**Engineer**

# Datenbank

ein geordneter Datenspeicher, den Programme abfragen können

## Was es ist

Eine Datenbank ist ein Datenbestand mit fester Struktur und Regeln für zulässige Inhalte, damit Software die Daten finden, ändern und prüfen kann. Dasselbe Wort, oft zu DB abgekürzt, bezeichnet im lockeren Gebrauch auch das verwaltende Programm. Dessen genauer Name lautet Datenbankmanagementsystem oder DBMS. Das NIST-Glossar führt zwei Definitionen auf. Die weiter gefasste beschreibt einen Informations- oder Datenspeicher, „der ein herkömmliches relationales Datenbanksystem sein kann, aber nicht muss“.

Das Wort erfüllt drei Aufgaben. Wenn zwei Menschen Verschiedenes meinen, dreht sich das Gespräch im Kreis. Es bezeichnet die Daten: Kunden, Bestellungen und Sensormesswerte einer Organisation. Es bezeichnet das verwaltende System: PostgreSQL, SQL Server, SQLite. Und innerhalb eines Servers bezeichnet es einen von mehreren Containern. Das PostgreSQL-Tutorial beschreibt die Ebenen: Zeilen mit benannten, typisierten Spalten bilden Tabellen, Tabellen werden zu Datenbanken gruppiert und die von einer Serverinstanz gemeinsam verwalteten Datenbanken bilden einen Datenbankcluster. SQL Server zieht dieselbe Grenze zwischen einer Instanz und den von ihr gehosteten Datenbanken. „Die Datenbank ist langsam“ kann eine Tabelle, einen Container oder einen ganzen Rechner meinen.

Die meisten im Alltag verwendeten Systeme sind relational: Sie speichern Daten als Relationen, den mathematischen Begriff für Tabellen, und nehmen Fragen in SQL entgegen. Das PostgreSQL-Tutorial nennt andere Anordnungen. Dateien und Verzeichnisse eines Unix-ähnlichen Systems bilden eine hierarchische Datenbank. Objektorientierte Datenbanken kamen später hinzu. Redis bezeichnet sich als Datenstrukturserver: Statt Tabellen bietet es Zeichenfolgen, Hashes, Listen, Mengen und weitere native Typen für Caching, Warteschlangen und Ereignisverarbeitung.

Die Systeme unterscheiden sich auch durch ihren Ausführungsort. Ein Client-Server-System ist ein dauerhaft laufender Prozess, den viele Programme über ein Netzwerk erreichen. Ein eingebettetes Datenbanksystem läuft als Bibliothek innerhalb eines Programms. SQLite hält eine ganze Datenbank mit Tabellen, Indizes, Triggern und Sichten in einer gewöhnlichen Datei. Das Projekt sieht sich eher im Wettbewerb mit `fopen()` als mit Serverdatenbanken. Der Unterschied zeigt sich beim Verbinden eines Programms:

```
# 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
```

Eine dritte Familie ist für Analysen gebaut. BigQuery speichert jede Spalte getrennt, während viele klassische Systeme die Felder einer Zeile gemeinsam auf dem Datenträger halten. Zeilenorientierte Speicherung passt zum Nachschlagen eines Datensatzes. Spaltenorientierte Speicherung passt zum Summieren eines Felds über Millionen Zeilen: Die Abfrage liest nur diese Spalte und lässt den Rest aus.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Datenbank?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Eine Struktur, die zur tatsächlichen Arbeitsweise Ihres Unternehmens passt, damit sich auch spätere Fragen noch beantworten lassen.

**Unsere Leistungen entdecken** Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

**6 ETL/ELT-Pipelines entworfen und betrieben** Kontakt aufnehmen →

## Warum es wichtig ist

Auch Dateien halten Daten. Eine Datenbank ergänzt Zusagen für den Moment, in dem viele Programme gleichzeitig darauf zugreifen.

Die erste Zusage ist ein konsistentes Bild bei gleichzeitiger Nutzung. PostgreSQL gibt jeder Anweisung einen Snapshot der Daten zu einem Zeitpunkt. Das heißt Multiversion Concurrency Control oder MVCC: Ein Lesevorgang wartet dadurch nie auf einen Schreibvorgang, und ein Schreibvorgang nie auf einen Lesevorgang.

SQLite erlaubt beliebig viele gleichzeitige Leser und zu jedem Zeitpunkt einen Schreiber. Die übrigen Schreiber warten, bis sie an der Reihe sind. Beide passen zu unterschiedlichen Anzahlen von Schreibern. Diese Zahl bestimmt einen großen Teil der Auswahl.

Die zweite Zusage ist, dass die Daten ihre Form behalten. Ein Schema legt Spalten und Typen einer Tabelle fest. Schlüssel und Constraints bestimmen, welche Zeilen existieren dürfen. Setzt die Datenbank diese Regeln durch, hält sich jedes schreibende Programm daran. Stehen die Regeln nur im Anwendungscode, hinterlässt das erste Programm, das sie überspringt, Zeilen, gegen die sich jede spätere Abfrage absichern muss.

Die dritte Zusage ist eine gemeinsame Sprache mit Akzenten. SQL ist der Standard ISO/IEC 9075, zuletzt als SQL:2023 überarbeitet. Der PostgreSQL-Anhang zur Konformität hält aber fest, dass kein aktuelles System vollständige Konformität mit dessen Core beansprucht. Ein Schema und seine Abfragen gehören zu ihrem Datenbanksystem. Der Umzug auf ein anderes ist eine Migration mit eigenen Tests.

Die SQLite-Dokumentation macht die Wahl zu einer kurzen Checkliste: Der SQL-ausführende Code erreicht die Daten über ein Netzwerk, viele Prozesse schreiben im selben Moment oder die Inhalte nähern sich dem Terabytebereich. Jeder dieser Punkte spricht für Client-Server. Andernfalls ist SQLite die einfachere Antwort. Berichte in großem Maßstab ergänzen eine Frage: Gehören aufwendige Lesevorgänge auf eine separate, per ETL befüllte Analysekopie, damit ein langer Bericht nie mit den Abfragen konkurriert, auf die Kunden warten?

## Wie es in der Praxis aussieht

Am großen Ende der Skala wurden im Rahmen einer Tätigkeit im Datenbankengineering bei einem schnell wachsenden Technologieunternehmen mindestens 100 Datenbanken auf PostgreSQL, Microsoft SQL Server und Google BigQuery erstellt, verwaltet und optimiert. Die meisten enthielten GIS- und Zeitreihendaten wie Sensormesswerte, Logs und Echtzeitmetriken. Die Arbeit lieferte normalisierte und denormalisierte Schemas für PostgreSQL und SQL Server mit räumlicher Indizierung, auf PostgreSQL-Seite mit PostGIS, und Zeitreihenpartitionierung. BigQuery hielt dagegen große Zeitreihenbestände in

zeitpartitionierten, geclusterten Tabellen. Nach Indizierung, materialisierten Sichten und Caching wurden GIS- und Zeitreihenabfragen in 40–60 % weniger Zeit beantwortet. Automatisiertes Monitoring senkte die Ausfallzeit um 50 %.

In einem kleineren System hielt zu einem dokumentierten Entwicklungsstand eine einzige SQLite-Datei die Fakten hinter einem Lebenslauf, einem Referenzblatt, einem Portfolio und einem Anschreiben in drei Sprachen. Diese Datenbank hatte dreiundzwanzig Tabellen. Ein Befehl baute sie in neunzehn geordneten Schritten auf, die sich jeweils sicher wiederholen ließen. Sechs Ausgabeformate wurden von 41 Python-Modulen mit insgesamt 10.580 Zeilen aus der Datenbank gerendert. Ein Dokument für eine bestimmte Lesergruppe war „eine Abfrage statt einer Kopie“: Ein Profil wählte die Leistungen und ihre Reihenfolge. Der damit verbundene Kompromiss war, dass ein Dokument damals nur aus dieser Datenbank und keiner anderen Quelle stammen konnte. Diese Implementierungszahlen beschreiben den dokumentierten Stand, nicht das aktuelle Repository oder seine Sprachabdeckung.

Zwischen diesen Größen liegt die Frage jeder stark genutzten Datenbank. Eine Notiz zur Optimierung vor dem Sharding beantwortet sie mit einer Arbeitsreihenfolge: hinsehen, straffen, Lesevorgänge vom Primärsystem nehmen, Tabellen um den Engpass herum partitionieren und erst dann sharden, wenn ein Rechner mit sinnvoller Arbeit ausgelastet ist. Hinsehen bedeutet, PostgreSQL selbst zu fragen: `pg_stat_statements` nach den Anweisungen mit den höchsten Gesamtkosten und dann `EXPLAIN (ANALYZE, BUFFERS)` danach, was die teuerste tut.

## Wo es schiefgeht

Die meisten Datenbankprobleme beginnen mit einer frühen Entscheidung, als die Datenmenge noch gering war und die Entscheidung kostenlos wirkte.

**Zwei Kopien einer Tatsache driften auseinander.** Der Lebenslaufgenerator oben existiert, weil in vier Dokumenten gehaltene Fakten viermal korrigiert werden, bis zwei Dokumente einander widersprechen und nichts erkennen lässt, welches aktuell ist. Dasselbe passiert mit einer in eine Tabellenkalkulation exportierten und dort bearbeiteten Tabelle oder einem zweiten Dienst mit eigener Kundenliste. Geben Sie jeder Tatsache einen verantwortlichen Speicher. Jede weitere Kopie muss ihre Herkunft und ihren Zeitpunkt nennen. Der

Cache der Notiz folgt dieser Regel streng: Er darf einem Käufer mitteilen, dass ein Artikel vollständig reserviert ist. Jede Zusage kommt weiterhin aus der Datenbank.

**Mehr Rechner kommen, bevor jemand die Pläne liest.** Daten auf Server aufzuteilen verteilt die langsamen Anweisungen auf mehr Hardware, ohne eine davon günstiger zu machen. Es verursacht dauerhaft Kosten bei allem anderen: Wie die Notiz aufzählt, laufen jede Migration, Sicherung, Wiederherstellung und jeder Bericht einmal pro Shard. Ein Join über Shards wird zu Anwendungscode. Lesen Sie zuerst `pg_stat_statements` und die Pläne und halten Sie ihre Ergebnisse als Ausgangsbasis fest.

**Ein Lesevorgang trifft auf eine zurückliegende Kopie.** Ein Streaming-Replikat übernimmt Lesevorgänge des Primärsystems und hat einen etwas älteren Datenstand. Hat ein Nutzer gerade etwas geschrieben und liest es vom Replikat zurück, kann er den alten Wert sehen. Lesen Sie die eigenen kürzlich geschriebenen Daten eines Nutzers vom Primärsystem.

**Berichte und Kunden teilen sich ein Datenbanksystem.** Ein Bericht über zwei Jahre Historie belegt Speicher und Datenträgerressourcen, die die Abfragen des Shops brauchen. Das spüren die Käufer. Verlagern Sie aufwendige Analysen auf eine Warehouse-Kopie. Lassen Sie die operative Datenbank ihre vielen kleinen Lese- und Schreibvorgänge bedienen.

## Weiterführende Lektüre

- NIST-Glossar: Datenbank — zwei kurze Definitionen aus NIST-Veröffentlichungen, von denen eine auch nicht relationale Datenbanken zulässt.
- PostgreSQL-Tutorial: Konzepte — Tabellen, Zeilen, Spalten, Datenbanken und Datenbankcluster auf einer Seite.
- PostgreSQL: Einführung in MVCC — wie Snapshots Leser und Schreiber ohne gegenseitiges Blockieren arbeiten lassen.
- PostgreSQL: SQL-Konformität — Teile und Versionen von ISO/IEC 9075 und die Bedeutung von Core-Konformität.
- Geeignete Einsatzbereiche für SQLite — wo ein eingebettetes Datenbanksystem passt, wo ein Client-Server-System besser passt und eine Checkliste zur Entscheidung.
- Überblick über die BigQuery-Speicherung — zeilen- und spaltenorientierte Speicherung und warum Analysen die zweite bevorzugen.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

**Preise ansehen** Kontakt aufnehmen →

# Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist eine Datenbank?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

**Unsere Leistungen entdecken**

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/database/