAR(t.date) = 2019  
GROUP BY p.product\_category  
ORDER BY total\_quantity DESC  
LIMIT 3;

## Q2: Top 5 Customers per Region in Top 3 Categories

WITH top\_categories AS (  
 SELECT p.product\_category  
 FROM Transactions t  
 JOIN Product p ON t.product\_id = p.product\_id  
 WHERE YEAR(t.date) = 2019  
 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: Orders Analysis

Given table:

* • Orders: marketplace\_id, order\_id, customer\_id, item, units, order\_date

## Q1: Percentage of 2021 Orders in Q1

SELECT  
 ROUND(100.0 \* SUM(CASE  
 WHEN order\_date BETWEEN '2021-01-01' AND '2021-03-31'  
 THEN 1 ELSE 0 END) / COUNT(\*), 2) AS q1\_pct  
FROM Orders  
WHERE YEAR(order\_date) = 2021;

## Q2: Top 10 Items from First Purchases in 2021 (US)

WITH first\_orders AS (  
 SELECT customer\_id, MIN(order\_date) AS first\_date  
 FROM Orders  
 WHERE YEAR(order\_date) = 2021  
 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 Common Questions

## 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 BETWEEN '2021-11-01' AND '2021-11-30';

## Average of Customer Averages

WITH customer\_avgs AS (  
 SELECT customer\_id, AVG(sales\_amount) AS avg\_sales  
 FROM Orders  
 WHERE order\_date BETWEEN '2021-11-01' AND '2021-11-30'  
 GROUP BY customer\_id  
)  
SELECT AVG(avg\_sales) AS avg\_of\_avgs  
FROM customer\_avgs;

## Why Average ≠ Average of Averages?

**Statistical insight:** The difference indicates **unequal purchase behavior** across customers. Simple average weights all transactions equally. Average of averages weights all *customers* equally.

If simple avg ($50) < avg of avgs ($60), it means high-volume customers spend less per transaction than low-volume customers.

## Histogram: Customer Order Counts (Including Zero)

WITH all\_customers AS (  
 SELECT DISTINCT customer\_id FROM Customers  
),  
nov\_orders AS (  
 SELECT customer\_id, COUNT(DISTINCT order\_id) AS order\_count  
 FROM Orders  
 WHERE order\_date BETWEEN '2021-11-01' AND '2021-11-30'  
 GROUP BY customer\_id  
)  
SELECT  
 COALESCE(no.order\_count, 0) AS order\_count,  
 COUNT(\*) AS customer\_count  
FROM all\_customers ac  
LEFT JOIN nov\_orders no ON ac.customer\_id = no.customer\_id  
GROUP BY COALESCE(no.order\_count, 0)  
ORDER BY order\_count;

## Week with Largest Customer Change vs Prior Week

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

# General Interview Tips

|  |  |
| --- | --- |
| **DO** | **DON'T** |
| Think out loud | Write in silence |
| Ask clarifying questions | Assume edge cases |
| Use CTEs for readability | Write one massive query |
| Mention tradeoffs | Only give one solution |
| Test with edge cases mentally | Assume it works |