Engineer

SQL

the language relational databases answer

What it is

SQL is a declarative language for defining, changing and reading data kept in relational tables: you write down the result you want, and the database decides how to produce it. The letters are usually expanded as Structured Query Language and pronounced either letter by letter or as “sequel”, while the international standard that defines it, ISO/IEC 9075, calls it simply Database Language SQL.

Declarative is the word doing the work. The statement below names a table, a condition, a grouping and an order. It says nothing about indexes, loops or which rows to read first:

```
SELECT country, count(*) AS signups
FROM accounts
WHERE created_at >= now() - interval '30 days'
GROUP BY country
ORDER BY signups DESC;
```

In PostgreSQL that text passes through four stages. A parser checks the syntax and builds a tree, a rewrite step expands any views the statement mentions, a planner compares the possible ways to run it, such as a sequential scan against an index, and an executor finally fetches the rows. The same sentence can take a millisecond or a minute depending on what the planner finds, which is why reading a plan belongs to writing SQL.

The language has parts that people name separately. Data definition creates and alters tables, schemas and indexes. Data manipulation inserts, updates, deletes and merges rows. Queries read them with SELECT. Grants and transactions

decide who may do what, and which changes land together. All of it travels as text over a single connection, so SQL is the common ground between an application, a reporting tool and a person at a terminal.

The standard has been revised many times, most recently as SQL:2023, and every engine implements its own subset of it. PostgreSQL's conformance appendix states that it meets at least 170 of the 177 mandatory Core features, and that at the time of writing no database system claimed full Core conformance. Each engine also adds a dialect of its own: Microsoft SQL Server speaks Transact-SQL, BigQuery speaks GoogleSQL, and PostgreSQL adds PL/pgSQL for procedural code that runs inside the database.

Use the concept in a decision

Put "What is SQL?" to work in a real decision. We can help relate the concept to the system you are building or buying. A structure that matches how the business actually works, so the questions you need answered later are still possible to ask.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Designed and ran 6 ETL/ELT pipelines — 6 ETL pipelines across three database engines

Designed and ran 6 ETL/ELT pipelines Get in touch →

Why it matters

The decision SQL puts in front of a team is where the work on data happens: inside the database, in the language it was built to execute, or in application code that fetches rows and processes them itself.

A set-based statement runs next to the data. Filtering a million rows down to forty inside the database sends forty rows over the network; filtering in the application sends a million and throws most of them away. Joins, aggregates and deduplication follow the same arithmetic. Moving that kind of work into SQL is often the cheapest speed-up available to a slow system, because it runs on the database already in place.

Ownership is the counterweight. Logic written in SQL lives in migrations and functions, which need the same review, tests and version control as any other code, and fewer engineers read a long PL/pgSQL function as fluently as they read Go or Python. A workable split sends whatever has to see many rows to the database and keeps rules that change with the business in the application, where they are easier to test in isolation.

Portability is the second trade-off. The core of the language, SELECT with joins, GROUP BY and ORDER BY, moves between engines with few edits. The edges move badly: date arithmetic, upserts, pagination, JSON functions and procedural code all differ, and a migration from one database to another spends most of its effort there. Choosing an engine is also choosing a dialect, and the more logic lives in that dialect, the more a later move costs.

Hiring is the third. SQL is one of the few skills that analysts, backend developers, data engineers and database administrators share, so a data layer written in plain SQL can be read and reviewed by any of them. An ORM hides the SQL until a page turns slow, and at that point someone still has to read what it generated.

What it looks like in practice

On one platform, PostgreSQL functions assemble each API response as a complete JSON document, and the Go handler forwards it as `json.RawMessage` without decoding or re-encoding it. The JSON passthrough architecture makes shaping the response a SQL concern, done where the data already sits. A small QueryJSON helper turned that pattern into the easiest one to follow, and the DTOs, mappers and response structs that a layered API usually collects went away. Handler code became much shorter, and a change to what a function returns reaches the client without anyone editing a handler.

The same language looks different in a data pipeline. Across six ETL/ELT pipelines spanning BigQuery, MSSQL and PostgreSQL, raw files were loaded into staging schemas with bcp, psql and SSIS, and the cleaning, normalising and enriching were written as PL/pgSQL and T-SQL procedures. Three engines, each with its own dialect, inside one job: the portability question above, seen from the inside. The page records the outcome as at least three hours of manual work reduced to under 20 minutes end to end, data refreshed daily instead of weekly, and the Python APIs that read it running 30% faster.

Where it goes wrong

Statements glued together from strings. A customer named O’Brien breaks a search page, or a crafted input reads someone else’s data. The cause is user input pasted into the SQL text, where the database cannot tell data from command. Send values as parameters instead, separately from the statement, as every

mainstream driver allows. In PostgreSQL's own client library a parameterised call also accepts only one command per string, which adds a second line of defence against injection.

Order that was only luck. A list that came back sorted for months arrives shuffled after a data load or a new plan. Without ORDER BY, PostgreSQL returns rows in whatever order is fastest to produce, and a LIMIT without ORDER BY may even return a different subset each time. Any result whose order matters says so in the query.

NULL treated as a value. A filter written `WHERE discount = NULL` returns nothing, and `NOT IN` against a subquery that contains a NULL returns nothing either. Any comparison with NULL yields unknown rather than true or false, and a WHERE clause keeps only the rows where its condition is true. Write `IS NULL`, and prefer `NOT EXISTS` to `NOT IN` when the inner column can hold a NULL.

Dialect that leaks into tests. A test suite passes against SQLite and the same code fails against PostgreSQL in production, or a move to SQL Server trips on every `LIMIT`, which that engine spells `TOP` or `OFFSET ... FETCH`. Each engine accepts things the others reject. Test against the engine you deploy, and keep engine-specific syntax in as few places as you can.

Further reading

- Part II. The SQL Language — PostgreSQL's full tutorial and reference for the language, from syntax to performance tips.
- Appendix D. SQL Conformance — the parts of ISO/IEC 9075 and which features PostgreSQL implements.
- Transact-SQL Reference — the SQL Server dialect, documented by Microsoft.
- Introduction to SQL in BigQuery — GoogleSQL and the legacy dialect it replaced.
- Query Language Understood by SQLite — the syntax of an embedded engine, statement by statement.



Glossary

Choose for your actual constraints

Bring a representative dataset, the questions it must answer and the current bottleneck. We can agree how to measure quality, throughput and recovery.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is SQL?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/sql/