Engineer

Tabellenpartitionierung

Eine Tabelle in physisch getrennte Teile aufteilen

Was es ist

Partitionierung, auch Tabellenpartitionierung genannt, speichert die Zeilen einer logischen Tabelle in mehreren gewöhnlichen Tabellen. Jede enthält diejenigen Zeilen, deren Schlüssel innerhalb ihrer Grenzen liegt. Abfragen und Schreibzugriffe verwenden weiterhin denselben einzelnen Namen wie zuvor. PostgreSQL, dem diese Seite folgt, nennt seine eingebaute Form deklarative Partitionierung: Die Tabelle wird mit einem Verfahren und einem Partitionsschlüssel als partitioniert deklariert, und jede Partition entsteht als eigene Tabelle.

Die partitionierte Tabelle selbst hält keine Zeilen. Eine Einfügung wird in die Partition geleitet, deren Grenzen zum Schlüssel passen. Ändert eine Aktualisierung den Schlüssel, wandert die Zeile in die nun passende Partition. Partitionen teilen sich die Spalten der übergeordneten Tabelle, können aber eigene Indizes, Speichereinstellungen und Tablespaces haben. Eine Partition kann selbst wiederum partitioniert werden.

Es gibt drei Verfahren. Bereichspartitionierung gibt jeder Partition einen Wertebereich, meist einen Datumsbereich. Die Untergrenze ist eingeschlossen, die Obergrenze nicht. Bei den Bereichen 1–10 und 10–20 gehört der Wert 10 deshalb in den zweiten. Listenpartitionierung nennt die Werte jeder Partition, etwa eine Menge von Ländercodes. Hashpartitionierung weist jeder Partition einen Divisor und einen Rest zu. Eine Zeile landet dort, wo der Hash ihres Schlüssels diesen Rest ergibt. Das verteilt die Zeilen gleichmäßig, wenn es keinen natürlichen Bereich für die Aufteilung gibt. Eine Standardpartition kann Werte auffangen, die in keine andere passen, allerdings nur bei Bereichs- und Listenpartitionierung.

Partitionierung findet innerhalb eines Datenbankservers statt. Daten auf getrennte Server mit jeweils einer eigenen Kopie des Schemas zu verteilen heißt Sharding und ist ein wesentlich größerer Schritt. SQL Server und BigQuery haben eigene Formen der Partitionierung mit eigener Syntax. Die folgenden Überlegungen lassen sich auch dann übertragen, wenn die Befehle anders lauten.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Tabellenpartitionierung?“ 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

Die meisten großen Tabellen wachsen an einem Ende. Bestellungen, Messwerte, Logs und Ereignisse kommen täglich hinzu, bleiben Monate oder Jahre erhalten, und die meisten Abfragen interessieren sich für den neueren Teil. Ohne Partitionierung arbeitet jede Abfrage gegen eine Tabelle und einen Satz von Indizes, die immer größer werden.

Passt der Partitionsschlüssel zu den Abfragen, kann der Planer beweisen, dass die meisten Partitionen keine passende Zeile enthalten können, und sie vollständig aus dem Plan weglassen. PostgreSQL nennt das Partition Pruning. Ein Bericht über die letzte Woche liest nur die Partition der letzten Woche. Die Dokumentation nennt einen weniger offensichtlichen Vorteil: Weil die Partitionen die Aufgabe der oberen Ebenen eines Indexbaums übernehmen, passen die am häufigsten verwendeten Teile der Indizes mit höherer Wahrscheinlichkeit in den Arbeitsspeicher.

Am stärksten ändert sich das Entfernen alter Daten. Die Zeilen eines Monats aus einer einzelnen Tabelle zu löschen ist ein langes `DELETE`, das tote Zeilen für die spätere Bereinigung durch `VACUUM` hinterlässt. Mit einer Partition pro Monat lässt sich dieselbe Aufbewahrungsregel durch ein einzelnes `DROP TABLE` oder `DETACH PARTITION` umsetzen. Laut Dokumentation ist das wesentlich schneller als eine Massenoperation und vermeidet den Aufwand von `VACUUM` vollständig. Eine

abgetrennte Partition bleibt eine gewöhnliche Tabelle. Sie kann daher vor dem Löschen gesichert oder zusammengefasst werden. Selten gelesene Partitionen können auf günstigerem, langsamerem Speicher liegen.

Der Gewinn hat einen Preis, und die Dokumentation sagt klar, wann er sich lohnt. Jede Partition macht die Planung aufwendiger. Normalerweise lohnt sie sich erst, wenn die Tabelle andernfalls sehr groß wäre. Als Faustregel sollte die Tabelle größer als der physische Arbeitsspeicher des Datenbankservers sein. Eines ändert Partitionierung außerdem nicht: Eine einzelne Zeile, um die viele Transaktionen konkurrieren, bleibt eine einzelne Zeile in einer Partition hinter einer Sperre.

Wie es in der Praxis aussieht

Der einfachste Fall ist eine Tabelle `orders`, die anhand von `created_at` nach Monaten aufgeteilt ist. Eine Notiz über Datenbankskalierung vor dem Sharding widmet einen Abschnitt der Partitionierung der Tabellen um eine umkämpfte Zeile herum. Der Primärschlüssel ist `(id, created_at)`, weil der Schlüssel die Partitionsspalte enthalten muss. Eine Abfrage mit Datumsbereich liest nur die darin liegenden Partitionen. Die Indizes dieses Monats bleiben klein genug für den Arbeitsspeicher, und die Aufbewahrungsregel lässt sich mit einem einzelnen `DETACH PARTITION` statt einem `DELETE` von Millionen Zeilen umsetzen. Die Grenze ist ebenso klar: Partitionierung beschleunigt die umkämpfte Zeile nicht, und eine Abfrage ohne Angabe des Schlüssels liest jede Partition.

Ein PostgreSQL-Performanceprojekt für eine Anwendung mit Benutzerkonten, Mandanten, geografischen Daten und täglichem Stromverbrauch teilte die Stromtabelle nach Zeitbereichen auf, in Tage und Monate. Für die geografischen Daten kam Hashpartitionierung zum Einsatz. Dazu kamen GiST- und zusammengesetzte Indizes sowie umgeschriebene Abfragen. Die gemeinsamen Änderungen verkürzten die Abfrageantwortzeiten um den Faktor 10 und senkten den Ressourcenverbrauch um 40 %.

In wesentlich größerem Maßstab war Partitionierung eine von mehreren Techniken für mindestens 100 Datenbanken in PostgreSQL, SQL Server und BigQuery, wobei ein großer Teil der Daten aus Geodaten und Zeitreihen bestand. Zeitreihenpartitionierung gehörte in PostgreSQL und SQL Server zum Schemaentwurf. BigQuery verwendete zeitpartitionierte und geclusterte Tabellen. Zusammen mit räumlichen Indizes, materialisierten Sichten und Caching sanken die Antwortzeiten für Abfragen dieser Daten um 40–60 %.

Wo es schiefgeht

Partitionierung verlagert einen Teil der Arbeit von der Abfrage in den Entwurf. Die meisten Probleme entstehen durch eine Entwurfsentscheidung, bei der die Abfragen nicht berücksichtigt wurden.

Am ersten Tag eines neuen Monats scheitern Einfügungen. Keine Partition deckt den neuen Bereich ab. PostgreSQL lehnt eine Einfügung in die übergeordnete Tabelle ab, wenn sie keiner bestehenden Partition zugeordnet werden kann. Erzeugen Sie Partitionen im Voraus über einen geplanten Job, der wie in der Dokumentation vorgeschlagen das DDL generiert. Lösen Sie einen Alarm aus, wenn die neueste Partition ihrem Ende nahekommt:

```
CREATE TABLE orders_2026_11
  PARTITION OF orders
  FOR VALUES FROM ('2026-11-01')
              TO ('2026-12-01');
```

Eine Standardpartition verhindert diese Fehler. Dann landen die Zeilen jedes nicht abgedeckten Monats jedoch gemeinsam in dieser einen Tabelle. Das spätere Anhängen einer neuen Partition durchsucht die Standardpartition unter einer `ACCESS EXCLUSIVE` -Sperre, sofern dort nicht bereits eine `CHECK` -Bedingung den neuen Bereich ausschließt. Behandeln Sie eine nicht leere Standardpartition als Alarm.

Nach der Partitionierung sind Abfragen langsamer als vorher. Die häufigen Abfragen filtern nicht nach dem Partitionsschlüssel und lesen daher jeweils jede Partition samt Indizes. Eine zweite Ursache ist die Zahl der Partitionen: Laut Dokumentation kommt der Planer mit bis zu einigen Tausend gut zurecht, sofern typische Abfragen alle bis auf wenige ausschließen. Darüber hinaus wachsen Planungszeit und Speicherbedarf. Wählen Sie den Schlüssel anhand der bereits von der Anwendung verwendeten `WHERE` -Klauseln. Prüfen Sie mit `EXPLAIN` , dass nur die erwarteten Partitionen erscheinen, und halten Sie ihre Zahl angemessen. Eine Partition pro Kunde ist eine typische Falle, da die Zahl mit dem Geschäft wächst. Stattdessen empfiehlt die Dokumentation Hashpartitionierung mit einer vernünftigen Zahl von Partitionen.

Ein Primärschlüssel oder eine Eindeutigkeitsbedingung lässt sich nicht anlegen. Auf einer partitionierten Tabelle muss eine Eindeutigkeitsbedingung oder ein Primärschlüssel jede Partitionsschlüsselspalte enthalten. Der Partitionsschlüssel darf weder Ausdrücke noch Funktionsaufrufe enthalten. Der Index jeder Partition kann Eindeutigkeit nur innerhalb dieser Partition durchsetzen. Ergänzen Sie die Schlüsselspalte in der Bedingung, wie es die Tabelle `orders` der Notiz mit `(id, created_at)` tut.

Ein Indexaufbau auf der übergeordneten Tabelle blockiert Schreibzugriffe lange. Ein gewöhnliches `CREATE INDEX` auf ihr sperrt Schreibzugriffe, bis jede Partition indiziert ist. Die Option `CONCURRENTLY` funktioniert auf einzelnen Partitionen statt auf der übergeordneten Tabelle. Legen Sie den Index auf der übergeordneten Tabelle mit `ON ONLY` an, wodurch er als ungültig markiert bleibt. Bauen Sie den Index jeder Partition mit `CONCURRENTLY` auf und hängen Sie ihn jeweils mit `ALTER INDEX ... ATTACH PARTITION` an. Sobald die Indizes aller Partitionen angehängt sind, wird der übergeordnete Index von selbst gültig.

Eine bestehende große Tabelle lässt sich nicht direkt umstellen. PostgreSQL verwandelt keine gewöhnliche Tabelle in eine partitionierte. Legen Sie die neue partitionierte Tabelle an und hängen Sie die alte als eine ihrer Partitionen an. Ergänzen Sie vorher eine zu den Partitionsgrenzen passende `CHECK` -Bedingung, damit beim Anhängen nicht die gesamte Tabelle unter einer `ACCESS EXCLUSIVE` -Sperre durchsucht wird.

Weiterführende Lektüre

- Tabellenpartitionierung — das PostgreSQL-Kapitel zu den drei Verfahren, Pruning, Wartung, Einschränkungen und bewährten Vorgehensweisen.
- CREATE TABLE — `PARTITION BY` , `PARTITION OF` , Partitionsgrenzen und Standardpartition.
- ALTER TABLE — `ATTACH PARTITION` und `DETACH PARTITION` mit den jeweiligen Sperren.
- CREATE INDEX — Indizes auf partitionierten Tabellen, `ON ONLY` und Aufbau mit `CONCURRENTLY` .

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 Tabellenpartitionierung?“ 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/partitioning/