Query Dojo SQL cheat sheet

Run every example at querydojo.com/sql-cheat-sheet

The order of a query

You write the clauses top to bottom. The database runs them in the numbered order.

`SELECT`	pick columns	5
`FROM`	read a table	1
`JOIN`	add a table	1
`WHERE`	filter rows	2
`GROUP BY`	make groups	3
`HAVING`	filter groups	4
`ORDER BY`	sort rows	6
`LIMIT`	keep N rows	7

Select and filter

Pick columns

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

Keep rows that match

```
SELECT first_name, salary
FROM employees
WHERE salary > 100000;
```

Combine conditions

```
SELECT first_name, title, salary
FROM employees
WHERE department_id = 1
  AND salary >= 90000;
```

Match a list or a range

```
SELECT first_name, department_id, salary
FROM employees
WHERE department_id IN (2, 3)
  AND salary BETWEEN 80000 AND 100000;
```

Match a pattern

```
SELECT first_name, title
FROM employees
WHERE title LIKE '%Manager%';
```

Find missing values

```
SELECT first_name, title
FROM employees
WHERE manager_id IS NULL;
```

Sort and keep the top rows

```
SELECT first_name, salary
FROM employees
ORDER BY salary DESC
LIMIT 5;
```

Unique values

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

Count and group

Count rows

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

Sum, average, lowest, highest

```
SELECT SUM(salary) AS total,
       ROUND(AVG(salary)) AS average,
       MIN(salary) AS lowest,
       MAX(salary) AS highest
FROM employees;
```

One row per group

```
SELECT department_id,
       COUNT(*) AS people,
       ROUND(AVG(salary)) AS avg_salary
FROM employees
GROUP BY department_id;
```

Keep groups that match

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

Count only some rows

```
SELECT department_id,
       SUM(CASE WHEN salary >= 100000
           THEN 1 ELSE 0 END) AS high_paid
FROM employees
GROUP BY department_id;
```

Join tables

INNER JOIN: rows with a match on both sides

```
SELECT e.first_name, d.name AS department
FROM employees e
JOIN departments d
  ON d.id = e.department_id;
```

LEFT JOIN: keep every left row

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

Rows with no match

```
SELECT m.first_name, m.title
FROM employees m
LEFT JOIN employees e
  ON e.manager_id = m.id
WHERE e.id IS NULL;
```

Join, then group

```
SELECT d.name AS department,
       COUNT(*) AS people
FROM employees e
JOIN departments d
  ON d.id = e.department_id
GROUP BY d.name
ORDER BY people DESC;
```

Subqueries and CTEs

Compare with one value

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

Match a list from another query

```
SELECT first_name
FROM employees
WHERE department_id IN (
  SELECT id FROM departments
  WHERE location = 'New York'
);
```

Check that a match exists

```
SELECT d.name
FROM departments d
WHERE EXISTS (
  SELECT 1 FROM employees e
  WHERE e.department_id = d.id
    AND e.hire_date >= '2023-01-01'
);
```

Name a step with WITH

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

Window functions

A window function adds a value to every row without collapsing the rows the way GROUP BY does.

Number rows inside each group

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

RANK leaves gaps after ties, DENSE_RANK does not

```
SELECT first_name, salary,
       RANK() OVER (
         ORDER BY salary DESC
       ) AS rnk,
       DENSE_RANK() OVER (
         ORDER BY salary DESC
       ) AS dense
FROM employees;
```

Running total

```
SELECT first_name, hire_date, salary,
       SUM(salary) OVER (
         ORDER BY hire_date, id
       ) AS payroll_so_far
FROM employees;
```

Look at the previous row

```
SELECT first_name, hire_date,
       LAG(hire_date) OVER (
         ORDER BY hire_date, id
       ) AS previous_hire
FROM employees;
```

Compare a row with its group

```
SELECT first_name, salary,
       ROUND(AVG(salary) OVER (
         PARTITION BY department_id
       )) AS dept_avg
FROM employees;
```

CASE, NULLs, text and dates

Text and date functions differ between databases. These run on SQLite; the idea is the same everywhere.

If and else

```
SELECT first_name, salary,
  CASE
    WHEN salary >= 120000 THEN 'senior'
    WHEN salary >= 80000 THEN 'mid'
    ELSE 'junior'
  END AS band
FROM employees;
```

Replace a missing value

```
SELECT first_name,
       COALESCE(manager_id, 0)
         AS manager_id
FROM employees;
```

Join and measure text

```
SELECT first_name || ' ' || last_name
         AS full_name,
       UPPER(last_name) AS shout,
       LENGTH(email) AS email_length
FROM employees;
```

Group by part of a date

```
SELECT strftime('%Y', hire_date) AS year,
       COUNT(*) AS hires
FROM employees
GROUP BY year
ORDER BY year;
```

Change data

Each run works on a private copy of the database, so nothing here is permanent.

Add a row

```
INSERT INTO departments
  (id, name, location)
VALUES (6, 'Support', 'Austin');

SELECT * FROM departments;
```

Change rows

```
UPDATE employees
SET salary = salary + 5000
WHERE department_id = 4;

SELECT first_name, salary
FROM employees
WHERE department_id = 4;
```

Remove rows

```
DELETE FROM employees
WHERE hire_date < '2017-01-01';

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