SQL Commands

Standard SQL that works across PostgreSQL, MySQL, and SQLite unless noted. Keywords are shown uppercase by convention; SQL itself is case-insensitive for keywords.

Querying

Clause	What it does
`SELECT col1, col2`	pick columns (`*` for all)
`FROM table`	the source table
`WHERE cond`	filter rows before grouping
`ORDER BY col DESC`	sort results (default `ASC`)
`LIMIT 10 OFFSET 20`	paginate: 10 rows, skipping 20
`SELECT DISTINCT col`	unique values only
`AS alias`	rename a column or table in the output

A typical read query

```
SELECT name, email, created_at
FROM users
WHERE active = true
ORDER BY created_at DESC
LIMIT 25;
```

Filtering

Operator	Matches
`col = 5`	exact equality
`col <> 5`	not equal (also `≠`)
`col LIKE 'a%'`	starts with a (`%` = any, `_` = one char)
`col ILIKE 'a%'`	case-insensitive LIKE (Postgres)
`col IN (1, 2, 3)`	matches any value in the list
`col BETWEEN 1 AND 9`	inclusive range
`col IS NULL`	test for NULL (never use `= NULL`)
`a = 1 AND b = 2`	combine conditions (`OR`, `NOT` too)

Joins

Join	Returns
`INNER JOIN`	only rows matching in both tables
`LEFT JOIN`	all left rows + matches (NULLs if none)
`RIGHT JOIN`	all right rows + matches
`FULL OUTER JOIN`	all rows from both sides
`CROSS JOIN`	every combination (Cartesian product)

Join with an aggregate

```sql
SELECT u.name, COUNT(o.id) AS orders
FROM users u
LEFT JOIN orders o ON o.user_id = u.id
GROUP BY u.name
ORDER BY orders DESC;
```

Aggregation

Function / clause	Purpose
`COUNT(*)`	number of rows
`SUM(col)`	total of a numeric column
`AVG(col)`	average
`MIN(col) / MAX(col)`	smallest / largest
`GROUP BY col`	collapse rows into groups
`HAVING COUNT(*) > 1`	filter groups (WHERE filters rows)

> **WHERE vs HAVING:** `WHERE` filters individual rows *before* grouping; `HAVING` filters *after* grouping, so it can reference aggregates like `COUNT(*)`.

Modifying Data

Insert, update, delete

```sql
INSERT INTO users (name, email)
VALUES ('Ada', 'ada@x.io');

UPDATE users SET active = false
WHERE last_seen < '2025-01-01';

DELETE FROM users WHERE id = 42;
```

Upsert (Postgres / SQLite)

```sql
INSERT INTO users (email, name)
VALUES ('ada@x.io', 'Ada')
ON CONFLICT (email)
DO UPDATE SET name = EXCLUDED.name;
```

> **Always pair UPDATE / DELETE with a WHERE.** Without one, you change *every* row. Run the matching `SELECT` first, and wrap risky changes in a `BEGIN; ... ROLLBACK;` transaction while you check.

Tables & Indexes

Create a table

```
CREATE TABLE orders (
  id        SERIAL PRIMARY KEY,
  user_id   INTEGER REFERENCES users(id),
  total     NUMERIC(10,2) NOT NULL,
  created_at TIMESTAMPTZ DEFAULT now()
);
```

Statement	Effect
`ALTER TABLE t ADD COLUMN c TEXT`	add a column
`ALTER TABLE t DROP COLUMN c`	remove a column
`CREATE INDEX ON t (col)`	speed up lookups on col
`DROP TABLE t`	delete a table and its data