SQL Query language

START HERE

`SELECT <cols> FROM <table>;`
Read rows

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

`WHERE <cond>`
Filter rows

`DELETE FROM <t> WHERE <cond>;`
Remove rows

`ORDER BY <col> DESC`
Sort the result

`JOIN <t2> ON <t1>.id = <t2>.<fk>`
Combine tables

`INSERT INTO <t> (<cols>) VALUES (...);`
Add a row

`GROUP BY <col>`
Aggregate per group

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).

End statements with ; . Strings use single quotes, identifiers double quotes: 'text' vs "Column".

QUERYING

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

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

Fine for exploring; list columns in application code.

`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

Large OFFSETs are slow; page by the last seen key instead.

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

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

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

ILIKE is case-insensitive in PostgreSQL; MySQL and SQLite LIKE already ignore case for ASCII.

`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

Unmatched right columns are NULL; put right-side filters in ON, not WHERE.

JOINS (cont.)

`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

`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

WHERE filters rows before grouping, HAVING filters groups.

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

Safer than NOT IN, which returns nothing if the list contains a NULL.

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

WINDOW FUNCTIONS (cont.)

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

Window functions run after WHERE and GROUP BY; filter on their result by wrapping the query in a CTE.

CHANGING DATA

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

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

Without WHERE every row changes.

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

Without WHERE every row goes.

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

Also on UPDATE and DELETE; SQLite 3.35+, MariaDB, not MySQL.

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

MySQL: ON DUPLICATE KEY UPDATE.

SCHEMA

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

`ALTER TABLE <t> ADD COLUMN <col> <type>;`
Add a column

`CREATE INDEX <name> ON <t> (<col>);`
Speed up lookups on a column

Common types

Type	Holds
`integer / bigint`	Whole numbers
`numeric(p,s)`	Exact decimals, money
`text / varchar(n)`	Strings
`boolean`	true / false
`timestamptz`	Point in time with zone
`date`	Calendar day
`jsonb`	Binary JSON (PostgreSQL)
`uuid`	128-bit identifier

CONSTRAINTS

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

`<col> <type> NOT NULL`
Forbid missing values

CONSTRAINTS (cont.)

`<col> <type> UNIQUE`
No duplicates in this column

`<col> <type> DEFAULT <v>`
Value used when none is given

`<col> <type> REFERENCES <parent>(id)`
Foreign key to another table

`<col> bigint GENERATED ALWAYS AS IDENTITY`
Auto-increment key

MySQL: AUTO_INCREMENT; SQLite: INTEGER PRIMARY KEY.

EXPRESSIONS

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

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

`CAST(<x> AS <type>) / <x>::<type>`
Convert type

The :: form is PostgreSQL only.

`date_trunc('month', <ts>)`
Round a timestamp down

TRANSACTIONS & PLANS

`BEGIN;`
Start a transaction

`COMMIT;`
Make the changes permanent

`ROLLBACK;`
Undo everything since BEGIN

`EXPLAIN <query>;`
Show the planned execution

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

ANALYZE executes the statement, so wrap UPDATE or DELETE in BEGIN ... ROLLBACK.

RECIPES

Latest row per group

```
SELECT DISTINCT ON
(customer_id) *
FROM orders
ORDER BY customer_id,
created_at DESC;

-- portable: row_number() OVER
(PARTITION BY ... ORDER BY ...
DESC) = 1
```

Top N per group

```
WITH r AS (
  SELECT *, row_number() OVER (
    PARTITION BY dept ORDER BY
salary DESC) AS rn
  FROM employees)
SELECT * FROM r WHERE rn ⩽ 3;
```