Engineer

CTE

en navngiven delforespørgsel, der læses som en tabel

Hvad det er

En CTE, kort for common table expression, er en forespørgsel, der får et navn i en WITH-klausul øverst i en sætning, så resten af sætningen kan læse dens rækker, som om de kom fra en tabel. En sætning kan definere flere CTE'er, hver af dem kan læse dem, der er navngivet før den, og den sidste forespørgsel nederst sætter dem sammen. PostgreSQL's manual beskriver dem som midlertidige tabeller, der kun findes for én forespørgsel. Billedet holder med ét forbehold: Der gemmes ikke noget under navnet, og den næste sætning kan ikke se det. En CTE kan være en SELECT, eller i PostgreSQL også en INSERT, UPDATE, DELETE eller MERGE, og sætningen, som WITH-klausulen hænger på, kan være en hvilken som helst af dem.

Et spørgsmål som »hvilke målere brugte på en enkelt dag i de seneste 30 dage mere end tre gange deres eget normale forbrug« deler sig naturligt i trin:

```
WITH dagligt AS (
  SELECT maaler_id, maalt_kl::date AS dag, sum(kwh) AS kwh
  FROM forbrug
  WHERE maalt_kl >= current_date - 30
  GROUP BY maaler_id, maalt_kl::date
),
normalt AS (
  SELECT maaler_id, avg(kwh) AS snit_kwh
  FROM dagligt
  GROUP BY maaler_id
)
SELECT d.maaler_id, d.dag, d.kwh, n.snit_kwh
FROM dagligt AS d
JOIN normalt AS n USING (maaler_id)
WHERE d.kwh > 3 * n.snit_kwh
ORDER BY d.dag, d.maaler_id;
```

Første trin lægger hver målers aflæsninger sammen pr. dag, andet trin tager gennemsnittet af de dagstal pr. måler, og det sidste sammenligner hver dag med sin egen målers gennemsnit. Samme svar kan skrives med indlejrede delforespørgsler, men så står dagstallene to gange i teksten, den ene inde i den anden, og læseren må arbejde sig indefra og ud. Med WITH læses logikken oppefra og ned, og hvert trin kan køres for sig, mens man kontrollerer det.

Tilføjer man RECURSIVE, kan en CTE læse sit eget output, og det er den del, almindelig SQL ellers ikke har nogen måde at udtrykke på. En rekursiv CTE har en startforespørgsel, derefter UNION eller UNION ALL og så en forespørgsel, der henviser tilbage til CTE'en. PostgreSQL kører startforespørgslen, lægger dens rækker i en arbejdstabel og bliver ved med at køre den anden forespørgsel mod de nyeste rækker, indtil en runde ikke tilføjer flere. Sådan går en sætning gennem et træ af ukendt dybde, for eksempel kategorierne under én overskrift:

```
WITH RECURSIVE undertrae AS (
  SELECT id, foraelder_id, navn, 1 AS dybde
  FROM kategorier
  WHERE id = 42
  UNION ALL
  SELECT k.id, k.foraelder_id, k.navn, u.dybde + 1
  FROM kategorier AS k
  JOIN undertrae AS u ON k.foraelder_id = u.id
)
SELECT id, navn, dybde FROM undertrae ORDER BY dybde, navn;
```

Brug begrebet i en beslutning

Brug »Hvad er en CTE?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. At finde de forespørgsler og manglende indeks bag et system, der er blevet tungere hvert år, og rette dem, der betyder noget.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. 30 dages eldata på 180 ms — 180 ms til at aggregere en måneds måleraflæsninger

30 dages eldata på 180 ms Kontakt os →

Hvorfor det betyder noget

Den første grund er læsbarhed. En analytisk forespørgsel på hundrede linjer, der er indlejret fire niveauer ned, er svær at gennemgå og sværere at ændre. Den samme logik som en kæde af navngivne trin giver den, der gennemgår koden,

én sætning pr. trin, giver forfatteren et sted at skrive en kommentar og lader enhver køre ét trin alene for at se, om rækkerne ser rigtige ud, før man stoler på det næste.

Den anden grund er, hvordan databasen evaluerer trinnene, og her har detaljerne ændret sig over tid. PostgreSQL evaluerer normalt en WITH-forespørgsel én gang pr. kørsel af sætningen, også når sætningen henviser til den flere gange, så en dyr beregning, der skal bruges to steder, kun kører én gang. Prisen for den garanti er, at planlæggeren ikke kan skubbe et filter fra den ydre forespørgsel ned i en CTE, der bruges flere steder, fordi filteret kun skal gælde én af brugene. CTE'en beregnes fuldt ud, og rækkerne smides væk bagefter.

Indtil PostgreSQL 12 gjaldt det for alle CTE'er: Hver blev evalueret før resten af forespørgslen og aldrig flettet ind i den. Version 12 ændrede standarden. En CTE, der ikke er rekursiv, ikke har sideeffekter og kun bruges én gang, foldes nu ind i den ydre forespørgsel, så planlæggeren optimerer de to niveauer samlet og kan bruge et indeks på det filter, den før ikke kunne se. MATERIALIZED tvinger den gamle opførsel frem, og NOT MATERIALIZED tvinger foldningen igennem, også for en CTE, der bruges flere gange, med risiko for at arbejdet kører to gange. SQLite accepterer de samme to nøgleord som uforpligtende hints og skriver selv, at idéen er lånt fra PostgreSQL. Andre motorer beslutter selv, og planen er stedet at kontrollere det.

Den tredje grund er, at en CTE kan ændre data. En DELETE med RETURNING i en WITH-klausul kan fodre en INSERT i hovedsætningen, og så flyttes rækker fra én tabel til en anden i én sætning, der enten sker helt eller slet ikke. Dataændrende CTE'er kører præcis én gang og altid til ende, uanset om noget læser deres output. Det gør dem til et redskab til at udføre flere skrivninger samlet, snarere end en måde at beskrive et resultat på.

Sådan ser det ud i praksis

Arbejdet bag PostgreSQL-ydeevnen, der blev accelereret 10 gange gjaldt en database med brugerkonti, tenants, lokationer og daglige eldata, der blev langsom, efterhånden som datamængderne voksede. Løsningen havde tre dele: målrettede indeks, herunder GiST-indeks til de geospatiale data, tidsbaseret partitionering af eltabellen pr. dag eller måned, og omskrivning af forespørgsler. Komplekse forespørgsler blev skrevet om for at undgå full table scans, med CTE'er og materialized views, mens EXPLAIN ANALYZE pegede på de ineffektive

joins. Svartider, der blev 10 gange bedre, og et ressourceforbrug, der faldt med 40 %, kom fra de tre dele tilsammen og ikke fra CTE'er alene. Kombinationen er værd at bemærke, fordi de to gør forskelligt arbejde: En CTE navngiver et mellemresultat, så længe én sætning varer, mens et materialized view gemmer et resultat, som mange sætninger kan læse, indtil det opdateres.

Den anden side af CTE'er, læsbarheden, bar de 500 rapporter om el- og GIS-dataanalyse. Hver rapport blev skrevet i ren SQL direkte i PostgreSQL og BigQuery, over data fra smart-målere med geolokationskoordinater, måler-ID'er og tidsstemplet forbrug, og scriptsene brugte CTE'er sammen med window functions, delforespørgsler og geospatiale joins. Hver rapport kom med annoteret SQL-kode, så kolleger kunne reproducere og revidere analysen, og med noter om almindelige datakvalitetsproblemer som manglende GPS-koordinater og beskadigede målerværdier. Rapporterne blev en standardreference på tværs af afdelinger og bidrog til en forbedring af driftsplanlægningens effektivitet på 10–15 %. Navngivne trin er det, der overhovedet gør den slags SQL mulig at annotere: En note kan stå over hvert trin og sige, hvad det fjerner, eller hvad det forudsætter.

Hvor det går galt

Et filter, der først rammer, når arbejdet er gjort. En CTE, der bruges to gange eller er markeret MATERIALIZED, beregnes fuldt ud, før den ydre forespørgsel filtrerer den, og den kopi, den laver, har ingen indeks. En forespørgsel, der burde røre få rækker, læser en hel tabel. Læs planen: Står CTE'en som sit eget trin med rækkeantal langt over det, forespørgslen returnerer, så prøv NOT MATERIALIZED, eller skriv forespørgslen om, så CTE'en bruges én gang. Behold den materialiserede form, hvor CTE'en kalder en dyr funktion, så funktionen kører én gang pr. række og ikke én gang pr. henvisning.

En rekursiv forespørgsel, der bliver ved. Data med en cyklus, for eksempel to kategorier, der hver står som den andens forælder, sender en rekursiv CTE med UNION ALL rundt i løkken, så længe serveren tillader det. UNION hjælper kun, når hele rækker gentager sig, og det holder sjældent, når der er en dybdekolonne med. Gem stien af besøgte nøgler i et array og stop, når en nøgle dukker op igen, eller brug CYCLE-klausulen, som PostgreSQL 14 tilføjede. En LIMIT på den ydre forespørgsel virker som et trick under test, og manualen fraråder at regne med det i produktion. SQL Server begrænser rekursion til 100 niveauer som standard og afbryder sætningen derefter, så den samme fejl stopper i stedet for at køre videre.

Ændringer, som resten af sætningen ikke ser. En dataændrende CTE og hovedforespørgslen kører samtidig på det samme snapshot, så en hovedforespørgsel, der læser måltabellen direkte, ser rækkerne, som de var før ændringen. Den eneste kanal mellem dem er RETURNING. Værre er det, når to dele af én sætning ændrer den samme række: Kun én af ændringerne slår igennem, og hvilken er svær at forudsige. Læs ændrede rækker fra CTE'ens RETURNING-output, og lad hver række have én skriver pr. sætning.

En plan, der flyttede sig med en opgradering. Kode skrevet til PostgreSQL 11 eller tidligere regnede nogle gange med, at en CTE blev evalueret først, som en bevidst barriere mod en plan, forfatteren ikke ville have. Efter en opgradering til 12 eller nyere foldes den samme CTE ind i den ydre forespørgsel, og planlæggeren vælger noget nyt, som regel hurtigere og en gang imellem meget langsommere. Sammenlign EXPLAIN-output for de langsomme sætninger før og efter opgraderingen, og tilføj MATERIALIZED, hvor barrieren var tilsigtet.

Læs mere

- 7.8. WITH Queries (Common Table Expressions) — manualens kapitel om CTE'er: rekursion, cyklusdetektion, materialisering og dataændrende sætninger.
- E.23. Release 12 — udgivelsen, der begyndte at folde CTE'er, der bruges én gang, ind i den ydre forespørgsel.
- E.25. Release 14 — udgivelsen, der tilføjede SQL-standardens SEARCH- og CYCLE-klausuler.
- The WITH Clause — SQLite's udgave, med dens materialiseringshints og hvor de er lånt fra.
- WITH common_table_expression (Transact-SQL) — SQL Servers regler for CTE'er, herunder MAXRECURSION-grænsen for rekursion.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring et repræsentativt datasæt, de spørgsmål det skal besvare, og den aktuelle flaskehals. Vi kan aftale målinger af kvalitet, gennemløb og gendannelse.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er en CTE?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/cte/