Engineer

Database

a store of data organised so that programs can query it

What it is

A database is a body of data kept in a fixed structure, with rules about what may go into it, so that software can find, change and check its contents; the same word, often shortened to DB, is also used loosely for the program that does the keeping, whose exact name is a database management system or DBMS. NIST's glossary prints two definitions, and the broader one describes a repository of information or data "which may or may not be a traditional relational database system".

The word does three jobs, and a conversation goes in circles when two people mean different ones. It names the data: the customers, orders and sensor readings an organisation keeps. It names the engine that keeps them: PostgreSQL, SQL Server, SQLite. And inside a server it names one container among several. PostgreSQL's tutorial sets out the levels: rows with named, typed columns make up tables, tables are grouped into databases, and the databases that one server instance manages together form a database cluster. SQL Server draws the same line between an instance and the databases it hosts. "The database is slow" can mean one table, one container or one whole machine.

Most engines in daily use are relational: they store data as relations, the mathematical term for tables, and take questions in SQL. The PostgreSQL tutorial names other arrangements. The files and directories of a Unix-like system form a hierarchical database, and object-oriented databases came later. Redis calls itself a data structure server: in place of tables it offers strings, hashes, lists, sets and other native types, put to work for caching, queuing and event processing.

Engines also differ in where they run. A client/server engine is a long-running process that many programs reach over a network. An embedded engine is a library inside one program, and SQLite keeps a whole database, tables, indexes, triggers and views included, in one ordinary file; the project says it competes with `fopen()` rather than with server databases. The difference shows in how a program connects:

```
# client/server: connect to a service that is already running
psql "host=db.internal dbname=shop user=app"

# embedded: the program opens a file and the engine runs inside it
sqlite3 shop.db
```

A third family is built for analysis. BigQuery stores each column separately, where many traditional systems keep a row's fields together on disk. Row storage suits looking up one record; column storage suits summing one field across millions of rows, because the query reads that column and leaves the rest alone.

Use the concept in a decision

Put "What is a database?" 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

Files hold data too. A database adds promises about that data for the moment many programs use it at once.

The first promise is a consistent picture under concurrent use. PostgreSQL gives each statement a snapshot of the data as it stood at a moment, a design called multiversion concurrency control, or MVCC, so a read never waits for a write and a write never waits for a read. SQLite allows any number of readers at once and one writer at any instant, with the other writers waiting their turn. The two suit different numbers of writers, and that number does much of the choosing.

The second promise is that data keeps its shape. A schema states which columns a table has and what type each holds; keys and constraints state which rows may exist. When the database enforces those rules, every program that writes to it obeys them. When the rules live only in application code, the first program that skips them leaves rows behind that every later query has to defend against.

The third is a shared language, with accents. SQL is the standard ISO/IEC 9075, last revised as SQL:2023, yet PostgreSQL's conformance appendix notes that no current system claims full conformance to its Core. A schema and its queries belong to their engine, and moving them to another is a migration with its own testing.

SQLite's own documentation turns the choice of engine into a short checklist: data across a network from the code that issues the SQL, many processes writing at the same instant, or content heading for the terabyte range each point to client/server, and otherwise SQLite is the simpler answer. Reporting at scale adds one more question: whether heavy reads belong on a separate analytical copy, filled by ETL, so that a long report never competes with the queries customers wait on.

What it looks like in practice

At the large end of the scale, a database engineer role at a rapidly growing tech company created, managed and optimised 100+ databases across PostgreSQL, Microsoft SQL Server and Google BigQuery, most of them holding GIS data and time-series data such as sensor readings, logs and real-time metrics. The work produced normalised and denormalised schemas for PostgreSQL and SQL Server, with spatial indexing (PostGIS on the PostgreSQL side) and time-series partitioning, while BigQuery carried large time-series data in time-partitioned, clustered tables. Once those databases had indexing, materialised views and caching, GIS and time-series queries were answered in 40–60% less time, and automated monitoring reduced downtime by 50%.

In a smaller system, at one documented stage of development, a single SQLite file held the facts behind a CV, a reference sheet, a portfolio and a cover letter in three languages. That database had twenty-three tables, and one command built them in nineteen ordered steps, any of which could safely run again. Six output formats were rendered from the database by 41 Python modules totalling 10,580 lines. A document aimed at one kind of reader was "a query rather than a copy": a profile selected which achievements appeared and in what order. The

trade-off was that a document could then come from that database and nowhere else. These implementation counts describe that recorded stage, not the current repository or its language coverage.

Between those sizes sits the question every busy database meets, and a note on optimising before sharding answers it with an order of work: look, tighten, take the reads off the primary, partition the tables around the hot spot, and shard only when one machine is full of good work. Looking means asking PostgreSQL itself: `pg_stat_statements` for the statements that cost most in total, then `EXPLAIN (ANALYZE, BUFFERS)` for what the costliest one does.

Where it goes wrong

Most database trouble starts in a decision made early, when the data was small and the decision looked free.

Two copies of one fact drift apart. The CV generator above exists because facts kept as four documents get corrected four times, until two of them disagree and nothing says which is current. The same happens to a table exported to a spreadsheet and edited there, or a second service with its own list of customers. Give each fact one store that owns it, and make every other copy say where it came from and when. The note's cache follows this rule strictly: it may tell a buyer an item is fully reserved, and every yes still comes from the database.

More machines arrive before anyone reads the plans. Splitting data across servers spreads the slow statements over more hardware without making any of them cheaper, and it puts a lasting bill on everything else: as the note lists it, every migration, backup, restore and report runs once per shard, and a join across shards becomes application code. Read `pg_stat_statements` and the plans first, and keep what they show as the baseline.

A read lands on a copy that is behind. A streaming replica takes reads off the primary and runs a little behind it. A user who has just written something and then reads it back from the replica may see the old value. Read a user's own recent writes from the primary.

Reports and customers share one engine. A report that scans two years of history holds memory and disk that the shop's own queries need, and the people buying feel it. Move heavy analysis to a warehouse copy, and keep the working

database for the many small reads and writes it serves.

Further reading

- NIST glossary: database — two short definitions from NIST publications, one of which allows for databases that are not relational.
- PostgreSQL tutorial: Concepts — tables, rows, columns, databases and the database cluster, in one page.
- PostgreSQL: Introduction to MVCC — how snapshots let readers and writers work without blocking each other.
- PostgreSQL: SQL Conformance — the parts and versions of ISO/IEC 9075 and what Core conformance means.
- Appropriate Uses For SQLite — where an embedded engine fits, where a client/server one fits better, and a checklist for deciding.
- Overview of BigQuery storage — row-oriented and column-oriented storage, and why analysis favours the second.

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 a database?” 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/database/