150 SQL Query Questions to Master SQL

Database Schema Reference

For these questions, assume the following tables exist:

employees (employee_id, first_name, last_name, email, hire_date, salary, department_id, manager_id)
departments (department_id, department_name, location)
orders (order_id, customer_id, order_date, total_amount, status)
customers (customer_id, name, email, city, country, registration_date)
products (product_id, product_name, category, price, stock_quantity)
order_items (order_item_id, order_id, product_id, quantity, price)
projects (project_id, project_name, start_date, end_date, budget, department_id)
sales (sale_id, employee_id, sale_date, amount, region)

BASIC LEVEL (1-50)

1. Select all columns from employees

```
SELECT * FROM employees;
```

2. Select only first_name and last_name from employees

```
SELECT first_name, last_name FROM employees;
```

3. Select all customers from USA

```
SELECT * FROM customers WHERE country = 'USA';
```

4. Select employees with salary greater than 50000

```
SELECT * FROM employees WHERE salary > 50000;
```

5. Select distinct cities from customers

```
SELECT DISTINCT city FROM customers;
```

6. Count total number of employees

```
SELECT COUNT(*) FROM employees;
```

7. Select employees ordered by salary descending

```
SELECT * FROM employees ORDER BY salary DESC;
```

8. Select top 10 highest paid employees

```
SELECT * FROM employees ORDER BY salary DESC LIMIT 10;
```

9. Select employees whose first name starts with 'J'

```
SELECT * FROM employees WHERE first_name LIKE 'J%';
```

10. Select employees hired in 2023

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

11. Select products with price between 100 and 500

```
SELECT * FROM products WHERE price BETWEEN 100 AND 500;
```

12. Select orders with status 'Completed' or 'Shipped'

```
SELECT * FROM orders WHERE status IN ('Completed', 'Shipped');
```

13. Select employees with NULL manager_id

```
SELECT * FROM employees WHERE manager_id IS NULL;
```

14. Select average salary of all employees

```
SELECT AVG(salary) AS average_salary FROM employees;
```

15. Select minimum and maximum product prices

```
SELECT MIN(price) AS min_price, MAX(price) AS max_price FROM products;
```

16. Select total number of orders by status

```
SELECT status, COUNT(*) AS order_count FROM orders GROUP BY status;
```

17. Select employees and their departments (INNER JOIN)

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id;
```

18. Select customers who registered after January 1, 2023

```
SELECT * FROM customers WHERE registration_date > '2023-01-01';
```

19. Select products ordered by name alphabetically

```
SELECT * FROM products ORDER BY product_name ASC;
```

20. Select sum of all order totals

```
SELECT SUM(total_amount) AS total_revenue FROM orders;
```

21. Select employees with salary less than 40000

```
SELECT * FROM employees WHERE salary < 40000;
```

22. Select count of products in each category

```
SELECT category, COUNT(*) AS product_count FROM products GROUP BY category;
```

23. Select orders placed in the last 30 days

```
SELECT * FROM orders WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

24. Select employees whose email contains 'gmail'

```
SELECT * FROM employees WHERE email LIKE '%gmail%';
```

25. Select distinct countries from customers

```
SELECT DISTINCT country FROM customers;
```

26. Select products with stock quantity greater than 100

```
SELECT * FROM products WHERE stock_quantity > 100;
```

27. Select employees ordered by hire_date (oldest first)

```
SELECT * FROM employees ORDER BY hire_date ASC;
```

28. Select total amount spent by each customer

```
SELECT customer_id, SUM(total_amount) AS total_spent
FROM orders
GROUP BY customer_id;
```

29. Select employees with names ending in 'son'

```
SELECT * FROM employees WHERE last_name LIKE '%son';
```

30. Select count of orders per customer

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

31. Select all departments located in 'New York'

```
SELECT * FROM departments WHERE location = 'New York';
```

32. Select employees with salary equal to 60000

```
SELECT * FROM employees WHERE salary = 60000;
```

33. Select products in 'Electronics' category

```
SELECT * FROM products WHERE category = 'Electronics';
```

34. Select average order total

```
SELECT AVG(total_amount) AS average_order FROM orders;
```

35. Select employees hired between 2020 and 2023

```
SELECT * FROM employees WHERE hire_date BETWEEN '2020-01-01' AND '2023-12-31';
```

36. Select customers ordered by name

```
SELECT * FROM customers ORDER BY name ASC;
```

37. Select total sales amount by region

```
SELECT region, SUM(amount) AS total_sales
FROM sales
GROUP BY region;
```

38. Select orders where total_amount is greater than 1000

```
SELECT * FROM orders WHERE total_amount > 1000;
```

39. Select employees with non-NULL email addresses

```
SELECT * FROM employees WHERE email IS NOT NULL;
```

40. Select count of employees in each department

```
SELECT department_id, COUNT(*) AS employee_count
FROM employees
GROUP BY department_id;
```

41. Select products with price less than 50

```
SELECT * FROM products WHERE price < 50;
```

42. Select customers from 'London' or 'Paris'

```
SELECT * FROM customers WHERE city IN ('London', 'Paris');
```

43. Select employees with salary between 50000 and 70000

```
SELECT * FROM employees WHERE salary BETWEEN 50000 AND 70000;
```

44. Select the 5 most recent orders

```
SELECT * FROM orders ORDER BY order_date DESC LIMIT 5;
```

45. Select total number of products

```
SELECT COUNT(*) AS total_products FROM products;
```

46. Select departments with their employee count

```
SELECT d.department_name, COUNT(e.employee_id) AS employee_count
FROM departments d
LEFT JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_name;
```

47. Select orders with status 'Pending'

```
SELECT * FROM orders WHERE status = 'Pending';
```

48. Select employees whose first name is 'John' or 'Jane'

```
SELECT * FROM employees WHERE first_name IN ('John', 'Jane');
```

49. Select maximum salary per department

```
SELECT department_id, MAX(salary) AS max_salary
FROM employees
GROUP BY department_id;
```

50. Select products ordered by price descending

```
SELECT * FROM products ORDER BY price DESC;
```

MEDIUM LEVEL (51-100)

51. Select employees earning above average salary

```
SELECT * FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

52. Select customers with more than 5 orders

```
SELECT customer_id, COUNT(*) AS order_count
FROM orders
GROUP BY customer_id
HAVING COUNT(*) > 5;
```

53. Select employee names with their manager names

```
SELECT e.first_name AS employee, m.first_name AS manager
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.employee_id;
```

54. Select total revenue by month in 2023

```
SELECT MONTH(order_date) AS month, SUM(total_amount) AS revenue
FROM orders
WHERE YEAR(order_date) = 2023
GROUP BY MONTH(order_date)
ORDER BY month;
```

55. Select departments with average salary greater than 60000

```
SELECT department_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 60000;
```

56. Select products that have never been ordered

```
SELECT p.* FROM products p
LEFT JOIN order_items oi ON p.product_id = oi.product_id
WHERE oi.product_id IS NULL;
```

57. Select top 5 customers by total spending

```
SELECT c.name, SUM(o.total_amount) AS total_spent
```

```
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name
ORDER BY total_spent DESC
LIMIT 5;
```

58. Select employees and calculate tenure in years

```
SELECT first_name, last_name,
    TIMESTAMPDIFF(YEAR, hire_date, CURDATE()) AS years_employed
FROM employees;
```

59. Select orders with customer information

```
SELECT o.order_id, o.order_date, c.name, c.email, o.total_amount
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id;
```

60. Select products with average order quantity

```
SELECT p.product_name, AVG(oi.quantity) AS avg_quantity
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.product_name;
```

61. Select departments with no employees

```
SELECT d.* FROM departments d
LEFT JOIN employees e ON d.department_id = e.department_id
WHERE e.employee_id IS NULL;
```

62. Select second highest salary

```
SELECT MAX(salary) AS second_highest
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```

63. Select employees hired in the same year

```
SELECT YEAR(hire_date) AS hire_year, COUNT(*) AS employee_count
FROM employees
GROUP BY YEAR(hire_date)
```

```
ORDER BY hire_year;
```

64. Select customers who haven't ordered in 2024

```
SELECT c.* FROM customers c
WHERE c.customer_id NOT IN (
   SELECT DISTINCT customer_id FROM orders
   WHERE YEAR(order_date) = 2024
);
```

65. Select products with price above category average

```
SELECT p.* FROM products p
WHERE p.price > (
   SELECT AVG(price) FROM products p2
   WHERE p2.category = p.category
);
```

66. Select monthly order count for each customer

```
SELECT customer_id,
      DATE_FORMAT(order_date, '%Y-%m') AS month,
      COUNT(*) AS order_count
FROM orders
GROUP BY customer_id, DATE_FORMAT(order_date, '%Y-%m');
```

67. Select employees with salary rank in department

```
SELECT employee_id, first_name, last_name, salary, department_id,
      RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS salary_rank
FROM employees;
```

68. Select total quantity sold per product

```
SELECT p.product_name, SUM(oi.quantity) AS total_sold
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.product_name
ORDER BY total_sold DESC;
```

69. Select customers with their first and last order dates

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

70. Select departments with budget utilization from projects

```
SELECT d.department_name,
    SUM(p.budget) AS total_budget,
    COUNT(p.project_id) AS project_count
FROM departments d
LEFT JOIN projects p ON d.department_id = p.department_id
GROUP BY d.department_id, d.department_name;
```

71. Select products with stock below average for their category

```
SELECT p.* FROM products p
WHERE p.stock_quantity < (
  SELECT AVG(stock_quantity) FROM products p2
  WHERE p2.category = p.category
);
```

72. Select employees who earn more than their manager

```
SELECT e.first_name, e.last_name, e.salary AS emp_salary, m.salary AS mgr_salary
FROM employees e
JOIN employees m ON e.manager_id = m.employee_id
WHERE e.salary > m.salary;
```

73. Select running total of sales by date

```
SELECT sale_date, amount,
    SUM(amount) OVER (ORDER BY sale_date) AS running_total
FROM sales
ORDER BY sale_date;
```

74. Select customers and their order frequency

```
SELECT c.name,
    COUNT(o.order_id) AS order_count,
    DATEDIFF(MAX(o.order_date), MIN(o.order_date)) AS days_span,
```

```
    COUNT(o.order_id) / NULLIF(DATEDIFF(MAX(o.order_date), MIN(o.order_date)), 0) AS
orders_per_day
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name;
```

75. Select product categories ranked by revenue

```
SELECT p.category, SUM(oi.quantity * oi.price) AS revenue
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.category
ORDER BY revenue DESC;
```

76. Select employees with duplicate email domains

```
SELECT SUBSTRING_INDEX(email, '@', -1) AS domain, COUNT(*) AS count
FROM employees
GROUP BY SUBSTRING_INDEX(email, '@', -1)
HAVING COUNT(*) > 1;
```

77. Select orders with all item details

```
SELECT o.order_id, o.order_date, p.product_name, oi.quantity, oi.price
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN products p ON oi.product_id = p.product_id
ORDER BY o.order_id;
```

78. Select customers with above-average order value

```
SELECT c.customer_id, c.name, AVG(o.total_amount) AS avg_order_value
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name
HAVING AVG(o.total_amount) > (SELECT AVG(total_amount) FROM orders);
```

79. Select year-over-year sales growth

```
SELECT YEAR(sale_date) AS year,
    SUM(amount) AS total_sales,
    LAG(SUM(amount)) OVER (ORDER BY YEAR(sale_date)) AS previous_year,
```

```
    (SUM(amount) - LAG(SUM(amount)) OVER (ORDER BY YEAR(sale_date))) /
    LAG(SUM(amount)) OVER (ORDER BY YEAR(sale_date)) * 100 AS growth_pct
FROM sales
GROUP BY YEAR(sale_date);
```

80. Select employees and their subordinate count

```
SELECT m.employee_id, m.first_name, m.last_name, COUNT(e.employee_id) AS subordinates
FROM employees m
LEFT JOIN employees e ON m.employee_id = e.manager_id
GROUP BY m.employee_id, m.first_name, m.last_name;
```

81. Select products ordered more than 100 times

```
SELECT p.product_id, p.product_name, SUM(oi.quantity) AS total_ordered
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.product_name
HAVING SUM(oi.quantity) > 100;
```

82. Select quarterly revenue comparison

```
SELECT YEAR(order_date) AS year,
    QUARTER(order_date) AS quarter,
    SUM(total_amount) AS revenue
FROM orders
GROUP BY YEAR(order_date), QUARTER(order_date)
ORDER BY year, quarter;
```

83. Select customers with consecutive monthly orders

```
SELECT DISTINCT o1.customer_id
FROM orders o1
JOIN orders o2 ON o1.customer_id = o2.customer_id
WHERE PERIOD_DIFF(DATE_FORMAT(o2.order_date, '%Y%m'),
            DATE_FORMAT(o1.order_date, '%Y%m')) = 1;
```

84. Select products with price changes (self-join simulation)

```
SELECT p1.product_id, p1.product_name, p1.price AS current_price
FROM products p1
WHERE p1.price != (SELECT AVG(price) FROM products WHERE category = p1.category);
```

85. Select departments ordered by total salary expense

```
SELECT d.department_name, SUM(e.salary) AS total_salary
FROM departments d
JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
ORDER BY total_salary DESC;
```

86. Select employee salary percentile within department

```
SELECT employee_id, first_name, last_name, salary, department_id,
    PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY salary) AS
percentile
FROM employees;
```

87. Select orders with multiple items

```
SELECT o.order_id, COUNT(oi.order_item_id) AS item_count
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY o.order_id
HAVING COUNT(oi.order_item_id) > 1;
```

88. Select customers who ordered every product in a category

```
SELECT c.customer_id, c.name
FROM customers c
WHERE (SELECT COUNT(DISTINCT p.product_id)
    FROM products p
    WHERE p.category = 'Electronics') =
   (SELECT COUNT(DISTINCT oi.product_id)
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    JOIN products p ON oi.product_id = p.product_id
    WHERE o.customer_id = c.customer_id AND p.category = 'Electronics');
```

89. Select moving average of sales (3-day window)

```
SELECT sale_date, amount,
    AVG(amount) OVER (ORDER BY sale_date ROWS BETWEEN 2 PRECEDING AND
CURRENT ROW) AS moving_avg
```

```
FROM sales;
```

90. Select products with highest profit margin

```
SELECT p.product_name,
    p.price AS selling_price,
    AVG(oi.price) AS avg_cost,
    (p.price - AVG(oi.price)) / p.price * 100 AS profit_margin_pct
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.product_name, p.price
ORDER BY profit_margin_pct DESC;
```

91. Select employees hired in each quarter

```
SELECT YEAR(hire_date) AS year,
    QUARTER(hire_date) AS quarter,
    COUNT(*) AS hires
FROM employees
GROUP BY YEAR(hire_date), QUARTER(hire_date)
ORDER BY year, quarter;
```

92. Select customers with gaps in ordering (no orders for 90+ days)

```
SELECT o1.customer_id, o1.order_date AS order1, o2.order_date AS order2,
    DATEDIFF(o2.order_date, o1.order_date) AS days_gap
FROM orders o1
JOIN orders o2 ON o1.customer_id = o2.customer_id AND o2.order_date > o1.order_date
WHERE DATEDIFF(o2.order_date, o1.order_date) > 90;
```

93. Select products with low stock alerts (below 10% of max stock)

```
SELECT product_name, stock_quantity,
    (SELECT MAX(stock_quantity) FROM products) AS max_stock
FROM products
WHERE stock_quantity < (SELECT MAX(stock_quantity) FROM products) * 0.1;
```

94. Select department performance by average project budget

```
SELECT d.department_name, AVG(p.budget) AS avg_budget, COUNT(p.project_id) AS
projects
FROM departments d
```

```
LEFT JOIN projects p ON d.department_id = p.department_id
GROUP BY d.department_id, d.department_name
ORDER BY avg_budget DESC;
```

95. Select employees with same hire date

```
SELECT hire_date, GROUP_CONCAT(CONCAT(first_name, ' ', last_name)) AS employees
FROM employees
GROUP BY hire_date
HAVING COUNT(*) > 1;
```

96. Select cumulative customer acquisition by month

```
SELECT DATE_FORMAT(registration_date, '%Y-%m') AS month,
      COUNT(*) AS new_customers,
      SUM(COUNT(*)) OVER (ORDER BY DATE_FORMAT(registration_date, '%Y-%m')) AS
cumulative_customers
FROM customers
GROUP BY DATE_FORMAT(registration_date, '%Y-%m');
```

97. Select top 3 products per category by sales

```
SELECT * FROM (
   SELECT p.category, p.product_name, SUM(oi.quantity * oi.price) AS revenue,
         ROW_NUMBER() OVER (PARTITION BY p.category ORDER BY SUM(oi.quantity *
oi.price) DESC) AS rank_num
   FROM products p
   JOIN order_items oi ON p.product_id = oi.product_id
   GROUP BY p.category, p.product_id, p.product_name
) ranked
WHERE rank_num <= 3;
```

98. Select orders with same-day delivery (order and delivery on same day)

```
SELECT o.order_id, o.order_date, o.status
FROM orders o
WHERE o.status = 'Delivered'
  AND DATE(order_date) = DATE(order_date);
```

99. Select employee salary distribution by range

```
SELECT
```

```
    CASE
        WHEN salary < 40000 THEN 'Under 40k'
        WHEN salary BETWEEN 40000 AND 60000 THEN '40k-60k'
        WHEN salary BETWEEN 60001 AND 80000 THEN '60k-80k'
        ELSE 'Above 80k'
    END AS salary_range,
    COUNT(*) AS employee_count
FROM employees
GROUP BY salary_range
ORDER BY MIN(salary);
```

100. Select repeat customers (more than one order)

```
SELECT c.customer_id, c.name, COUNT(o.order_id) AS order_count
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.name
HAVING COUNT(o.order_id) > 1
ORDER BY order_count DESC;
```

ADVANCED LEVEL (101-150)

101. Find the Nth highest salary by department

```
WITH RankedSalaries AS (
    SELECT employee_id, first_name, last_name, salary, department_id,
           DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS
salary_rank
    FROM employees
)
SELECT * FROM RankedSalaries
WHERE salary_rank = 3;  -- Change N as needed
```

102. Find employees who earn more than the average salary in their department

```
SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id
FROM employees e
WHERE e.salary > (
    SELECT AVG(salary)
    FROM employees e2
```

```
    WHERE e2.department_id = e.department_id
);
```

103. Calculate running total of sales per employee

```
SELECT employee_id, sale_date, amount,
       SUM(amount) OVER (PARTITION BY employee_id ORDER BY sale_date) AS
running_total
FROM sales
ORDER BY employee_id, sale_date;
```

104. Find customers who made purchases in consecutive months

```
WITH MonthlyPurchases AS (
    SELECT DISTINCT customer_id,
           DATE_FORMAT(order_date, '%Y-%m') AS purchase_month
    FROM orders
)
SELECT DISTINCT m1.customer_id
FROM MonthlyPurchases m1
JOIN MonthlyPurchases m2
    ON m1.customer_id = m2.customer_id
    AND DATE_FORMAT(DATE_ADD(STR_TO_DATE(CONCAT(m1.purchase_month, '-01'),
'%Y-%m-%d'),
                    INTERVAL 1 MONTH), '%Y-%m') = m2.purchase_month;
```

105. Find duplicate records (same name and email)

```
SELECT name, email, COUNT(*) AS duplicate_count
FROM customers
GROUP BY name, email
HAVING COUNT(*) > 1;
```

106. Calculate month-over-month growth rate

```
WITH MonthlySales AS (
    SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
           SUM(total_amount) AS monthly_revenue
    FROM orders
    GROUP BY DATE_FORMAT(order_date, '%Y-%m')
)
SELECT month, monthly_revenue,
       LAG(monthly_revenue) OVER (ORDER BY month) AS prev_month_revenue,
```

```
    ROUND(((monthly_revenue - LAG(monthly_revenue) OVER (ORDER BY month)) /
        LAG(monthly_revenue) OVER (ORDER BY month) * 100), 2) AS growth_rate_pct
FROM MonthlySales;
```

107. Find employees with no subordinates (leaf nodes in org hierarchy)

```
SELECT e1.employee_id, e1.first_name, e1.last_name
FROM employees e1
LEFT JOIN employees e2 ON e1.employee_id = e2.manager_id
WHERE e2.employee_id IS NULL;
```

108. Calculate cumulative distribution of salaries

```
SELECT salary,
    COUNT(*) AS frequency,
    SUM(COUNT(*)) OVER (ORDER BY salary) AS cumulative_count,
    ROUND(SUM(COUNT(*)) OVER (ORDER BY salary) * 100.0 /
        SUM(COUNT(*)) OVER (), 2) AS cumulative_pct
FROM employees
GROUP BY salary
ORDER BY salary;
```

109. Find the most recent order for each customer

```
SELECT o.*
FROM orders o
INNER JOIN (
  SELECT customer_id, MAX(order_date) AS max_date
  FROM orders
  GROUP BY customer_id
) latest ON o.customer_id = latest.customer_id
     AND o.order_date = latest.max_date;
```

110. Find products never ordered together

```
SELECT p1.product_id AS product1, p2.product_id AS product2
FROM products p1
CROSS JOIN products p2
WHERE p1.product_id < p2.product_id
 AND NOT EXISTS (
  SELECT 1
  FROM order_items oi1
  JOIN order_items oi2 ON oi1.order_id = oi2.order_id
```

```
   WHERE oi1.product_id = p1.product_id
     AND oi2.product_id = p2.product_id
);
```

111. Calculate average time between orders per customer

```
WITH OrderGaps AS (
   SELECT customer_id,
          order_date,
          LAG(order_date) OVER (PARTITION BY customer_id ORDER BY order_date) AS
prev_order_date,
          DATEDIFF(order_date, LAG(order_date) OVER (PARTITION BY customer_id ORDER
BY order_date)) AS days_between
   FROM orders
)
SELECT customer_id,
       AVG(days_between) AS avg_days_between_orders,
       MIN(days_between) AS min_gap,
       MAX(days_between) AS max_gap
FROM OrderGaps
WHERE prev_order_date IS NOT NULL
GROUP BY customer_id;
```

112. Find employees hired in same month and year

```
SELECT DATE_FORMAT(hire_date, '%Y-%m') AS hire_month,
       COUNT(*) AS employees_hired,
       GROUP_CONCAT(CONCAT(first_name, ' ', last_name) SEPARATOR ', ') AS
employee_names
FROM employees
GROUP BY DATE_FORMAT(hire_date, '%Y-%m')
HAVING COUNT(*) > 1
ORDER BY hire_month;
```

113. Identify top 3 selling products per category

```
WITH ProductSales AS (
   SELECT p.category, p.product_id, p.product_name,
          SUM(oi.quantity * oi.price) AS total_revenue,
          ROW_NUMBER() OVER (PARTITION BY p.category ORDER BY SUM(oi.quantity *
oi.price) DESC) AS rank_in_category
   FROM products p
   JOIN order_items oi ON p.product_id = oi.product_id
```

```
    GROUP BY p.category, p.product_id, p.product_name
)
SELECT category, product_name, total_revenue, rank_in_category
FROM ProductSales
WHERE rank_in_category <= 3
ORDER BY category, rank_in_category;
```

114. Find customers with declining order values

```
WITH OrderTrends AS (
    SELECT customer_id, order_date, total_amount,
        LAG(total_amount, 1) OVER (PARTITION BY customer_id ORDER BY order_date) AS
prev_amount_1,
        LAG(total_amount, 2) OVER (PARTITION BY customer_id ORDER BY order_date) AS
prev_amount_2
    FROM orders
)
SELECT DISTINCT customer_id
FROM OrderTrends
WHERE total_amount < prev_amount_1
  AND prev_amount_1 < prev_amount_2;
```

115. Calculate retention rate by cohort

```
WITH FirstPurchase AS (
    SELECT customer_id,
        DATE_FORMAT(MIN(order_date), '%Y-%m') AS cohort_month
    FROM orders
    GROUP BY customer_id
),
CohortActivity AS (
    SELECT fp.cohort_month,
        DATE_FORMAT(o.order_date, '%Y-%m') AS activity_month,
        COUNT(DISTINCT o.customer_id) AS active_customers
    FROM FirstPurchase fp
    JOIN orders o ON fp.customer_id = o.customer_id
    GROUP BY fp.cohort_month, DATE_FORMAT(o.order_date, '%Y-%m')
)
SELECT cohort_month,
      MAX(CASE WHEN cohort_month = activity_month THEN active_customers END) AS
month_0,
```

```
    MAX(CASE WHEN activity_month =
DATE_FORMAT(DATE_ADD(STR_TO_DATE(CONCAT(cohort_month, '-01'), '%Y-%m-%d'),
INTERVAL 1 MONTH), '%Y-%m')
            THEN active_customers END) AS month_1,
    MAX(CASE WHEN activity_month =
DATE_FORMAT(DATE_ADD(STR_TO_DATE(CONCAT(cohort_month, '-01'), '%Y-%m-%d'),
INTERVAL 2 MONTH), '%Y-%m')
            THEN active_customers END) AS month_2
FROM CohortActivity
GROUP BY cohort_month
ORDER BY cohort_month;
```

116. Find missing dates in a sequence

```
WITH RECURSIVE DateRange AS (
  SELECT MIN(order_date) AS date_val, MAX(order_date) AS max_date
  FROM orders
  UNION ALL
  SELECT DATE_ADD(date_val, INTERVAL 1 DAY), max_date
  FROM DateRange
  WHERE date_val < max_date
)
SELECT dr.date_val AS missing_date
FROM DateRange dr
LEFT JOIN orders o ON dr.date_val = o.order_date
WHERE o.order_date IS NULL
LIMIT 100;  -- Prevent infinite results
```

117. Calculate customer lifetime value (CLV)

```
WITH CustomerMetrics AS (
  SELECT customer_id,
      COUNT(*) AS total_orders,
      SUM(total_amount) AS total_revenue,
      AVG(total_amount) AS avg_order_value,
      DATEDIFF(MAX(order_date), MIN(order_date)) AS customer_lifetime_days
  FROM orders
  GROUP BY customer_id
)
SELECT customer_id,
    total_orders,
    total_revenue,
    avg_order_value,
```

```
    customer_lifetime_days,
    CASE
      WHEN customer_lifetime_days > 0
      THEN (total_revenue / customer_lifetime_days) * 365
      ELSE total_revenue
    END AS estimated_annual_value
FROM CustomerMetrics
ORDER BY estimated_annual_value DESC;
```

118. Find overlapping date ranges (project conflicts)

```
SELECT p1.project_id AS project1_id, p1.project_name AS project1,
    p2.project_id AS project2_id, p2.project_name AS project2,
    p1.department_id,
    GREATEST(p1.start_date, p2.start_date) AS overlap_start,
    LEAST(p1.end_date, p2.end_date) AS overlap_end
FROM projects p1
JOIN projects p2
  ON p1.department_id = p2.department_id
  AND p1.project_id < p2.project_id
WHERE p1.start_date <= p2.end_date
 AND p2.start_date <= p1.end_date;
```

119. Pivot table: Convert rows to columns (quarterly sales)

```
SELECT customer_id,
    SUM(CASE WHEN QUARTER(order_date) = 1 THEN total_amount ELSE 0 END) AS
Q1_sales,
    SUM(CASE WHEN QUARTER(order_date) = 2 THEN total_amount ELSE 0 END) AS
Q2_sales,
    SUM(CASE WHEN QUARTER(order_date) = 3 THEN total_amount ELSE 0 END) AS
Q3_sales,
    SUM(CASE WHEN QUARTER(order_date) = 4 THEN total_amount ELSE 0 END) AS
Q4_sales
FROM orders
WHERE YEAR(order_date) = 2024
GROUP BY customer_id;
```

120. Find first and last transaction per customer per year

```
SELECT customer_id, YEAR(order_date) AS year,
    MIN(order_date) AS first_order,
    MAX(order_date) AS last_order,
```

```
    COUNT(*) AS orders_in_year,
    SUM(total_amount) AS year_total
FROM orders
GROUP BY customer_id, YEAR(order_date)
ORDER BY customer_id, year;
```

121. Calculate moving average (3-month window)

```
WITH MonthlySales AS (
  SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
      SUM(total_amount) AS monthly_total
  FROM orders
  GROUP BY DATE_FORMAT(order_date, '%Y-%m')
)
SELECT month, monthly_total,
    ROUND(AVG(monthly_total) OVER (
      ORDER BY month
      ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ), 2) AS moving_avg_3_months
FROM MonthlySales
ORDER BY month;
```

122. Find employees with salary gaps compared to peers

```
WITH DeptSalaries AS (
  SELECT employee_id, first_name, last_name, salary, department_id,
      AVG(salary) OVER (PARTITION BY department_id) AS dept_avg_salary,
      PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary) OVER (PARTITION BY
department_id) AS dept_median_salary
  FROM employees
)
SELECT employee_id, first_name, last_name, salary, department_id,
    ROUND(dept_avg_salary, 2) AS dept_avg,
    ROUND(dept_median_salary, 2) AS dept_median,
    ROUND(salary - dept_avg_salary, 2) AS variance_from_avg,
    ROUND((salary - dept_avg_salary) / dept_avg_salary * 100, 2) AS pct_diff
FROM DeptSalaries
WHERE ABS(salary - dept_avg_salary) > 10000
ORDER BY department_id, variance_from_avg DESC;
```

123. Identify products with seasonal patterns

```
SELECT p.product_id, p.product_name,
```

```
    SUM(CASE WHEN MONTH(o.order_date) IN (12,1,2) THEN oi.quantity ELSE 0 END) AS
winter_sales,
    SUM(CASE WHEN MONTH(o.order_date) IN (3,4,5) THEN oi.quantity ELSE 0 END) AS
spring_sales,
    SUM(CASE WHEN MONTH(o.order_date) IN (6,7,8) THEN oi.quantity ELSE 0 END) AS
summer_sales,
    SUM(CASE WHEN MONTH(o.order_date) IN (9,10,11) THEN oi.quantity ELSE 0 END) AS
fall_sales
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
JOIN orders o ON oi.order_id = o.order_id
GROUP BY p.product_id, p.product_name
HAVING SUM(oi.quantity) > 100;
```

124. Find customers who bought product A but not product B

```
SELECT DISTINCT c.customer_id, c.name
FROM customers c
WHERE EXISTS (
   SELECT 1
   FROM orders o
   JOIN order_items oi ON o.order_id = oi.order_id
   WHERE o.customer_id = c.customer_id
     AND oi.product_id = 1  -- Product A
)
AND NOT EXISTS (
   SELECT 1
   FROM orders o
   JOIN order_items oi ON o.order_id = oi.order_id
   WHERE o.customer_id = c.customer_id
     AND oi.product_id = 2  -- Product B
);
```

125. Calculate year-to-date (YTD) sales

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS month,
    SUM(total_amount) AS monthly_sales,
    SUM(SUM(total_amount)) OVER (
       PARTITION BY YEAR(order_date)
       ORDER BY MONTH(order_date)
    ) AS ytd_sales
FROM orders
GROUP BY YEAR(order_date), MONTH(order_date)
```

```
ORDER BY month;
```

126. Find departments with highest turnover (hire/fire ratio)

```
WITH DeptHires AS (
    SELECT department_id,
        COUNT(*) AS total_employees,
        SUM(CASE WHEN YEAR(hire_date) = YEAR(CURDATE()) THEN 1 ELSE 0 END) AS
hires_this_year
    FROM employees
    GROUP BY department_id
)
SELECT d.department_name,
    dh.total_employees,
    dh.hires_this_year,
    ROUND((dh.hires_this_year * 100.0 / dh.total_employees), 2) AS turnover_rate_pct
FROM departments d
JOIN DeptHires dh ON d.department_id = dh.department_id
ORDER BY turnover_rate_pct DESC;
```

127. Identify customers at risk of churn

```
WITH CustomerActivity AS (
    SELECT customer_id,
        MAX(order_date) AS last_order_date,
        COUNT(*) AS total_orders,
        AVG(DATEDIFF(order_date, LAG(order_date) OVER (PARTITION BY customer_id
ORDER BY order_date))) AS avg_days_between_orders
    FROM orders
    GROUP BY customer_id
)
SELECT customer_id,
    last_order_date,
    DATEDIFF(CURDATE(), last_order_date) AS days_since_last_order,
    total_orders,
    ROUND(avg_days_between_orders, 1) AS avg_order_gap,
    CASE
        WHEN DATEDIFF(CURDATE(), last_order_date) > avg_days_between_orders * 2 THEN
'High Risk'
        WHEN DATEDIFF(CURDATE(), last_order_date) > avg_days_between_orders * 1.5
THEN 'Medium Risk'
        ELSE 'Active'
    END AS churn_risk
```

```
FROM CustomerActivity
WHERE total_orders > 1
ORDER BY days_since_last_order DESC;
```

128. Calculate percentile ranks for employee salaries

```
SELECT employee_id, first_name, last_name, salary, department_id,
      PERCENT_RANK() OVER (ORDER BY salary) AS overall_percentile,
      PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY salary) AS
dept_percentile,
      CASE
         WHEN PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY salary)
>= 0.9 THEN 'Top 10%'
         WHEN PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY salary)
>= 0.75 THEN 'Top 25%'
         WHEN PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY salary)
>= 0.5 THEN 'Above Average'
         ELSE 'Below Average'
      END AS salary_bracket
FROM employees;
```

129. Find products frequently bought together (Market Basket Analysis)

```
WITH ProductPairs AS (
   SELECT oi1.product_id AS product_a,
         oi2.product_id AS product_b,
         COUNT(DISTINCT oi1.order_id) AS times_bought_together
   FROM order_items oi1
   JOIN order_items oi2
      ON oi1.order_id = oi2.order_id
      AND oi1.product_id < oi2.product_id
   GROUP BY oi1.product_id, oi2.product_id
)
SELECT p1.product_name AS product_1,
      p2.product_name AS product_2,
      pp.times_bought_together,
      ROUND(pp.times_bought_together * 100.0 / (SELECT COUNT(DISTINCT order_id) FROM
orders), 2) AS co_occurrence_pct
FROM ProductPairs pp
JOIN products p1 ON pp.product_a = p1.product_id
JOIN products p2 ON pp.product_b = p2.product_id
WHERE pp.times_bought_together >= 5
ORDER BY pp.times_bought_together DESC
```

```
LIMIT 20;
```

130. Calculate customer segmentation (RFM Analysis)

```
WITH RFM AS (
    SELECT customer_id,
        DATEDIFF(CURDATE(), MAX(order_date)) AS recency,
        COUNT(*) AS frequency,
        SUM(total_amount) AS monetary
    FROM orders
    GROUP BY customer_id
),
RFMScores AS (
    SELECT customer_id, recency, frequency, monetary,
        NTILE(5) OVER (ORDER BY recency DESC) AS r_score,
        NTILE(5) OVER (ORDER BY frequency ASC) AS f_score,
        NTILE(5) OVER (ORDER BY monetary ASC) AS m_score
    FROM RFM
)
SELECT customer_id, recency, frequency, monetary,
    r_score, f_score, m_score,
    CONCAT(r_score, f_score, m_score) AS rfm_score,
    CASE
        WHEN r_score >= 4 AND f_score >= 4 AND m_score >= 4 THEN 'Champions'
        WHEN r_score >= 3 AND f_score >= 3 AND m_score >= 3 THEN 'Loyal Customers'
        WHEN r_score >= 4 AND f_score <= 2 THEN 'New Customers'
        WHEN r_score <= 2 AND f_score >= 3 THEN 'At Risk'
        WHEN r_score <= 2 AND f_score <= 2 THEN 'Lost'
        ELSE 'Regular'
    END AS customer_segment
FROM RFMScores;
```

131. Find gaps in sequences (missing order IDs)

```
WITH OrderSequence AS (
    SELECT order_id,
        LAG(order_id) OVER (ORDER BY order_id) AS prev_order_id
    FROM orders
)
SELECT prev_order_id + 1 AS missing_start,
    order_id - 1 AS missing_end,
    order_id - prev_order_id - 1 AS gap_size
FROM OrderSequence
```

```
WHERE order_id - prev_order_id > 1
ORDER BY missing_start;
```

132. Calculate employee hierarchy depth

```
WITH RECURSIVE EmployeeHierarchy AS (
    SELECT employee_id, first_name, last_name, manager_id, 1 AS level
    FROM employees
    WHERE manager_id IS NULL

    UNION ALL

    SELECT e.employee_id, e.first_name, e.last_name, e.manager_id, eh.level + 1
    FROM employees e
    JOIN EmployeeHierarchy eh ON e.manager_id = eh.employee_id
)
SELECT employee_id, first_name, last_name, manager_id, level,
       (SELECT MAX(level) FROM EmployeeHierarchy) AS max_depth
FROM EmployeeHierarchy
ORDER BY level, employee_id;
```

133. Find outliers using standard deviation

```
WITH SalesStats AS (
    SELECT AVG(total_amount) AS mean_amount,
           STDDEV(total_amount) AS stddev_amount
    FROM orders
)
SELECT o.order_id, o.customer_id, o.total_amount,
       ROUND((o.total_amount - ss.mean_amount) / ss.stddev_amount, 2) AS z_score
FROM orders o
CROSS JOIN SalesStats ss
WHERE ABS(o.total_amount - ss.mean_amount) > 2 * ss.stddev_amount
ORDER BY ABS(o.total_amount - ss.mean_amount) DESC;
```

134. Calculate inventory turnover ratio

```
WITH ProductMovement AS (
    SELECT p.product_id, p.product_name, p.stock_quantity,
           SUM(oi.quantity) AS units_sold,
           COUNT(DISTINCT o.order_id) AS times_ordered
    FROM products p
    LEFT JOIN order_items oi ON p.product_id = oi.product_id
```

```
    LEFT JOIN orders o ON oi.order_id = o.order_id
    WHERE o.order_date >= DATE_SUB(CURDATE(), INTERVAL 12 MONTH)
    GROUP BY p.product_id, p.product_name, p.stock_quantity
)
SELECT product_id, product_name, stock_quantity, units_sold,
       CASE
          WHEN stock_quantity > 0
          THEN ROUND(units_sold / stock_quantity, 2)
          ELSE NULL
       END AS turnover_ratio,
       CASE
          WHEN stock_quantity > 0 AND units_sold / stock_quantity > 5 THEN 'Fast Moving'
          WHEN stock_quantity > 0 AND units_sold / stock_quantity BETWEEN 2 AND 5 THEN
'Moderate'
          WHEN stock_quantity > 0 AND units_sold / stock_quantity < 2 THEN 'Slow Moving'
          ELSE 'No Movement'
       END AS inventory_status
FROM ProductMovement
ORDER BY turnover_ratio DESC;
```

135. Find correlated products (often viewed together)

```
SELECT oi1.product_id AS product1,
       oi2.product_id AS product2,
       COUNT(*) AS co_occurrence,
       ROUND(COUNT(*) * 100.0 / (
          SELECT COUNT(DISTINCT order_id)
          FROM order_items
          WHERE product_id = oi1.product_id
       ), 2) AS lift_score
FROM order_items oi1
JOIN order_items oi2
    ON oi1.order_id = oi2.order_id
    AND oi1.product_id < oi2.product_id
GROUP BY oi1.product_id, oi2.product_id
HAVING COUNT(*) > 10
ORDER BY lift_score DESC
LIMIT 20;
```

136. Calculate daily active users (DAU) and monthly active users (MAU)

```
WITH DailyActive AS (
    SELECT DATE(order_date) AS activity_date,
```

```
        COUNT(DISTINCT customer_id) AS dau
    FROM orders
    GROUP BY DATE(order_date)
),
MonthlyActive AS (
    SELECT DATE_FORMAT(order_date, '%Y-%m') AS activity_month,
        COUNT(DISTINCT customer_id) AS mau
    FROM orders
    GROUP BY DATE_FORMAT(order_date, '%Y-%m')
)
SELECT da.activity_date,
    da.dau,
    ma.mau,
    ROUND(da.dau * 100.0 / ma.mau, 2) AS dau_mau_ratio
FROM DailyActive da
JOIN MonthlyActive ma
    ON DATE_FORMAT(da.activity_date, '%Y-%m') = ma.activity_month
ORDER BY da.activity_date DESC;
```

137. Find employees with longest tenure by department

```
WITH EmployeeTenure AS (
    SELECT employee_id, first_name, last_name, department_id, hire_date,
        TIMESTAMPDIFF(YEAR, hire_date, CURDATE()) AS years_employed,
        RANK() OVER (PARTITION BY department_id ORDER BY hire_date) AS tenure_rank
    FROM employees
)
SELECT d.department_name, et.first_name, et.last_name,
    et.hire_date, et.years_employed
FROM EmployeeTenure et
JOIN departments d ON et.department_id = d.department_id
WHERE et.tenure_rank = 1
ORDER BY et.years_employed DESC;
```

138. Calculate conversion funnel with drop-off rates

```
WITH Funnel AS (
    SELECT
        COUNT(DISTINCT customer_id) AS registered_users,
        COUNT(DISTINCT CASE WHEN EXISTS (
            SELECT 1 FROM orders o WHERE o.customer_id = c.customer_id
        ) THEN customer_id END) AS users_with_orders,
        COUNT(DISTINCT CASE WHEN EXISTS (
```

```
            SELECT 1 FROM orders o
            WHERE o.customer_id = c.customer_id
            GROUP BY o.customer_id
            HAVING COUNT(*) > 1
        ) THEN customer_id END) AS repeat_customers
    FROM customers c
)
SELECT registered_users,
       users_with_orders,
       repeat_customers,
       ROUND(users_with_orders * 100.0 / registered_users, 2) AS conversion_rate,
       ROUND(repeat_customers * 100.0 / users_with_orders, 2) AS retention_rate,
       ROUND((registered_users - users_with_orders) * 100.0 / registered_users, 2) AS
first_step_dropoff
FROM Funnel;
```

139. Find salary bands and distribution

```
SELECT
    CASE
        WHEN salary < 40000 THEN '0-40K'
        WHEN salary BETWEEN 40000 AND 60000 THEN '40K-60K'
        WHEN salary BETWEEN 60001 AND 80000 THEN '60K-80K'
        WHEN salary BETWEEN 80001 AND 100000 THEN '80K-100K'
        ELSE '100K+'
    END AS salary_band,
    COUNT(*) AS employee_count,
    ROUND(AVG(salary), 2) AS avg_salary,
    MIN(salary) AS min_salary,
    MAX(salary) AS max_salary,
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM employees), 2) AS percentage
FROM employees
GROUP BY salary_band
ORDER BY MIN(salary);
```

140. Find products with negative profit margin

```
WITH ProductCosts AS (
    SELECT p.product_id, p.product_name, p.price AS selling_price,
           AVG(oi.price) AS avg_cost_price,
           SUM(oi.quantity) AS units_sold
    FROM products p
    JOIN order_items oi ON p.product_id = oi.product_id
```

```
    GROUP BY p.product_id, p.product_name, p.price
)
SELECT product_id, product_name, selling_price,
       ROUND(avg_cost_price, 2) AS avg_cost,
       ROUND((selling_price - avg_cost_price), 2) AS profit_per_unit,
       ROUND(((selling_price - avg_cost_price) / selling_price * 100), 2) AS profit_margin_pct,
       units_sold
FROM ProductCosts
WHERE selling_price < avg_cost_price
ORDER BY profit_margin_pct;
```

141. Calculate customer lifetime by cohort

```
WITH CustomerCohort AS (
    SELECT customer_id,
           DATE_FORMAT(MIN(order_date), '%Y-%m') AS cohort_month,
           MIN(order_date) AS first_order,
           MAX(order_date) AS last_order,
           COUNT(*) AS order_count
    FROM orders
    GROUP BY customer_id
)
SELECT cohort_month,
       COUNT(*) AS customers,
       AVG(DATEDIFF(last_order, first_order)) AS avg_lifetime_days,
       AVG(order_count) AS avg_orders_per_customer,
       SUM(CASE WHEN DATEDIFF(last_order, first_order) > 365 THEN 1 ELSE 0 END) AS
customers_1year_plus
FROM CustomerCohort
GROUP BY cohort_month
ORDER BY cohort_month;
```

142. Find anomalies in daily order patterns

```
WITH DailySales AS (
    SELECT DATE(order_date) AS sale_date,
           COUNT(*) AS order_count,
           SUM(total_amount) AS daily_revenue
    FROM orders
    GROUP BY DATE(order_date)
),
SalesStats AS (
    SELECT AVG(daily_revenue) AS avg_revenue,
```

```
        STDDEV(daily_revenue) AS stddev_revenue
    FROM DailySales
)
SELECT ds.sale_date, ds.order_count, ds.daily_revenue,
       ROUND(ss.avg_revenue, 2) AS avg_daily_revenue,
       ROUND((ds.daily_revenue - ss.avg_revenue) / ss.stddev_revenue, 2) AS z_score,
       CASE
           WHEN ABS(ds.daily_revenue - ss.avg_revenue) > 2 * ss.stddev_revenue THEN
'Anomaly'
           ELSE 'Normal'
       END AS status
FROM DailySales ds
CROSS JOIN SalesStats ss
WHERE ABS(ds.daily_revenue - ss.avg_revenue) > 2 * ss.stddev_revenue
ORDER BY ABS(ds.daily_revenue - ss.avg_revenue) DESC;
```

143. Calculate week-over-week growth

```
WITH WeeklySales AS (
    SELECT YEARWEEK(order_date) AS week_num,
           DATE(DATE_SUB(order_date, INTERVAL WEEKDAY(order_date) DAY)) AS week_start,
           SUM(total_amount) AS weekly_revenue
    FROM orders
    GROUP BY YEARWEEK(order_date), DATE(DATE_SUB(order_date, INTERVAL
WEEKDAY(order_date) DAY))
)
SELECT week_num, week_start, weekly_revenue,
       LAG(weekly_revenue) OVER (ORDER BY week_num) AS prev_week_revenue,
       ROUND(weekly_revenue - LAG(weekly_revenue) OVER (ORDER BY week_num), 2) AS
absolute_growth,
       ROUND((weekly_revenue - LAG(weekly_revenue) OVER (ORDER BY week_num)) /
            LAG(weekly_revenue) OVER (ORDER BY week_num) * 100, 2) AS growth_pct
FROM WeeklySales
ORDER BY week_num DESC
LIMIT 12;
```

144. Find employees who manage multiple departments

```
SELECT m.employee_id, m.first_name, m.last_name,
       COUNT(DISTINCT e.department_id) AS departments_managed,
       GROUP_CONCAT(DISTINCT d.department_name SEPARATOR ', ') AS department_list,
       COUNT(e.employee_id) AS total_direct_reports
FROM employees m
```

```
JOIN employees e ON m.employee_id = e.manager_id
JOIN departments d ON e.department_id = d.department_id
GROUP BY m.employee_id, m.first_name, m.last_name
HAVING COUNT(DISTINCT e.department_id) > 1
ORDER BY departments_managed DESC;
```

145. Calculate ABC analysis for inventory (Pareto principle)

```
WITH ProductRevenue AS (
    SELECT p.product_id, p.product_name,
        SUM(oi.quantity * oi.price) AS total_revenue
    FROM products p
    JOIN order_items oi ON p.product_id = oi.product_id
    GROUP BY p.product_id, p.product_name
),
RankedProducts AS (
    SELECT product_id, product_name, total_revenue,
        SUM(total_revenue) OVER (ORDER BY total_revenue DESC) AS cumulative_revenue,
        SUM(total_revenue) OVER () AS total_revenue_all
    FROM ProductRevenue
)
SELECT product_id, product_name,
    ROUND(total_revenue, 2) AS revenue,
    ROUND(cumulative_revenue / total_revenue_all * 100, 2) AS cumulative_pct,
    CASE
        WHEN cumulative_revenue / total_revenue_all <= 0.80 THEN 'A - High Value'
        WHEN cumulative_revenue / total_revenue_all <= 0.95 THEN 'B - Medium Value'
        ELSE 'C - Low Value'
    END AS abc_category
FROM RankedProducts
ORDER BY total_revenue DESC;
```

146. Find customers with erratic purchasing behavior

```
WITH CustomerOrderFrequency AS (
    SELECT customer_id,
        order_date,
        LAG(order_date) OVER (PARTITION BY customer_id ORDER BY order_date) AS
prev_order_date,
        DATEDIFF(order_date, LAG(order_date) OVER (PARTITION BY customer_id ORDER
BY order_date)) AS days_between
    FROM orders
),
```

```
CustomerStats AS (
    SELECT customer_id,
        AVG(days_between) AS avg_gap,
        STDDEV(days_between) AS stddev_gap,
        MIN(days_between) AS min_gap,
        MAX(days_between) AS max_gap,
        COUNT(*) AS order_count
    FROM CustomerOrderFrequency
    WHERE prev_order_date IS NOT NULL
    GROUP BY customer_id
)
SELECT c.customer_id, c.name,
       ROUND(cs.avg_gap, 1) AS avg_days_between_orders,
       ROUND(cs.stddev_gap, 1) AS stddev,
       cs.min_gap, cs.max_gap, cs.order_count,
       ROUND(cs.stddev_gap / NULLIF(cs.avg_gap, 0), 2) AS coefficient_of_variation
FROM CustomerStats cs
JOIN customers c ON cs.customer_id = c.customer_id
WHERE cs.order_count >= 5
  AND cs.stddev_gap / NULLIF(cs.avg_gap, 0) > 1  -- High variability
ORDER BY coefficient_of_variation DESC;
```

147. Calculate employee promotion eligibility

```
WITH EmployeeMetrics AS (
    SELECT e.employee_id, e.first_name, e.last_name, e.salary, e.department_id,
        TIMESTAMPDIFF(MONTH, e.hire_date, CURDATE()) AS tenure_months,
        AVG(e2.salary) AS avg_peer_salary,
        COUNT(s.sale_id) AS total_sales,
        COALESCE(SUM(s.amount), 0) AS total_sales_amount
    FROM employees e
    LEFT JOIN employees e2 ON e.department_id = e2.department_id AND e.employee_id !=
e2.employee_id
    LEFT JOIN sales s ON e.employee_id = s.employee_id AND s.sale_date >=
DATE_SUB(CURDATE(), INTERVAL 12 MONTH)
    GROUP BY e.employee_id, e.first_name, e.last_name, e.salary, e.department_id, e.hire_date
)
SELECT employee_id, first_name, last_name, salary, tenure_months,
       ROUND(avg_peer_salary, 2) AS avg_peer_salary,
       total_sales, ROUND(total_sales_amount, 2) AS sales_amount,
       CASE
           WHEN tenure_months >= 24 AND salary <= avg_peer_salary AND total_sales > 50
THEN 'Highly Eligible'
           WHEN tenure_months >= 18 AND total_sales > 30 THEN 'Eligible'
```

```
        WHEN tenure_months >= 12 THEN 'Review Needed'
        ELSE 'Not Eligible'
    END AS promotion_status
FROM EmployeeMetrics
WHERE tenure_months >= 12
ORDER BY
  CASE
    WHEN tenure_months >= 24 AND salary <= avg_peer_salary AND total_sales > 50 THEN
1
    WHEN tenure_months >= 18 AND total_sales > 30 THEN 2
    ELSE 3
  END,
  total_sales_amount DESC;
```

148. Find temporal patterns (day of week analysis)

```
SELECT DAYNAME(order_date) AS day_of_week,
    DAYOFWEEK(order_date) AS day_num,
    COUNT(*) AS total_orders,
    ROUND(AVG(total_amount), 2) AS avg_order_value,
    SUM(total_amount) AS total_revenue,
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM orders), 2) AS pct_of_orders,
    ROUND(SUM(total_amount) * 100.0 / (SELECT SUM(total_amount) FROM orders), 2) AS
pct_of_revenue
FROM orders
GROUP BY DAYNAME(order_date), DAYOFWEEK(order_date)
ORDER BY day_num;
```

149. Calculate product recommendation score

```
WITH CustomerPurchases AS (
  SELECT DISTINCT o.customer_id, oi.product_id
  FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
),
ProductAffinities AS (
  SELECT cp1.product_id AS product_owned,
      cp2.product_id AS product_to_recommend,
      COUNT(DISTINCT cp1.customer_id) AS co_purchase_count
  FROM CustomerPurchases cp1
  JOIN CustomerPurchases cp2
    ON cp1.customer_id = cp2.customer_id
    AND cp1.product_id != cp2.product_id
```

```
    GROUP BY cp1.product_id, cp2.product_id
)
SELECT p1.product_name AS owned_product,
       p2.product_name AS recommended_product,
       pa.co_purchase_count,
       ROUND(pa.co_purchase_count * 100.0 / (
          SELECT COUNT(DISTINCT customer_id)
          FROM CustomerPurchases
          WHERE product_id = pa.product_owned
       ), 2) AS recommendation_score
FROM ProductAffinities pa
JOIN products p1 ON pa.product_owned = p1.product_id
JOIN products p2 ON pa.product_to_recommend = p2.product_id
WHERE pa.co_purchase_count >= 5
ORDER BY pa.product_owned, recommendation_score DESC;
```

150. Create a comprehensive business dashboard query

```
WITH CurrentMonth AS (
    SELECT
        COUNT(DISTINCT o.customer_id) AS active_customers,
        COUNT(o.order_id) AS total_orders,
        SUM(o.total_amount) AS total_revenue,
        ROUND(AVG(o.total_amount), 2) AS avg_order_value
    FROM orders o
    WHERE DATE_FORMAT(o.order_date, '%Y-%m') = DATE_FORMAT(CURDATE(), '%Y-%m')
),
PreviousMonth AS (
    SELECT
        COUNT(DISTINCT o.customer_id) AS active_customers,
        COUNT(o.order_id) AS total_orders,
        SUM(o.total_amount) AS total_revenue,
        ROUND(AVG(o.total_amount), 2) AS avg_order_value
    FROM orders o
    WHERE DATE_FORMAT(o.order_date, '%Y-%m') =
DATE_FORMAT(DATE_SUB(CURDATE(), INTERVAL 1 MONTH), '%Y-%m')
),
TopProducts AS (
    SELECT p.product_name, SUM(oi.quantity) AS units_sold
    FROM order_items oi
    JOIN products p ON oi.product_id = p.product_id
    JOIN orders o ON oi.order_id = o.order_id
    WHERE DATE_FORMAT(o.order_date, '%Y-%m') = DATE_FORMAT(CURDATE(), '%Y-%m')
    GROUP BY p.product_id, p.product_name
```

```
    ORDER BY units_sold DESC
    LIMIT 1
),
CustomerMetrics AS (
    SELECT
        COUNT(DISTINCT CASE WHEN order_count = 1 THEN customer_id END) AS
new_customers,
        COUNT(DISTINCT CASE WHEN order_count > 1 THEN customer_id END) AS
returning_customers
    FROM (
        SELECT customer_id, COUNT(*) AS order_count
        FROM orders
        WHERE DATE_FORMAT(order_date, '%Y-%m') = DATE_FORMAT(CURDATE(), '%Y-%m')
        GROUP BY customer_id
    ) customer_orders
)
SELECT
    'Current Month' AS period,
    cm.active_customers,
    cm.total_orders,
    ROUND(cm.total_revenue, 2) AS revenue,
    cm.avg_order_value,
    ROUND((cm.total_revenue - pm.total_revenue) / pm.total_revenue * 100, 2) AS
revenue_growth_pct,
    ROUND((cm.total_orders - pm.total_orders) / pm.total_orders * 100, 2) AS order_growth_pct,
    cust.new_customers,
    cust.returning_customers,
    ROUND(cust.returning_customers * 100.0 / cm.active_customers, 2) AS retention_rate,
    tp.product_name AS top_product,
    tp.units_sold AS top_product_sales
FROM CurrentMonth cm
CROSS JOIN PreviousMonth pm
CROSS JOIN TopProducts tp
CROSS JOIN CustomerMetrics cust;
```

Interview Tips

How to Practice

1. **Understand the business question first** - What insight is needed?
2. **Break down complex queries** - Build them step by step

3. **Explain your logic** - Interviewers want to see your thought process
4. **Optimize for readability** - Use CTEs instead of nested subqueries
5. **Consider performance** - Think about indexes and query costs
6. **Test edge cases** - NULL values, empty results, etc.

Study Tips

1. **Start with the basics** - Master SELECT, WHERE, and JOIN before moving to complex queries
2. **Practice CTEs** - Common Table Expressions make complex queries more readable
3. **Understand window functions** - These are powerful for analytics and rankings
4. **Master aggregations** - GROUP BY and HAVING are essential for data analysis
5. **Learn subqueries** - Both correlated and non-correlated subqueries are crucial
6. **Practice recursive queries** - Useful for hierarchical data like org charts
7. **Understand performance** - Learn about indexes and query optimization
8. **Test incrementally** - Build complex queries step by step
9. **Use proper formatting** - Well-formatted SQL is easier to debug
10. **Practice regularly** - The more you write queries, the more natural it becomes

Key Concepts Covered

- **Basic**: SELECT, WHERE, ORDER BY, LIMIT, LIKE, IN, BETWEEN, NULL handling, basic aggregations (COUNT, SUM, AVG, MIN, MAX), GROUP BY, simple JOINs
- **Medium**: Subqueries, multiple JOINs, HAVING clause, date functions, string functions, window functions (RANK, ROW_NUMBER), CASE statements, self-joins, UNION operations
- **Advanced**: CTEs (Common Table Expressions), recursive queries, complex window functions (LEAD, LAG, NTILE, PERCENTILE), advanced analytics (cohort analysis, RFM, funnel analysis), time series analysis, statistical calculations, hierarchical queries, pivoting/unpivoting

Practice these queries with your own sample data to truly master SQL!