SQL Interview Preparation Guide

100 Essential SQL Queries for Data Analysts

Section 1: Basic Queries (1–10)

Select all columns

```
SELECT * FROM employees;
```

Select specific columns

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

Count all rows

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

Count non-null values

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

Find unique values

```
SELECT DISTINCT department FROM employees;
```

Filter using WHERE

```
SELECT * FROM employees WHERE department = 'Sales';
```

Filter with AND

```
SELECT * FROM employees WHERE department = 'Sales' AND salary > 60000;
```

Filter with OR

```
SELECT * FROM employees WHERE department = 'Sales' OR department = 'Marketing';
```

Complex filter (AND + OR)

```
SELECT * FROM employees WHERE (department = 'Sales' OR department = 'Marketing') AND salary > 70000;
```

Exclude a value using NOT

```
SELECT * FROM employees WHERE department <> 'IT';
```

Section 2: Filtering & Pattern Matching (11–20)

Filter by list using IN

```
SELECT * FROM employees WHERE department IN ('Sales', 'Marketing', 'IT');
```

Exclude list using NOT IN

```
SELECT * FROM employees WHERE department NOT IN ('Sales', 'Marketing');
```

Filter values in a range

```
SELECT * FROM employees WHERE salary BETWEEN 50000 AND 75000;
```

String contains (case-insensitive)

```
SELECT * FROM employees WHERE first_name ILIKE '%jo%';
```

String starts with pattern

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

String ends with pattern

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

NULL values

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

Non-NULL values

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

Filter by date range

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

Pattern at specific position

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

Section 3: Sorting & Limiting (21–30)

Sort ascending

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

Sort descending

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

Sort by multiple columns

```
SELECT * FROM employees ORDER BY department ASC, salary DESC;
```

Top 10 highest paid

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

Latest 5 hired employees

```
SELECT * FROM employees ORDER BY hire_date DESC LIMIT 5;
```

DISTINCT + ORDER BY

```
SELECT DISTINCT department FROM employees ORDER BY department;
```

Pagination: rows 11–20

```
SELECT * FROM employees ORDER BY employee_id OFFSET 10 LIMIT 10;
```

Highest salary employee

```
SELECT * FROM employees ORDER BY salary DESC LIMIT 1;
```

Second-highest salary

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

Order by calculated field

```
SELECT first_name, last_name, (salary * 0.1) AS bonus FROM employees ORDER BY bonus DESC;
```

Section 4: Aggregation & Grouping (31–40)

Count rows per group

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

Average salary per department

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

Total salary per department

```
SELECT department_id, SUM(salary) AS total_salary FROM employees GROUP BY
department_id;
```

Maximum salary per department

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

Minimum salary per department

```
SELECT department_id, MIN(salary) AS min_salary FROM employees GROUP BY department_id;
```

Filter groups using HAVING

```
SELECT department_id, AVG(salary) AS avg_salary

FROM employees

GROUP BY department_id

HAVING AVG(salary) > 65000;
```

Departments with more than 5 employees

```
SELECT department_id, COUNT(*) AS emp_count

FROM employees

GROUP BY department_id

HAVING COUNT(*) > 5;
```

Filter then group

```
SELECT department_id, AVG(salary) AS avg_salary

FROM employees

WHERE hire_date >= '2022-01-01'

GROUP BY department_id;
```

Group by multiple columns

```
SELECT department_id, job_id, AVG(salary) AS avg_salary

FROM employees
```

```
GROUP BY department_id, job_id;
```

Highest earner per department

```
SELECT department_id, first_name, last_name, salary

FROM employees e

WHERE salary = (SELECT MAX(salary) FROM employees WHERE department_id =
e.department_id);
```

Section 5: Joins (41–60)

INNER JOIN

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

LEFT JOIN

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

LEFT JOIN with no match

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

RIGHT JOIN

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

FULL OUTER JOIN

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

Self JOIN (employees & managers)

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

Join three tables

```
SELECT e.first_name, p.project_name, d.department_name
FROM employees e
JOIN projects p ON e.employee_id = p.employee_id
```

```
JOIN departments d ON e.department_id = d.department_id;
```

Cross Join

```
SELECT * FROM employees CROSS JOIN departments;
```

Left Join with filtering (unassigned products)

```
SELECT p.product_name, s.sale_id
FROM products p
LEFT JOIN sales s ON p.product_id = s.product_id
WHERE s.sale_id IS NULL;
```

Aggregation on joined table

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

Section 6: Subqueries (61–75)

Subquery in WHERE clause (single value)

```
SELECT first_name, last_name
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Subquery in WHERE clause (multiple values)

```
SELECT first_name, last_name
FROM employees
WHERE department_id IN (SELECT department_id FROM departments WHERE location_city = 'San Francisco');
```

Subquery in FROM clause (derived table)

```
SELECT d.department_name, a.avg_salary
FROM departments d
JOIN (
   SELECT department_id, AVG(salary) AS avg_salary
   FROM employees
   GROUP BY department_id
) a ON d.department_id = a.department_id;
```

Subquery in SELECT clause (scalar subquery)

```
SELECT first_name, salary, (SELECT AVG(salary) FROM employees) AS company_avg_salary
FROM employees;
```

Correlated subquery: salary > dept average

```
SELECT e.first_name, e.salary
FROM employees e
WHERE e.salary > (
   SELECT AVG(salary)
   FROM employees
   WHERE department_id = e.department_id
);
```

EXISTS to check existence

```
SELECT department_name
FROM departments d
WHERE EXISTS (
  SELECT 1
  FROM employees e
  WHERE e.department_id = d.department_id AND e.salary > 100000
);
```

NOT EXISTS to find non-matching rows

```
SELECT department_name
FROM departments d
WHERE NOT EXISTS (
  SELECT 1
  FROM employees e
  WHERE e.department_id = d.department_id
);
```

Employees who have placed orders

```
SELECT e.first_name, e.last_name
FROM employees e
WHERE EXISTS (
  SELECT 1
  FROM orders o
  WHERE o.employee_id = e.employee_id
);
```

Total salary per department using subquery

```
SELECT department_name,
    (SELECT SUM(salary) FROM employees WHERE department_id = d.department_id) AS total_salary
FROM departments d;
```

Department with highest average salary

```
SELECT department_name
FROM departments
```

```
WHERE department_id = (
  SELECT department_id
  FROM employees
  GROUP BY department_id
  ORDER BY AVG(salary) DESC
  LIMIT 1
);
```

Customers who ordered a specific product

```
SELECT customer_name
FROM customers
WHERE customer_id IN (
  SELECT customer_id
  FROM orders
  WHERE product_id = 123
);
```

Employees earning more than average salary

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

Customers who have not placed an order

```
SELECT customer_name
FROM customers
WHERE customer_id NOT IN (SELECT DISTINCT customer_id FROM orders);
```

Employees earning less than average by job title

```
SELECT e.first_name, e.salary, e.job_title
FROM employees e
WHERE e.salary < (SELECT AVG(salary) FROM employees WHERE job_title = e.job_title);
```

Employees in department of any employee with name containing 'T'

```
SELECT employee_id, first_name, last_name
FROM employees
```

```
WHERE department_id IN (
    SELECT department_id
    FROM employees
    WHERE first_name LIKE '%T%'
);
```

Section 7: Window Functions (76–85)

ROW_NUMBER()

```
SELECT first_name, department_id, salary,
    ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn
FROM employees;
```

RANK()

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

DENSE_RANK()

```
SELECT first_name, department_id, salary,
    DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS dense_rank
FROM employees;
```

NTILE()

```
SELECT first_name, salary,
    NTILE(4) OVER (ORDER BY salary DESC) AS quartile
FROM employees;
```

LEAD()

```
SELECT order_date, total_amount,
    LEAD(total_amount, 1) OVER (ORDER BY order_date) AS next_order_amount
FROM orders;
```

LAG()

```
SELECT order_date, total_amount,
    LAG(total_amount, 1, 0) OVER (ORDER BY order_date) AS prev_order_amount
FROM orders;
```

Running total

```
SELECT order_date, total_amount,
    SUM(total_amount) OVER (ORDER BY order_date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS running_total
FROM orders;
```

Moving average (last 3 rows)

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

Top 3 employees by department using RANK()

```
SELECT * FROM (
  SELECT first_name, department_id, salary,
      RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn
  FROM employees
) AS ranked_employees
WHERE rn <= 3;
```

Percentage of total sales per product

```
SELECT product_id,
    SUM(sales) AS product_sales,
    SUM(SUM(sales)) OVER () AS total_sales,
    (SUM(sales)/SUM(SUM(sales)) OVER())*100 AS percentage_of_total
FROM sales
GROUP BY product_id;
```

Section 8: CTEs & Data Manipulation (86–100)

Basic CTE

```
WITH dept_avg AS (
  SELECT department_id, AVG(salary) AS avg_salary
  FROM employees
  GROUP BY department_id )
SELECT e.first_name, e.salary, da.avg_salary
FROM employees e
JOIN dept_avg da ON e.department_id = da.department_id;
```

INSERT INTO

```
INSERT INTO employees (first_name, last_name, salary)
VALUES ('John', 'Doe', 60000);
```

UPDATE

```
UPDATE employees
SET salary = salary * 1.05
WHERE department_id = 'IT';
```

DELETE

```
DELETE FROM employees WHERE employee_id = 101;
```

TRUNCATE TABLE

```
TRUNCATE TABLE old_data;
```

UNION

```
SELECT first_name FROM employees
UNION
SELECT first_name FROM customers;
```

UNION ALL

```
SELECT first_name FROM employees
UNION ALL
SELECT first_name FROM customers;
```

CASE statement

```
SELECT first_name, salary,
```

```
    CASE
      WHEN salary > 100000 THEN 'High Earner'
      WHEN salary BETWEEN 50000 AND 100000 THEN 'Mid-Range'
      ELSE 'Junior'
    END AS salary_level
FROM employees;
```

Pivot with CASE

```
SELECT department_id,
    COUNT(CASE WHEN salary >= 70000 THEN 1 END) AS high_salary_count,
    COUNT(CASE WHEN salary < 70000 THEN 1 END) AS low_salary_count
FROM employees
GROUP BY department_id;
```

CAST

```
SELECT CAST(order_date AS DATE) FROM orders;
```

COALESCE

```
SELECT COALESCE(email, 'No Email Provided') FROM employees;
```

NULLIF

```
SELECT NULLIF(salary, 0) FROM employees;
```

Recursive CTE

```
WITH RECURSIVE subordinates AS (
  SELECT employee_id, manager_id
  FROM employees
  WHERE manager_id = 1
  UNION ALL
  SELECT e.employee_id, e.manager_id
  FROM employees e
  JOIN subordinates s ON e.manager_id = s.employee_id )
SELECT * FROM subordinates;
```

GROUPING SETS

```
SELECT department_id, job_id, SUM(salary)
```

```
FROM employees
GROUP BY GROUPING SETS ((department_id), (job_id), ());
```

ROLLUP

```
SELECT department_id, job_id, SUM(salary)
FROM employees
GROUP BY ROLLUP(department_id, job_id);
```

Thank you!