Engineer

SQL

Sprache, die relationale Datenbanken verstehen

Was es ist

SQL ist eine deklarative Sprache zum Definieren, Ändern und Lesen von Daten in relationalen Tabellen: Man beschreibt das gewünschte Ergebnis, und die Datenbank entscheidet, wie sie es erzeugt. Die Buchstaben werden meist als Structured Query Language ausgeschrieben und entweder einzeln oder wie „sequel“ ausgesprochen. Der internationale Standard ISO/IEC 9075, der die Sprache definiert, nennt sie hingegen schlicht Database Language SQL.

Entscheidend ist das Wort „deklarativ“. Die folgende Anweisung nennt eine Tabelle, eine Bedingung, eine Gruppierung und eine Sortierung. Sie sagt nichts über Indizes, Schleifen oder darüber, welche Zeilen zuerst gelesen werden sollen:

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

In PostgreSQL durchläuft dieser Text vier Stufen. Ein Parser prüft die Syntax und baut einen Baum auf. Ein Umschreibeschritt erweitert die in der Anweisung genannten Sichten. Ein Planer vergleicht mögliche Ausführungswege, etwa einen sequenziellen Scan mit der Nutzung eines Index. Schließlich ruft ein Executor die Zeilen ab. Dieselbe Anweisung kann eine Millisekunde oder eine Minute dauern, je nachdem, was der Planer findet. Deshalb gehört das Lesen eines Plans zum Schreiben von SQL.

Die Sprache umfasst Teile mit eigenen Namen. Datendefinition erzeugt und verändert Tabellen, Schemas und Indizes. Datenmanipulation fügt Zeilen ein, aktualisiert, löscht und führt sie zusammen. Abfragen lesen sie mit SELECT.

Berechtigungen und Transaktionen bestimmen, wer was darf und welche Änderungen gemeinsam wirksam werden. Alles wird als Text über eine einzige Verbindung übertragen. SQL ist damit die gemeinsame Grundlage für eine Anwendung, ein Berichtswerkzeug und einen Menschen am Terminal.

Der Standard wurde häufig überarbeitet, zuletzt als SQL:2023, und jede Engine implementiert ihre eigene Teilmenge. PostgreSQLs Anhang zur Standardkonformität erklärt, dass mindestens 170 der 177 verpflichtenden Core-Merkmale erfüllt werden und dass zum Zeitpunkt der Abfassung kein Datenbanksystem vollständige Core-Konformität beanspruchte. Jede Engine ergänzt außerdem einen eigenen Dialekt: Microsoft SQL Server verwendet Transact-SQL, BigQuery GoogleSQL, und PostgreSQL ergänzt PL/pgSQL für prozeduralen Code innerhalb der Datenbank.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist SQL?“ 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

SQL stellt ein Team vor die Entscheidung, wo die Datenverarbeitung stattfindet: innerhalb der Datenbank in der Sprache, für deren Ausführung sie gebaut wurde, oder im Anwendungscode, der Zeilen abruft und selbst verarbeitet.

Eine mengenbasierte Anweisung läuft direkt bei den Daten. Wer eine Million Zeilen in der Datenbank auf vierzig filtert, überträgt vierzig Zeilen über das Netzwerk. Wer erst in der Anwendung filtert, überträgt eine Million und verwirft die meisten davon. Für Joins, Aggregationen und das Entfernen von Duplikaten gilt dieselbe Rechnung. Solche Arbeit nach SQL zu verlagern ist oft die günstigste Beschleunigung für ein langsames System, weil sie auf der bereits vorhandenen Datenbank läuft.

Das Gegengewicht ist die Verantwortung für den Code. SQL-Logik steckt in Migrationen und Funktionen, die dieselben Reviews, Tests und dieselbe Versionsverwaltung benötigen wie jeder andere Code. Weniger Entwickler lesen eine lange PL/pgSQL-Funktion so flüssig wie Go oder Python. Eine praktikable Aufteilung verlagert alles, was viele Zeilen sehen muss, in die Datenbank und belässt Regeln, die sich mit dem Geschäft ändern, in der Anwendung. Dort lassen sie sich leichter isoliert testen.

Die zweite Abwägung betrifft die Portabilität. Der Kern der Sprache, SELECT mit Joins, GROUP BY und ORDER BY, lässt sich mit wenigen Änderungen zwischen Engines übertragen. An den Rändern gelingt das schlecht: Datumsarithmetik, Upserts, Paginierung, JSON-Funktionen und prozeduraler Code unterscheiden sich. Der größte Teil des Aufwands einer Datenbankmigration fällt dort an. Mit einer Engine wählt man auch einen Dialekt. Je mehr Logik darin steckt, desto teurer wird ein späterer Wechsel.

Der dritte Punkt ist die Personalgewinnung. SQL gehört zu den wenigen Fähigkeiten, die Analysten, Backend-Entwickler, Data Engineers und Datenbankadministratoren gemeinsam haben. Eine in reinem SQL geschriebene Datenschicht kann daher von allen gelesen und geprüft werden. Ein ORM versteckt das SQL, bis eine Seite langsam wird. Dann muss trotzdem jemand lesen, was es erzeugt hat.

Wie es in der Praxis aussieht

Auf einer Plattform bauen PostgreSQL-Funktionen jede API-Antwort als vollständiges JSON-Dokument zusammen, und der Go-Handler reicht es als `json.RawMessage` weiter, ohne es zu dekodieren oder erneut zu kodieren. Die JSON-Passthrough-Architektur verlagert die Form der Antwort in SQL, direkt zu den bereits vorhandenen Daten. Eine kleine Hilfsfunktion namens QueryJSON machte dieses Muster zum einfachsten Weg. Die DTOs, Mapper und Antwortstrukturen, die sich in einer geschichteten API üblicherweise ansammeln, entfielen. Der Handler-Code wurde deutlich kürzer, und eine Änderung am Rückgabewert einer Funktion erreicht den Client, ohne dass jemand einen Handler bearbeitet.

In einer Datenpipeline zeigt dieselbe Sprache ein anderes Bild. In sechs ETL-/ELT-Pipelines über BigQuery, MSSQL und PostgreSQL wurden Rohdateien mit bcp, psql und SSIS in Staging-Schemas geladen. Bereinigung, Normalisierung und Anreicherung wurden als PL/pgSQL- und T-SQL-Prozeduren geschrieben. Drei

Engines mit je einem eigenen Dialekt innerhalb eines Auftrags: die zuvor beschriebene Portabilitätsfrage, von innen betrachtet. Die Seite dokumentiert das Ergebnis: mindestens drei Stunden Handarbeit wurden auf unter 20 Minuten von Anfang bis Ende verkürzt, die Daten täglich statt wöchentlich aktualisiert, und die darauf zugreifenden Python-APIs liefen 30 % schneller.

Wo es schiefgeht

Aus Zeichenketten zusammengesetzte Anweisungen. Ein Kunde namens O’Brien legt eine Suchseite lahm, oder eine präparierte Eingabe liest fremde Daten aus. Ursache sind Benutzereingaben, die in den SQL-Text eingefügt werden. Dort kann die Datenbank Daten und Befehle nicht unterscheiden. Übergeben Sie Werte stattdessen getrennt von der Anweisung als Parameter, wie es alle gängigen Treiber erlauben. In PostgreSQLs eigener Client-Bibliothek akzeptiert ein parametrisierter Aufruf außerdem nur einen Befehl pro Zeichenkette. Das schafft eine zweite Schutzschicht gegen Injektionen.

Eine Reihenfolge, die nur Zufall war. Eine monatelang sortiert gelieferte Liste kommt nach einem Datenimport oder einem neuen Plan durcheinander an. Ohne ORDER BY gibt PostgreSQL Zeilen in der Reihenfolge zurück, die sich am schnellsten erzeugen lässt. Ein LIMIT ohne ORDER BY kann sogar jedes Mal eine andere Teilmenge liefern. Wenn die Reihenfolge eines Ergebnisses wichtig ist, muss die Abfrage das ausdrücklich festlegen.

NULL wird wie ein Wert behandelt. Ein Filter mit `WHERE discount = NULL` liefert nichts zurück. Dasselbe gilt für `NOT IN` mit einer Unterabfrage, die NULL enthält. Jeder Vergleich mit NULL ergibt „unbekannt“ statt wahr oder falsch, und eine WHERE-Klausel behält nur Zeilen, deren Bedingung wahr ist. Schreiben Sie `IS NULL` und bevorzugen Sie `NOT EXISTS` gegenüber `NOT IN`, wenn die innere Spalte NULL enthalten kann.

Ein Dialekt schleicht sich in die Tests ein. Die Tests laufen mit SQLite erfolgreich durch, während derselbe Code in Produktion mit PostgreSQL scheitert. Oder beim Wechsel zu SQL Server scheitert jedes `LIMIT`, für das diese Engine `TOP` oder `OFFSET ... FETCH` verwendet. Jede Engine akzeptiert Dinge, die andere ablehnen. Testen Sie gegen die tatsächlich eingesetzte Engine und konzentrieren Sie enginespezifische Syntax auf möglichst wenige Stellen.

Weiterführende Lektüre

- Teil II: Die SQL-Sprache — PostgreSQLs vollständige Einführung und Referenz zur Sprache, von der Syntax bis zu Leistungstipps.
- Anhang D: SQL-Konformität — Die Teile von ISO/IEC 9075 und die von PostgreSQL implementierten Merkmale.
- Transact-SQL-Referenz — Der von Microsoft dokumentierte SQL-Server-Dialekt.
- Einführung in SQL in BigQuery — GoogleSQL und der dadurch abgelöste frühere Dialekt.
- Die von SQLite verstandene Abfragesprache — Die Syntax einer eingebetteten Engine, Anweisung für Anweisung.

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 SQL?“ 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/sql/