Engineer

CTE

названий підзапит, який читають як таблицю

Що це таке

CTE, скорочення від common table expression («загальний табличний вираз»), — це запит, якому дають ім’я в конструкції WITH на початку інструкції, щоб решта інструкції читала його рядки так, ніби вони надходять із таблиці. Одна інструкція може визначити кілька CTE, кожна з них бачить ті, що названі раніше, а останній запит унизу складає їх докупи. Посібник PostgreSQL описує їх як тимчасові таблиці, що існують лише для одного запиту. Образ точний з одним застереженням: під цим ім’ям нічого не зберігається, і наступна інструкція його вже не побачить. CTE може бути SELECT, а в PostgreSQL також INSERT, UPDATE, DELETE або MERGE, і інструкція, до якої прикріплено WITH, теж може бути будь-якою з них.

Питання на кшталт «у яких лічильників хоча б за один з останніх 30 днів споживання більш ніж утричі перевищило звичайний для них рівень» природно розпадається на кроки:

```
WITH per_day AS (
  SELECT meter_id, taken_at::date AS day, sum(kwh) AS kwh
  FROM meter_readings
  WHERE taken_at >= current_date - 30
  GROUP BY meter_id, taken_at::date
),
baseline AS (
  SELECT meter_id, avg(kwh) AS avg_kwh
  FROM per_day
  GROUP BY meter_id
)
SELECT p.meter_id, p.day, p.kwh, b.avg_kwh
FROM per_day AS p
JOIN baseline AS b USING (meter_id)
WHERE p.kwh > 3 * b.avg_kwh
ORDER BY p.day, p.meter_id;
```

Перший крок підсумовує показники кожного лічильника за день, другий бере середнє цих денних сум для кожного лічильника, а останній порівнює кожен день із середнім свого лічильника. Ту саму відповідь можна записати вкладеними підзапитами, але тоді денні суми двічі з’являться в тексті, одна всередині іншої, і читачеві доведеться розбирати запит зсередини назовні. З WITH логіка читається згори донизу, а кожен крок можна запустити окремо, поки його перевіряють.

Слово RECURSIVE дозволяє CTE читати власний результат, і саме цього звичайний SQL інакше виразити не може. Рекурсивна CTE складається з початкового запиту, далі UNION або UNION ALL, а потім запиту, що посилається на саму CTE. PostgreSQL виконує початковий запит, кладе його рядки в робочу таблицю і знову й знову виконує другий запит над найновішими рядками, доки черговий прохід не додасть жодного. Так інструкція обходить дерево невідомої глибини, наприклад усі підрозділи під одним відділом:

```
WITH RECURSIVE unit_tree AS (
  SELECT id, parent_id, title, 1 AS depth
  FROM org_units
  WHERE id = 7
  UNION ALL
  SELECT u.id, u.parent_id, u.title, t.depth + 1
  FROM org_units AS u
  JOIN unit_tree AS t ON u.parent_id = t.id
)
SELECT id, title, depth FROM unit_tree ORDER BY depth, title;
```

Застосуйте поняття до рішення

Застосуйте «Що таке CTE?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Пошук запитів і відсутніх індексів у системі, що рік за роком важчає, та виправлення тих, що справді мають значення.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. 30 днів даних про електроенергію за 180 мс — 180 ms на агрегацію місяця показань лічильників

30 днів даних про електроенергію за 180 мс Зв'язатися з нами →

Чому це важливо

Перша причина — читабельність. Аналітичний запит на сотню рядків із чотирма рівнями вкладеності важко переглядати і ще важче змінювати. Та сама логіка у вигляді ланцюжка названих кроків дає рецензентові одне речення на крок, дає авторові місце для коментаря і дозволяє будь-кому запустити один крок окремо, щоб побачити, чи правильні в нього рядки, перш ніж довіряти наступному.

Друга причина — те, як база обчислює кроки, і тут деталі з часом змінилися. PostgreSQL зазвичай обчислює запит із WITH один раз за виконання інструкції, навіть коли інструкція звертається до нього кілька разів, тож дороге обчислення, потрібне у двох місцях, виконується один раз. Ціна цієї гарантії в тому, що планувальник не може протиснути фільтр із зовнішнього запиту всередину CTE, яку використовують кілька разів, адже фільтр має стосуватися лише одного з використань. CTE обчислюється повністю, а зайві рядки відкидаються потім.

До PostgreSQL 12 так було з усіма CTE: кожну обчислювали раніше за решту запиту і ніколи не вбудовували в нього. Версія 12 змінила типову поведінку. CTE, яка не рекурсивна, не має побічних ефектів і використовується лише один раз, тепер вбудовується в зовнішній запит, тож планувальник оптимізує обидва рівні разом і може скористатися індексом на фільтрі, якого раніше не бачив. MATERIALIZED примусово повертає стару поведінку, а NOT MATERIALIZED примусово вбудовує навіть CTE, яку використовують кілька разів, ризикуючи виконати її роботу двічі. SQLite приймає ці самі два ключові слова як необов’язкові підказки й прямо пише, що запозичила ідею в PostgreSQL. Інші рушії вирішують самі, і перевіряти це слід у плані.

Третя причина — CTE може змінювати дані. DELETE з RETURNING усередині WITH може передати рядки в INSERT основної інструкції, і тоді рядки переходять з однієї таблиці в іншу за одну інструкцію, яка або виконується повністю, або не виконується зовсім. CTE, що змінюють дані, виконуються рівно один раз і завжди до кінця, незалежно від того, чи читає хтось їхній результат. Це робить їх інструментом для кількох записів разом, а не способом описати результат.

Як це виглядає на практиці

Робота, описана в Прискорила продуктивність PostgreSQL у 10 разів, стосувалася бази з обліковими записами користувачів, орендарями, географічними даними й щоденними показниками споживання електроенергії, яка сповільнювалася з ростом обсягів. Рішення мало три частини: цільові індекси, зокрема GiST для геопросторових даних, партиціонування таблиці електроенергетичних даних за днями або місяцями та переписування запитів. Складні запити переписали, щоб уникнути повного сканування таблиць, з використанням CTE і матеріалізованих представлень для часто потрібних наборів даних, а `EXPLAIN ANALYZE` вказав на неефективні з'єднання й підзапити. Десятикратне покращення часу відповіді й зниження використання ресурсів на 40% дали всі три частини разом, а не самі CTE. Поєднання варте уваги, бо ці двоє виконують різну роботу: CTE дає ім'я проміжному результату на час однієї інструкції, а матеріалізоване представлення зберігає результат, який читають багато інструкцій, доки його не оновлять.

Інший бік CTE, читабельність, лежав в основі 500 звітів з аналізу даних електроенергетики та GIS. Кожен звіт написано чистим SQL безпосередньо в PostgreSQL і BigQuery над даними розумних лічильників із геолокаційними координатами, ідентифікаторами лічильників і споживанням із часовими мітками, а скрипти поєднували CTE з віконними функціями, підзапитами й геопросторовими з'єднаннями. До кожного звіту додавався анотований SQL-код, щоб колеги могли відтворити й перевірити аналіз, а також нотатки про типові проблеми якості даних, як-от відсутні GPS-координати чи пошкоджені значення лічильників. Звіти стали стандартним довідковим матеріалом у різних підрозділах і сприяли покращенню ефективності операційного планування на 10–15%. Саме названі кроки роблять такий SQL придатним до анотування: над кожним кроком можна поставити примітку про те, що він відкидає або що припускає.

Де виникають проблеми

Фільтр, що спрацьовує, коли робота вже зроблена. CTE, до якої звертаються двічі або яку позначено MATERIALIZED, обчислюється повністю ще до того, як зовнішній запит її відфільтрує, а копія, яку вона створює, не має індексів. Запит, якому вистачило б кількох рядків, читає всю таблицю. Читайте план: якщо CTE стоїть окремим кроком і кількість рядків у ньому значно більша за ту, яку повертає запит, спробуйте NOT MATERIALIZED або

перепишіть запит так, щоб звернення до CTE було одне. Залишайте матеріалізовану форму там, де CTE викликає дорогу функцію, щоб та виконувалася раз на рядок, а не раз на кожне звернення.

Рекурсивний запит, що не зупиняється. Дані з циклом, наприклад два підрозділи, кожен із яких записаний батьківським для іншого, женуть рекурсивну CTE з UNION ALL по колу, доки це дозволяє сервер. UNION допомагає лише тоді, коли повторюються цілі рядки, а з колонкою глибини таке трапляється рідко. Зберігайте шлях відвіданих ключів у масиві й зупиняйтеся, коли ключ з'являється вдруге, або користуйтеся конструкцією CYCLE, яку додав PostgreSQL 14. LIMIT у зовнішньому запиті годиться як прийом для тестування, а посібник не радить покладатися на нього в робочому середовищі. SQL Server типово обмежує рекурсію 100 рівнями й скасовує інструкцію далі, тож та сама помилка зупиняється, замість крутитися без кінця.

Зміни, яких решта інструкції не бачить. CTE, що змінює дані, і основний запит виконуються одночасно на одному знімку даних, тож основний запит, який читає цільову таблицю напряму, бачить рядки такими, якими вони були до зміни. Єдиний канал між ними — RETURNING. Ще гірше, коли дві частини однієї інструкції змінюють той самий рядок: спрацьовує лише одна зміна, і яка саме, передбачити важко. Читайте змінені рядки з результату RETURNING у CTE і залишайте кожному рядку одного автора змін на інструкцію.

План, що змінився разом з оновленням. Код, написаний для PostgreSQL 11 або давніших версій, інколи розраховував на те, що CTE обчислюється першою, як свідомий бар'єр проти плану, якого автор не хотів. Після оновлення до 12 або новішої версії та сама CTE вбудовується в зовнішній запит, і планувальник обирає щось нове, зазвичай швидше, а часом значно повільніше. Порівняйте вивід EXPLAIN для повільних інструкцій до й після оновлення і додайте MATERIALIZED там, де бар'єр був навмисним.

Що почитати далі

- 7.8. WITH Queries (Common Table Expressions) — розділ посібника про CTE: рекурсія, виявлення циклів, матеріалізація та інструкції, що змінюють дані.
- E.23. Release 12 — випуск, з якого PostgreSQL почав вбудовувати одноразові CTE в зовнішній запит.

- E.25. Release 14 — випуск, що додав стандартні для SQL конструкції SEARCH і CYCLE.
- The WITH Clause — версія SQLite з її підказками щодо матеріалізації та поясненням, звідки вони взялися.
- WITH common_table_expression (Transact-SQL) — правила SQL Server для CTE, зокрема обмеження рекурсії MAXRECURSION.

Словник

Обирайте за фактичними умовами

Надайте репрезентативний набір даних, запитання до нього та поточне вузьке місце. Узгодимо вимірювання якості, пропускної здатності та відновлення.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке CTE?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/cte/