By - Krishna Singh

300 Real SQL Interview Questions Asked at PwC, Deloitte, EY, KPMG, Tredence, Persistent Systems and Accenture & More.

Medium to Advanced SQL Questions (**01–300**)

1. Find the second highest salary from the Employee table.

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

2. Find duplicate records in a table.

```
SELECT name, COUNT(*)
FROM employees
GROUP BY name
HAVING COUNT(*) > 1;
```

3. Retrieve employees who earn more than their manager.

```
SELECT e.name AS Employee, e.salary, m.name AS
Manager, m.salary AS ManagerSalary
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.salary > m.salary;
```

4. Count employees in each department having more than 5 employees.

```
SELECT department_id, COUNT(*) AS
num_employees
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

5. Find employees who joined in the last 6 months.

```
SELECT *
FROM employees
WHERE join_date >= CURRENT_DATE -
INTERVAL '6 months';
```

6. Get departments with no employees.

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

7. Write a query to find the median salary.

```
SELECT AVG(salary) AS median_salary
```

```
FROM (
  SELECT salary
  FROM employees
  ORDER BY salary
  LIMIT 2 - (SELECT COUNT(*) FROM employees)
% 2
  OFFSET (SELECT (COUNT(*) - 1) / 2 FROM
employees)
) AS median_subquery;
```

8. Running total of salaries by department.

```
SELECT name, department_id, salary,
       SUM(salary) OVER (PARTITION BY
department_id ORDER BY id) AS running_total
FROM employees;
```

9. Find the longest consecutive streak of daily logins for each user.

```
WITH login_dates AS (
  SELECT user_id, login_date,
         login_date - INTERVAL ROW_NUMBER()
OVER (PARTITION BY user_id ORDER BY
login_date) DAY AS grp
  FROM user_logins
)
SELECT user_id, COUNT(*) AS streak_length,
MIN(login_date) AS start_date, MAX(login_date) AS
end_date
FROM login_dates
GROUP BY user_id, grp
```

```
ORDER BY streak_length DESC;
```

10. Recursive query to find the full reporting chain for each employee.

```
WITH RECURSIVE reporting_chain AS (
  SELECT id, name, manager_id, 1 AS level
  FROM employees
  WHERE manager_id IS NULL

  UNION ALL

  SELECT e.id, e.name, e.manager_id, rc.level + 1
  FROM employees e
  JOIN reporting_chain rc ON e.manager_id = rc.id
)
SELECT * FROM reporting_chain ORDER BY level,
id;
```

11. Write a query to find gaps in a sequence of numbers (missing IDs).

```
SELECT (id + 1) AS missing_id
FROM employees e1
WHERE NOT EXISTS (
  SELECT 1 FROM employees e2 WHERE e2.id =
e1.id + 1
)
ORDER BY missing_id;
```

12. Calculate cumulative distribution (CDF) of salaries.

```
SELECT name, salary,
```

```
    CUME_DIST() OVER (ORDER BY salary) AS
salary_cdf
FROM employees;
```

13. Compare two tables and find rows with differences in any column (all columns).

```
SELECT *
FROM table1 t1
FULL OUTER JOIN table2 t2 ON t1.id = t2.id
WHERE t1.col1 IS DISTINCT FROM t2.col1
  OR t1.col2 IS DISTINCT FROM t2.col2
  OR t1.col3 IS DISTINCT FROM t2.col3;
```

14. Write a query to rank employees based on salary with ties handled properly.

```
SELECT name, salary,
    RANK() OVER (ORDER BY salary DESC) AS
salary_rank
FROM employees;
```

15. Find customers who have not made any purchase.

```
SELECT c.customer_id, c.name
FROM customers c
LEFT JOIN sales s ON c.customer_id = s.customer_id
WHERE s.sale_id IS NULL;
```

16. Write a query to perform a conditional aggregation (count males and females in each department).

```
SELECT department_id,
```

```
    COUNT(CASE WHEN gender = 'M' THEN 1
END) AS male_count,
    COUNT(CASE WHEN gender = 'F' THEN 1
END) AS female_count
FROM employees
GROUP BY department_id;
```

17. Write a query to calculate the difference between current row and previous row's salary (lag function).

```
SELECT name, salary,
    salary - LAG(salary) OVER (ORDER BY id) AS
salary_diff
FROM employees;
```

18. Identify overlapping date ranges for bookings.

```
SELECT b1.booking_id, b2.booking_id
FROM bookings b1
JOIN bookings b2 ON b1.booking_id <> b2.booking_id
WHERE b1.start_date <= b2.end_date
  AND b1.end_date >= b2.start_date;
```

19. Write a query to find employees with salary greater than average salary in the entire company, ordered by salary descending.

```
SELECT name, salary
FROM employees
```

```
WHERE salary > (SELECT AVG(salary) FROM
employees)
ORDER BY salary DESC;
```

20. Aggregate JSON data (if supported) to list all employee names in a department as a JSON array.

```
SELECT department_id, JSON_AGG(name) AS
employee_names
FROM employees
GROUP BY department_id;
```

21. Find employees who have the same salary as their manager.

```
SELECT e.name AS Employee, e.salary, m.name AS
Manager
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.salary = m.salary;
```

22. Write a query to get the first and last purchase date for each customer.

```
SELECT customer_id,
    MIN(purchase_date) AS first_purchase,
    MAX(purchase_date) AS last_purchase
FROM sales
GROUP BY customer_id;
```

23. Find departments with the highest average salary.

```
WITH avg_salaries AS (
  SELECT department_id, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY department_id
)
SELECT *
FROM avg_salaries
WHERE avg_salary = (SELECT MAX(avg_salary)
FROM avg_salaries);
```

24. Write a query to find the number of employees in each job title.

```
SELECT job_title, COUNT(*) AS num_employees
FROM employees
GROUP BY job_title;
```

25. Find employees who don't have a department assigned.

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

21. Write a query to find the difference in days between two dates in the same table.

```
SELECT id, DATEDIFF(day, start_date, end_date) AS
days_difference
FROM projects;
```

Note: DATEDIFF syntax varies — replace accordingly (e.g., DATEDIFF('day', start_date, end_date) in some systems).

22. Calculate the moving average of salaries over the last 3 employees ordered by hire date.

```
SELECT name, hire_date, salary,
       AVG(salary) OVER (ORDER BY hire_date
ROWS BETWEEN 2 PRECEDING AND CURRENT
ROW) AS moving_avg_salary
FROM employees;
```

23. Find the most recent purchase per customer using window functions.

```
SELECT *
FROM (
  SELECT *,
         ROW_NUMBER() OVER (PARTITION BY
customer_id ORDER BY purchase_date DESC) AS rn
  FROM sales
) sub
WHERE rn = 1;
```

24. Detect hierarchical depth of each employee in the org chart.

```
WITH RECURSIVE employee_depth AS (
  SELECT id, name, manager_id, 1 AS depth
  FROM employees
  WHERE manager_id IS NULL
```

```
  UNION ALL

  SELECT e.id, e.name, e.manager_id, ed.depth + 1
  FROM employees e
  JOIN employee_depth ed ON e.manager_id = ed.id
)
SELECT * FROM employee_depth;
```

25. Write a query to perform a self-join to find pairs of employees in the same department.

```
SELECT e1.name AS Employee1, e2.name AS
Employee2, e1.department_id
FROM employees e1
JOIN employees e2 ON e1.department_id =
e2.department_id AND e1.id < e2.id;
```

26. Write a query to pivot rows into columns dynamically (if dynamic pivot is not supported, simulate it for fixed values).

```
SELECT
    department_id,
    SUM(CASE WHEN job_title = 'Manager' THEN 1
ELSE 0 END) AS Managers,
    SUM(CASE WHEN job_title = 'Developer' THEN 1
ELSE 0 END) AS Developers,
    SUM(CASE WHEN job_title = 'Tester' THEN 1
ELSE 0 END) AS Testers
FROM employees
GROUP BY department_id;
```

27. Find customers who made purchases in every category available.

```
SELECT customer_id
FROM sales s
GROUP BY customer_id
HAVING COUNT(DISTINCT category_id) =
(SELECT COUNT(DISTINCT category_id) FROM
sales);
```

28. Identify employees who haven't received a salary raise in more than a year.

```
SELECT e.name
FROM employees e
JOIN salary_history sh ON e.id = sh.employee_id
GROUP BY e.id, e.name
HAVING MAX(sh.raise_date) < CURRENT_DATE -
INTERVAL '1 year';
```

29. Write a query to rank salespeople by monthly sales, resetting the rank every month.

```
SELECT salesperson_id, sale_month, total_sales,
      RANK() OVER (PARTITION BY sale_month
ORDER BY total_sales DESC) AS monthly_rank
FROM (
  SELECT salesperson_id, DATE_TRUNC('month',
sale_date) AS sale_month, SUM(amount) AS
total_sales
  FROM sales
  GROUP BY salesperson_id, sale_month
) AS monthly_sales;
```

30. Calculate the percentage change in sales compared to the previous month for each product.

```
SELECT product_id, sale_month, total_sales,
       (total_sales - LAG(total_sales) OVER
(PARTITION BY product_id ORDER BY
sale_month)) * 100.0 /
       LAG(total_sales) OVER (PARTITION BY
product_id ORDER BY sale_month) AS pct_change
FROM (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS sale_month, SUM(amount) AS
total_sales
  FROM sales
  GROUP BY product_id, sale_month
) monthly_sales;
```

31. Find employees who earn more than the average salary across the company but less than the highest salary in their department.

```
SELECT *
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM
employees)
  AND salary < (
    SELECT MAX(salary)
    FROM employees
    WHERE department_id = e.department_id
  );
```

32. Retrieve the last 5 orders for each customer.

```
SELECT *
FROM (
  SELECT o.*, ROW_NUMBER() OVER
(PARTITION BY customer_id ORDER BY order_date
DESC) AS rn
  FROM orders o
) sub
WHERE rn <= 5;
```

33. Find employees with no salary changes in the last 2 years.

```
SELECT e.*
FROM employees e
LEFT JOIN salary_history sh ON e.id =
sh.employee_id AND sh.change_date >=
CURRENT_DATE - INTERVAL '2 years'
WHERE sh.employee_id IS NULL;
```

34. Find the department with the lowest average salary.

```
SELECT department_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id
ORDER BY avg_salary
LIMIT 1;
```

35. List employees whose names start and end with the same letter.

```
SELECT *
```

```
FROM employees
WHERE LEFT(name, 1) = RIGHT(name, 1);
```

Questions

31. Write a query to detect circular references in employee-manager hierarchy (cycles).

```
WITH RECURSIVE mgr_path (id, manager_id, path)
AS (
  SELECT id, manager_id, ARRAY[id]
  FROM employees
  WHERE manager_id IS NOT NULL

  UNION ALL

  SELECT e.id, e.manager_id, path || e.id
  FROM employees e
  JOIN mgr_path mp ON e.manager_id = mp.id
  WHERE NOT e.id = ANY(path)
)
SELECT DISTINCT id
FROM mgr_path
WHERE id = ANY(path);
```

32. Write a query to get the running total of sales per customer, ordered by sale date.

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

33. Find the department-wise salary percentile (e.g., 90th percentile) using window functions.

```
SELECT department_id, salary,
       PERCENTILE_CONT(0.9) WITHIN GROUP
(ORDER BY salary) OVER (PARTITION BY
department_id) AS pct_90_salary
FROM employees;
```

34. Find employees whose salary is a prime number.

```
WITH primes AS (
  SELECT generate_series(2, (SELECT MAX(salary)
FROM employees)) AS num
  EXCEPT
  SELECT num FROM (
    SELECT num, UNNEST(ARRAY(
      SELECT generate_series(2, FLOOR(SQRT(num)))
AS divisor
    )) AS divisor
    WHERE num % divisor = 0
  ) AS composite
)
SELECT *
FROM employees
WHERE salary IN (SELECT num FROM primes);
```

Note: This is a conceptual approach—some databases may not support this syntax fully.

35. Find employees who have worked for multiple departments over time.

```
SELECT employee_id
FROM employee_department_history
GROUP BY employee_id
HAVING COUNT(DISTINCT department_id) > 1;
```

36. Use window function to find the difference between current row's sales and previous row's sales partitioned by product.

```
SELECT product_id, sale_date, amount,
       amount - LAG(amount) OVER (PARTITION BY
product_id ORDER BY sale_date) AS sales_diff
FROM sales;
```

37. Write a query to find all employees who are at the lowest level in the hierarchy (no subordinates).

```
SELECT *
FROM employees e
WHERE NOT EXISTS (
  SELECT 1 FROM employees sub WHERE
sub.manager_id = e.id
);
```

38. Find average order value per month and product category.

```
SELECT DATE_TRUNC('month', order_date) AS
order_month, category_id, AVG(order_value) AS
avg_order_value
FROM orders
```

```
GROUP BY order_month, category_id;
```

39. Write a query to create a running count of how many employees joined in each year.

```
SELECT join_year, COUNT(*) AS yearly_hires,
       SUM(COUNT(*)) OVER (ORDER BY join_year)
AS running_total_hires
FROM (
  SELECT EXTRACT(YEAR FROM hire_date) AS
join_year
  FROM employees
) sub
GROUP BY join_year
ORDER BY join_year;
```

40. Write a query to find the second most recent order date per customer.

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

41. Find employees who have never made a sale.

```
SELECT e.id, e.name
FROM employees e
```

```
LEFT JOIN sales s ON e.id = s.employee_id
WHERE s.sale_id IS NULL;
```

42. Find the average tenure of employees by department.

```
SELECT department_id, AVG(DATE_PART('year', CURRENT_DATE - hire_date)) AS avg_tenure_years
FROM employees
GROUP BY department_id;
```

43. Get employees with salary in the top 10% in their department.

```
SELECT *
FROM (
  SELECT e.*, NTILE(10) OVER (PARTITION BY department_id ORDER BY salary DESC) AS decile
  FROM employees e
) sub
WHERE decile = 1;
```

44. Find customers who purchased more than once in the same day.

```
SELECT customer_id, purchase_date, COUNT(*) AS purchase_count
FROM sales
GROUP BY customer_id, purchase_date
HAVING COUNT(*) > 1;
```

45. List all departments and their employee counts, including departments with zero employees.

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

41. Write a query to find duplicate rows based on multiple columns.

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

42. Write a recursive query to calculate factorial of a number (e.g., 5).

```
WITH RECURSIVE factorial(n, fact) AS (
  SELECT 1, 1
  UNION ALL
  SELECT n + 1, fact * (n + 1)
  FROM factorial
  WHERE n < 5
)
SELECT fact FROM factorial WHERE n = 5;
```

43. Write a query to calculate the cumulative percentage of total sales per product.

```
SELECT product_id, sale_amount,
```

```
        SUM(sale_amount) OVER (ORDER BY
sale_amount DESC) * 100.0 / SUM(sale_amount)
OVER () AS cumulative_pct
FROM sales;
```

44. Write a query to get employees who reported directly or indirectly to a given manager (hierarchy traversal).

```
WITH RECURSIVE reporting AS (
  SELECT id, name, manager_id
  FROM employees
  WHERE manager_id = 101  -- replace 101 with
manager's id

  UNION ALL

  SELECT e.id, e.name, e.manager_id
  FROM employees e
  INNER JOIN reporting r ON e.manager_id = r.id
)
SELECT * FROM reporting;
```

45. Find the average number of orders per customer and standard deviation.

```
SELECT AVG(order_count) AS avg_orders,
STDDEV(order_count) AS stddev_orders
FROM (
  SELECT customer_id, COUNT(*) AS order_count
  FROM orders
  GROUP BY customer_id
```

```
) sub;
```

46. Find gaps in date sequences for each customer (missing days).

```
WITH dates AS (
  SELECT customer_id, purchase_date,
        LAG(purchase_date) OVER (PARTITION BY customer_id ORDER BY purchase_date) AS prev_date
  FROM sales
)
SELECT customer_id, prev_date + INTERVAL '1 day' AS missing_date
FROM dates
WHERE purchase_date > prev_date + INTERVAL '1 day';
```

47. Rank employees by salary within their department, and calculate percent rank.

```
SELECT name, department_id, salary,
      RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS salary_rank,
      PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS salary_percent_rank
FROM employees;
```

48. Find products that have never been sold.

```
SELECT p.product_id, p.product_name
FROM products p
LEFT JOIN sales s ON p.product_id = s.product_id
```

```
WHERE s.sale_id IS NULL;
```

49. Write a query to find consecutive days where sales were above a threshold.

```
WITH flagged_sales AS (
  SELECT sale_date, amount,
        CASE WHEN amount > 1000 THEN 1 ELSE 0
END AS flag
  FROM sales
),
groups AS (
  SELECT sale_date, amount, flag,
        sale_date - INTERVAL ROW_NUMBER()
OVER (ORDER BY sale_date) DAY AS grp
  FROM flagged_sales
  WHERE flag = 1
)
SELECT MIN(sale_date) AS start_date,
MAX(sale_date) AS end_date, COUNT(*) AS
consecutive_days
FROM groups
GROUP BY grp
ORDER BY consecutive_days DESC;
```

50. Write a query to concatenate employee names in each department (string aggregation).

```
SELECT department_id, STRING_AGG(name, ', ') AS
employee_names
FROM employees
GROUP BY department_id;
```

51. Find employees whose salary is above the average salary of their department but below the company-wide average.

```
SELECT *
FROM employees e
WHERE salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department_id = e.department_id
)
AND salary < (SELECT AVG(salary) FROM employees);
```

52. List the customers who purchased all products in a specific category.

```
SELECT customer_id
FROM sales
WHERE category_id = 10 -- example category
GROUP BY customer_id
HAVING COUNT(DISTINCT product_id) = (
    SELECT COUNT(DISTINCT product_id)
    FROM products
    WHERE category_id = 10
);
```

53. Retrieve the Nth highest salary from the employees table.

```
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
LIMIT 1 OFFSET N-1;
```

(Replace N with the desired rank, e.g., N=3 for third highest)

54. Find employees with no corresponding entries in the salary_history table.

```
SELECT e.*
FROM employees e
LEFT JOIN salary_history sh ON e.id =
sh.employee_id
WHERE sh.employee_id IS NULL;
```

55. Show the department with the highest number of employees and the count.

```
SELECT department_id, COUNT(*) AS
employee_count
FROM employees
GROUP BY department_id
ORDER BY employee_count DESC
LIMIT 1;
```

51. Write a recursive query to list all ancestors (managers) of a given employee.

```
WITH RECURSIVE ancestors AS (
    SELECT id, name, manager_id
```

```
    FROM employees
    WHERE id = 123 -- given employee id

    UNION ALL

    SELECT e.id, e.name, e.manager_id
    FROM employees e
    JOIN ancestors a ON e.id = a.manager_id
)
SELECT * FROM ancestors WHERE id != 123;
```

52. Calculate the median salary by department using window functions.

```
SELECT DISTINCT department_id,
       PERCENTILE_CONT(0.5) WITHIN GROUP
(ORDER BY salary) OVER (PARTITION BY
department_id) AS median_salary
FROM employees;
```

53. Write a query to find the first purchase date and last purchase date for each customer, including customers who never purchased anything.

```
SELECT c.customer_id,
       MIN(s.purchase_date) AS first_purchase,
       MAX(s.purchase_date) AS last_purchase
FROM customers c
LEFT JOIN sales s ON c.customer_id = s.customer_id
GROUP BY c.customer_id;
```

54. Find the percentage difference between each month's total sales and the previous month's total sales.

```
WITH monthly_sales AS (
  SELECT DATE_TRUNC('month', sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY month
)
SELECT month, total_sales,
       (total_sales - LAG(total_sales) OVER (ORDER BY month)) * 100.0 / LAG(total_sales) OVER (ORDER BY month) AS pct_change
FROM monthly_sales;
```

55. Write a query to find employees who have the longest tenure within their department.

```
WITH tenure AS (
  SELECT *,
         RANK() OVER (PARTITION BY department_id ORDER BY hire_date ASC) AS tenure_rank
  FROM employees
)
SELECT *
FROM tenure
WHERE tenure_rank = 1;
```

56. Generate a report that shows sales and sales growth percentage compared to the same month last year.

```
WITH monthly_sales AS (
```

```
  SELECT DATE_TRUNC('month', sale_date) AS
month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY month
)
SELECT ms1.month, ms1.total_sales,
      ((ms1.total_sales - ms2.total_sales) * 100.0 /
ms2.total_sales) AS growth_pct
FROM monthly_sales ms1
LEFT JOIN monthly_sales ms2 ON ms1.month =
ms2.month + INTERVAL '1 year';
```

57. Write a query to identify overlapping shifts for employees.

```
SELECT s1.employee_id, s1.shift_id AS shift1,
s2.shift_id AS shift2
FROM shifts s1
JOIN shifts s2 ON s1.employee_id = s2.employee_id
AND s1.shift_id <> s2.shift_id
WHERE s1.start_time < s2.end_time AND s1.end_time
> s2.start_time;
```

58. Calculate the total revenue for each customer, and rank them from highest to lowest spender.

```
SELECT customer_id, SUM(amount) AS
total_revenue,
      RANK() OVER (ORDER BY SUM(amount)
DESC) AS revenue_rank
FROM sales
GROUP BY customer_id;
```

59. Write a query to find the employee(s) who have never received a promotion.

```
SELECT e.*
FROM employees e
LEFT JOIN promotions p ON e.id = p.employee_id
WHERE p.employee_id IS NULL;
```

60. Write a query to find the top 3 products with the highest total sales amount each month.

```
WITH monthly_product_sales AS (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY product_id, month
),
ranked_sales AS (
  SELECT *, RANK() OVER (PARTITION BY month
ORDER BY total_sales DESC) AS sales_rank
  FROM monthly_product_sales
)
SELECT product_id, month, total_sales
FROM ranked_sales
WHERE sales_rank <= 3
ORDER BY month, sales_rank;
```

61. Find the customers who placed orders only in the last 30 days.

```
SELECT DISTINCT customer_id
FROM orders
WHERE order_date >= CURRENT_DATE -
INTERVAL '30 days'
AND customer_id NOT IN (
  SELECT DISTINCT customer_id
  FROM orders
  WHERE order_date < CURRENT_DATE -
INTERVAL '30 days'
);
```

62. Find products that have never been ordered.

```
SELECT p.product_id, p.product_name
FROM products p
LEFT JOIN orders o ON p.product_id = o.product_id
WHERE o.order_id IS NULL;
```

63. Find employees whose salary is above their department's average but below the overall average salary.

```
SELECT *
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM
employees WHERE department_id = e.department_id)
AND salary < (SELECT AVG(salary) FROM
employees);
```

64. Calculate the total sales amount and number of orders per customer in the last year.

```
SELECT customer_id, COUNT(*) AS total_orders,
SUM(amount) AS total_sales
FROM sales
WHERE sale_date >= CURRENT_DATE -
INTERVAL '1 year'
GROUP BY customer_id;
```

65. List the top 5 highest-paid employees per department.

```
SELECT *
FROM (
  SELECT e.*, ROW_NUMBER() OVER
(PARTITION BY department_id ORDER BY salary
DESC) AS rn
  FROM employees e
) sub
WHERE rn <= 5;
```

61. Write a query to identify “gaps and islands” in attendance records (consecutive dates present).

```
WITH attendance_groups AS (
  SELECT employee_id, attendance_date,
         attendance_date - INTERVAL ROW_NUMBER()
OVER (PARTITION BY employee_id ORDER BY
attendance_date) DAY AS grp
  FROM attendance
)
SELECT employee_id, MIN(attendance_date) AS
start_date, MAX(attendance_date) AS end_date,
COUNT(*) AS consecutive_days
```

```
FROM attendance_groups
GROUP BY employee_id, grp
ORDER BY employee_id, start_date;
```

62. Write a recursive query to list all descendants of a manager in an organizational hierarchy.

```
WITH RECURSIVE descendants AS (
  SELECT id, name, manager_id
  FROM employees
  WHERE manager_id = 100 -- starting manager id

  UNION ALL

  SELECT e.id, e.name, e.manager_id
  FROM employees e
  INNER JOIN descendants d ON e.manager_id = d.id
)
SELECT * FROM descendants;
```

63. Calculate a 3-month moving average of monthly sales per product.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY product_id, month
)
SELECT product_id, month, total_sales,
       AVG(total_sales) OVER (PARTITION BY
product_id ORDER BY month ROWS BETWEEN 2
```

```
PRECEDING AND CURRENT ROW) AS
moving_avg
FROM monthly_sales;
```

64. Write a query to find employees who have the same hire date as their managers.

```
SELECT e.name AS employee_name, m.name AS
manager_name, e.hire_date
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.hire_date = m.hire_date;
```

65. Write a query to find products with increasing sales over the last 3 months.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY product_id, month
),
ranked_sales AS (
  SELECT product_id, month, total_sales,
        ROW_NUMBER() OVER (PARTITION BY
product_id ORDER BY month DESC) AS rn
  FROM monthly_sales
)
SELECT ms1.product_id
FROM ranked_sales ms1
JOIN ranked_sales ms2 ON ms1.product_id =
ms2.product_id AND ms1.rn = 1 AND ms2.rn = 2
```

```
JOIN ranked_sales ms3 ON ms1.product_id =
ms3.product_id AND ms3.rn = 3
WHERE ms3.total_sales < ms2.total_sales AND
ms2.total_sales < ms1.total_sales;
```

66. Write a query to get the nth highest salary per department.

```
SELECT department_id, salary
FROM (
  SELECT department_id, salary, ROW_NUMBER()
OVER (PARTITION BY department_id ORDER BY
salary DESC) AS rn
  FROM employees
) sub
WHERE rn = N;
```

(Replace N with the desired rank.)

67. Find employees who have managed more than 3 projects.

```
SELECT manager_id, COUNT(DISTINCT project_id)
AS project_count
FROM projects
GROUP BY manager_id
HAVING COUNT(DISTINCT project_id) > 3;
```

--68. Write a query to calculate the difference in days between each employee's hire date and their manager’s hire date.

```
SELECT e.name AS employee, m.name AS manager,
      DATEDIFF(day, m.hire_date, e.hire_date) AS
days_difference
FROM employees e
JOIN employees m ON e.manager_id = m.id;
```

(Syntax of DATEDIFF varies by SQL dialect)

69. Write a query to find the department with the highest average years of experience.

```
SELECT department_id, AVG(EXTRACT(year FROM
CURRENT_DATE - hire_date)) AS
avg_experience_years
FROM employees
GROUP BY department_id
ORDER BY avg_experience_years DESC
LIMIT 1;
```

70. Identify employees who had overlapping project assignments.

```
SELECT p1.employee_id, p1.project_id AS project1,
p2.project_id AS project2
FROM project_assignments p1
JOIN project_assignments p2 ON p1.employee_id =
p2.employee_id AND p1.project_id <> p2.project_id
WHERE p1.start_date < p2.end_date AND p1.end_date
> p2.start_date;
```

71. Find customers who made purchases in every month of the current year.

```
WITH months AS (
  SELECT generate_series(1, 12) AS month
),
customer_months AS (
  SELECT customer_id, EXTRACT(MONTH FROM purchase_date) AS month
  FROM sales
  WHERE EXTRACT(YEAR FROM purchase_date) = EXTRACT(YEAR FROM CURRENT_DATE)
  GROUP BY customer_id, EXTRACT(MONTH FROM purchase_date)
)
SELECT customer_id
FROM customer_months
GROUP BY customer_id
HAVING COUNT(DISTINCT month) = 12;
```

72. List employees who earn more than all their subordinates.

```
SELECT e.id, e.name, e.salary
FROM employees e
WHERE e.salary > ALL (
  SELECT salary
  FROM employees sub
  WHERE sub.manager_id = e.id
);
```

73. Get the product with the highest sales for each category.

```
WITH category_sales AS (
  SELECT category_id, product_id, SUM(amount) AS total_sales,
        RANK() OVER (PARTITION BY category_id ORDER BY SUM(amount) DESC) AS sales_rank
  FROM sales
  GROUP BY category_id, product_id
)
SELECT category_id, product_id, total_sales
FROM category_sales
WHERE sales_rank = 1;
```

74. Find customers who haven't ordered in the last 6 months.

```
SELECT customer_id
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id
HAVING MAX(o.order_date) < CURRENT_DATE - INTERVAL '6 months' OR MAX(o.order_date) IS NULL;
```

75. Find the maximum salary gap between any two employees within the same department.

```
SELECT department_id, MAX(salary) - MIN(salary) AS salary_gap
FROM employees
```

```
GROUP BY department_id;
```

71. Write a recursive query to compute the total budget under each manager (including subordinates).

```
WITH RECURSIVE manager_budget AS (
  SELECT id, manager_id, budget
  FROM departments

  UNION ALL

  SELECT d.id, d.manager_id, mb.budget
  FROM departments d
  JOIN manager_budget mb ON d.manager_id = mb.id
)
SELECT manager_id, SUM(budget) AS total_budget
FROM manager_budget
GROUP BY manager_id;
```

72. Write a query to detect gaps in a sequence of invoice numbers.

```
WITH numbered_invoices AS (
  SELECT invoice_number, ROW_NUMBER() OVER
(ORDER BY invoice_number) AS rn
  FROM invoices
)
SELECT invoice_number + 1 AS missing_invoice
FROM numbered_invoices ni
WHERE (invoice_number + 1) <> (
  SELECT invoice_number FROM numbered_invoices
WHERE rn = ni.rn + 1
```

```
);
```

73. Calculate the rank of employees by salary within their department but restart rank numbering every 10 employees.

```
WITH ranked_employees AS (
  SELECT e.*,
         ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn
  FROM employees e
)
SELECT *, ((rn - 1) / 10) + 1 AS rank_group
FROM ranked_employees;
```

74. Find the moving median of daily sales over the last 7 days for each product.

```
WITH daily_sales AS (
  SELECT product_id, sale_date, SUM(amount) AS total_sales
  FROM sales
  GROUP BY product_id, sale_date
)
SELECT product_id, sale_date,
       PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY total_sales)
       OVER (PARTITION BY product_id ORDER BY sale_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS moving_median
FROM daily_sales;
```

75. Find customers who purchased both product A and product B.

```
SELECT customer_id
FROM sales
WHERE product_id IN ('A', 'B')
GROUP BY customer_id
HAVING COUNT(DISTINCT product_id) = 2;
```

76. Write a query to generate a calendar table with all dates for the current year.

```
SELECT generate_series(
  DATE_TRUNC('year', CURRENT_DATE),
  DATE_TRUNC('year', CURRENT_DATE) +
INTERVAL '1 year' - INTERVAL '1 day',
  INTERVAL '1 day'
) AS calendar_date;
```

77. Find employees who have worked in more than 3 different departments.

```
SELECT employee_id
FROM employee_department_history
GROUP BY employee_id
HAVING COUNT(DISTINCT department_id) > 3;
```

78. Calculate the percentage contribution of each product's sales to the total sales per month.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS month, SUM(amount) AS product_sales
  FROM sales
```

```
  GROUP BY product_id, month
),
total_monthly_sales AS (
  SELECT month, SUM(product_sales) AS total_sales
  FROM monthly_sales
  GROUP BY month
)
SELECT ms.product_id, ms.month, ms.product_sales,
       (ms.product_sales * 100.0) / tms.total_sales AS
pct_contribution
FROM monthly_sales ms
JOIN total_monthly_sales tms ON ms.month =
tms.month;
```

79. Write a query to pivot monthly sales data for each product into columns.

```
SELECT product_id,
       SUM(CASE WHEN EXTRACT(MONTH FROM
sale_date) = 1 THEN amount ELSE 0 END) AS Jan,
       SUM(CASE WHEN EXTRACT(MONTH FROM
sale_date) = 2 THEN amount ELSE 0 END) AS Feb,
       -- Repeat for other months
       SUM(CASE WHEN EXTRACT(MONTH FROM
sale_date) = 12 THEN amount ELSE 0 END) AS Dec
FROM sales
GROUP BY product_id;
```

80. Find the 3 most recent orders per customer including order details.

```
SELECT *
```

```
FROM (
  SELECT o.*, ROW_NUMBER() OVER
(PARTITION BY customer_id ORDER BY order_date
DESC) AS rn
  FROM orders o
) sub
WHERE rn <= 3;
```

81. Find employees who have never taken any leave.

```
SELECT e.*
FROM employees e
LEFT JOIN leaves l ON e.id = l.employee_id
WHERE l.leave_id IS NULL;
```

82. List customers who placed orders in January but not in February.

```
WITH jan_orders AS (
  SELECT DISTINCT customer_id
  FROM orders
  WHERE EXTRACT(MONTH FROM order_date) = 1
),
feb_orders AS (
  SELECT DISTINCT customer_id
  FROM orders
  WHERE EXTRACT(MONTH FROM order_date) = 2
)
SELECT customer_id
```

```
FROM jan_orders
WHERE customer_id NOT IN (SELECT customer_id
FROM feb_orders);
```

83. Find products that have seen a price increase in the last 6 months.

```
WITH price_changes AS (
  SELECT product_id, price, effective_date,
        LAG(price) OVER (PARTITION BY product_id
ORDER BY effective_date) AS prev_price
  FROM product_prices
  WHERE effective_date >= CURRENT_DATE -
INTERVAL '6 months'
)
SELECT DISTINCT product_id
FROM price_changes
WHERE price > prev_price;
```

84. Find the department(s) with the second highest average salary.

```
WITH avg_salaries AS (
  SELECT department_id, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY department_id
),
ranked_salaries AS (
  SELECT department_id, avg_salary,
DENSE_RANK() OVER (ORDER BY avg_salary
DESC) AS rnk
  FROM avg_salaries
```

```
)
SELECT department_id, avg_salary
FROM ranked_salaries
WHERE rnk = 2;
```

85. Find employees who joined in the same month and year.

```
SELECT e1.id AS emp1_id, e2.id AS emp2_id,
e1.hire_date
FROM employees e1
JOIN employees e2 ON e1.id <> e2.id
AND EXTRACT(MONTH FROM e1.hire_date) =
EXTRACT(MONTH FROM e2.hire_date)
AND EXTRACT(YEAR FROM e1.hire_date) =
EXTRACT(YEAR FROM e2.hire_date);
```

Advanced-Level SQL Questions

81. Write a recursive query to find all employees and their level of reporting (distance from CEO).

```
WITH RECURSIVE hierarchy AS (
  SELECT id, name, manager_id, 1 AS level
  FROM employees
  WHERE manager_id IS NULL -- CEO level

  UNION ALL

  SELECT e.id, e.name, e.manager_id, h.level + 1
  FROM employees e
  JOIN hierarchy h ON e.manager_id = h.id
)
```

```
SELECT * FROM hierarchy ORDER BY level,
manager_id;
```

82. Find the second highest salary per department without using window functions.

```
SELECT department_id, MAX(salary) AS
second_highest_salary
FROM employees e1
WHERE salary < (
  SELECT MAX(salary)
  FROM employees e2
  WHERE e2.department_id = e1.department_id
)
GROUP BY department_id;
```

83. Calculate the percentage change in sales for each product comparing current month to previous month.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY product_id, month
)
SELECT product_id, month, total_sales,
      (total_sales - LAG(total_sales) OVER
(PARTITION BY product_id ORDER BY month)) *
100.0
      / LAG(total_sales) OVER (PARTITION BY
product_id ORDER BY month) AS pct_change
FROM monthly_sales;
```

84. Write a query to identify duplicate rows (all columns) in a table.

```
SELECT *, COUNT(*) OVER (PARTITION BY col1, col2, col3, ...) AS cnt
FROM table_name
WHERE cnt > 1;
```

(Replace col1, col2, col3, ... with actual column names)

85. Write a query to unpivot quarterly sales data into rows.

```
SELECT product_id, 'Q1' AS quarter, Q1_sales AS sales FROM sales_data
UNION ALL
SELECT product_id, 'Q2', Q2_sales FROM sales_data
UNION ALL
SELECT product_id, 'Q3', Q3_sales FROM sales_data
UNION ALL
SELECT product_id, 'Q4', Q4_sales FROM sales_data;
```

86. Find employees whose salary is above the average salary of their department but below the company-wide average.

```
SELECT *
FROM employees e
WHERE salary > (SELECT AVG(salary) FROM employees WHERE department_id = e.department_id)
```

```
AND salary < (SELECT AVG(salary) FROM
employees);
```

87. Write a query to find customers with the highest purchase amount per year.

```
WITH yearly_sales AS (
  SELECT customer_id, EXTRACT(YEAR FROM
sale_date) AS year, SUM(amount) AS total_amount
  FROM sales
  GROUP BY customer_id, year
),
ranked_sales AS (
  SELECT *, RANK() OVER (PARTITION BY year
ORDER BY total_amount DESC) AS rnk
  FROM yearly_sales
)
SELECT customer_id, year, total_amount
FROM ranked_sales
WHERE rnk = 1;
```

88. Write a query to list all employees who have a salary equal to the average salary of their department.

```
SELECT e.*
FROM employees e
JOIN (
  SELECT department_id, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY department_id
) d ON e.department_id = d.department_id AND
e.salary = d.avg_salary;
```

89. Find the first order date for each customer.

```
SELECT customer_id, MIN(order_date) AS
first_order_date
FROM orders
GROUP BY customer_id;
```

90. Find employees who have been promoted more than twice.

```
SELECT employee_id, COUNT(*) AS
promotion_count
FROM promotions
GROUP BY employee_id
HAVING COUNT(*) > 2;
```

91. Find employees who have not been assigned to any project.

```
SELECT e.*
FROM employees e
LEFT JOIN project_assignments pa ON e.id =
pa.employee_id
WHERE pa.project_id IS NULL;
```

92. Find the total sales per customer including those with zero sales.

```
SELECT c.customer_id, COALESCE(SUM(s.amount),
0) AS total_sales
FROM customers c
LEFT JOIN sales s ON c.customer_id = s.customer_id
```

```
GROUP BY c.customer_id;
```

93. Find the highest salary by department and the employee(s) who earn it.

```
WITH dept_max AS (
  SELECT department_id, MAX(salary) AS max_salary
  FROM employees
  GROUP BY department_id
)
SELECT e.*
FROM employees e
JOIN dept_max d ON e.department_id =
d.department_id AND e.salary = d.max_salary;
```

94. Find customers with no orders in the last year.

```
SELECT customer_id
FROM customers c
LEFT JOIN orders o ON c.customer_id =
o.customer_id AND o.order_date >=
CURRENT_DATE - INTERVAL '1 year'
WHERE o.order_id IS NULL;
```

95. Find employees whose salary is within 10% of the highest salary in their department.

```
WITH dept_max AS (
  SELECT department_id, MAX(salary) AS max_salary
  FROM employees
  GROUP BY department_id
)
SELECT e.*
```

```
FROM employees e
JOIN dept_max d ON e.department_id =
d.department_id
WHERE e.salary >= 0.9 * d.max_salary;
```

96. Find the running total of sales by date.

```
SELECT sale_date, SUM(amount) OVER (ORDER BY
sale_date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS
running_total
FROM sales
ORDER BY sale_date;
```

97. Find employees who earn more than the average salary of the entire company.

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

98. Get the last 3 orders placed by each customer.

```
SELECT *
FROM (
  SELECT o.*, ROW_NUMBER() OVER
(PARTITION BY customer_id ORDER BY order_date
DESC) AS rn
  FROM orders o
) sub
WHERE rn <= 3;
```

99. Find the difference in days between the earliest and latest orders per customer.

```
SELECT customer_id, MAX(order_date) -
MIN(order_date) AS days_between
FROM orders
GROUP BY customer_id;
```

100. Find employees who have worked on all projects.

```
SELECT employee_id
FROM project_assignments
GROUP BY employee_id
HAVING COUNT(DISTINCT project_id) = (SELECT
COUNT(*) FROM projects);
```

101. Find customers who placed orders only in the last 6 months.

```
SELECT customer_id
FROM orders
GROUP BY customer_id
HAVING MIN(order_date) >= CURRENT_DATE -
INTERVAL '6 months';
```

102. Get the total number of orders per day, including days with zero orders.

```
WITH dates AS (
  SELECT generate_series(MIN(order_date),
MAX(order_date), INTERVAL '1 day') AS day
  FROM orders
)
SELECT d.day, COUNT(o.order_id) AS order_count
```

```
FROM dates d
LEFT JOIN orders o ON d.day = o.order_date
GROUP BY d.day
ORDER BY d.day;
```

103. Find the department with the most employees.

```
SELECT department_id, COUNT(*) AS
employee_count
FROM employees
GROUP BY department_id
ORDER BY employee_count DESC
LIMIT 1;
```

104. Write a query to find gaps in employee IDs.

```
WITH numbered AS (
  SELECT id, ROW_NUMBER() OVER (ORDER BY
id) AS rn
  FROM employees
)
SELECT rn + 1 AS missing_id
FROM numbered
WHERE id <> rn;
```

105. Find employees who were hired before their managers.

```
SELECT e.name AS employee, m.name AS manager,
e.hire_date, m.hire_date AS manager_hire_date
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.hire_date < m.hire_date;
```

106. List departments with average salary greater than the overall average.

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

107. Find employees with the highest number of dependents.

```
SELECT employee_id, COUNT(*) AS
dependent_count
FROM dependents
GROUP BY employee_id
ORDER BY dependent_count DESC
LIMIT 1;
```

108. Find customers with the longest gap between two consecutive orders.

```
WITH ordered_orders AS (
  SELECT customer_id, order_date,
        LAG(order_date) OVER (PARTITION BY
customer_id ORDER BY order_date) AS
prev_order_date
  FROM orders
),
gaps AS (
  SELECT customer_id, order_date - prev_order_date
AS gap_days
```

```
  FROM ordered_orders
  WHERE prev_order_date IS NOT NULL
)
SELECT customer_id, MAX(gap_days) AS
longest_gap
FROM gaps
GROUP BY customer_id
ORDER BY longest_gap DESC
LIMIT 1;
```

109. Find customers who ordered all products in a category.

```
SELECT customer_id
FROM sales
WHERE product_id IN (SELECT product_id FROM
products WHERE category_id = 1)
GROUP BY customer_id
HAVING COUNT(DISTINCT product_id) = (SELECT
COUNT(*) FROM products WHERE category_id = 1);
```

110. Get the most recent order date per customer.

```
SELECT customer_id, MAX(order_date) AS
last_order_date
FROM orders
GROUP BY customer_id;
```

111. List all employees and their manager names.

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

112. Find employees with the same salary as their manager.

```
SELECT e.name AS employee, m.name AS manager, e.salary
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.salary = m.salary;
```

113. List products with sales above the average sales amount.

```
WITH avg_sales AS (
  SELECT AVG(amount) AS avg_amount
  FROM sales
)
SELECT product_id, SUM(amount) AS total_sales
FROM sales
GROUP BY product_id
HAVING SUM(amount) > (SELECT avg_amount
FROM avg_sales);
```

114. Get the number of employees hired each year.

```
SELECT EXTRACT(YEAR FROM hire_date) AS hire_year, COUNT(*) AS count
FROM employees
GROUP BY hire_year
ORDER BY hire_year;
```

115. Find the number of employees with the same job title per department.

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

116. Find employees with no manager assigned.

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

117. Calculate average salary by department and job title.

```
SELECT department_id, job_title, AVG(salary) AS
avg_salary
FROM employees
GROUP BY department_id, job_title;
```

118. Find the median salary of employees.

```
SELECT PERCENTILE_CONT(0.5) WITHIN
GROUP (ORDER BY salary) AS median_salary
FROM employees;
```

119. Find employees who have been promoted more than once.

```
SELECT employee_id, COUNT(*) AS
promotion_count
FROM promotions
GROUP BY employee_id
HAVING COUNT(*) > 1;
```

120. Calculate total sales by product category.

```
SELECT p.category_id, SUM(s.amount) AS total_sales
FROM sales s
JOIN products p ON s.product_id = p.product_id
GROUP BY p.category_id;
```

121. Find the top 3 products by sales amount.

```
SELECT product_id, SUM(amount) AS total_sales
FROM sales
GROUP BY product_id
ORDER BY total_sales DESC
LIMIT 3;
```

122. Get employees who joined after their department was created.

```
SELECT e.*
FROM employees e
JOIN departments d ON e.department_id =
d.department_id
WHERE e.hire_date > d.creation_date;
```

123. Find customers with no sales records.

```
SELECT c.*
FROM customers c
LEFT JOIN sales s ON c.customer_id = s.customer_id
WHERE s.sale_id IS NULL;
```

124. Find the second highest salary in the company.

```
SELECT MAX(salary) AS second_highest_salary
FROM employees
```

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

125. Find products with sales only in the current month.

```
SELECT product_id
FROM sales
GROUP BY product_id
HAVING MAX(sale_date) >=
DATE_TRUNC('month', CURRENT_DATE)
AND MIN(sale_date) >= DATE_TRUNC('month',
CURRENT_DATE);
```

126. Find employees with consecutive workdays.

```
WITH attendance AS (
  SELECT employee_id, work_date,
         work_date - INTERVAL '1 day' *
ROW_NUMBER() OVER (PARTITION BY
employee_id ORDER BY work_date) AS grp
  FROM work_log
)
SELECT employee_id, COUNT(*) AS
consecutive_days
FROM attendance
GROUP BY employee_id, grp
HAVING COUNT(*) > 1;
```

127. Find the average number of orders per customer.

```
SELECT AVG(order_count) AS
avg_orders_per_customer
FROM (
```

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

128. Find employees who have worked on more than 5 projects.

```
SELECT employee_id, COUNT(DISTINCT project_id)
AS project_count
FROM project_assignments
GROUP BY employee_id
HAVING COUNT(DISTINCT project_id) > 5;
```

129. Find the total number of products sold each day.

```
SELECT sale_date, SUM(quantity) AS
total_quantity_sold
FROM sales
GROUP BY sale_date
ORDER BY sale_date;
```

130. Find customers with orders totaling more than $10,000.

```
SELECT customer_id, SUM(amount) AS total_amount
FROM sales
GROUP BY customer_id
HAVING SUM(amount) > 10000;
```

♦ Medium to Advanced SQL Questions (131–200)

131. Find employees who have never received a bonus.

```
SELECT e.*
FROM employees e
LEFT JOIN bonuses b ON e.id = b.employee_id
WHERE b.bonus_id IS NULL;
```

132. Find the department with the lowest average salary.

```
SELECT department_id, AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id
ORDER BY avg_salary
LIMIT 1;
```

133. Get cumulative count of orders per customer over time.

```
SELECT customer_id, order_date,
      COUNT(*) OVER (PARTITION BY customer_id
ORDER BY order_date) AS cumulative_orders
FROM orders;
```

134. Find customers who ordered products only from one category.

```
SELECT customer_id
FROM sales s
JOIN products p ON s.product_id = p.product_id
GROUP BY customer_id
HAVING COUNT(DISTINCT p.category_id) = 1;
```

135. Write a query to display employee names alongside their manager names, including those without managers.

```
SELECT e.name AS employee_name, m.name AS manager_name
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.id;
```

136. Find products with sales increasing every month for the last 3 months.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month', sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  WHERE sale_date >= CURRENT_DATE - INTERVAL '3 months'
  GROUP BY product_id, month
)
SELECT product_id
FROM monthly_sales
GROUP BY product_id
HAVING COUNT(*) = 3
   AND MIN(total_sales) < MAX(total_sales)
   AND total_sales[1] < total_sales[2] AND total_sales[2] < total_sales[3]; -- depends on DB support
```

Note: Exact syntax varies per RDBMS; may need window functions.

137. Write a recursive query to get all descendants of a manager.

```
WITH RECURSIVE subordinates AS (
  SELECT id, name, manager_id
  FROM employees
  WHERE id = :manager_id

  UNION ALL

  SELECT e.id, e.name, e.manager_id
  FROM employees e
  JOIN subordinates s ON e.manager_id = s.id
)
SELECT * FROM subordinates WHERE id !=
:manager_id;
```

138. Find the department with the highest variance in salaries.

```
SELECT department_id, VAR_SAMP(salary) AS
salary_variance
FROM employees
GROUP BY department_id
ORDER BY salary_variance DESC
LIMIT 1;
```

139. Calculate the difference between each order amount and the previous order amount per customer.

```
SELECT customer_id, order_date, amount,
```

```
    amount - LAG(amount) OVER (PARTITION BY
customer_id ORDER BY order_date) AS diff
FROM orders;
```

140. Find customers who purchased both Product A and Product B.

```
SELECT customer_id
FROM sales
WHERE product_id IN ('A', 'B')
GROUP BY customer_id
HAVING COUNT(DISTINCT product_id) = 2;
```

141. Find the top N customers by total sales amount.

```
SELECT customer_id, SUM(amount) AS total_sales
FROM sales
GROUP BY customer_id
ORDER BY total_sales DESC
LIMIT N;
```

142. Find the month with the highest sales in the current year.

```
SELECT DATE_TRUNC('month', sale_date) AS
month, SUM(amount) AS total_sales
FROM sales
WHERE EXTRACT(YEAR FROM sale_date) =
EXTRACT(YEAR FROM CURRENT_DATE)
GROUP BY month
ORDER BY total_sales DESC
LIMIT 1;
```

143. Write a query to display all employees who have worked on a project longer than 6 months.

```
SELECT employee_id
FROM project_assignments
WHERE end_date - start_date > INTERVAL '6 months';
```

144. Find the nth highest salary in a company (e.g., 5th highest).

```
SELECT DISTINCT salary
FROM employees
ORDER BY salary DESC
OFFSET n - 1 ROWS FETCH NEXT 1 ROW ONLY;
```

145. Get the average salary of employees hired each year.

```
SELECT EXTRACT(YEAR FROM hire_date) AS year, AVG(salary) AS avg_salary
FROM employees
GROUP BY year
ORDER BY year;
```

146. Find employees whose salaries are between the 25th and 75th percentile.

```
WITH percentiles AS (
  SELECT
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY salary) AS p25,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY salary) AS p75
```

```
  FROM employees
)
SELECT e.*
FROM employees e, percentiles p
WHERE e.salary BETWEEN p.p25 AND p.p75;
```

147. Find employees with salaries higher than their department average.

```
SELECT e.*
FROM employees e
JOIN (
  SELECT department_id, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY department_id
) d ON e.department_id = d.department_id
WHERE e.salary > d.avg_salary;
```

```
--148. Find the difference between each row's value and the previous row’s value in sales.

SELECT sale_date, amount,
       amount - LAG(amount) OVER (ORDER BY sale_date) AS diff
FROM sales;
```

149. List employees who have been in the company for more than 10 years.

```
SELECT *
```

```
FROM employees
WHERE CURRENT_DATE - hire_date > INTERVAL
'10 years';
```

150. Find the department with the most promotions.

```
SELECT e.department_id, COUNT(*) AS
promotion_count
FROM promotions p
JOIN employees e ON p.employee_id = e.id
GROUP BY e.department_id
ORDER BY promotion_count DESC
LIMIT 1;
```

151. Find customers who ordered products from at least 3 different categories.

```
SELECT customer_id
FROM sales s
JOIN products p ON s.product_id = p.product_id
GROUP BY customer_id
HAVING COUNT(DISTINCT p.category_id) >= 3;
```

152. Find the average gap (in days) between orders per customer.

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

```
gaps AS (
  SELECT customer_id, order_date - prev_order_date
AS gap
  FROM ordered_orders
  WHERE prev_order_date IS NOT NULL
)
SELECT customer_id, AVG(gap) AS avg_gap_days
FROM gaps
GROUP BY customer_id;
```

153. List all customers who have never ordered product X.

```
SELECT customer_id
FROM customers
WHERE customer_id NOT IN (
  SELECT DISTINCT customer_id
  FROM sales
  WHERE product_id = 'X'
);
```

154. Calculate total revenue and number of orders per country.

```
SELECT c.country, COUNT(o.order_id) AS
order_count, SUM(o.amount) AS total_revenue
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.country;
```

155. Find the employees who were hired on the same day as their managers.

```
SELECT e.name AS employee, m.name AS manager,
e.hire_date
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.hire_date = m.hire_date;
```

156. Find the top 3 products by quantity sold in each category.

```
SELECT category_id, product_id, total_quantity
FROM (
  SELECT p.category_id, s.product_id,
SUM(s.quantity) AS total_quantity,
        ROW_NUMBER() OVER (PARTITION BY
p.category_id ORDER BY SUM(s.quantity) DESC) AS
rn
  FROM sales s
  JOIN products p ON s.product_id = p.product_id
  GROUP BY p.category_id, s.product_id
) sub
WHERE rn <= 3;
```

157. Find the difference in days between the first and last order for each customer.

```
SELECT customer_id, MAX(order_date) -
MIN(order_date) AS days_between
FROM orders
GROUP BY customer_id;
```

158. Find customers who have increased their order volume every month for the last 3 months.

```
WITH monthly_orders AS (
  SELECT customer_id, DATE_TRUNC('month',
order_date) AS month, COUNT(*) AS orders_count
  FROM orders
  WHERE order_date >= CURRENT_DATE -
INTERVAL '3 months'
  GROUP BY customer_id, month
)
SELECT customer_id
FROM monthly_orders
GROUP BY customer_id
HAVING COUNT(*) = 3
   AND MIN(orders_count) < MAX(orders_count); --
Needs detailed window check for strict increase
```

159. Find employees who have the same salary as the average salary in their job title.

```
SELECT e.*
FROM employees e
JOIN (
  SELECT job_title, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY job_title
) j ON e.job_title = j.job_title
WHERE e.salary = j.avg_salary;
```

160. Write a query to calculate the difference in salary between employees and their managers.

```
SELECT e.name, m.name AS manager_name, e.salary,
m.salary AS manager_salary, e.salary - m.salary AS
salary_diff
FROM employees e
JOIN employees m ON e.manager_id = m.id;
```

161. List the departments with no employees.

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

162. Find the employee with the maximum salary in each department.

```
WITH dept_max AS (
  SELECT department_id, MAX(salary) AS max_salary
  FROM employees
  GROUP BY department_id
)
SELECT e.*
FROM employees e
JOIN dept_max d ON e.department_id =
d.department_id AND e.salary = d.max_salary;
```

163. Find customers with orders on every day in the last week.

```
WITH days AS (
```

```
  SELECT generate_series(CURRENT_DATE -
INTERVAL '6 days', CURRENT_DATE, INTERVAL
'1 day') AS day
)
SELECT customer_id
FROM orders o
JOIN days d ON o.order_date = d.day
GROUP BY customer_id
HAVING COUNT(DISTINCT o.order_date) = 7;
```

164. Find the product that has been sold in the highest quantity in a single order.

```
SELECT product_id, MAX(quantity) AS
max_quantity_in_order
FROM sales
GROUP BY product_id
ORDER BY max_quantity_in_order DESC
LIMIT 1;
```

165. Find employees who joined before their department was created.

```
SELECT e.*
FROM employees e
JOIN departments d ON e.department_id =
d.department_id
WHERE e.hire_date < d.creation_date;
```

166. Find customers with sales in at least 3 different years.

```
SELECT customer_id
FROM sales
GROUP BY customer_id
HAVING COUNT(DISTINCT EXTRACT(YEAR
FROM sale_date)) >= 3;
```

167. Find employees whose salary is above the company's average but below their department's average.

```
WITH company_avg AS (SELECT AVG(salary) AS
avg_salary FROM employees),
    dept_avg AS (
      SELECT department_id, AVG(salary) AS
avg_salary
      FROM employees
      GROUP BY department_id
    )
SELECT e.*
FROM employees e, company_avg ca
JOIN dept_avg da ON e.department_id =
da.department_id
WHERE e.salary > ca.avg_salary AND e.salary <
da.avg_salary;
```

168. Find the average order amount per customer per year.

```
SELECT customer_id, EXTRACT(YEAR FROM
order_date) AS year, AVG(amount) AS
avg_order_amount
FROM orders
```

```
GROUP BY customer_id, year;
```

169. Find employees who have worked on at least one project with a budget over $1,000,000.

```
SELECT DISTINCT pa.employee_id
FROM project_assignments pa
JOIN projects p ON pa.project_id = p.project_id
WHERE p.budget > 1000000;
```

170. Find the most recent promotion date per employee.

```
SELECT employee_id, MAX(promotion_date) AS
last_promotion_date
FROM promotions
GROUP BY employee_id;
```

171. Find customers who made orders totaling more than the average order amount.

```
WITH avg_order AS (
  SELECT AVG(amount) AS avg_amount FROM
orders
)
SELECT customer_id, SUM(amount) AS total_amount
FROM orders
GROUP BY customer_id
HAVING SUM(amount) > (SELECT avg_amount
FROM avg_order);
```

172. Find products never ordered.

```
SELECT product_id
FROM products
```

```
WHERE product_id NOT IN (SELECT DISTINCT
product_id FROM sales);
```

173. Find the month with the lowest sales in the past year.

```
SELECT DATE_TRUNC('month', sale_date) AS
month, SUM(amount) AS total_sales
FROM sales
WHERE sale_date >= CURRENT_DATE -
INTERVAL '1 year'
GROUP BY month
ORDER BY total_sales
LIMIT 1;
```

174. Calculate the number of employees hired each month in the last year.

```
SELECT DATE_TRUNC('month', hire_date) AS
month, COUNT(*) AS hires
FROM employees
WHERE hire_date >= CURRENT_DATE -
INTERVAL '1 year'
GROUP BY month
ORDER BY month;
```

175. Find the department with the highest number of projects.

```
SELECT department_id, COUNT(*) AS project_count
FROM projects
GROUP BY department_id
ORDER BY project_count DESC
```

```
LIMIT 1;
```

176. Find employees who do not have dependents.

```
SELECT e.*
FROM employees e
LEFT JOIN dependents d ON e.id = d.employee_id
WHERE d.dependent_id IS NULL;
```

177. Get the total sales amount for each product category including categories with zero sales.

```
SELECT c.category_id, COALESCE(SUM(s.amount), 0) AS total_sales
FROM categories c
LEFT JOIN products p ON c.category_id = p.category_id
LEFT JOIN sales s ON p.product_id = s.product_id
GROUP BY c.category_id;
```

178. Find employees who have been promoted but their salary didn't increase.

```
SELECT e.id, e.name
FROM employees e
JOIN promotions p ON e.id = p.employee_id
WHERE e.salary <= (SELECT salary_before FROM promotion_history WHERE employee_id = e.id
ORDER BY promotion_date DESC LIMIT 1);
```

179. Find customers with average order amount above $500.

```
SELECT customer_id, AVG(amount) AS
avg_order_amount
FROM orders
GROUP BY customer_id
HAVING AVG(amount) > 500;
```

180. Find orders where the total quantity exceeds 100 units.

```
SELECT order_id, SUM(quantity) AS total_quantity
FROM order_items
GROUP BY order_id
HAVING SUM(quantity) > 100;
```

181. Find products whose sales have doubled compared to the previous month.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY product_id, month
),
sales_comparison AS (
  SELECT product_id, month, total_sales,
        LAG(total_sales) OVER (PARTITION BY
product_id ORDER BY month) AS prev_month_sales
  FROM monthly_sales
)
SELECT product_id, month
FROM sales_comparison
```

```
WHERE prev_month_sales IS NOT NULL AND
total_sales >= 2 * prev_month_sales;
```

182. Write a query to find employees who worked on more than 3 projects in 2023.

```
SELECT employee_id, COUNT(DISTINCT project_id)
AS project_count
FROM project_assignments
WHERE assignment_date BETWEEN '2023-01-01'
AND '2023-12-31'
GROUP BY employee_id
HAVING COUNT(DISTINCT project_id) > 3;
```

183. Find customers whose last order was placed more than 1 year ago.

```
SELECT customer_id, MAX(order_date) AS
last_order_date
FROM orders
GROUP BY customer_id
HAVING MAX(order_date) < CURRENT_DATE -
INTERVAL '1 year';
```

184. Find the average salary increase percentage per department.

```
SELECT e.department_id, AVG((e.salary -
p.old_salary) / p.old_salary * 100) AS avg_increase_pct
FROM employees e
JOIN promotions p ON e.id = p.employee_id
GROUP BY e.department_id;
```

185. Find employees who have never been promoted.

```
SELECT *
FROM employees
WHERE id NOT IN (SELECT DISTINCT
employee_id FROM promotions);
```

186. Find products ordered by all customers.

```
SELECT product_id
FROM sales
GROUP BY product_id
HAVING COUNT(DISTINCT customer_id) =
(SELECT COUNT(*) FROM customers);
```

187. Find customers with orders totaling more than $5000 in the last 6 months.

```
SELECT customer_id, SUM(amount) AS total_amount
FROM orders
WHERE order_date >= CURRENT_DATE -
INTERVAL '6 months'
GROUP BY customer_id
HAVING SUM(amount) > 5000;
```

188. Find the rank of employees based on salary within their department.

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

189. Find customers who purchased a product but never reordered it.

```
WITH order_counts AS (
  SELECT customer_id, product_id, COUNT(*) AS order_count
  FROM sales
  GROUP BY customer_id, product_id
)
SELECT customer_id, product_id
FROM order_counts
WHERE order_count = 1;
```

190. Find the day with the highest number of new hires.

```
SELECT hire_date, COUNT(*) AS hires
FROM employees
GROUP BY hire_date
ORDER BY hires DESC
LIMIT 1;
```

191. Find the number of employees who have worked in more than one department.

```
SELECT employee_id
FROM employee_department_history
GROUP BY employee_id
HAVING COUNT(DISTINCT department_id) > 1;
```

192. Find customers who ordered the most products in 2023.

```
SELECT customer_id, SUM(quantity) AS total_quantity
```

```
FROM sales
WHERE EXTRACT(YEAR FROM sale_date) = 2023
GROUP BY customer_id
ORDER BY total_quantity DESC
LIMIT 1;
```

193. Find the average days taken to ship orders per shipping method.

```
SELECT shipping_method, AVG(shipping_date - order_date) AS avg_shipping_days
FROM orders
GROUP BY shipping_method;
```

194. Find employees with overlapping project assignments.

```
SELECT pa1.employee_id, pa1.project_id, pa2.project_id AS overlapping_project
FROM project_assignments pa1
JOIN project_assignments pa2 ON pa1.employee_id = pa2.employee_id AND pa1.project_id <> pa2.project_id
WHERE pa1.start_date <= pa2.end_date AND pa1.end_date >= pa2.start_date;
```

195. Find the total number of unique customers per product category.

```
SELECT p.category_id, COUNT(DISTINCT s.customer_id) AS unique_customers
FROM sales s
JOIN products p ON s.product_id = p.product_id
```

```
GROUP BY p.category_id;
```

196. Find customers whose orders increased by at least 20% compared to the previous month.

```
WITH monthly_orders AS (
  SELECT customer_id, DATE_TRUNC('month', order_date) AS month, COUNT(*) AS order_count
  FROM orders
  GROUP BY customer_id, month
),
orders_comparison AS (
  SELECT customer_id, month, order_count,
         LAG(order_count) OVER (PARTITION BY customer_id ORDER BY month) AS prev_order_count
  FROM monthly_orders
)
SELECT customer_id, month
FROM orders_comparison
WHERE prev_order_count IS NOT NULL AND order_count >= 1.2 * prev_order_count;
```

197. Find employees with no projects assigned in the last 6 months.

```
SELECT e.*
FROM employees e
LEFT JOIN project_assignments pa ON e.id = pa.employee_id AND pa.start_date >= CURRENT_DATE - INTERVAL '6 months'
WHERE pa.project_id IS NULL;
```

198. Find the number of employees who have changed departments more than twice.

```
SELECT employee_id
FROM employee_department_history
GROUP BY employee_id
HAVING COUNT(DISTINCT department_id) > 2;
```

199. Find the product with the highest average rating.

```
SELECT product_id, AVG(rating) AS avg_rating
FROM product_reviews
GROUP BY product_id
ORDER BY avg_rating DESC
LIMIT 1;
```

200. Find customers who have placed orders but never used a discount.

```
SELECT DISTINCT customer_id
FROM orders
WHERE discount_used = FALSE;
```

🔺 Medium to Advanced SQL Questions (201–300)

201. Find employees who have worked on every project in their department.

```
SELECT e.id, e.name
FROM employees e
JOIN projects p ON e.department_id = p.department_id
LEFT JOIN project_assignments pa ON e.id =
pa.employee_id AND p.project_id = pa.project_id
GROUP BY e.id, e.name, p.department_id
```

```
HAVING COUNT(p.project_id) =
COUNT(pa.project_id);
```

202. Find the average order amount excluding the top 5% largest orders.

```
WITH ordered_orders AS (
  SELECT amount,
        NTILE(100) OVER (ORDER BY amount DESC)
AS percentile
  FROM orders
)
SELECT AVG(amount)
FROM ordered_orders
WHERE percentile > 5;
```

203. Find the top 3 employees with the highest salary increase over last year.

```
WITH salary_last_year AS (
  SELECT employee_id, salary AS last_year_salary
  FROM salaries
  WHERE year = EXTRACT(YEAR FROM
CURRENT_DATE) - 1
),
salary_this_year AS (
  SELECT employee_id, salary AS this_year_salary
  FROM salaries
  WHERE year = EXTRACT(YEAR FROM
CURRENT_DATE)
)
```

```
SELECT t.employee_id, t.this_year_salary -
l.last_year_salary AS salary_increase
FROM salary_this_year t
JOIN salary_last_year l ON t.employee_id =
l.employee_id
ORDER BY salary_increase DESC
LIMIT 3;
```

204. Find employees with the longest consecutive workdays.

```
WITH workdays AS (
  SELECT employee_id, work_date,
        ROW_NUMBER() OVER (PARTITION BY
employee_id ORDER BY work_date) -
        ROW_NUMBER() OVER (PARTITION BY
employee_id ORDER BY work_date) AS grp
  FROM attendance
),
consecutive_days AS (
  SELECT employee_id, COUNT(*) AS
consecutive_days
  FROM workdays
  GROUP BY employee_id, grp
)
SELECT employee_id, MAX(consecutive_days) AS
max_consecutive_days
FROM consecutive_days
GROUP BY employee_id;
```

205. Find all managers who do not manage any employee.

```
SELECT DISTINCT manager_id
FROM employees
WHERE manager_id NOT IN (SELECT DISTINCT id
FROM employees);
```

206. Find the average salary of employees hired each month.

```
SELECT EXTRACT(YEAR FROM hire_date) AS
year, EXTRACT(MONTH FROM hire_date) AS
month, AVG(salary) AS avg_salary
FROM employees
GROUP BY year, month
ORDER BY year, month;
```

--207. Find the first 5 orders after a customer's registration date.

```
SELECT order_id, customer_id, order_date
FROM (
  SELECT order_id, customer_id, order_date,
        ROW_NUMBER() OVER (PARTITION BY
customer_id ORDER BY order_date) AS rn
  FROM orders
  JOIN customers c ON orders.customer_id =
c.customer_id
  WHERE order_date >= c.registration_date
```

```
) sub
WHERE rn <= 5;
```

208. Find customers who placed orders every month for the last 6 months.

```
WITH months AS (
  SELECT generate_series(
    DATE_TRUNC('month', CURRENT_DATE) - INTERVAL '5 months',
    DATE_TRUNC('month', CURRENT_DATE),
    INTERVAL '1 month'
  ) AS month
),
customer_months AS (
  SELECT customer_id, DATE_TRUNC('month', order_date) AS month
  FROM orders
  WHERE order_date >= CURRENT_DATE - INTERVAL '6 months'
  GROUP BY customer_id, month
)
SELECT customer_id
FROM customer_months cm
JOIN months m ON cm.month = m.month
GROUP BY customer_id
HAVING COUNT(DISTINCT cm.month) = 6;
```

209. Calculate the moving average of sales over the last 3 days.

```
SELECT sale_date, product_id, amount,
       AVG(amount) OVER (PARTITION BY
product_id ORDER BY sale_date ROWS BETWEEN 2
PRECEDING AND CURRENT ROW) AS
moving_avg_3_days
FROM sales;
```

210. Find the number of employees who share the same birthday.

```
SELECT birth_date, COUNT(*) AS count_employees
FROM employees
GROUP BY birth_date
HAVING COUNT(*) > 1;
```

211. Find customers who ordered the same product multiple times in one day.

```
SELECT customer_id, product_id, order_date,
COUNT(*) AS order_count
FROM sales
GROUP BY customer_id, product_id, order_date
HAVING COUNT(*) > 1;
```

212. Find the total sales for each product including products with zero sales.

```
SELECT p.product_id, COALESCE(SUM(s.amount),
0) AS total_sales
FROM products p
LEFT JOIN sales s ON p.product_id = s.product_id
```

```
GROUP BY p.product_id;
```

213. List the top 5 employees by number of projects in each department.

```
SELECT department_id, employee_id, project_count
FROM (
  SELECT e.department_id, pa.employee_id,
COUNT(DISTINCT pa.project_id) AS project_count,
         ROW_NUMBER() OVER (PARTITION BY
e.department_id ORDER BY COUNT(DISTINCT
pa.project_id) DESC) AS rn
  FROM project_assignments pa
  JOIN employees e ON pa.employee_id = e.id
  GROUP BY e.department_id, pa.employee_id
) sub
WHERE rn <= 5;
```

214. Find the day with the largest difference between maximum and minimum temperature.

```
SELECT weather_date, MAX(temperature) -
MIN(temperature) AS temp_diff
FROM weather_data
GROUP BY weather_date
ORDER BY temp_diff DESC
LIMIT 1;
```

215. Find the 3 most recent orders per customer.

```
SELECT order_id, customer_id, order_date
FROM (
  SELECT order_id, customer_id, order_date,
```

```
        ROW_NUMBER() OVER (PARTITION BY
customer_id ORDER BY order_date DESC) AS rn
  FROM orders
) sub
WHERE rn <= 3;
```

216. Find products with sales only in a specific country.

```
SELECT product_id
FROM sales s
JOIN customers c ON s.customer_id = c.customer_id
GROUP BY product_id
HAVING COUNT(DISTINCT c.country) = 1;
```

217. Find employees with a salary greater than all employees in department 10.

```
SELECT *
FROM employees
WHERE salary > ALL (
  SELECT salary FROM employees WHERE
department_id = 10
);
```

218. Find the percentage of employees in each department.

```
WITH total_employees AS (SELECT COUNT(*) AS
total FROM employees)
SELECT department_id, COUNT(*) * 100.0 /
(SELECT total FROM total_employees) AS percentage
```

```
FROM employees
GROUP BY department_id;
```

219. Find the median salary per department.

```
SELECT department_id,
      PERCENTILE_CONT(0.5) WITHIN GROUP
(ORDER BY salary) AS median_salary
FROM employees
GROUP BY department_id;
```

220. Find the employee who worked the most hours in a project.

```
SELECT employee_id, project_id,
MAX(hours_worked) AS max_hours
FROM project_assignments
GROUP BY employee_id, project_id
ORDER BY max_hours DESC
LIMIT 1;
```

221. Find the first order date for each customer.

```
SELECT customer_id, MIN(order_date) AS
first_order_date
FROM orders
GROUP BY customer_id;
```

222. Find the second most expensive product per category.

```
SELECT category_id, product_id, price
FROM (
  SELECT category_id, product_id, price,
```

```
        ROW_NUMBER() OVER (PARTITION BY
category_id ORDER BY price DESC) AS rn
  FROM products
) sub
WHERE rn = 2;
```

223. Find employees with the highest salary in each job title.

```
WITH max_salary_per_job AS (
  SELECT job_title, MAX(salary) AS max_salary
  FROM employees
  GROUP BY job_title
)
SELECT e.*
FROM employees e
JOIN max_salary_per_job m ON e.job_title =
m.job_title AND e.salary = m.max_salary;
```

224. Calculate the ratio of males to females in each department.

```
SELECT department_id,
      SUM(CASE WHEN gender = 'M' THEN 1 ELSE 0
END) * 1.0 / NULLIF(SUM(CASE WHEN gender = 'F'
THEN 1 ELSE 0 END), 0) AS male_to_female_ratio
FROM employees
GROUP BY department_id;
```

225. Find customers who spent more than average in their country.

```
WITH avg_spent_per_country AS (
```

```
  SELECT c.country, AVG(o.amount) AS avg_amount
  FROM customers c
  JOIN orders o ON c.customer_id = o.customer_id
  GROUP BY c.country
)
SELECT c.customer_id, SUM(o.amount) AS
total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN avg_spent_per_country a ON c.country =
a.country
GROUP BY c.customer_id, c.country, a.avg_amount
HAVING SUM(o.amount) > a.avg_amount;
```

226. Find employees who have not been assigned to any project in the last year.

```
SELECT e.*
FROM employees e
LEFT JOIN project_assignments pa ON e.id =
pa.employee_id AND pa.assignment_date >=
CURRENT_DATE - INTERVAL '1 year'
WHERE pa.project_id IS NULL;
```

227. Find the top 3 customers by total order amount in each region.

```
SELECT region, customer_id, total_amount
FROM (
  SELECT c.region, o.customer_id, SUM(o.amount) AS
total_amount,
```

```
        ROW_NUMBER() OVER (PARTITION BY
c.region ORDER BY SUM(o.amount) DESC) AS rn
  FROM customers c
  JOIN orders o ON c.customer_id = o.customer_id
  GROUP BY c.region, o.customer_id
) sub
WHERE rn <= 3;
```

228. Find employees hired after their managers.

```
SELECT e.name AS employee_name, m.name AS
manager_name, e.hire_date, m.hire_date AS
manager_hire_date
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.hire_date > m.hire_date;
```

229. Find customers who ordered all products from a specific category.

```
WITH category_products AS (
  SELECT product_id
  FROM products
  WHERE category_id = :category_id
),
customer_products AS (
  SELECT customer_id, product_id
  FROM sales
  WHERE product_id IN (SELECT product_id FROM
category_products)
  GROUP BY customer_id, product_id
)
```

```
SELECT customer_id
FROM customer_products
GROUP BY customer_id
HAVING COUNT(DISTINCT product_id) = (SELECT
COUNT(*) FROM category_products);
```

230. Find employees with the highest number of direct reports.

```
SELECT manager_id, COUNT(*) AS report_count
FROM employees
WHERE manager_id IS NOT NULL
GROUP BY manager_id
ORDER BY report_count DESC
LIMIT 1;
```

231. Calculate the retention rate of customers month-over-month.

```
WITH monthly_customers AS (
  SELECT customer_id, DATE_TRUNC('month',
order_date) AS month
  FROM orders
  GROUP BY customer_id, month
),
retention AS (
  SELECT current.month AS current_month,
current.customer_id
  FROM monthly_customers current
  JOIN monthly_customers previous ON
current.customer_id = previous.customer_id
```

```
    AND current.month = previous.month + INTERVAL
'1 month'
)
SELECT current_month, COUNT(DISTINCT
customer_id) * 100.0 /
      (SELECT COUNT(DISTINCT customer_id)
FROM monthly_customers WHERE month =
current_month - INTERVAL '1 month') AS
retention_rate
FROM retention
GROUP BY current_month
ORDER BY current_month;
```

232. Find the average time difference between order and delivery.

```
SELECT AVG(delivery_date - order_date) AS
avg_delivery_time
FROM orders
WHERE delivery_date IS NOT NULL;
```

233. Find the department with the youngest average employee age.

```
SELECT department_id, AVG(EXTRACT(YEAR
FROM AGE(CURRENT_DATE, birth_date))) AS
avg_age
FROM employees
GROUP BY department_id
ORDER BY avg_age
LIMIT 1;
```

234. Find products that were sold in every quarter of the current year.

```
WITH quarterly_sales AS (
  SELECT product_id, DATE_TRUNC('quarter', sale_date) AS quarter
  FROM sales
  WHERE EXTRACT(YEAR FROM sale_date) = EXTRACT(YEAR FROM CURRENT_DATE)
  GROUP BY product_id, quarter
)
SELECT product_id
FROM quarterly_sales
GROUP BY product_id
HAVING COUNT(DISTINCT quarter) = 4;
```

235. Find customers whose orders decreased consecutively for 3 months.

```
WITH monthly_orders AS (
  SELECT customer_id, DATE_TRUNC('month', order_date) AS month, COUNT(*) AS order_count
  FROM orders
  GROUP BY customer_id, month
),
orders_with_lag AS (
  SELECT customer_id, month, order_count,
         LAG(order_count, 1) OVER (PARTITION BY customer_id ORDER BY month) AS prev_1,
         LAG(order_count, 2) OVER (PARTITION BY customer_id ORDER BY month) AS prev_2
```

```
  FROM monthly_orders
)
SELECT DISTINCT customer_id
FROM orders_with_lag
WHERE order_count < prev_1 AND prev_1 < prev_2;
```

236. Find the employee(s) with the highest number of late arrivals.

```
SELECT employee_id, COUNT(*) AS late_count
FROM attendance
WHERE arrival_time > scheduled_start_time
GROUP BY employee_id
ORDER BY late_count DESC
LIMIT 1;
```

237. Find the most common product combinations in orders (pairs).

```
WITH order_pairs AS (
  SELECT o1.order_id, o1.product_id AS product1,
o2.product_id AS product2
  FROM order_items o1
  JOIN order_items o2 ON o1.order_id = o2.order_id
AND o1.product_id < o2.product_id
)
SELECT product1, product2, COUNT(*) AS
pair_count
FROM order_pairs
GROUP BY product1, product2
ORDER BY pair_count DESC
LIMIT 10;
```

238. Find employees who have worked more than 40 hours in a week.

```
WITH weekly_hours AS (
  SELECT employee_id, DATE_TRUNC('week', work_date) AS week_start, SUM(hours_worked) AS total_hours
  FROM work_logs
  GROUP BY employee_id, week_start
)
SELECT employee_id, week_start, total_hours
FROM weekly_hours
WHERE total_hours > 40;
```

239. Find the total revenue generated per sales representative.

```
SELECT sales_rep_id, SUM(amount) AS total_revenue
FROM sales
GROUP BY sales_rep_id;
```

240. Find customers with no orders in the last year.

```
SELECT customer_id
FROM customers
WHERE customer_id NOT IN (
  SELECT DISTINCT customer_id FROM orders
WHERE order_date >= CURRENT_DATE - INTERVAL '1 year'
);
```

241. Find products with an increasing sales trend over the last 3 months.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month', sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  WHERE sale_date >= CURRENT_DATE - INTERVAL '3 months'
  GROUP BY product_id, month
),
sales_ranked AS (
  SELECT product_id, month, total_sales,
         LAG(total_sales) OVER (PARTITION BY product_id ORDER BY month) AS prev_month_sales,
         LAG(total_sales, 2) OVER (PARTITION BY product_id ORDER BY month) AS prev_2_month_sales
  FROM monthly_sales
)
SELECT DISTINCT product_id
FROM sales_ranked
WHERE total_sales > prev_month_sales AND prev_month_sales > prev_2_month_sales;
```

242. Find departments where average salary is higher than the company average.

```
WITH company_avg AS (
  SELECT AVG(salary) AS avg_salary FROM employees
)
```

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

243. Find customers with orders where no product quantity is less than 5.

```
SELECT DISTINCT order_id
FROM order_items
GROUP BY order_id
HAVING MIN(quantity) >= 5;
```

244. Find products ordered only by customers from one country.

```
SELECT product_id
FROM sales s
JOIN customers c ON s.customer_id = c.customer_id
GROUP BY product_id
HAVING COUNT(DISTINCT c.country) = 1;
```

245. Find employees who have not submitted their timesheets in the last month.

```
SELECT e.id, e.name
FROM employees e
LEFT JOIN timesheets t ON e.id = t.employee_id AND
t.timesheet_date >= CURRENT_DATE - INTERVAL
'1 month'
WHERE t.timesheet_id IS NULL;
```

246. Find the total discount given in each month.

```
SELECT DATE_TRUNC('month', order_date) AS
month, SUM(discount_amount) AS total_discount
FROM orders
GROUP BY month
ORDER BY month;
```

247. Find customers who have placed orders but never paid by credit card.

```
SELECT DISTINCT customer_id
FROM orders
WHERE payment_method != 'Credit Card';
```

248. Find employees whose salaries are within 10% of their department's average salary.

```
WITH dept_avg AS (
  SELECT department_id, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY department_id
)
SELECT e.*
FROM employees e
JOIN dept_avg d ON e.department_id =
d.department_id
WHERE e.salary BETWEEN d.avg_salary * 0.9 AND
d.avg_salary * 1.1;
```

249. Find customers who ordered the most products in each category.

```
WITH product_totals AS (
```

```
  SELECT c.customer_id, p.category_id,
SUM(s.quantity) AS total_quantity,
        RANK() OVER (PARTITION BY p.category_id
ORDER BY SUM(s.quantity) DESC) AS rank
  FROM sales s
  JOIN products p ON s.product_id = p.product_id
  JOIN customers c ON s.customer_id = c.customer_id
  GROUP BY c.customer_id, p.category_id
)
SELECT customer_id, category_id, total_quantity
FROM product_totals
WHERE rank = 1;
```

250. Find the top 5 longest projects.

```
SELECT project_id, start_date, end_date,
       end_date - start_date AS duration
FROM projects
ORDER BY duration DESC
LIMIT 5;
```

251. Find employees who have not taken any leave in the last 6 months.

```
SELECT e.id, e.name
FROM employees e
LEFT JOIN leaves l ON e.id = l.employee_id AND
l.leave_date >= CURRENT_DATE - INTERVAL '6
months'
WHERE l.leave_id IS NULL;
```

252. Find the department with the most projects completed last year.

```
SELECT department_id, COUNT(*) AS
completed_projects
FROM projects
WHERE status = 'Completed' AND completion_date
BETWEEN DATE_TRUNC('year',
CURRENT_DATE) - INTERVAL '1 year' AND
DATE_TRUNC('year', CURRENT_DATE) -
INTERVAL '1 day'
GROUP BY department_id
ORDER BY completed_projects DESC
LIMIT 1;
```

253. Find customers who have increased their order frequency month-over-month for 3 consecutive months.

```
WITH monthly_orders AS (
  SELECT customer_id, DATE_TRUNC('month',
order_date) AS month, COUNT(*) AS order_count
  FROM orders
  GROUP BY customer_id, month
),
orders_with_lag AS (
  SELECT customer_id, month, order_count,
        LAG(order_count, 1) OVER (PARTITION BY
customer_id ORDER BY month) AS prev_1,
        LAG(order_count, 2) OVER (PARTITION BY
customer_id ORDER BY month) AS prev_2
  FROM monthly_orders
)
```

```
SELECT DISTINCT customer_id
FROM orders_with_lag
WHERE order_count > prev_1 AND prev_1 > prev_2;
```

254. Find employees who have been assigned projects outside their department.

```
SELECT DISTINCT e.id, e.name
FROM employees e
JOIN project_assignments pa ON e.id =
pa.employee_id
JOIN projects p ON pa.project_id = p.project_id
WHERE e.department_id != p.department_id;
```

255. Calculate the average time to close tickets per support agent.

```
SELECT support_agent_id, AVG(closed_date -
opened_date) AS avg_close_time
FROM support_tickets
WHERE closed_date IS NOT NULL
GROUP BY support_agent_id;
```

256. Find the first and last login date for each user.

```
SELECT user_id, MIN(login_date) AS first_login,
MAX(login_date) AS last_login
FROM user_logins
GROUP BY user_id;
```

257. Find customers who made purchases only in one month of the year.

```
WITH customer_months AS (
```

```
  SELECT customer_id, DATE_TRUNC('month',
order_date) AS month
  FROM orders
  GROUP BY customer_id, month
)
SELECT customer_id
FROM customer_months
GROUP BY customer_id
HAVING COUNT(*) = 1;
```

258. Find products with sales revenue above the average revenue per product.

```
WITH avg_revenue AS (
  SELECT AVG(total_revenue) AS avg_rev
  FROM (
    SELECT product_id, SUM(amount) AS
total_revenue
    FROM sales
    GROUP BY product_id
  ) sub
)
SELECT product_id, SUM(amount) AS total_revenue
FROM sales
GROUP BY product_id
HAVING SUM(amount) > (SELECT avg_rev FROM
avg_revenue);
```

259. Find departments where more than 50% of employees have a salary above $60,000.

```
SELECT department_id
```

```
FROM employees
GROUP BY department_id
HAVING AVG(CASE WHEN salary > 60000 THEN 1
ELSE 0 END) > 0.5;
```

260. Find employees who worked on all projects in the company.

```
WITH total_projects AS (
  SELECT COUNT(DISTINCT project_id) AS
project_count FROM projects
),
employee_projects AS (
  SELECT employee_id, COUNT(DISTINCT
project_id) AS projects_worked
  FROM project_assignments
  GROUP BY employee_id
)
SELECT ep.employee_id
FROM employee_projects ep
JOIN total_projects tp ON 1=1
WHERE ep.projects_worked = tp.project_count;
```

261. Find customers who ordered products from all categories.

```
WITH category_count AS (
  SELECT COUNT(DISTINCT category_id) AS
total_categories FROM products
),
customer_categories AS (
```

```
  SELECT customer_id, COUNT(DISTINCT
p.category_id) AS categories_ordered
  FROM sales s
  JOIN products p ON s.product_id = p.product_id
  GROUP BY customer_id
)
SELECT customer_id
FROM customer_categories
JOIN category_count ON 1=1
WHERE categories_ordered = total_categories;
```

262. Find the average tenure of employees by department.

```
SELECT department_id, AVG(DATE_PART('year',
CURRENT_DATE - hire_date)) AS avg_tenure_years
FROM employees
GROUP BY department_id;
```

263. Find the number of orders placed on weekends vs weekdays.

```
SELECT CASE
        WHEN EXTRACT(DOW FROM order_date) IN
(0,6) THEN 'Weekend'
        ELSE 'Weekday'
      END AS day_type,
      COUNT(*) AS order_count
FROM orders
GROUP BY day_type;
```

264. Find the percentage of orders with discounts per month.

```
SELECT DATE_TRUNC('month', order_date) AS month,
       100.0 * SUM(CASE WHEN discount > 0 THEN 1 ELSE 0 END) / COUNT(*) AS discount_percentage
FROM orders
GROUP BY month
ORDER BY month;
```

265. Find the employees who have never been late to work.

```
SELECT e.id, e.name
FROM employees e
LEFT JOIN attendance a ON e.id = a.employee_id
AND a.arrival_time > a.scheduled_start_time
WHERE a.employee_id IS NULL;
```

266. Find products with sales only during holiday seasons.

```
SELECT product_id
FROM sales s
JOIN holidays h ON s.sale_date = h.holiday_date
GROUP BY product_id
HAVING COUNT(*) = (SELECT COUNT(*) FROM sales WHERE product_id = s.product_id);
```

267. Find the department with the largest increase in employee count compared to last year.

```
WITH current_year AS (
```

```
  SELECT department_id, COUNT(*) AS emp_count
  FROM employees
  WHERE hire_date <= CURRENT_DATE AND
(termination_date IS NULL OR termination_date >=
CURRENT_DATE)
  GROUP BY department_id
),
last_year AS (
  SELECT department_id, COUNT(*) AS emp_count
  FROM employees
  WHERE hire_date <= CURRENT_DATE -
INTERVAL '1 year' AND (termination_date IS NULL
OR termination_date >= CURRENT_DATE -
INTERVAL '1 year')
  GROUP BY department_id
)
SELECT c.department_id, c.emp_count -
COALESCE(l.emp_count,0) AS increase
FROM current_year c
LEFT JOIN last_year l ON c.department_id =
l.department_id
ORDER BY increase DESC
LIMIT 1;
```

268. Find the average order value per customer segment.

```
SELECT segment, AVG(o.amount) AS
avg_order_value
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
```

```
GROUP BY segment;
```

269. Find employees who manage more than 3 projects.

```
SELECT manager_id, COUNT(DISTINCT project_id)
AS project_count
FROM projects
GROUP BY manager_id
HAVING COUNT(DISTINCT project_id) > 3;
```

270. Find products that have never been returned.

```
SELECT p.product_id
FROM products p
LEFT JOIN returns r ON p.product_id = r.product_id
WHERE r.return_id IS NULL;
```

271. Find customers with orders but no shipments.

```
SELECT DISTINCT o.customer_id
FROM orders o
LEFT JOIN shipments s ON o.order_id = s.order_id
WHERE s.shipment_id IS NULL;
```

272. Find employees whose salaries increased every year.

```
WITH salary_diff AS (
  SELECT employee_id, year, salary,
        LAG(salary) OVER (PARTITION BY
employee_id ORDER BY year) AS prev_salary
  FROM salaries
)
```

```
SELECT DISTINCT employee_id
FROM salary_diff
WHERE salary > prev_salary OR prev_salary IS NULL
GROUP BY employee_id
HAVING COUNT(*) = (SELECT COUNT(*) FROM
salaries s2 WHERE s2.employee_id =
salary_diff.employee_id);
```

273. Find the total number of unique products sold in the last quarter.

```
SELECT COUNT(DISTINCT product_id) AS
unique_products_sold
FROM sales
WHERE sale_date >= DATE_TRUNC('quarter',
CURRENT_DATE) - INTERVAL '3 months'
  AND sale_date < DATE_TRUNC('quarter',
CURRENT_DATE);
```

274. Find the day with the highest sales in each month.

```
WITH daily_sales AS (
  SELECT DATE(order_date) AS day, SUM(amount)
AS total_sales
  FROM orders
  GROUP BY day
),
ranked_sales AS (
  SELECT day, total_sales,
        RANK() OVER (PARTITION BY
DATE_TRUNC('month', day) ORDER BY total_sales
DESC) AS sales_rank
```

```
  FROM daily_sales
)
SELECT day, total_sales
FROM ranked_sales
WHERE sales_rank = 1;
```

275. Find the products with the highest sales increase compared to the previous month.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY product_id, month
),
sales_diff AS (
  SELECT product_id, month, total_sales,
        LAG(total_sales) OVER (PARTITION BY
product_id ORDER BY month) AS prev_month_sales
  FROM monthly_sales
)
SELECT product_id, month, total_sales -
prev_month_sales AS increase
FROM sales_diff
WHERE prev_month_sales IS NOT NULL
ORDER BY increase DESC
LIMIT 10;
```

276. Find the top 5 customers by total order value in the last year.

```
SELECT customer_id, SUM(amount) AS
total_order_value
FROM orders
WHERE order_date >= CURRENT_DATE -
INTERVAL '1 year'
GROUP BY customer_id
ORDER BY total_order_value DESC
LIMIT 5;
```

277. Find the number of employees who changed departments in the last year.

```
SELECT COUNT(DISTINCT employee_id)
FROM employee_department_history
WHERE change_date >= CURRENT_DATE -
INTERVAL '1 year';
```

278. Find the average salary for each job title within each department.

```
SELECT department_id, job_title, AVG(salary) AS
avg_salary
FROM employees
GROUP BY department_id, job_title;
```

279. Find customers who placed orders with multiple payment methods.

```
SELECT customer_id
FROM orders
GROUP BY customer_id
HAVING COUNT(DISTINCT payment_method) > 1;
```

280. Find products with the lowest average rating per category.

```
SELECT category_id, product_id, AVG(rating) AS avg_rating
FROM product_reviews
GROUP BY category_id, product_id
QUALIFY ROW_NUMBER() OVER (PARTITION BY category_id ORDER BY AVG(rating)) = 1;
```

281. Find employees who have never received a promotion.

```
SELECT e.id, e.name
FROM employees e
LEFT JOIN promotions p ON e.id = p.employee_id
WHERE p.promotion_id IS NULL;
```

282. Find the total number of orders placed each day in the last week.

```
SELECT DATE(order_date) AS order_day, COUNT(*) AS orders_count
FROM orders
WHERE order_date >= CURRENT_DATE - INTERVAL '7 days'
GROUP BY order_day
ORDER BY order_day;
```

283. Find customers with orders in both online and in-store channels.

```
SELECT customer_id
FROM orders
```

```
GROUP BY customer_id
HAVING COUNT(DISTINCT order_channel) > 1;
```

284. Find the top 3 sales reps by number of deals closed this quarter.

```
SELECT sales_rep_id, COUNT(*) AS deals_closed
FROM sales_deals
WHERE deal_close_date >= DATE_TRUNC('quarter',
CURRENT_DATE)
GROUP BY sales_rep_id
ORDER BY deals_closed DESC
LIMIT 3;
```

285. Find products that have been discontinued but still have sales.

```
SELECT p.product_id
FROM products p
JOIN sales s ON p.product_id = s.product_id
WHERE p.discontinued = TRUE;
```

286. Find employees who report to a manager hired after them.

```
SELECT e.id, e.name, m.id AS manager_id, m.name
AS manager_name
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE e.hire_date < m.hire_date;
```

287. Find the average delivery time by shipping method.

```
SELECT shipping_method, AVG(delivery_date - order_date) AS avg_delivery_time
FROM shipments
GROUP BY shipping_method;
```

288. Find orders where the total quantity exceeds 100.

```
SELECT order_id, SUM(quantity) AS total_quantity
FROM order_items
GROUP BY order_id
HAVING SUM(quantity) > 100;
```

289. Find customers who made orders but never returned a product.

```
SELECT DISTINCT o.customer_id
FROM orders o
LEFT JOIN returns r ON o.order_id = r.order_id
WHERE r.return_id IS NULL;
```

290. Find products that have been ordered but never reviewed.

```
SELECT p.product_id
FROM products p
LEFT JOIN product_reviews pr ON p.product_id = pr.product_id
JOIN sales s ON p.product_id = s.product_id
WHERE pr.review_id IS NULL
GROUP BY p.product_id;
```

291. Find employees who have worked on projects for more than 2 years.

```
SELECT employee_id
FROM project_assignments
GROUP BY employee_id
HAVING MAX(assignment_end_date) -
MIN(assignment_start_date) > INTERVAL '2 years';
```

292. Find the product with the highest sales for each month.

```
WITH monthly_sales AS (
  SELECT product_id, DATE_TRUNC('month',
sale_date) AS month, SUM(amount) AS total_sales
  FROM sales
  GROUP BY product_id, month
),
ranked_sales AS (
  SELECT product_id, month, total_sales,
        ROW_NUMBER() OVER (PARTITION BY
month ORDER BY total_sales DESC) AS rn
  FROM monthly_sales
)
SELECT product_id, month, total_sales
FROM ranked_sales
WHERE rn = 1;
```

293. Find customers with the highest order count in each region.

```
WITH customer_order_counts AS (
  SELECT c.region, o.customer_id, COUNT(*) AS
order_count,
```

```
        ROW_NUMBER() OVER (PARTITION BY
c.region ORDER BY COUNT(*) DESC) AS rn
  FROM customers c
  JOIN orders o
```

294. Flag customers with an increase in orders every month this year.

```
WITH monthly_counts AS (
  SELECT customer_id, DATE_TRUNC('month',
order_date) AS month, COUNT(*) AS cnt
  FROM orders
  WHERE EXTRACT(YEAR FROM order_date) =
EXTRACT(YEAR FROM CURRENT_DATE)
  GROUP BY customer_id, month
),
increase_check AS (
  SELECT customer_id,
        LAG(cnt) OVER (PARTITION BY customer_id
ORDER BY month) AS prev_cnt,
        cnt
  FROM monthly_counts
)
SELECT DISTINCT customer_id
FROM increase_check
WHERE cnt > prev_cnt AND prev_cnt IS NOT NULL;
```

295. Find employees whose hire date is the same weekday as their manager's.

```
SELECT e.id, e.name
```

```
FROM employees e
JOIN employees m ON e.manager_id = m.id
WHERE EXTRACT(DOW FROM e.hire_date) =
EXTRACT(DOW FROM m.hire_date);
```

296. Get total working hours per employee per week.

```
SELECT employee_id, DATE_TRUNC('week',
work_date) AS week_start, SUM(hours_worked) AS
total_hours
FROM work_logs
GROUP BY employee_id, week_start;
```

297. Identify suppliers who delivered to all regions.

```
WITH total_regions AS (
  SELECT COUNT(DISTINCT region) AS total FROM
suppliers
),
supplier_regions AS (
  SELECT supplier_id, COUNT(DISTINCT
delivery_region) AS regions_served
  FROM deliveries
  GROUP BY supplier_id
)
SELECT supplier_id
FROM supplier_regions
JOIN total_regions ON regions_served = total;
```

298. Find products ordered on their launch date.

```
SELECT o.product_id
FROM orders o
```

```
JOIN products p ON o.product_id = p.product_id
WHERE CAST(o.order_date AS DATE) =
p.launch_date;
```

299. Retrieve employees with salary in top 5% company-wide.

```
WITH salary_ranks AS (
  SELECT salary, PERCENT_RANK() OVER
(ORDER BY salary DESC) AS pr
  FROM employees
)
SELECT e.*
FROM employees e
JOIN salary_ranks sr ON e.salary = sr.salary
WHERE sr.pr <= 0.05;
```

300. List departments with no open positions.

```
SELECT d.department_id
FROM departments d
LEFT JOIN job_openings j ON d.department_id =
j.department_id AND j.status = 'Open'
WHERE j.job_id IS NULL;
```

----------THE END----------