CHEAT SHEET · ALIJABBARY.COM

SQL Quick Reference for Data Analysts

The SQL you'll actually use as a data analyst — window functions, CTEs, JOIN mechanics, and the patterns behind every dashboard you've shipped.

01 · QUERY ORDER

SQL is written one way, executed another

Knowing the logical execution order explains why aliases sometimes don't work and why WHERE can't see SUM().

. FROM + JOIN	Build the working table — sources are materialised first
. WHERE	Filter rows BEFORE grouping — uses raw column values only
. GROUP BY	Collapse rows into groups
. HAVING	Filter GROUPS (aggregates) — use this for SUM > 100, not WHERE
. SELECT	Pick columns + compute aggregates — this is when aliases come alive
. ORDER BY → LIMIT	Sort + slice last — can reference SELECT aliases

02 · JOIN MECHANICS

Five JOINs, four shapes you'll actually use

- INNER JOIN — only rows with a match on BOTH sides (default; use when you mean 'rows that have both')
- LEFT JOIN — every left row, NULLs where no right match. The default for 'fact + dimension' lookups
- FULL OUTER JOIN — every row from either side; rare but useful for reconciliation reports
- CROSS JOIN — Cartesian product; intentional only (date spines, calendar tables)
- Anti-join idiom: LEFT JOIN … WHERE right.id IS NULL → 'rows that DON'T have a match'
- Row explosion is the #1 JOIN bug — confirm each join key is unique on at least one side before joining

03 · CTES

WITH … AS — readable > clever

CTEs (Common Table Expressions) turn nested-subquery spaghetti into a sequence of named steps. Read them top to bottom.

- WITH base AS (SELECT …), enriched AS (SELECT … FROM base JOIN …) SELECT * FROM enriched
- Each CTE is a virtual table; the final SELECT uses them like real tables
- PostgreSQL ≥ 12, Snowflake, BigQuery, DuckDB all inline CTEs by default — no performance penalty
- Recursive CTEs (WITH RECURSIVE) traverse hierarchies — org charts, threaded comments, graph walks

- Stop at 4-5 CTEs in one query. Beyond that, materialise into a real table or a dbt model

04 · WINDOW FUNCTIONS

Aggregate without collapsing rows

ROW_NUMBER() OVER (PARTITION BY x ORDER BY y)	Rank within group — pick latest per user etc.
RANK() / DENSE_RANK()	Like ROW_NUMBER but ties get the same rank
LAG(col) / LEAD(col) OVER (...)	Previous / next row value — period-over-period deltas
SUM(x) OVER (PARTITION BY u ORDER BY d)	Running total per user, in date order
AVG(x) OVER (ORDER BY d ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)	7-day rolling average — the dashboard classic
NTILE(4) OVER (ORDER BY x)	Bucket rows into quartiles

05 · PATTERNS THAT PAY RENT

Three queries you'll write 100 times

- Latest row per group: ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY created_at DESC) = 1
- Cohort retention: GROUP BY signup_month, period_index → COUNT(DISTINCT user_id)
- Funnel conversion: COUNT(*) FILTER (WHERE step = 'signup') / COUNT(*) FILTER (WHERE step = 'visit')
- FILTER clause beats CASE WHEN inside aggregates — cleaner and faster in PostgreSQL / DuckDB
- Date spines: generate_series('2026-01-01', '2026-12-31', '1 day') for gap-filled time series

06 · PERFORMANCE

Make the database do the work — but read the plan

- EXPLAIN ANALYZE shows what actually happened — read it for any query that runs more than 1 second
- Indexes help equality + range scans; they don't help SELECT * with no WHERE
- Avoid SELECT * in production queries — bloats network I/O and breaks when columns are added
- Prefer EXISTS over IN for membership checks on large tables — EXISTS short-circuits
- Push filters EARLY — filter inside the CTE, not after joining; less data flowing downstream

Beyond the basics

COMMON PITFALLS

- ! NULL is not equal to anything, including itself — use IS NULL / IS NOT NULL, never = NULL
- ! COUNT(col) skips NULLs; COUNT(*) counts every row — they answer different questions
- ! Integer division: 5 / 2 = 2 in many engines. Cast: 5::float / 2 or 5.0 / 2
- ! GROUP BY with no aggregate on selected columns is a 'bare column' error in standard SQL — list every non-aggregated column
- ! Comparing timestamps across time zones — store and compare in UTC; convert only at display time

FURTHER READING

- → PostgreSQL window functions tutorial (https://www.postgresql.org/docs/current/tutorial-window.html)
- → Mode SQL Tutorial (free, hands-on) (https://mode.com/sql-tutorial/)
- → Use The Index, Luke! — performance tuning by example (https://use-the-index-luke.com/)
- → DuckDB SQL reference (modern analytical engine) (https://duckdb.org/docs/sql/introduction)