Engineer

CTE

benannte Unterabfrage, die außen wie eine Tabelle lesbar ist

Was es ist

Eine CTE, kurz für Common Table Expression, ist eine Abfrage, die in einer WITH-Klausel am Anfang einer Anweisung einen Namen erhält. Der Rest der Anweisung kann ihre Zeilen dann lesen, als kämen sie aus einer Tabelle. Eine Anweisung kann mehrere CTEs definieren. Jede kann die zuvor benannten lesen, und die abschließende Abfrage am Ende führt sie zusammen. Das PostgreSQL-Handbuch beschreibt sie als „temporäre Tabellen, die nur für eine Abfrage existieren“. Das ist ein treffendes Bild, mit einer Einschränkung: Unter diesem Namen wird nichts gespeichert, und die nächste Anweisung kann nicht darauf zugreifen. Jede CTE kann ein SELECT sein oder in PostgreSQL auch ein INSERT, UPDATE, DELETE oder MERGE. Auch die Anweisung, zu der die WITH-Klausel gehört, kann jede dieser Arten haben.

Eine Frage wie „Welche Zähler haben an irgendeinem Tag der letzten 30 Tage mehr als das Dreifache ihrer eigenen üblichen Menge verbraucht?“ lässt sich natürlich in Schritte zerlegen:

```
WITH daily AS (
  SELECT meter_id, read_at::date AS day, sum(kwh) AS kwh
  FROM readings
  WHERE read_at >= current_date - 30
  GROUP BY meter_id, read_at::date
),
typical AS (
  SELECT meter_id, avg(kwh) AS avg_kwh
  FROM daily
  GROUP BY meter_id
)
SELECT d.meter_id, d.day, d.kwh, t.avg_kwh
FROM daily AS d
```

```
JOIN typical AS t USING (meter_id)
WHERE d.kwh > 3 * t.avg_kwh
ORDER BY d.day, d.meter_id;
```

Der erste Schritt summiert die Messwerte jedes Zählers pro Tag, der zweite bildet je Zähler den Durchschnitt dieser Summen. Der letzte vergleicht jeden Tag mit dem Durchschnitt seines eigenen Zählers. Dasselbe Ergebnis lässt sich mit verschachtelten Unterabfragen schreiben. Die Tagessummen stünden dann aber zweimal im Text, einmal innerhalb der anderen Berechnung, und der Leser müsste sich von innen nach außen durcharbeiten. Mit WITH liest sich die Logik von oben nach unten, und jeder Schritt lässt sich während der Prüfung einzeln ausführen.

Mit RECURSIVE kann eine CTE ihre eigene Ausgabe lesen. Genau das lässt sich in einfachem SQL auf keinem anderen Weg ausdrücken. Eine rekursive CTE besteht aus einer Startabfrage, dann UNION oder UNION ALL und dann einer Abfrage, die auf die CTE zurückverweist. PostgreSQL führt die Startabfrage aus, hält ihre Zeilen in einer Arbeitstabelle und führt die zweite Abfrage immer wieder auf den neuesten Zeilen aus, bis eine Runde keine mehr hinzufügt. So durchläuft eine Anweisung einen Baum unbekannter Tiefe, etwa die Kategorien unter einer Überschrift:

```
WITH RECURSIVE subtree AS (
  SELECT id, parent_id, name, 1 AS depth
  FROM categories
  WHERE id = 42
  UNION ALL
  SELECT c.id, c.parent_id, c.name, s.depth + 1
  FROM categories AS c
  JOIN subtree AS s ON c.parent_id = s.id
)
SELECT id, name, depth FROM subtree ORDER BY depth, name;
```

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine CTE?“ 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

Der erste Grund ist Lesbarkeit. Eine analytische Abfrage mit hundert Zeilen und vier Verschachtelungsebenen ist schwer zu prüfen und noch schwerer zu ändern. Dieselbe Logik als Kette benannter Schritte gibt einem Reviewer einen Satz pro Schritt, dem Autor einen Platz für einen Kommentar und jedem die Möglichkeit, einen Schritt allein auszuführen. So lässt sich prüfen, ob dessen Zeilen plausibel sind, bevor man dem nächsten vertraut.

Der zweite Grund ist die Auswertung der Schritte durch die Datenbank. Hier haben sich die Einzelheiten mit der Zeit geändert. PostgreSQL wertet eine WITH-Abfrage normalerweise einmal pro Ausführung der Anweisung aus, auch wenn die Anweisung mehrfach darauf verweist. Eine an zwei Stellen benötigte teure Berechnung läuft somit einmal. Diese Garantie kostet allerdings etwas: Der Planer kann einen Filter aus der äußeren Abfrage nicht in eine mehrfach referenzierte CTE verschieben, weil der Filter für eine Verwendung gelten soll und nicht für die anderen. Die CTE wird vollständig berechnet, und die Zeilen werden erst danach verworfen.

Vor PostgreSQL 12 galt das für jede CTE: Jede wurde vor dem Rest der Abfrage ausgewertet und nie mit ihm zusammengeführt. Version 12 änderte das Standardverhalten. Eine nicht rekursive CTE ohne Seiteneffekte, auf die nur einmal verwiesen wird, wird jetzt in die äußere Abfrage eingebettet. Der Planer optimiert beide Ebenen gemeinsam und kann den äußeren Filter bis zu einem Index auf der Basistabelle weiterreichen. MATERIALIZED erzwingt das alte Verhalten. NOT MATERIALIZED erzwingt die Einbettung auch bei einer mehrfach verwendeten CTE, mit dem Risiko, ihre Arbeit zweimal auszuführen. SQLite akzeptiert dieselben beiden Schlüsselwörter als unverbindliche Hinweise und erklärt, die Idee von PostgreSQL übernommen zu haben. Andere Datenbanksysteme entscheiden selbst. Im Ausführungsplan lässt sich das prüfen.

Der dritte Grund ist, dass eine CTE Daten ändern kann. Ein DELETE mit RETURNING in einer WITH-Klausel kann seine Ausgabe an ein INSERT in der Hauptanweisung übergeben. Damit werden Zeilen in einer einzigen Anweisung von einer Tabelle in eine andere verschoben, entweder vollständig oder gar nicht. Datenändernde CTEs laufen genau einmal und immer vollständig,

unabhängig davon, ob etwas ihre Ausgabe liest. Damit sind sie ein Werkzeug für mehrere gemeinsame Schreibvorgänge statt nur zur Beschreibung eines Ergebnisses.

Wie es in der Praxis aussieht

Hinter der Arbeit PostgreSQL-Leistung auf das 10-Fache gesteigert stand eine Datenbank für Nutzerkonten, Mandanten, Standorte und tägliche Strommesswerte, die mit wachsender Datenmenge langsamer wurde. Die Lösung bestand aus drei Teilen: gezielten Indizes und GiST-Indizes, zeitbasierter Partitionierung der Messwerttabelle und überarbeiteten Abfragen. Die Überarbeitung nutzte CTEs und materialisierte Sichten für häufig abgerufene Daten, um vollständige Tabellenscans zu vermeiden. EXPLAIN ANALYZE zeigte dabei, welche Joins und Unterabfragen sich umzustrukturieren lohnten. Die Verkürzung der Antwortzeiten auf ein Zehntel und die Verringerung des Ressourcenverbrauchs um 40 % ergaben sich aus allen drei Teilen gemeinsam, nicht aus CTEs allein. Die Kombination ist beachtenswert, weil beide unterschiedliche Aufgaben haben: Eine CTE benennt ein Zwischenergebnis für die Dauer einer Anweisung, während eine materialisierte Sicht ein Ergebnis speichert, das viele Anweisungen bis zur nächsten Aktualisierung lesen können.

Die andere Stärke von CTEs, ihre Lesbarkeit, bildete die Grundlage für die 500 Strom- und GIS-Datenanalyseberichte. Jeder Bericht bestand aus reinem SQL für PostgreSQL und BigQuery über Smart-Meter-Daten mit Koordinaten, Zähler-IDs und zeitgestempeltem Verbrauch. Die Skripte nutzten CTEs zusammen mit Fensterfunktionen, Unterabfragen und räumlichen Joins. Jeder Bericht wurde mit kommentiertem SQL ausgeliefert, damit Kollegen die Analyse reproduzieren und prüfen konnten, sowie mit Hinweisen auf häufige Datenqualitätsprobleme wie fehlende GPS-Koordinaten und beschädigte Zählerwerte. Schrittweise lesbares SQL lässt sich gut weitergeben: Die Berichte wurden abteilungsübergreifend zum Standardnachschlagewerk und halfen, die operative Planung um 10–15 % effizienter zu machen. Benannte Schritte machen solches SQL überhaupt erst gezielt kommentierbar. Über jedem Schritt kann stehen, was er entfernt oder voraussetzt.

Wo es schiefgeht

Ein Filter greift erst, wenn die ganze Arbeit erledigt ist. Eine zweimal referenzierte oder mit MATERIALIZED markierte CTE wird vollständig berechnet, bevor die äußere Abfrage sie filtert. Die dabei erzeugte Kopie hat keine Indizes. Eine Abfrage, die wenige Zeilen berühren sollte, liest eine ganze Tabelle. Lesen

Sie den Plan: Erscheint die CTE als eigener Schritt mit weit mehr Zeilen, als die Abfrage zurückgibt, versuchen Sie NOT MATERIALIZED oder formulieren Sie die Abfrage so um, dass sie nur einmal auf die CTE verweist. Behalten Sie die materialisierte Form dort bei, wo die CTE eine teure Funktion aufruft. So läuft diese einmal pro Zeile und nicht einmal pro Verweis.

Eine rekursive Abfrage läuft immer weiter. Daten mit einem Zyklus, etwa zwei Kategorien, die jeweils die andere als übergeordnet angeben, schicken eine rekursive CTE mit UNION ALL so lange im Kreis, wie der Server es zulässt. UNION hilft nur bei vollständig identischen Zeilen, was nach dem Hinzufügen einer Tiefenspalte selten zutrifft. Verfolgen Sie den Pfad besuchter Schlüssel in einem Array und stoppen Sie, sobald ein Schlüssel wiederkehrt, oder verwenden Sie die mit PostgreSQL 14 eingeführte CYCLE-Klausel. Ein LIMIT an der äußeren Abfrage funktioniert als Testtrick. Das Handbuch rät davon ab, sich im Produktivbetrieb darauf zu verlassen. SQL Server begrenzt die Rekursion standardmäßig auf 100 Ebenen und bricht die Anweisung darüber hinaus ab. Derselbe Fehler stoppt dort also, statt weiterzulaufen.

Änderungen bleiben für den Rest der Anweisung unsichtbar. Eine datenändernde CTE und die Hauptabfrage laufen gleichzeitig auf demselben Snapshot. Liest die Hauptabfrage direkt aus der Zieltabelle, sieht sie deshalb die Zeilen vor der Änderung. Der einzige Kanal zwischen beiden ist RETURNING. Noch problematischer sind zwei Teile derselben Anweisung, die dieselbe Zeile ändern: Nur eine Änderung wird wirksam, und welche, lässt sich schwer vorhersagen. Lesen Sie geänderte Zeilen aus der RETURNING-Ausgabe der CTE und lassen Sie jede Zeile pro Anweisung nur von einer Stelle ändern.

Ein Upgrade verschiebt den Plan. Für PostgreSQL 11 oder älter geschriebener Code verließ sich gelegentlich darauf, dass eine CTE zuerst ausgewertet wird, als bewusste Barriere gegen einen unerwünschten Plan. Nach einem Upgrade auf 12 oder neuer wird dieselbe CTE in die äußere Abfrage eingebettet. Der Planer wählt etwas Neues, meist Schnelleres, gelegentlich deutlich Langsameres. Vergleichen Sie die EXPLAIN-Ausgabe langsamer Anweisungen vor und nach dem Upgrade und ergänzen Sie MATERIALIZED, wo die Barriere beabsichtigt war.

Weiterführende Lektüre

- 7.8. WITH-Abfragen (Common Table Expressions) — das Handbuchkapitel zu CTEs: Rekursion, Zyklenerkennung, Materialisierung und datenändernde Anweisungen.

- E.23. Release 12 — die Version, die einmal verwendete CTEs erstmals in die äußere Abfrage einbettete.
- E.25. Release 14 — die Version, die die SQL-Standardklauseln SEARCH und CYCLE ergänzte.
- Die WITH-Klausel — die SQLite-Variante mit ihren Materialisierungshinweisen und deren Herkunft.
- WITH common_table_expression (Transact-SQL) — die CTE-Regeln von SQL Server einschließlich der Rekursionsgrenze MAXRECURSION.

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 CTE?“ 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/cte/