Engineer

PostgreSQL

🐘 Datenbank, auf der der Großteil dieser Arbeit aufbaut

Was es ist

PostgreSQL ist ein freier, quelloffener relationaler Datenbankserver, der Daten in Tabellen hält, in SQL formulierte Fragen beantwortet und jede Transaktion nach dem Alles-oder-nichts-Prinzip ausführt. Die meisten Menschen, die damit arbeiten, nennen ihn Postgres; das Projekt betrachtet beide Namen als seine eigenen. Auf die Frage „Was ist PostgreSQL?“ würde jemand aus der Entwicklung es schlicht so sagen: die universell einsetzbare Datenbank, mit der ein neues System anfängt, wenn niemand einen bestimmten Grund für eine andere Wahl hat.

Es ist älter als die meisten Programme, deren Grundlage es heute bildet. Das Projekt POSTGRES begann an der University of California in Berkeley unter Professor Michael Stonebraker; seine Implementierung startete 1986. Andrew Yu und Jolly Chen ergänzten 1994 einen SQL-Interpreter und veröffentlichten das Ergebnis als Postgres95. Zwei Jahre später wurde es in PostgreSQL umbenannt. Die Versionsnummerierung setzte wieder bei 6.0 an, um die in Berkeley begonnene Folge fortzuführen. Das Projekt selbst bezeichnet das System als objektrelational: Über Tabellen und Joins hinaus können Nutzer neue Datentypen, Funktionen, Operatoren, Aggregate, Indexverfahren und prozedurale Sprachen hinzufügen, die der Server wie seine eigenen behandelt.

Die Architektur besteht aus Client und Server. Ein Serverprozess verwaltet die Datendateien, und jede Clientverbindung erhält einen eigenen, ausschließlich für diese Verbindung abgespaltenen Prozess. Anwendungen senden darüber SQL, über einen Treiber für die jeweilige Programmiersprache. Die Nebenläufigkeit beruht auf der Mehrversionen-Nebenläufigkeitskontrolle: Eine Aktualisierung schreibt eine neue Version einer Zeile, statt die alte zu überschreiben. Dadurch liest jede Anweisung einen konsistenten Datenstand; lesende Vorgänge warten nie auf schreibende und schreibende nie auf lesende.

Wo steht es im Vergleich zu den Alternativen? Neben MySQL und MariaDB als weiteren quelloffenen relationalen Servern, neben Oracle und Microsoft SQL Server als kommerziellen Angeboten, neben SQLite als eingebetteter Datenbank innerhalb eines einzelnen Programms und gegenüber Dokumentendatenbanken wie MongoDB, die Datensätze ohne festes Schema speichern.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist PostgreSQL?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Die Abfragen und fehlenden Indizes hinter einem von Jahr zu Jahr schwerfälligeren System finden und die entscheidenden Probleme beheben.

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. 30 Tage Stromdaten in 180 ms — 180 ms für die Aggregation eines Monats an Zählerständen

30 Tage Stromdaten in 180 ms Kontakt aufnehmen →

Warum es wichtig ist

Mit der Wahl einer Datenbank lebt ein Team jahrelang, denn die Daten überdauern den dafür geschriebenen Code. PostgreSQL ist aus drei Gründen eine Standardwahl, die man selten bereut und deren Vorteile ein Auftraggeber prüfen kann.

Der erste Grund sind Kosten und Anbieterbindung. PostgreSQL steht unter der PostgreSQL License, einer kurzen, freizügigen Lizenz aus der Familie von BSD und MIT. Es gibt deshalb keine Gebühr pro Kern, kein Lizenzaudit und keinen Anbieter, der die Bedingungen ändern kann. Amazon, Microsoft und Google bieten es jeweils als verwalteten Dienst an; dieselbe Datenbank läuft auch auf einem Laptop. Ein Wechsel des Hosts bedeutet damit eine Datenmigration statt einer Neuentwicklung der Anwendung.

Der zweite Grund ist die Breite. Ein Server deckt relationale Tabellen, JSON-Dokumente in `jsonb`-Spalten, Volltextsuche, räumliche Abfragen über die Erweiterung PostGIS und geplante Jobs über `pg_cron` ab. Ohne diese Bündelung müsste ein Team für jeden dieser Bereiche ein eigenes Produkt betreiben und absichern, Sicherungskopien davon erstellen und Personal dafür einstellen. Für die meisten Produkte entscheidet sich hier die Wahl zwischen PostgreSQL und MySQL: Einfache Lese- und Schreibzugriffe beherrschen beide gut. Der Unterschied zeigt sich, wenn reichhaltigere Datentypen, strengere Constraints oder eine Erweiterung nötig werden.

Der dritte Grund ist die Langlebigkeit. Das Projekt veröffentlicht jährlich eine Hauptversion und unterstützt jede fünf Jahre lang. Ein Upgrade lässt sich dadurch planen, statt von einem Notfall erzwungen zu werden. PostgreSQL 18, erschienen im September 2025, ergänzte asynchrone Ein- und Ausgabe sowie eine native Funktion für zeitlich geordnete UUIDs. SQL, das für zehn Jahre ältere Versionen geschrieben wurde, läuft darauf im Allgemeinen unverändert.

Die Nachteile sind real. Ein Prozess je Verbindung macht Verbindungen teuer; eine stark genutzte Anwendung braucht deshalb einen Pooler. Auf einem größeren Rechner wächst PostgreSQL leichter als über mehrere hinweg: Schreibzugriffe auf viele Server zu verteilen erfordert Partitionierung, Replikation oder eine Erweiterung, und jede davon erhöht den Betriebsaufwand. Außerdem sind die Voreinstellungen auf einen kleinen Rechner abgestimmt. Eine anspruchsvolle Installation muss daher bewusst konfiguriert werden.

Wie es in der Praxis aussieht

Viele Projekte auf dieser Website laufen mit PostgreSQL. Drei davon zeigen drei unterschiedliche Arten, seine Möglichkeiten zu nutzen.

Auf einer Plattform wurde die Datenbank zur gesamten Datenschicht. Jeder Lese- und Schreibzugriff läuft über eine von 1.275 PostgreSQL-Funktionen in 34 Schemas; die Go-API hat keinen direkten Zugriff auf irgendeine Tabelle. Die Anmelderolle der API, `mariner`, besitzt EXECUTE auf den Funktionen und USAGE auf den Schemas und sonst nichts, insgesamt rund 4.000 ausdrücklich erteilte Rechte. Die Funktionen laufen als SECURITY DEFINER unter einem separaten Eigentümer ohne Anmeldemöglichkeit und mit fest gesetztem `search_path`. Postgres selbst setzt die Zugriffsgrenze durch. Der Preis dafür: Ein neues Feld erfordert eine Migration und eine Änderung der Funktion.

Bei einer Anwendung mit Mandanten, Standorten und täglichen Stromablesungen ging es um Optimierung statt um den Entwurf. Das Ziel war eine Verbesserung um den Faktor 10, ohne die Architektur grundlegend umzubauen. GiST-Indizes für räumliche Daten, zusammengesetzte Indizes auf Mandant und Datumsbereich, Bereichspartitionierung der Stromtabelle nach Tag und Monat sowie nach der Auswertung ihrer Pläne mit `EXPLAIN ANALYZE` umgeschriebene Abfragen verkürzten die Abfrageantwortzeiten um den Faktor 10 und senkten die Ressourcenauslastung um 40 %.

Auf einer Plattform mit langsamen Hintergrundaufgaben liegt die Jobwarteschlange ebenfalls in der Datenbank. River hält seine Warteschlange in PostgreSQL, sodass ein Job und die von ihm betroffenen Zeilen gemeinsam bestätigt oder zurückgerollt werden. Dahinter stehen 15 Worker-Module, 8 geplante Aufgaben und 20 `pg_cron` -Jobs, die Partitionen bereinigen, Salts rotieren und Aggregate aktualisieren. Die Seite benennt die Grenze klar: Ab einem gewissen Volumen ist eine Warteschlange in Postgres nicht mehr die richtige Antwort. Eine Betriebskomponente weniger war mehr wert als Leistungsreserven, die niemand brauchen würde.

Wo es schiefgeht

PostgreSQL verliert selten Daten. Es überrascht Menschen aber durchaus, und diese Überraschungen folgen einigen bekannten Mustern.

Die Tabelle wächst, obwohl die Zeilenzahl gleich bleibt. Ein `UPDATE` oder `DELETE` lässt die alte Zeilenversion zurück, bis `VACUUM` sie entfernt, weil eine andere Transaktion sie noch lesen könnte. Eine stundenlang offene Transaktion hält diese Bereinigung für die gesamte Datenbank auf, und die Autovacuum-Voreinstellungen passen zu kleinen Tabellen. Schließen Sie inaktive Transaktionen, stimmen Sie Autovacuum für große Tabellen einzeln ab und partitionieren Sie Daten mit Ablaufdatum, damit alte Monate entfernt werden können, statt ihre Zeilen einzeln zu löschen. Vacuum hat noch eine zweite Aufgabe: Transaktions-IDs sind 32 Bit breit, jede Tabelle muss spätestens nach jeweils zwei Milliarden Transaktionen bereinigt werden, und ein zu weit zurückgefallener Server verweigert neue Schreibzugriffe, um die Daten zu schützen.

Beim ersten Lastanstieg geht der Speicher aus. Jede Verbindung ist ein Serverprozess mit eigenem Speicher; das Standardlimit liegt bei 100 Verbindungen. Zwanzig Anwendungsinstanzen mit jeweils fünfzig Poolverbindungen fordern tausend Prozesse an. Schalten Sie einen Pooler wie PgBouncer davor und bemessen Sie den Pool jeder Anwendung nach der Arbeit. Meist genügt eine Handvoll Verbindungen je Instanz.

Eine Abfrage braucht plötzlich Sekunden statt Millisekunden. Mit den wachsenden Daten hat der Planer einen anderen Plan gewählt, oft weil seine Statistiken veraltet sind oder die Indizes zu den Abfragen des Vorjahres

passen. Lesen Sie den Plan, bevor Sie etwas ändern, aktualisieren Sie die Statistiken mit `ANALYZE` und indexieren Sie das Prädikat, nach dem die Abfrage tatsächlich filtert:

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT id, total FROM orders
WHERE customer_id = 42 AND placed_at >= '2026-01-01';
```

Einfügungen werden mit wachsender Tabelle langsamer. Zufällige UUIDs als Primärschlüssel landen überall im B-Baum-Index. Jede Einfügung greift dadurch auf eine andere Seite zu, und der Index fragmentiert. Zeitlich geordnete Kennungen vermeiden das: Eine Plattform machte UUID v7 über `uuidv7()` von PostgreSQL 18 zum Standardschlüssel. Neue Zeilen fügen sich damit geordnet in den Index ein, statt sich über ihn zu verteilen.

Weiterführende Lektüre

- Was ist PostgreSQL? — die Definition des Projekts selbst mit den dort genannten Funktionen und Erweiterungspunkten.
- Eine kurze Geschichte von PostgreSQL — Berkeley, Postgres95 und die Umbenennung, von den daran Beteiligten.
- Nebenläufigkeitskontrolle: Einführung — wie konsistente Datenstände Lese- und Schreibvorgänge ohne gegenseitige Blockierung ermöglichen.
- Regelmäßige Vacuum-Bereinigung — nicht mehr benötigte Zeilenversionen, Autovacuum und der Überlauf der Transaktions-IDs im Detail.
- EXPLAIN verwenden — Abfragepläne lesen, mit ausgearbeiteten Beispielen.
- Versionspolitik — die jährliche Hauptversion und die jeweils fünfjährige Unterstützung.

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 PostgreSQL?“ 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/postgresql/