Engineer

Materialisierte Sicht

Sicht, die ihr Abfrageergebnis wie eine Tabelle speichert

Was es ist

Eine materialisierte Sicht ist eine Abfrage, deren Ergebnis die Datenbank aufbewahrt. Beim Lesen liefert sie deshalb früher berechnete Zeilen, statt die Abfrage erneut auszuführen. Im britischen Englisch heißt sie materialised view statt materialized view. Sie liegt zwischen zwei älteren Objekten. Eine gewöhnliche Sicht speichert nur ihre Abfrage. Jeder Lesezugriff führt sie auf den zugrunde liegenden Tabellen aus. Eine mit `CREATE TABLE AS` erzeugte Tabelle speichert ein Ergebnis, vergisst aber die Abfrage, die es erzeugt hat. Eine materialisierte Sicht behält beides: die gespeicherten Zeilen, die beim Lesen zurückgegeben werden, und die Abfrage, die zum Ersetzen dieser Zeilen erneut ausgeführt werden kann.

PostgreSQL führte materialisierte Sichten in Version 9.3 ein, die am 9. September 2013 erschien. Die Versionshinweise schreiben die Arbeit Kevin Grittner zu. Laut Dokumentation verwenden sie wie gewöhnliche Sichten das Regelsystem, halten das Ergebnis aber in einer tabellenähnlichen Form dauerhaft fest. Für den Abfrageparser ist eine materialisierte Sicht eine Relation wie jede Tabelle oder Sicht. Ein `SELECT` darauf liest die gespeicherten Zeilen deshalb direkt. Die gespeicherte Abfrage ist nur beim Füllen der Sicht wichtig. Die Sicht lässt sich nicht direkt aktualisieren. Um ihren Inhalt durch ein frisches Abfrageergebnis zu ersetzen, muss die gespeicherte Abfrage erneut laufen.

Zwei Befehle erledigen diese Arbeit. `CREATE MATERIALIZED VIEW` definiert die Sicht und führt die Abfrage sofort zum Füllen aus, sofern nicht `WITH NO DATA` angegeben wird. `REFRESH MATERIALIZED VIEW` führt die gespeicherte Abfrage erneut aus und ersetzt den gesamten Inhalt durch das neue Ergebnis. Version 9.4 vom 18. Dezember 2014 ergänzte `CONCURRENTLY`. Damit kann eine Aktualisierung laufen, ohne die lesenden Sitzungen zu blockieren. Beide Befehle sind PostgreSQL-Erweiterungen und nicht Teil des SQL-Standards. Andere Datenbanken bieten die Idee unter eigener Syntax und eigenen Regeln an.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine materialisierte Sicht?“ 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

Manche Abfragen sind teuer und werden oft gestellt: ein Dashboard, das die Bestellungen eines Jahres nach Tagen summiert, ein Bericht mit einem Join über sechs Tabellen oder eine nach Verkäufen sortierte Produktseite. Auf den Quelltabellen verursacht jede Anfrage erneut die Kosten für einen vollständigen Scan, einen Join und eine Aggregation, obwohl sich das Ergebnis zwischen Anfragen kaum ändert. Eine materialisierte Sicht trägt diese Kosten einmal pro Aktualisierung. Jeder spätere Lesezugriff scannt eine kleine, vorab berechnete Tabelle. Die PostgreSQL-Dokumentation sagt, das Lesen der gespeicherten Daten sei „oft viel schneller als der direkte Zugriff auf die zugrunde liegenden Tabellen oder der Zugriff über eine Sicht“.

Der Preis ist die eingeschränkte Aktualität der Daten. Zwischen Aktualisierungen zeigt die Sicht den Datenstand der letzten Aktualisierung. In PostgreSQL wird sie nur aktualisiert, wenn jemand oder ein zeitgesteuerter Auftrag dies anfordert. Viele Anwendungen können damit leben. Gestrige Verkäufe ändern sich heute nicht, und ein Monatsbericht braucht keine Bestellungen aus dieser Minute. Eine materialisierte Sicht erzwingt eine ausdrückliche Entscheidung: Wie alt darf die Antwort sein, und wer sorgt dafür, dass sie nicht älter wird?

Sie gibt auch Indizes einen Ort: Das gespeicherte Ergebnis kann eigene, auf die lesenden Abfragen zugeschnittene Indizes tragen. Die PostgreSQL-Dokumentation zeigt dafür einen zweiten Einsatz. Über einen Foreign Data Wrapper eingelesene Daten aus einem anderen System können in eine materialisierte Sicht mit Indizes kopiert werden, auch wenn der Wrapper selbst überhaupt keine Indizes unterstützt.

Für ein Team mit normalisiertem Schema ist dies oft der Ort für die flache Datenstruktur. Datenbanknormalisierung speichert jede Tatsache einmal und beantwortet die meisten Fragen über Joins. Eine materialisierte Sicht kann das zusammengeführte, summierte Ergebnis für Berichte halten. Die Quelltabellen behalten dabei ihre einzelne Kopie jeder Tatsache, und jeder Schreibzugriff geht weiterhin an eine Stelle.

Wie es in der Praxis aussieht

Nehmen wir die Bestelltabellen eines Shops: `orders` mit einer Zeile pro Bestellung und ihrem Bestelldatum sowie `order_lines` mit SKU, Menge und Stückpreis jeder Position. Ein Verkaufsdashboard braucht Stückzahlen und Umsatz je Produkt und Tag. Die Abfrage ist kurz, liest bei einem Bestellbestand mehrerer Jahre aber jede Position:

```
CREATE MATERIALIZED VIEW daily_sales AS
SELECT o.ordered_on                  AS day,
       l.sku,
       sum(l.quantity)               AS units,
       sum(l.quantity * l.unit_price) AS revenue
FROM   orders o
JOIN   order_lines l USING (order_id)
WHERE  o.ordered_on < CURRENT_DATE
GROUP  BY o.ordered_on, l.sku;
```

Die `WHERE` -Klausel lässt den heutigen Tag mit seinen noch veränderlichen Zahlen aus. Das Dashboard zeigt deshalb nur vollständige Tage. Das Beispiel in PostgreSQLs eigener Dokumentation behandelt einen Tag noch nicht abgeschlossener Verkäufe genauso. Der Befehl führt die Abfrage einmal aus und speichert das Ergebnis. Danach kommt ein Index:

```
CREATE UNIQUE INDEX daily_sales_sku_day ON daily_sales (sku, day);
```

Der Index erfüllt zwei Aufgaben. Er lässt das Dashboard die Tage eines Produkts finden, ohne die ganze Sicht zu lesen, und ermöglicht später eine Aktualisierung bei laufenden Lesezugriffen. Die Abfrage gruppiert nach Tag und SKU. Jedes Paar erscheint deshalb einmal, und der Index kann eindeutig sein. Ab jetzt liest das Dashboard die Sicht wie eine Tabelle:

```
SELECT day, units, revenue
FROM   daily_sales
WHERE  sku = 'TEA-250'
ORDER  BY day;
```

Jede Nacht kurz nach Mitternacht bringt ein zeitgesteuerter Auftrag sie auf den aktuellen Stand:

```
REFRESH MATERIALIZED VIEW CONCURRENTLY daily_sales;
```

Ohne `CONCURRENTLY` braucht eine Aktualisierung vieler Zeilen tendenziell weniger Ressourcen und wird früher fertig. Sie könnte aber die Lesezugriffe des Dashboards bis zum Abschluss blockieren. Mit der Option können Leser während der Aktualisierung weiterlesen. Laut Dokumentation kann sie sogar schneller sein, wenn sich nur wenige Zeilen ändern, wie hier, wo jede Nacht ein neuer Tag hinzukommt. Die Option hat zwei Bedingungen. Die Sicht muss bereits Daten enthalten und mindestens einen eindeutigen Index besitzen, der ausschließlich gewöhnliche Spaltennamen verwendet und alle Zeilen abdeckt. Dieser Index darf also weder ein Ausdrucksindex sein noch eine `WHERE` -Klausel enthalten. Der obige Index erfüllt die zweite Bedingung, das Erstellen der Sicht mit Daten erfüllte die erste.

Andere Datenbanken ziehen die Grenze anders. BigQuery hält materialisierte Sichten im Hintergrund aktuell: Änderungen an Basistabellen werden ohne Aufforderung inkrementell ergänzt. Eine Abfrage der Sicht bezieht jüngste Änderungen mit ein, damit das Ergebnis frisch bleibt. Smart Tuning kann eine auf die Basistabellen gerichtete Abfrage stattdessen zur Sicht umleiten. Das gilt für inkrementelle Sichten, die nur eine begrenzte Auswahl von SQL zulassen. Eine mit `allow_non_incremental_definition = true` definierte Sicht akzeptiert die meisten Abfragen. Jede Aktualisierung führt aber die volle Abfrage aus, und ihre Ergebnisse sind nicht immer aktuell.

SQL Server erreicht ein ähnliches Ergebnis mit indizierten Sichten. Die Sicht muss mit `WITH SCHEMABINDING` erstellt werden und eine deterministische Definition haben. Wird ihr ein gruppierter Index hinzugefügt, der Eindeutigkeit erzwingt, speichert sie ihr Ergebnis in derselben Speicheranordnung wie eine indizierte Tabelle. Der Optimierer kann es auch für eine Abfrage nutzen, die die Sicht nicht nennt. SQL Server aktualisiert die indizierte Sicht bei jedem Einfügen, Ändern

und Löschen in den Tabellen, von denen die Sicht abhängt. Es muss also keine Aktualisierung geplant werden. Die Kosten wandern zu den Schreibzugriffen, deren Leistung laut Microsofts Dokumentation erheblich sinken kann.

Dieselbe Abwägung zeigt sich auch bei der Arbeit an Produktivsystemen. In einer früheren Rolle entwickelte die Gründerin PL/pgSQL-ETL- und -ELT-Pipelines, die partitionierte Tabellen und materialisierte Sichten nutzten, um das Lesen großer Datenbestände zu beschleunigen. Hinzu kamen Primär- und Fremdschlüsselbedingungen für referenzielle Integrität sowie eindeutige und zusammengesetzte Indizes für die Joins.

Wo es schiefgeht

Veraltete Zahlen sehen aktuell aus. Ein seit einer Woche nicht mehr laufender Aktualisierungsauftrag lässt die Sicht weiter jede Abfrage beantworten, schnell und falsch. Nichts im Ergebnis verrät sein Alter. Lassen Sie den Auftrag den Zeitpunkt jeder erfolgreichen Aktualisierung an einer vom Dashboard anzeigbaren Stelle festhalten. Alarmieren Sie, wenn die letzte Aktualisierung länger zurückliegt, als mit den Lesern vereinbart.

Eine Aktualisierung blockiert die Leser. Ein gewöhnliches `REFRESH MATERIALIZED VIEW` einer großen Sicht könnte jede lesende Sitzung bis zum Abschluss blockieren. Ergänzen Sie einen eindeutigen Index und verwenden Sie `CONCURRENTLY` oder planen Sie gewöhnliche Aktualisierungen zu Zeiten ohne Leser. Wenn sich bei jedem Lauf die meisten Zeilen ändern, kann die gewöhnliche Aktualisierung weiterhin günstiger sein.

`CONCURRENTLY` **wird verweigert.** Die Option benötigt einen eindeutigen Index auf gewöhnlichen Spalten, der alle Zeilen abdeckt. Ein eindeutiger Index auf `lower(email)` oder ein mit einer `WHERE` -Klausel beschränkter genügt nicht. Die Sicht muss außerdem bereits Daten enthalten. Die Option scheitert deshalb bei einer mit `WITH NO DATA` erstellten Sicht, und beide Optionen können nicht kombiniert werden. Fügen Sie einen passenden Index hinzu und füllen Sie die Sicht einmal mit einer gewöhnlichen Aktualisierung.

Eine nicht lesbare Sicht. `WITH NO DATA` erstellt die Sicht, ohne ihre Abfrage auszuführen, und markiert sie als nicht scanbar. Jede Abfrage darauf scheitert bis zur ersten Aktualisierung. Das kann in einer Migration richtig sein, die nicht auf eine lange Abfrage warten soll. Die erste Aktualisierung muss dann aber geplant werden und darf nicht in Vergessenheit geraten.

Eine geänderte Definition kam nie an. `CREATE MATERIALIZED VIEW IF NOT EXISTS` in einer Migration tut außer einem Hinweis nichts, wenn bereits eine Sicht dieses Namens existiert. Die Dokumentation garantiert nicht, dass die bestehende Sicht der neuen Definition ähnelt. Eine Migration, die so die Abfrage ändert, lässt die alte stehen. Löschen Sie die Sicht mit `DROP MATERIALIZED VIEW` und erstellen Sie sie in derselben Migration neu.

Aktualisierungsaufträge überlappen sich. Eine materialisierte Sicht kann immer nur eine laufende Aktualisierung haben, selbst mit `CONCURRENTLY`. Ein engerer Zeitplan als die Aktualisierungsdauer lässt die Sicht deshalb nur weiter zurückfallen. Messen Sie die Dauer einer Aktualisierung und planen Sie danach.

Die Reihenfolge der Zeilen wird für verlässlich gehalten. Hat die definierende Abfrage ein `ORDER BY`, erscheint der erste Inhalt in dieser Reihenfolge. Eine Aktualisierung verspricht aber nicht, sie zu erhalten. Setzen Sie `ORDER BY` in die Abfragen, die die Sicht lesen.

Funktionen verlieren ihre Tabellen. Während die definierende Abfrage beim Erstellen oder Aktualisieren läuft, setzt PostgreSQL `search_path` vorübergehend auf `pg_catalog, pg_temp` und beschränkt die erlaubten Aktionen: Eine Funktion, die temporäre Tabellen erstellt, scheitert. Eine Funktion, die ihre Tabellen über unqualifizierte Namen findet, kann jeden von Hand ausgeführten Test bestehen und nur bei der Aktualisierung fehlschlagen. Qualifizieren Sie die Tabellennamen in jeder von der Sicht aufgerufenen Funktion mit dem Schema.

Die falsche Rolle führt den Auftrag aus. Aktualisieren erfordert das Recht `MAINTAIN` auf der materialisierten Sicht. Ein Auftrag unter einer Berichtsrolle, die die Sicht lesen, aber nicht pflegen darf, scheitert jede Nacht. Geben Sie der aktualisierenden Rolle dieses Recht.

Eine gespeicherte Kopie, wo ein Index reichen würde. Ist eine Abfrage wegen eines fehlenden Index langsam, versteckt eine materialisierte Sicht das Problem und bringt zusätzlich das Problem veralteter Daten mit sich. Lesen Sie zuerst den Abfrageplan. Braucht eine Abfrage nur einen Namen, genügt eine gewöhnliche Sicht, die immer aktuell ist. Behalten Sie materialisierte Sichten für teuer zu berechnende Ergebnisse, die gefahrlos etwas älter gelesen werden können.

Weiterführende Lektüre

- Materialisierte Sichten — PostgreSQLs Kapitel zum Verhältnis zwischen materialisierten Sichten, Sichten und Tabellen mit einer nachts aktualisierten Verkaufsübersicht und einem Foreign-Data-Wrapper-Beispiel.
- CREATE MATERIALIZED VIEW — die Befehlsreferenz einschließlich `WITH NO DATA`, `IF NOT EXISTS` und des beschränkten `search_path` während der Abfrage.
- REFRESH MATERIALIZED VIEW — der Aktualisierungsbefehl, die Bedingungen für `CONCURRENTLY` und das benötigte Recht `MAINTAIN`.
- Versionshinweise zu PostgreSQL 9.3 — die Veröffentlichung im September 2013, die materialisierte Sichten ergänzte.
- Einführung in materialisierte Sichten — BigQuerys inkrementelle und nicht inkrementelle materialisierte Sichten, automatische Aktualisierung und Smart Tuning.
- Indizierte Sichten erstellen — SQL Servers indizierte Sichten, ihre Anforderungen und ihre Kosten für Schreibzugriffe.

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 materialisierte Sicht?“ 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/materialized-view/