

# 50 SQL Interview Queries



## 1. Find duplicate records in a table



```
SELECT column1, column2, COUNT(*)  
FROM your_table  
GROUP BY column1, column2  
HAVING COUNT(*) > 1;
```



## 2. Retrieve the second highest salary from the Employee table



```
SELECT MAX(salary) AS  
SecondHighestSalary  
FROM Employee  
WHERE salary < (SELECT MAX(salary)  
FROM Employee);
```



### 3. Find employees without department (Left Join usage)



Uber

```
SELECT e.*  
FROM Employee e  
LEFT JOIN Department d  
ON e.department_id =  
d.department_id  
WHERE d.department_id IS NULL;
```

### 4. Calculate the total revenue per product



```
SELECT product_id,  
SUM(quantity * price) AS  
total_revenue  
FROM Sales  
GROUP BY product_id;
```

PayPal

## 5. Get the top 3 highest-paid employees.



```
SELECT TOP 3 *
FROM Employee
ORDER BY salary DESC;
```



## 6. Customers who made purchases but never returned products.



```
SELECT DISTINCT c.customer_id
FROM Customers c
JOIN Orders o ON c.customer_id =
o.customer_id
WHERE c.customer_id NOT IN (
SELECT customer_id FROM Returns
);
```



## 7. Show the count of orders per customer.



```
SELECT customer_id,  
COUNT(*) AS order_count  
FROM Orders  
GROUP BY customer_id;
```



## 8. Retrieve all employees who joined in 2023.



```
SELECT *  
FROM Employee  
WHERE YEAR(hire_date) = 2023;
```



Comment "**SQL**" and get the complete in-depth PDF in your DMs.

## 9. Calculate the average order value per customer.



```
SELECT customer_id,  
AVG(total_amount) AS  
avg_order_value FROM  
Orders GROUP BY  
customer_id;
```



## 10. Get the latest order placed by each customer.



```
SELECT customer_id,  
MAX(order_date) AS  
latest_order_date  
FROM Orders  
GROUP BY customer_id;
```



## 11. Find products that were never sold.



```
SELECT p.product_id  
FROM Products p  
LEFT JOIN Sales s  
ON p.product_id = s.product_id  
WHERE s.product_id IS NULL;
```



## 12. Identify the most selling product.



```
SELECT TOP 1 product_id,  
SUM(quantity) AS total_qty  
FROM Sales  
GROUP BY product_id  
ORDER BY total_qty DESC;
```



Comment "**SQL**" and get the complete in-depth PDF in your DMs.

## 13. Get the total revenue and the number of orders per region.



```
SELECT region,  
SUM(total_amount) AS total_revenue,  
COUNT(*) AS order_count  
FROM Orders  
GROUP BY region;
```

∞ Meta

## 14. Count how many customers placed more than 5 orders.



```
SELECT COUNT(*) AS customer_count  
FROM (  
    SELECT customer_id FROM Orders  
    GROUP BY customer_id  
    HAVING COUNT(*) > 5  
) AS subquery;
```

amazon

## 15. Retrieve customers with orders above the average order value.



```
SELECT *
FROM Orders
WHERE total_amount >
(SELECT AVG(total_amount)
FROM Orders);
```



## 16. Find all employees hired on weekends.



```
SELECT *
FROM Employee
WHERE DATENAME(WEEKDAY, hire_date)
IN ('Saturday', 'Sunday');
```



## 17. Find all employees hired on weekends.



```
SELECT *
FROM Employee
WHERE salary BETWEEN 50000 AND
100000;
```



## 18. Get monthly sales revenue and order count.



```
SELECT
FORMAT(date, 'yyyy-MM') AS month,
SUM(amount) AS total_revenue,
COUNT(order_id) AS order_count
FROM Orders
GROUP BY
FORMAT(date, 'yyyy-MM');
```



## 19. Rank employees by salary within each department.



```
SELECT employee_id, department_id,  
salary, RANK() OVER (PARTITION BY  
department_id  
ORDER BY salary DESC) AS salary_rk  
FROM Employee;
```



## 20. Find customers who placed orders every month in 2023.



```
SELECT customer_id  
FROM Orders  
WHERE YEAR(order_date) = 2023  
GROUP BY customer_id  
HAVING COUNT(DISTINCT  
FORMAT(order_date, 'yyyy-MM')) = 12
```

## 21. Find moving average of sales over the last 3 days.



```
SELECT order_date,  
AVG(total_amount) OVER (ORDER BY  
order_date ROWS BETWEEN 2 PRECEDING  
AND CURRENT ROW) AS moving_avg  
FROM Orders;
```



## 22. Identify the first and last order date for each customer.



```
SELECT customer_id,  
MIN(order_date) AS first_order,  
MAX(order_date) AS last_order  
FROM Orders  
GROUP BY customer_id;
```



## 23. Show product sales distribution (percent of total revenue).



```
WITH TotalRevenue AS (
    SELECT
        SUM(quantity * price) AS total FROM Sales)
    SELECT s.product_id,
        SUM(s.quantity * s.price) AS revenue,
        SUM(s.quantity * s.price) * 100/ t.total
        AS revenue_pct
    FROM Sales s
    CROSS JOIN TotalRevenue t
    GROUP BY s.product_id, t.total;
```



## 24. Retrieve customers who made consecutive purchases (2 Days)



```
WITH cte AS (
    SELECT id, order_date,
        LAG(order_date) OVER (PARTITION BY id
        ORDER BY order_date) AS prev_order_date
    FROM Orders)
    SELECT id, order_date, prev_order_date
    FROM cte
    WHERE
        DATEDIFF(DAY, prev_order_date, order_date) = 1;
```

## 24. Retrieve customers who made consecutive purchases (2 Days)



```
WITH cte AS (
    SELECT id, order_date,
    LAG(order_date) OVER (PARTITION BY id
    ORDER BY order_date) AS prev_order_date
    FROM Orders)
    SELECT id, order_date, prev_order_date
    FROM cte
    WHERE
    DATEDIFF(DAY, prev_order_date, order_date) = 1;
```

## 25. Find churned customers (no orders in the last 6 months).

```
SELECT customer_id
FROM Orders
GROUP BY customer_id
HAVING
MAX(order_date) <
DATEADD(MONTH,-6,GETDATE());
```



## 26. Calculate cumulative revenue by day.



```
SELECT order_date,  
SUM(total_amount) OVER  
(ORDER BY order_date) AS  
cumulative_revenue  
FROM Orders;
```



## 27. Identify top-performing departments by average salary.



```
SELECT department_id,  
AVG(salary) AS avg_salary  
FROM Employee  
GROUP BY department_id  
ORDER BY avg_salary DESC;
```



## 28. Find customers who ordered more than the average number of orders per customer.



```
WITH customer_orders AS (
  SELECT customer_id, COUNT(*) AS order_count
  FROM Orders
  GROUP BY customer_id)
SELECT * FROM customer_orders
WHERE order_count > (SELECT
  AVG(order_count) FROM customer_orders);
```

## 29. Calculate revenue generated from new customers (first-time orders).



```
WITH first_orders AS (
  SELECT customer_id, MIN(order_date) AS
  first_order_date FROM Orders
  GROUP BY customer_id)
SELECT SUM(o.total_amount) AS new_revenue
  FROM Orders o JOIN first_orders f
    ON o.customer_id = f.customer_id
 WHERE o.order_date = f.first_order_date;
```

## 30. Find the percentage of employees in each department.



**SELECT**

```
department_id,  
COUNT(*) AS emp_count,  
COUNT(*) * 100.0 / (SELECT  
COUNT(*) FROM Employee)  
AS pct FROM Employee  
GROUP BY department_id;
```

Uber

## 31. Retrieve the maximum salary difference within each department.



**SELECT**

```
department_id,  
MAX(salary) - MIN(salary) AS  
salary_diff  
FROM Employee  
GROUP BY department_id;
```

PayPal

## 32. Find products that contribute to 80% of the revenue (Pareto Principle).



```
WITH sales_cte AS (
    SELECT product_id, SUM(qty * price) AS revenue
    FROM Sales GROUP BY product_id),
    total_revenue AS (
        SELECT SUM(revenue) AS total FROM sales_cte)
    SELECT s.product_id, s.revenue,
        SUM(s.revenue) OVER
        (ORDER BY s.revenue DESC ROWS BETWEEN UNBOUNDED
        PRECEDING AND CURRENT ROW) AS running_total
    FROM sales_cte s, total_revenue t
    WHERE SUM(s.revenue) OVER (ORDER BY s.revenue DESC
        ROWS BETWEEN UNBOUNDED PRECEDING AND
        CURRENT ROW) <= t.total * 0.8;
```



## 33. Calculate average time between two purchases for each customer.



```
WITH cte AS (
    SELECT customer_id, order_date,
        LAG(order_date) OVER (PARTITION BY
            customer_id
            ORDER BY order_date) AS prev_date
        FROM Orders)
    SELECT customer_id,
        AVG(DATEDIFF(DAY, prev_date, order_date))
        AS avg_gap_days FROM cte
    WHERE prev_date IS NOT NULL
    GROUP BY customer_id;
```



## 34. Show last purchase for each customer along with order amount.

```
WITH ranked_orders AS
(SELECT customer_id, order_id,
total_amount, ROW_NUMBER() OVER
(PARTITION BY customer_id ORDER BY
order_date DESC) AS rn FROM Orders)
SELECT customer_id, order_id,
total_amount
FROM ranked_orders
WHERE rn = 1;
```

Google

## 35. Calculate year-over-year growth in revenue.



```
SELECT FORMAT(order_date, 'yyyy') AS year,
SUM(total_amount) AS revenue,
SUM(total_amount) - LAG(SUM(total_amount))
OVER (ORDER BY FORMAT(order_date, 'yyyy'))
AS yoy_growth
FROM Orders
GROUP BY FORMAT(order_date, 'yyyy');
```

## 36. Detect customers whose purchase amount is higher than their historical 90th percentile.



```
WITH ranked_orders AS (
    SELECT customer_id, order_id,
    total_amount,
    NTILE(10) OVER (PARTITION BY customer_id
    ORDER BY total_amount) AS decile
    FROM Orders)
SELECT customer_id, order_id, total_amount
FROM ranked_orders
WHERE decile = 10;
```



## 37. Retrieve the longest gap between orders for each customer.



```
WITH cte AS (
    SELECT customer_id, order_date,
    LAG(order_date) OVER (PARTITION BY
    customer_id ORDER BY order_date) AS
    prev_order_date FROM Orders)
SELECT customer_id, MAX(DATEDIFF(DAY,
    prev_order_date, order_date)) AS max_gap
FROM cte
WHERE prev_order_date IS NOT NULL
GROUP BY customer_id;
```



## 38. Identify customers with revenue below the 10th percentile.



Google

```
WITH cte AS (
  SELECT customer_id, SUM(total_amount) AS
  total_revenue
  FROM Orders
  GROUP BY customer_id)
SELECT customer_id, total_revenue
FROM cte
WHERE total_revenue <
(SELECT PERCENTILE_CONT(0.1) WITHIN GROUP
 (ORDER BY total_revenue) FROM cte);
```