Engineer

CTE

en namngiven delfråga som en större fråga läser som tabell

Vad det är

En CTE är en fråga som får ett namn i en WITH-klausul överst i en SQL-sats, så att resten av satsen kan läsa dess rader som om de kom från en tabell. Förkortningen står för common table expression, på svenska ibland gemensamt tabelluttryck, och PostgreSQL:s manual kallar konstruktionen WITH-frågor. En sats kan definiera flera CTE:er, där var och en kan läsa dem som namngavs före den, och den sista frågan längst ned fogar ihop dem. Manualen beskriver dem som tillfälliga tabeller som bara finns för en enda fråga, och det är rätt bild med ett förbehåll: ingenting lagras under namnet, och nästa sats kan inte se det. Varje CTE kan vara en SELECT, eller i PostgreSQL en INSERT, UPDATE, DELETE eller MERGE, och den sats som WITH-klausulen hör till kan också vara vilken som helst av dem.

En fråga som ”vilka mätare förbrukade någon dag under de senaste 30 dagarna mer än tre gånger sin egen normala mängd” delar naturligt upp sig i steg:

```
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;
```

Det första steget summerar varje mätares avläsningar per dag, det andra räknar ut medelvärdet av de summorna per mätare, och det sista jämför varje dag med den egna mätarens medelvärde. Samma svar kan skrivas med nästlade underfrågor, men då skulle de dagliga summorna stå två gånger i texten, den ena inuti den andra, och en läsare skulle få arbeta sig inifrån och ut. Med WITH läses logiken uppifrån och ned, och varje steg kan köras för sig medan det kontrolleras.

Med RECURSIVE kan en CTE läsa sitt eget resultat, och det är den del som vanlig SQL inte har något annat sätt att uttrycka. En rekursiv CTE har en startfråga, sedan UNION eller UNION ALL, sedan en fråga som hänvisar tillbaka till CTE:n. PostgreSQL kör startfrågan, håller dess rader i en arbetstabell och kör sedan den andra frågan mot de senaste raderna, om och om igen, tills en omgång inte lägger till några. Så går en sats igenom ett träd av okänt djup, t.ex. kategorierna under en rubrik:

```
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;
```

Använd begreppet i ett beslut

Använd ”Vad är CTE?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Hittar databasfrågorna och de saknade indexen bakom ett system som har blivit tyngre för varje år, och åtgärdar dem som spelar roll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. 30 dagars eldata på 180 ms — 180 ms för att aggregera en månads mätaravläsningar

30 dagars eldata på 180 ms Hör av er →

Varför det spelar roll

Det första skälet är läsbarhet. En analytisk fråga på hundra rader som är nästlad fyra nivåer djupt är svår att granska och ännu svårare att ändra. Samma logik som en kedja av namngivna steg ger granskaren en mening per steg, ger författaren

en plats för en kommentar och låter vem som helst köra ett steg ensamt för att se om dess rader ser rimliga ut innan nästa steg får förtroende.

Det andra skälet är hur databasen utvärderar stegen, och här har detaljerna ändrats över tid. PostgreSQL utvärderar normalt en WITH-fråga en gång per körning av satsen, även när satsen hänvisar till den mer än en gång, så en dyr beräkning som behövs på två ställen görs en gång. Priset för den garantin är att planeraren inte kan flytta in ett filter från den yttre frågan i en CTE som det hänvisas till flera gånger, eftersom filtret bara ska gälla en av användningarna. CTE:n beräknas i sin helhet, och raderna kastas efteråt.

Fram till PostgreSQL 12 gällde det varje CTE: var och en utvärderades före resten av frågan och slogs aldrig ihop med den. Version 12 ändrade standardbeteendet. En CTE som inte är rekursiv, saknar sidoeffekter och bara används en gång bakas nu in i den yttre frågan, så att planeraren optimerar båda nivåerna tillsammans och kan föra ned det yttre filtret till ett index på bastabellen. MATERIALIZED tvingar fram det gamla beteendet, och NOT MATERIALIZED tvingar fram inbakning även för en CTE som används mer än en gång, med risk för att dess arbete görs två gånger. SQLite godtar samma två nyckelord som icke-bindande tips och säger att idén är lånad från PostgreSQL. Andra motorer avgör själva, och det är i planen man ser vad som blev av frågan.

Det tredje skälet är att en CTE kan ändra data. En DELETE med RETURNING i en WITH-klausul kan mata en INSERT i huvudsatsen, vilket flyttar rader från en tabell till en annan i en enda sats som antingen genomförs helt eller inte alls. Datamodifierande CTE:er körs exakt en gång och alltid till slut, oavsett om något läser deras resultat, och det gör dem till ett verktyg för att göra flera skrivningar tillsammans snarare än ett sätt att beskriva ett resultat.

Så ser det ut i praktiken

Arbetet bakom en PostgreSQL-databas som gjordes 10 gånger snabbare gällde en databas för användarkonton, tenants, platser och dagliga mätvärden för elförbrukning, som blev långsammare i takt med att datamängden växte. Åtgärden bestod av tre delar: riktade index och GiST-index, tidsbaserad partitionering av tabellen med eldata och omskrivna frågor. Omskrivningarna använde CTE:er och materialiserade vyer för data som användes ofta, för att undvika fullständiga tabellskanningar, och `EXPLAIN ANALYZE` pekade ut de joins och underfrågor som var värda att strukturera om. Den tiofaldiga förbättringen av svarstiderna och den 40 % lägre resursanvändningen kom från de tre delarna

tillsammans, inte från CTE:er ensamma. Kombinationen är värd att notera, eftersom de två gör olika jobb: en CTE ger ett mellanresultat ett namn under en enda sats, medan en materialiserad vy lagrar ett resultat som många satser kan läsa tills vyn uppdateras.

Den andra sidan av CTE:er, läsbarheten, bar de 500 rapporterna med analyser av el- och GIS-data. Varje rapport skrevs i ren SQL mot PostgreSQL och BigQuery, över data från smarta elmätare med koordinater, mätar-ID:n och tidsstämplad förbrukning, och skripten använde CTE:er tillsammans med fönsterfunktioner, underfrågor och geospatiala joins. Varje rapport levererades med kommenterad SQL så att kollegor kunde reproducera och granska analysen, plus anteckningar om vanliga datakvalitetsproblem som saknade GPS-koordinater och korrupta mätvärden. SQL som läses steg för steg är lätt att dela och använda vidare. Rapporterna blev ett standardunderlag på flera avdelningar och bidrog till att den operativa planeringen blev 10–15 % effektivare. Det är namngivna steg som över huvud taget gör sådan SQL möjlig att kommentera: en anteckning kan stå ovanför varje steg och säga vad steget tar bort eller förutsätter.

Var det går fel

Filtret tillämpas först när allt arbete är gjort. En CTE som det hänvisas till två gånger, eller som är märkt MATERIALIZED, beräknas i sin helhet innan den yttre frågan filtrerar den, och kopian den ger har inga index. En fråga som borde röra några få rader läser en hel tabell. Läs planen: när CTE:n syns som ett eget steg, med radantal långt över vad frågan returnerar, pröva NOT MATERIALIZED eller skriv om frågan så att det bara hänvisas till CTE:n en gång. Behåll den materialiserade formen där CTE:n anropar en dyr funktion, så att funktionen körs en gång per rad och inte en gång per hänvisning.

En rekursiv fråga fortsätter varv efter varv. Data med en cykel, t.ex. två kategorier som var och en står som den andras förälder, skickar en rekursiv CTE med UNION ALL runt slingan så länge servern tillåter det. UNION hjälper bara när hela rader upprepas, vilket sällan stämmer när en kolumn för djup har lagts till. Håll reda på vägen av besökta nycklar i en array och stanna när en nyckel dyker upp igen, eller använd CYCLE-klausulen som kom i PostgreSQL 14. En LIMIT på den yttre frågan fungerar som ett knep vid testning, och manualen avråder från att förlita sig på den i produktion. SQL Server begränsar rekursionen till 100 nivåer som standard och avbryter satsen bortom det, så samma bugg stannar i stället för att fortsätta.

Ändringar som resten av satsen inte ser. En datamodifierande CTE och huvudfrågan körs samtidigt mot samma ögonblicksbild, så en huvudfråga som läser måltabellen direkt ser raderna som de var före ändringen. Den enda kanalen mellan dem är RETURNING. Två delar av samma sats som ändrar samma rad är värre: bara en av ändringarna får effekt, och vilken är svårt att förutsäga. Läs ändrade rader ur CTE:ns RETURNING-utdata, och låt varje rad ha en enda skrivare per sats.

En plan som flyttade sig vid en uppgradering. Kod som skrevs för PostgreSQL 11 eller tidigare förlitade sig ibland på att en CTE utvärderades först, som en avsiktlig spärr mot en plan som författaren inte ville ha. Efter en uppgradering till 12 eller senare bakas samma CTE in i den yttre frågan och planeraren väljer något nytt, oftast snabbare och ibland mycket långsammare. Jämför utdata från EXPLAIN för de långsamma satserna före och efter uppgraderingen, och lägg till MATERIALIZED där spärren var avsiktlig.

Läs mer

- 7.8. WITH Queries (Common Table Expressions) — manualens kapitel om CTE:er: rekursion, upptäckt av cykler, materialisering och datamodifierande satser.
- E.23. Release 12 — versionen som började baka in CTE:er som används en gång i den yttre frågan.
- E.25. Release 14 — versionen som lade till klausulerna SEARCH och CYCLE ur SQL-standarden.
- The WITH Clause — SQLites variant, med dess tips om materialisering och varifrån de är lånade.
- WITH common_table_expression (Transact-SQL) — SQL Servers regler för CTE:er, bl.a. gränsen MAXRECURSION för rekursion.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med ett representativt dataset, frågorna det ska besvara och dagens flaskhals. Vi kan avtala mått på kvalitet, genomströmning och återställning.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är CTE?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/cte/