SQL Query language

FROM / JOIN pick rows → **WHERE** → **GROUP BY** build groups → **HAVING** → **SELECT** compute columns → **DISTINCT** → **ORDER BY** sort → **LIMIT** → **Result** rows returned

Logical order a SELECT is evaluated in

CONCEPTS

Write the clauses in one order, but the database evaluates FROM first and SELECT late: a column alias from SELECT is not visible in WHERE.

Examples use PostgreSQL. Most of the rest is portable; differences are named in tips (MySQL, SQLite).

QUERYING

`SELECT <col>, <col> FROM <table>;`
Read chosen columns

`SELECT * FROM <table>;`
Read all columns

`WHERE <col> = <v> AND <col> > <v>`
Filter rows

`ORDER BY <col> DESC, <col2>`
Sort, descending then ascending

`LIMIT <n> OFFSET <m>`
Take a page of rows

`SELECT DISTINCT <col> FROM <table>;`
Remove duplicate rows

QUERYING (cont.)

`WHERE <col> IN (<a>, <b>)`
Match any of a list

`WHERE <col> LIKE 'ab%'`
Pattern match (% any, _ one char)

`WHERE <col> IS NULL`
Find missing values

JOINS

`FROM <a> JOIN <b> ON <a>.id = <b>.<a>_id`
Inner join: only matching rows

`FROM <a> LEFT JOIN <b> ON <cond>`
Keep all rows of the left table

`FROM <a> LEFT JOIN <b> ON <cond> WHERE <b>.id IS NULL`
Anti-join: rows with no match

`<query1> UNION [ALL] <query2>`
Stack results, UNION removes duplicates

AGGREGATION

`COUNT(*)`
Count rows

AGGREGATION (cont.)

`COUNT(DISTINCT <col>)`
Count unique non-NULL values

`SUM(<col>), AVG(<col>)`
Total and mean

`MIN(<col>), MAX(<col>)`
Smallest and largest

`GROUP BY <col>`
One output row per distinct value

`HAVING COUNT(*) > <n>`
Filter groups after aggregation

SUBQUERIES & CTES

`WITH <name> AS (SELECT ...) SELECT ... FROM <name>;`
Name a subquery (CTE)

`WHERE <col> IN (SELECT <col> FROM <t>)`
Membership test with a subquery

`WHERE EXISTS (SELECT 1 FROM <t> WHERE <t>.<fk> = o.id)`
Is there at least one related row?

`WHERE NOT EXISTS (SELECT 1 FROM ...)`
No related row exists

WINDOW FUNCTIONS

`<agg>(<col>) OVER (PARTITION BY <g>)`
Aggregate without collapsing rows

`row_number() OVER (PARTITION BY <g> ORDER BY <c>)`
Number rows within a group

`lag(<col>) OVER (ORDER BY <c>)`
Value from the previous row

`sum(<col>) OVER (ORDER BY <c>)`
Running total

CHANGING DATA

`INSERT INTO <t> (<a>, <b>) VALUES (<v>, <w>);`
Insert a row

`UPDATE <t> SET <col> = <v> WHERE <cond>;`
Change matching rows

`DELETE FROM <t> WHERE <cond>;`
Delete matching rows

`INSERT ... RETURNING *`
Get the written rows back

`INSERT ... ON CONFLICT (<col>) DO UPDATE SET <c> = EXCLUDED.<c>`
Upsert

SCHEMA

`CREATE TABLE <t> (<col> <type> [constraints], ...);`
Create a table

CONSTRAINTS

`<col> <type> PRIMARY KEY`
Unique, non-NULL row identity

EXPRESSIONS

`CASE WHEN <cond> THEN <a> ELSE <b> END`
Conditional value

`COALESCE(<a>, <b>, <default>)`
First non-NULL value

TRANSACTIONS & PLANS

`BEGIN;`
Start a transaction

`COMMIT;`
Make the changes permanent

`ROLLBACK;`
Undo everything since BEGIN

`EXPLAIN ANALYZE <query>;`
Run it and show real timings