SQL

Mastery Series

SQL Mastery Series

The Complete Collection

Volumes 1 – 7 · Beginner to Genius

84 hand-picked SQL problems across 7 difficulty levels
Every problem with sample data and expected output
Full solution and step-by-step explanation for each
Pure problem-solving, start to finish — no theory chapters

by

Hatem M.

7 Volumes · 84 Problems · One Complete Path

Beginner → Easy-Medium → Medium → Medium-Hard → Hard → Expert → Genius

About This Book

This edition brings together all seven volumes of the SQL Mastery Series in a single, continuous book: 84 hand-picked problems that take you from your very first SELECT statement to genius-level analytical SQL.

Every volume follows the same structure — a short prompt, an input table, the expected output, a working solution, and a plain explanation of the key idea — so the format never gets in the way of the learning.

The difficulty rises one honest step at a time:

- **Volume 1 · Beginner** — single-table filtering, sorting, basic conditions
- **Volume 2 · Easy-Medium** — the same toolkit, one notch harder
- **Volume 3 · Medium** — joins and subqueries enter the picture
- **Volume 4 · Medium-Hard** — multi-table joins and CTEs
- **Volume 5 · Hard** — window functions
- **Volume 6 · Expert** — interview-grade problems
- **Volume 7 · Genius** — the final volume — everything at once

Solve each problem yourself first — open any free SQL playground (DB Fiddle, SQLite Online, db<>fiddle), recreate the input table, and try your own query before reading the solution. The reading comes second; the typing comes first.

Contents

Each volume contains 12 problems. 84 problems in total.

SQL

Mastery Series

SQL Mastery Series

Volume 1

Beginner Problem Set

12 hand-picked problems
Each with sample data and expected output
Full solution and step-by-step explanation
Pure problem-solving — no theory chapters

About This Volume

This is the first volume in a six-part series of SQL problem sets. Every volume covers the full breadth of standard SQL: filtering, sorting, aggregation, joins, subqueries, window functions, and beyond. What changes between volumes is the difficulty of the problems, not the topic list.

Volume 1 is the warm-up: short problems, single tables, one concept at a time. Solve each problem on your own first — open any free SQL playground (DB Fiddle, SQLite Online, db<>fiddle), recreate the input table, and try your query before reading the solution. The reading comes second; the typing comes first.

Each problem follows the same structure: a short prompt, the input table, the expected output, the solution query, and a brief explanation of the key idea. If a problem feels obvious — good. That is the point of Volume 1.

Contents

#01 List All Employees EASY

◆ Problem

Return every column and every row from the Employees table.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500

◉ Expected Output

Result

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500

✦ Solution

```
SELECT *
FROM Employees;
```

✎ Explanation

SELECT * returns every column. FROM Employees specifies the source table. When no WHERE clause is present, every row is returned in storage order.

#02 Show Names and Departments Only EASY

◆ Problem

Return only the name and department of each employee.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000

◉ Expected Output

Result

name	department
Alice	IT
Bob	HR
Charlie	Sales

✦ Solution

```
SELECT name, department
FROM Employees;
```

✎ Explanation

Listing the columns explicitly after SELECT projects only those columns into the output, in the order they are written.

#03 Employees Earning More Than 4500 EASY

◆ Problem

Return all columns for employees whose salary is strictly greater than 4500.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500
5	Eve	HR	6000

◉ Expected Output

Result

id	name	department	salary
1	Alice	IT	5000
4	Diana	IT	5500
5	Eve	HR	6000

✦ Solution

```
SELECT *
FROM Employees
WHERE salary > 4500;
```

✎ Explanation

WHERE filters rows by a boolean condition. The operator > is strict, so 4500 itself is excluded. Use >= if the boundary value should be included.

#04 Find IT Department Employees EASY

◆ Problem

Return the name and salary of employees in the IT department.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500

◉ Expected Output

Result

name	salary
Alice	5000
Diana	5500

✦ Solution

```
SELECT name, salary
FROM Employees
WHERE department = 'IT';
```

✎ Explanation

String literals are wrapped in single quotes. Most engines compare strings case-sensitively by default (MySQL is a notable exception).

#05 High-Paid IT Employees EASY

◆ Problem

Return employees who work in IT AND earn at least 5500.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500
5	Eve	IT	7000

◉ Expected Output

Result

id	name	department	salary
4	Diana	IT	5500
5	Eve	IT	7000

✦ Solution

```
SELECT *
FROM Employees
WHERE department = 'IT'
AND salary >= 5500;
```

✎ Explanation

Multiple conditions are combined with AND (both must be true) or OR (either must be true). When mixing the two, parentheses control evaluation order — AND binds tighter than OR.

#06 Sort Employees by Salary (High to Low) EASY

◆ Problem

Return all employees sorted by salary in descending order.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500
5	Eve	HR	6000

◉ Expected Output

Result

id	name	department	salary
5	Eve	HR	6000
4	Diana	IT	5500
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000

✦ Solution

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

✎ Explanation

ORDER BY sorts the result. ASC is ascending (the default), DESC is descending. Multiple columns can be supplied, separated by commas, each with its own direction.

#07 Top 3 Highest-Paid Employees EASY

◆ Problem

Return the names and salaries of the three highest-paid employees.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500
5	Eve	HR	6000

◉ Expected Output

Result

name	salary
Eve	6000
Diana	5500
Alice	5000

✦ Solution

```
SELECT name, salary
FROM Employees
ORDER BY salary DESC
LIMIT 3;
```

✎ Explanation

Sort first, then take the top N. LIMIT works in MySQL, PostgreSQL, and SQLite. SQL Server uses TOP, and Oracle uses FETCH FIRST N ROWS ONLY.

#08 List Unique Departments EASY

◆ Problem

Return each department name once, with no duplicates.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500
5	Eve	HR	6000

◉ Expected Output

Result

department
IT
HR
Sales

✦ Solution

```
SELECT DISTINCT department
FROM Employees;
```

✎ Explanation

DISTINCT eliminates duplicate rows from the result set. When applied to multiple columns, it keeps each unique combination of those columns.

#09 Employees in a Salary Range EASY

◆ Problem

Return employees whose salary is between 4500 and 5500, inclusive on both ends.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000

4	Diana	IT	5500
5	Eve	HR	6000

◉ Expected Output

Result

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
4	Diana	IT	5500

✦ Solution

```
SELECT *
FROM Employees
WHERE salary BETWEEN 4500 AND 5500;
```

✎ Explanation

BETWEEN x AND y is inclusive of both endpoints. It is equivalent to: column >= x AND column <= y. It also works with dates and strings.

#10 Employees in HR or Sales EASY

◆ Problem

Return employees who work in HR or Sales using a single condition.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Diana	IT	5500
5	Eve	HR	6000

◉ Expected Output

Result

id	name	department	salary

2	Bob	HR	4500
3	Charlie	Sales	4000
5	Eve	HR	6000

✦ Solution

```
SELECT *
FROM Employees
WHERE department IN ('HR', 'Sales');
```

✎ Explanation

IN matches any value in the given list. It is shorter and clearer than chaining several OR clauses. NOT IN excludes the listed values, but be careful: NOT IN with NULL in the list yields no rows at all.

#11 Names Starting With A — EASY

◆ Problem

Return all employees whose name starts with the letter 'A'.

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	HR	4500
3	Charlie	Sales	4000
4	Adam	IT	5500
5	Anna	HR	6000

◉ Expected Output

Result

id	name	department	salary
1	Alice	IT	5000
4	Adam	IT	5500
5	Anna	HR	6000

✦ Solution

```
SELECT *
FROM Employees
WHERE name LIKE 'A%';
```

✎ Explanation

LIKE matches a text pattern. The wildcard % stands for any sequence of characters (including none). The wildcard _ stands for exactly one character. Patterns starting with % cannot use a normal index efficiently.

#12 Employees Without a Department EASY

◆ Problem

Return employees whose department is unknown (NULL).

▣ Input

Employees

id	name	department	salary
1	Alice	IT	5000
2	Bob	NULL	4500
3	Charlie	Sales	4000
4	Diana	NULL	5500
5	Eve	HR	6000

◉ Expected Output

Result

id	name	department	salary
2	Bob	NULL	4500
4	Diana	NULL	5500

✦ Solution

```
SELECT *
FROM Employees
WHERE department IS NULL;
```

✎ Explanation

NULL means 'unknown', so it cannot be compared with = or <>. Use IS NULL to detect missing values, and IS NOT NULL for present values. Any arithmetic or comparison involving NULL produces NULL, never TRUE.

End of Volume 1

Volume 2 — same topics, harder problems.

Multi-condition filters, trickier patterns, and edge cases that catch beginners off guard.

SQL

Mastery Series

SQL Mastery Series

Volume 6

Expert Problem Set

12 interview-grade problems
Pivot, conditional rates, retention, sessionization
Recursive CTEs over real graphs and time spines
Performance-aware predicates and sargability

About This Volume

Volume 6 collects the kinds of problems you meet in serious technical interviews and in production analytics work. Several of these are direct relatives of well-known LeetCode Hard problems; others are the canonical patterns for cohort retention, sessionization, graph traversal, and pivoted reporting.

By this point in the series, no single keyword is new. What is new is the demand to combine what you already know — a recursive CTE generating a date spine, a window function counting first-time visitors, a conditional sum dividing a sargable date range. Each problem here is, in effect, a small composition.

The last problem deliberately introduces a performance theme: sargability. At the expert level, query correctness alone is not enough. How a query interacts with indexes — whether predicates can be pushed down, whether functions disable index usage — becomes part of the answer.

Contents

#01 Monthly Sales Pivot Per Product HARD

◆ Problem

Transform the long-format Sales table into a wide table: one row per product, with separate columns for total revenue in Jan, Feb, and Mar of 2024. Products with no sales in a month should show 0 there.

▣ Input

Sales

product	sale_date	amount
Laptop	2024-01-10	1500
Laptop	2024-01-25	1200
Laptop	2024-02-14	1800
Laptop	2024-03-05	900
Phone	2024-01-08	700
Phone	2024-03-22	650
Headset	2024-02-10	120

◉ Expected Output

Result

product	jan	feb	mar
Headset	0	120	0
Laptop	2700	1800	900
Phone	700	0	650

✦ Solution

```
SELECT
    product,
    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,
    SUM(CASE WHEN EXTRACT(MONTH FROM sale_date) = 3
             THEN amount ELSE 0 END) AS mar
FROM Sales
WHERE EXTRACT(YEAR FROM sale_date) = 2024
GROUP BY product
ORDER BY product;
```

✎ Explanation

Pivoting in standard SQL is just conditional aggregation. Each output column is a SUM wrapped in a CASE that keeps only the rows belonging to that bucket. ELSE 0 ensures products with no sales in a month show 0 instead of NULL.

Some engines have a dedicated PIVOT operator (SQL Server, Oracle), but the conditional-aggregation form is portable and reads better. The trade-off appears when the pivoted columns are not known in advance — then dynamic SQL or application-side pivoting is the right tool.

SQL

Mastery Series

You've seen the first volume.

Six more are waiting — all the way to genius level.

The complete collection includes:

- All 7 volumes · 84 hand-picked problems, beginner to genius
- Joins, subqueries, CTEs, window functions, and recursive queries
- Interview-grade challenges: medians, gaps-and-islands, graph traversal
- Every problem: sample data, expected output, solution, and explanation
- Every query tested against a real database — no broken code

In this sample:

- ✓ Volume 1 — Beginner (complete, all 12 problems)
- ✓ Volume 6 — Expert (a taste: one full interview-grade problem)

Still locked: Volumes 2, 3, 4, 5, 7 — and the rest of 6.

Get the full book on Leanpub

146 pages · 7 volumes · 84 problems · one complete path