

SQL Interview Queries Practice Sheet (with Answers)

31 SQL questions that come up in placement tests and technical interviews, from filters and joins to window functions and deleting duplicates. Every query was run against the sample data shown, and its real output is printed below it.

Free Free PDF 16 pages All levels By [wins.solutions team](#) Updated 2 Oct 2026

Short answer

SQL interview questions keep returning to the same skills: filtering and sorting, GROUP BY with HAVING, subqueries, joins including self joins and anti-joins, NULL handling, and window functions such as RANK, DENSE_RANK, ROW_NUMBER, LAG and running totals. This sheet has 31 of them with answers, each run against one small sample database whose setup script is included, so you can practise in any SQL editor.

WINS.SOLUTIONS · SQL

SQL Interview Queries Practice Sheet (with Answers)

Try each question yourself before reading the answer. Write the query on paper or in a free online SQL editor, then compare. All queries were run in SQLite 3, and the output under each one is exactly what it returned. Where MySQL or PostgreSQL syntax differs, a note says so.

The sample database

Five small tables describe a company. Load them into any SQL editor to practise: the full script is at the end of this sheet, along with a small `signups` table for question 31.

Table	Columns
<code>departments</code>	dept_id, dept_name
<code>employees</code>	emp_id, name, dept_id, manager_id, salary, hire_date, city
<code>projects</code>	project_id, project_name, city
<code>assignments</code>	emp_id, project_id, hours
<code>sales</code>	sale_id, emp_id, sale_date, amount

Things worth knowing about the data, because several questions depend on them: **Marketing has no employees**, **Tara has no department** (dept_id is NULL), **Asha has no manager**, two Engineering employees **share a salary** of 72,000, and **Dev earns more than his manager**.

Basics

1. Filter and sort

List Engineering employees from highest to lowest salary.

```
SELECT e.name, e.salary
FROM employees e
JOIN departments d ON d.dept_id = e.dept_id
WHERE d.dept_name = 'Engineering'
ORDER BY e.salary DESC;
```

Output (5 rows):

name	salary
Asha	150,000
Meera	110,000
Rohan	95,000
Kabir	72,000
Anika	72,000

2. Pattern matching

Find employees whose name starts with A or ends with a.

```
SELECT name
FROM employees
WHERE name LIKE 'A%' OR name LIKE '%a'
ORDER BY name;
```

Output (9 rows): Anika, Arjun, Asha, Ishita, Meera, Neha, Priya, Sara, Tara.

LIKE is case-insensitive for ASCII letters in SQLite and MySQL (default collation) but case-sensitive in PostgreSQL; use ILIKE there.

3. Date range

Find employees hired in 2024.

```
SELECT name, hire_date
FROM employees
WHERE hire_date ≥ '2024-01-01' AND hire_date < '2025-01-01'
ORDER BY hire_date;
```

Output (3 rows):

name	hire_date
Arjun	2024-01-08
Anika	2024-02-20
Ishita	2024-08-01

A half-open range works on DATE and TIMESTAMP columns and can use an index, unlike wrapping the column in YEAR().

4. Handling NULL

Show every employee's department, printing Unassigned when there is none.

```
SELECT e.name, COALESCE(d.dept_name, 'Unassigned') AS department
FROM employees e
LEFT JOIN departments d ON d.dept_id = e.dept_id
WHERE e.dept_id IS NULL OR e.emp_id IN (1, 6);
```

Output (3 rows):

name	department
Asha	Engineering
Vikram	Sales
Tara	Unassigned

The WHERE clause only keeps the output short. Remove it to see everyone.

5. CASE bands

Put each employee in a salary band: under 60k, 60k–100k, above 100k. Count each band.

```
SELECT CASE
  WHEN salary < 60000 THEN 'Under 60k'
  WHEN salary ≤ 100000 THEN '60k-100k'
  ELSE 'Above 100k'
END AS band,
```

```

COUNT(*) AS employees
FROM employees
GROUP BY band
ORDER BY MIN(salary);

```

Output (3 rows):

band	employees
Under 60k	4
60k-100k	8
Above 100k	2

Aggregation

6. Count per group, including zero

Count employees in every department, including departments with none.

```

SELECT d.dept_name, COUNT(e.emp_id) AS employees
FROM departments d
LEFT JOIN employees e ON e.dept_id = d.dept_id
GROUP BY d.dept_id, d.dept_name
ORDER BY employees DESC, d.dept_name;

```

Output (5 rows):

dept_name	employees
Engineering	5
Sales	4
Finance	2
HR	2
Marketing	0

COUNT(e.emp_id) counts only matched rows. COUNT(*) would wrongly give Marketing 1.

7. HAVING

Show departments whose average salary is above 70,000.

```

SELECT d.dept_name, ROUND(AVG(e.salary)) AS avg_salary
FROM employees e

```

```
JOIN departments d ON d.dept_id = e.dept_id
GROUP BY d.dept_name
HAVING AVG(e.salary) > 70000
ORDER BY avg_salary DESC;
```

Output (2 rows):

dept_name	avg_salary
Engineering	99,800
Finance	82,500

WHERE filters rows before grouping; HAVING filters the groups after.

8. Duplicates

Find salaries shared by more than one employee.

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

Output:

salary	how_many
72,000	2

9. Cities

Which cities have more than two employees?

```
SELECT city, COUNT(*) AS employees
FROM employees
GROUP BY city
HAVING COUNT(*) > 2
ORDER BY employees DESC, city;
```

Output (3 rows):

city	employees
Pune	5
Delhi	4

city	employees
Mumbai	3

10. Pivot with CASE

For each department, count employees in Pune and in Delhi as two columns.

```
SELECT d.dept_name,
       SUM(CASE WHEN e.city = 'Pune' THEN 1 ELSE 0 END) AS pune,
       SUM(CASE WHEN e.city = 'Delhi' THEN 1 ELSE 0 END) AS delhi
FROM departments d
JOIN employees e ON e.dept_id = d.dept_id
GROUP BY d.dept_name
ORDER BY d.dept_name;
```

Output (4 rows):

dept_name	pune	delhi
Engineering	3	1
Finance	0	1
HR	2	0
Sales	0	2

Subqueries

11. Highest salary

What is the highest salary?

```
SELECT MAX(salary) AS highest FROM employees;
```

Output: highest = 150,000.

12. Second-highest salary

Find the second-highest distinct salary.

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

Output: second_highest = 110,000.

Returns NULL, not an error, if everyone earns the same. Interviewers often ask about that edge case.

13. Above company average

List employees earning more than the company average.

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

Output (6 rows):

name	salary
Asha	150,000
Meera	110,000
Priya	98,000
Rohan	95,000
Dev	91,000
Vikram	88,000

14. Above their department's average

List employees earning more than their own department's average.

```
SELECT e.name, e.salary
FROM employees e
WHERE e.salary > (SELECT AVG(x.salary)
                  FROM employees x
                  WHERE x.dept_id = e.dept_id)
ORDER BY e.salary DESC;
```

Output (6 rows):

name	salary
Asha	150,000
Meera	110,000
Priya	98,000

name	salary
Dev	91,000
Vikram	88,000
Neha	64,000

This is a correlated subquery: it runs once per outer row, using that row's dept_id.

15. EXISTS

List employees who are assigned to at least one project.

```
SELECT e.name
FROM employees e
WHERE EXISTS (SELECT 1 FROM assignments a WHERE a.emp_id = e.emp_id)
ORDER BY e.name;
```

Output (7 rows): Anika, Dev, Farhan, Kabir, Meera, Priya, Rohan.

Joins

16. Self join

Show each employee with their manager's name, including the person with no manager.

```
SELECT e.name AS employee, COALESCE(m.name, '-') AS manager
FROM employees e
LEFT JOIN employees m ON m.emp_id = e.manager_id
WHERE e.emp_id IN (1, 2, 4, 11)
ORDER BY e.emp_id;
```

Output (4 rows):

employee	manager
Asha	-
Rohan	Asha
Kabir	Meera
Arjun	Neha

The WHERE clause only keeps the output short.

17. Earning more than the manager

Find employees who earn more than their manager.

```
SELECT e.name, e.salary, m.name AS manager, m.salary AS manager_salary
FROM employees e
JOIN employees m ON m.emp_id = e.manager_id
WHERE e.salary > m.salary;
```

Output:

name	salary	manager	manager_salary
Dev	91,000	Vikram	88,000

18. Anti-join

List employees with no project assignment.

```
SELECT e.name
FROM employees e
LEFT JOIN assignments a ON a.emp_id = e.emp_id
WHERE a.emp_id IS NULL
ORDER BY e.name;
```

Output (7 rows): Arjun, Asha, Ishita, Neha, Sara, Tara, Vikram.

NOT EXISTS gives the same result. Avoid NOT IN when the subquery can return NULL.

19. Departments with no employees

Which departments have no employees?

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

Output: Marketing.

20. Many-to-many

List employees working on more than one project, with the count.

```
SELECT e.name, COUNT(*) AS projects
FROM assignments a
JOIN employees e ON e.emp_id = a.emp_id
GROUP BY e.emp_id, e.name
HAVING COUNT(*) > 1
```

```
ORDER BY e.name;
```

Output (2 rows):

name	projects
Kabir	2
Meera	2

21. Totals with zero rows

Total hours booked on each project, showing 0 for projects with no hours.

```
SELECT p.project_name, COALESCE(SUM(a.hours), 0) AS total_hours
FROM projects p
LEFT JOIN assignments a ON a.project_id = p.project_id
GROUP BY p.project_id, p.project_name
ORDER BY total_hours DESC;
```

Output (5 rows):

project_name	total_hours
Mobile app	240
Payments revamp	200
Audit tool	100
Data platform	80
Partner portal	40

22. UNION vs UNION ALL

List every city that appears for an employee or a project, without repeats.

```
SELECT city FROM employees
UNION
SELECT city FROM projects
ORDER BY city;
```

Output (4 rows): Bengaluru, Delhi, Mumbai, Pune.

UNION removes duplicates; UNION ALL keeps them and is faster when you know there are none.

Window functions

23. Nth highest salary

Find the third-highest distinct salary.

```
SELECT DISTINCT salary
FROM (SELECT salary, DENSE_RANK() OVER (ORDER BY salary DESC) AS rnk
      FROM employees) ranked
WHERE rnk = 3;
```

Output: salary = 98,000.

DENSE_RANK gives tied salaries the same rank with no gaps, so N means the Nth distinct value.

24. Top earner per department

Find the highest-paid employee in each department.

```
SELECT dept_name, name, salary
FROM (SELECT d.dept_name, e.name, e.salary,
            RANK() OVER (PARTITION BY e.dept_id ORDER BY e.salary DESC) AS rnk
      FROM employees e
      JOIN departments d ON d.dept_id = e.dept_id) t
WHERE rnk = 1
ORDER BY dept_name;
```

Output (4 rows):

dept_name	name	salary
Engineering	Asha	150,000
Finance	Priya	98,000
HR	Neha	64,000
Sales	Dev	91,000

RANK keeps ties; ROW_NUMBER would pick exactly one row per department.

25. Top two per department

List the two highest-paid employees in each department.

```
SELECT dept_name, name, salary
FROM (SELECT d.dept_name, e.name, e.salary,
            ROW_NUMBER() OVER (PARTITION BY e.dept_id ORDER BY e.salary DESC, e.name) AS
rn
```

```

FROM employees e
JOIN departments d ON d.dept_id = e.dept_id) t
WHERE rn ≤ 2
ORDER BY dept_name, salary DESC;

```

Output (8 rows):

dept_name	name	salary
Engineering	Asha	150,000
Engineering	Meera	110,000
Finance	Priya	98,000
Finance	Farhan	67,000
HR	Neha	64,000
HR	Arjun	41,000
Sales	Dev	91,000
Sales	Vikram	88,000

26. Gap to department maximum

For Engineering, show how far each salary is below the department maximum.

```

SELECT name, salary,
       MAX(salary) OVER (PARTITION BY dept_id) - salary AS below_max
FROM employees
WHERE dept_id = 1
ORDER BY salary DESC;

```

Output (5 rows):

name	salary	below_max
Asha	150,000	0
Meera	110,000	40,000
Rohan	95,000	55,000
Kabir	72,000	78,000
Anika	72,000	78,000

27. Running total

Show a running total of sales by date.

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

Output (10 rows):

sale_date	amount	running_total
2026-01-05	12,000	12,000
2026-01-09	30,000	42,000
2026-01-20	8,000	50,000
2026-02-02	15,000	65,000
2026-02-11	22,000	87,000
2026-02-25	18,000	105,000
2026-03-03	9,500	114,500
2026-03-15	20,000	134,500
2026-03-28	26,000	160,500
2026-03-30	40,000	200,500

28. Previous row with LAG

For employee 8 (Dev), show each sale and the change from the previous sale.

```
SELECT sale_date, amount,
       amount - LAG(amount) OVER (ORDER BY sale_date) AS change
FROM sales
WHERE emp_id = 8
ORDER BY sale_date;
```

Output (4 rows):

sale_date	amount	change
2026-01-09	30,000	NULL
2026-02-11	22,000	-8,000
2026-02-25	18,000	-4,000

sale_date	amount	change
2026-03-28	26,000	8,000

The first row has no previous sale, so its change is NULL.

29. Share of total

Each salesperson's total sales and percentage of all sales.

```
SELECT e.name, SUM(s.amount) AS total,
       ROUND(100.0 * SUM(s.amount) / SUM(SUM(s.amount)) OVER (), 1) AS pct
FROM sales s
JOIN employees e ON e.emp_id = s.emp_id
GROUP BY e.emp_id, e.name
ORDER BY total DESC;
```

Output (4 rows):

name	total	pct
Dev	96,000	47.9
Sara	47,000	23.4
Vikram	40,000	20
Ishita	17,500	8.7

SUM(SUM(...)) OVER () is a window over the grouped rows: the grand total.

30. Monthly totals and rank

Total sales per month, ranked from best to worst month.

```
SELECT strftime('%Y-%m', sale_date) AS month,
       SUM(amount) AS total,
       RANK() OVER (ORDER BY SUM(amount) DESC) AS rnk
FROM sales
GROUP BY month
ORDER BY rnk;
```

Output (3 rows):

month	total	rnk
2026-03	95,500	1
2026-02	55,000	2

month	total	rnk
2026-01	50,000	3

strftime is SQLite. Use DATE_FORMAT(sale_date, '%Y-%m') in MySQL and TO_CHAR(sale_date, 'YYYY-MM') in PostgreSQL.

Data cleaning

31. Delete duplicate rows

The `signups` table (in the setup script below) has exact duplicate rows. Delete the extras, keeping one copy of each.

```
DELETE FROM signups
WHERE rowid NOT IN (SELECT MIN(rowid)
                    FROM signups
                    GROUP BY email, signed_up);
```

Rows left (3 of the original 5):

email	signed_up
a@x.com	2026-01-01
b@x.com	2026-01-02
c@x.com	2026-01-03

rowid is SQLite-specific: in PostgreSQL use ctid, or in any database use ROW_NUMBER() OVER (PARTITION BY email, signed_up) in a CTE and delete rows where it is above 1.

Interview tips

- **Say the order SQL runs in:** FROM and JOIN, then WHERE, GROUP BY, HAVING, SELECT, ORDER BY, LIMIT. It explains why you can't use a SELECT alias in WHERE.
- **Ask about NULLs and ties** before you write the query. "Second-highest salary" changes if two people share the top salary.
- **Prefer NOT EXISTS to NOT IN** when the subquery column can contain NULL.
- **Know three ways to rank:** ROW_NUMBER (always unique), RANK (ties share a rank, then a gap), DENSE_RANK (ties share a rank, no gap).
- **Read the query plan** in interviews about speed: `EXPLAIN` in MySQL and PostgreSQL, `EXPLAIN QUERY PLAN` in SQLite.

Load the sample data

Paste this into SQLite, MySQL or PostgreSQL to practise. It uses only standard types, so it runs unchanged in all three.

```
CREATE TABLE departments (dept_id INTEGER PRIMARY KEY, dept_name TEXT NOT NULL);
CREATE TABLE employees (
    emp_id INTEGER PRIMARY KEY, name TEXT NOT NULL, dept_id INTEGER REFERENCES
    departments(dept_id),
    manager_id INTEGER REFERENCES employees(emp_id), salary INTEGER NOT NULL, hire_date TEXT
    NOT NULL, city TEXT NOT NULL);
CREATE TABLE projects (project_id INTEGER PRIMARY KEY, project_name TEXT NOT NULL, city
    TEXT NOT NULL);
CREATE TABLE assignments (emp_id INTEGER REFERENCES employees(emp_id), project_id INTEGER
    REFERENCES projects(project_id), hours INTEGER NOT NULL);
CREATE TABLE sales (sale_id INTEGER PRIMARY KEY, emp_id INTEGER REFERENCES
    employees(emp_id), sale_date TEXT NOT NULL, amount INTEGER NOT NULL);
CREATE TABLE signups (email TEXT NOT NULL, signed_up TEXT NOT NULL);

INSERT INTO departments VALUES (1,'Engineering'),(2,'Sales'),(3,'HR'),(4,'Finance'),
(5,'Marketing');
INSERT INTO employees VALUES
(1, 'Asha', 1, NULL, 150000, '2019-04-01', 'Pune'),
(2, 'Rohan', 1, 1, 95000, '2021-07-15', 'Pune'),
(3, 'Meera', 1, 1, 110000, '2020-01-10', 'Bengaluru'),
(4, 'Kabin', 1, 3, 72000, '2023-06-01', 'Pune'),
(5, 'Anika', 1, 3, 72000, '2024-02-20', 'Delhi'),
(6, 'Vikram', 2, 1, 88000, '2018-11-05', 'Delhi'),
(7, 'Sara', 2, 6, 52000, '2022-03-14', 'Delhi'),
(8, 'Dev', 2, 6, 91000, '2021-09-30', 'Mumbai'),
(9, 'Ishita', 2, 6, 48000, '2024-08-01', 'Mumbai'),
(10, 'Neha', 3, 1, 64000, '2020-05-18', 'Pune'),
(11, 'Arjun', 3, 10, 41000, '2024-01-08', 'Pune'),
(12, 'Priya', 4, 1, 98000, '2019-12-02', 'Mumbai'),
(13, 'Farhan', 4, 12, 67000, '2022-10-11', 'Delhi'),
(14, 'Tara', NULL, 1, 45000, '2025-01-06', 'Bengaluru');
INSERT INTO projects VALUES (1, 'Payments revamp', 'Pune'), (2, 'Mobile app', 'Bengaluru'),
(3, 'Data platform', 'Pune'), (4, 'Partner portal', 'Delhi'), (5, 'Audit tool', 'Mumbai');
INSERT INTO assignments VALUES (2, 1, 120), (3, 1, 80), (3, 3, 60), (4, 2, 150), (5, 2, 90), (8, 4, 40),
(12, 5, 30), (13, 5, 70), (4, 3, 20);
INSERT INTO sales VALUES
(1, 7, '2026-01-05', 12000), (2, 8, '2026-01-09', 30000), (3, 9, '2026-01-20', 8000),
(4, 7, '2026-02-02', 15000), (5, 8, '2026-02-11', 22000), (6, 8, '2026-02-25', 18000),
(7, 9, '2026-03-03', 9500), (8, 7, '2026-03-15', 20000), (9, 8, '2026-03-28', 26000), (10, 6, '2026-03-
30', 40000);
INSERT INTO signups VALUES ('a@x.com', '2026-01-01'), ('b@x.com', '2026-01-02'),
('a@x.com', '2026-01-01'), ('c@x.com', '2026-01-03'), ('b@x.com', '2026-01-02');
```

For the theory behind these queries (keys, joins, normalization and transactions), see the [DBMS quick revision notes](#).