Engineer

CTE

en navngitt delspørring en ytre spørring leser som en tabell

Hva det er

En CTE, kort for *common table expression*, er en spørring som får et navn i en WITH-klausul øverst i en SQL-setning, slik at resten av setningen kan lese radene som om de kom fra en tabell. En setning kan definere flere CTE-er, og hver av dem kan lese dem som er navngitt før den, mens den endelige spørringen nederst setter dem sammen. Manualen til PostgreSQL beskriver dem som midlertidige tabeller som bare finnes for én spørring, og det er riktig bilde med ett forbehold: ingenting lagres under det navnet, og neste setning kan ikke se det. Hver CTE kan være en SELECT, eller i PostgreSQL en INSERT, UPDATE, DELETE eller MERGE, og setningen WITH-klausulen hører til, kan også være hvilken som helst av dem.

Et spørsmål som «hvilke målere brukte mer enn tre ganger sitt eget vanlige forbruk på en dag i løpet av de siste 30 dagene» deler seg naturlig i trinn:

```
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 første trinnet summerer avlesningene for hver måler per dag, det andre finner gjennomsnittet av de summene per måler, og det siste sammenligner hver dag med gjennomsnittet for sin egen måler. Det samme svaret kan skrives med nøstede delspørringer, men da står dagssummene to ganger i teksten, den ene inni den andre, og den som leser, må arbeide seg innenfra og ut. Med WITH leses logikken ovenfra og ned, og hvert trinn kan kjøres for seg mens det kontrolleres.

Med RECURSIVE kan en CTE lese sine egne resultater, og det er den delen vanlig SQL ellers ikke har noen måte å uttrykke på. En rekursiv CTE har en startspørring, deretter UNION eller UNION ALL, og deretter en spørring som viser tilbake til CTE-en. PostgreSQL kjører startspørringen, holder radene i en arbeidstabell og fortsetter å kjøre den andre spørringen mot de nyeste radene til en runde ikke legger til noen. Slik går en setning gjennom et tre av ukjent dybde, som kategoriene under én overskrift:

```
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;
```

Bruk begrepet i en beslutning

Bruk «Hva er CTE?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Finne spørringene og de manglende indeksene bak et system som har blitt tyngre for hvert år, og rette dem som betyr noe.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. 30 dagers strømdata på 180 ms — 180 ms for å aggregere en måned med måleravlesninger

30 dagers strømdata på 180 ms Ta kontakt →

Hvorfor det er viktig

Den første grunnen er lesbarhet. En analytisk spørring på hundre linjer som er nøstet fire nivåer dypt, er vanskelig å gjennomgå og enda vanskeligere å endre. Den samme logikken som en kjede av navngitte trinn gir den som gjennomgår,

ett trinn om gangen som kan forklares med én enkelt setning, gir forfatteren et sted å skrive en kommentar, og lar hvem som helst kjøre ett trinn alene for å se om radene ser riktige ut, før neste trinn får tillit.

Den andre grunnen er hvordan databasen evaluerer trinnene, og her har detaljene endret seg over tid. PostgreSQL evaluerer normalt en WITH-spørring én gang per kjøring av setningen, også når setningen viser til den flere ganger, så en kostbar beregning som trengs to steder, kjøres én gang. Prisen for den garantien er at planleggeren (*planner*) ikke kan skyve et filter fra den ytre spørringen inn i en CTE det vises til flere ganger, fordi filteret skal gjelde for én bruk og ikke for de andre. CTE-en regnes ut i sin helhet, og de overflødige radene kastes etterpå.

Fram til PostgreSQL 12 gjaldt det hver eneste CTE: hver av dem ble evaluert før resten av spørringen og aldri slått sammen med den. Versjon 12 endret standardoppførselen. En CTE som ikke er rekursiv, ikke har sideeffekter og bare vises til én gang, blir nå foldet inn i den ytre spørringen, så planleggeren optimaliserer de to nivåene sammen og kan ta det ytre filteret med ned til en indeks på basistabellen. MATERIALIZED tvinger fram den gamle oppførselen, og NOT MATERIALIZED tvinger fram foldingen også for en CTE som brukes mer enn én gang, med risiko for at arbeidet dens gjøres to ganger. SQLite godtar de samme to nøkkelordene som hint uten bindende kraft og sier selv at ideen er hentet fra PostgreSQL. Andre motorer avgjør selv, og planen er stedet å sjekke.

Den tredje grunnen er at en CTE kan endre data. En DELETE med RETURNING i en WITH-klausul kan levere radene sine til en INSERT i hovedsetningen, og det flytter rader fra én tabell til en annen i én enkelt setning som enten skjer helt eller ikke i det hele tatt. CTE-er som endrer data, kjøres nøyaktig én gang og alltid til ende, enten noe leser resultatet deres eller ikke, og det gjør dem til et verktøy for å utføre flere skrivinger samlet heller enn en måte å beskrive et resultat på.

Slik ser det ut i praksis

Arbeidet som økte PostgreSQL-ytelsen 10 ganger, gjaldt en database for brukerkontoer, leietakere, steder og daglige avlesninger av strømforbruk, som ble tregere etter hvert som datamengden vokste. Rettelsen kom i tre deler: målrettede indekser og GiST-indekser, tidsbasert partisjonering av tabellen med avlesninger, og omskrevne spørringer. Omskrivingene brukte CTE-er og materialiserte visninger for data som ofte ble brukt, for å unngå fulle tabellskanninger, og EXPLAIN ANALYZE pekte ut de joinene og delspørringene

som var verdt å strukturere om. Den tidoblede forbedringen i svartider og fallet på 40 % i ressursbruk kom fra de tre delene sammen, og ikke fra CTE-er alene. Paret er verdt å merke seg, fordi de to gjør ulike jobber: en CTE gir et mellomresultat et navn så lenge én setning varer, mens en materialisert visning lagrer et resultat som mange setninger kan lese helt til det oppdateres.

Den andre siden av CTE-er, lesbarheten, bar de 500 analyserapportene om strøm- og GIS-data. Hver rapport ble skrevet i ren SQL mot PostgreSQL og BigQuery, over data fra smartmålere med koordinater, måler-ID-er og tidsstemplet forbruk, og skriptene brukte CTE-er sammen med vindusfunksjoner, delspørringer og geospatiale joins. Hver rapport ble levert med kommentert SQL, slik at kolleger kunne reprodusere og etterprøve analysen, og med notater om vanlige datakvalitetsproblemer som manglende GPS-koordinater og korrupte målerverdier. SQL som leses trinn for trinn, er lett å dele og bruke videre: rapportene ble et standard oppslagsverk på tvers av avdelinger og bidro til at den operative planleggingen ble 10–15 % mer effektiv. Navngitte trinn er det som i det hele tatt gjør SQL av det slaget mulig å kommentere: en merknad kan stå over hvert trinn og si hva det fjerner eller forutsetter.

Hvor det går galt

Et filter som brukes etter at alt arbeidet er gjort. En CTE det vises til to ganger, eller som er merket MATERIALIZED, regnes ut i sin helhet før den ytre spørringen filtrerer den, og kopien den lager, har ingen indekser. En spørring som skulle berøre noen få rader, leser en hel tabell. Les planen: når CTE-en står som et eget trinn, med radtall langt over det spørringen returnerer, prøver du NOT MATERIALIZED eller skriver spørringen om slik at det bare vises til CTE-en én gang. Behold den materialiserte formen der CTE-en kaller en kostbar funksjon, slik at funksjonen kjøres én gang per rad og ikke én gang per henvisning.

En rekursiv spørring som bare fortsetter. Data med en syklus, som to kategorier der hver står oppført som forelder til den andre, sender en rekursiv CTE med UNION ALL rundt i løkken så lenge serveren tillater det. UNION hjelper bare når hele rader gjentas, og det holder sjelden så snart en kolonne for dybde er lagt til. Før stien av besøkte nøkler i et array, og stopp når en nøkkel dukker opp igjen, eller bruk CYCLE-klausulen som PostgreSQL 14 la til. En LIMIT på den ytre spørringen fungerer som et triks under testing, og manualen fraråder å stole på den i produksjon. SQL Server begrenser rekursjonen til 100 nivåer som standard og avbryter setningen utover det, så den samme feilen stopper i stedet for å løpe videre.

Endringer resten av setningen ikke ser. En CTE som endrer data, og hovedspørringen kjøres samtidig på det samme øyeblikksbildet (*snapshot*), så en hovedspørring som leser måltabellen direkte, ser radene slik de var før endringen. Den eneste kanalen mellom dem er RETURNING. To deler av én setning som endrer den samme raden, er verre: bare én av endringene får virkning, og hvilken er vanskelig å forutsi. Les endrede rader fra RETURNING-utdataene til CTE-en, og la hver rad ha bare én skriver per setning.

En plan som flyttet seg ved en oppgradering. Kode skrevet for PostgreSQL 11 eller tidligere støttet seg av og til på at en CTE ble evaluert først, som en bevisst sperre mot en plan forfatteren ikke ville ha. Etter en oppgradering til 12 eller nyere blir den samme CTE-en foldet inn i den ytre spørringen, og planleggeren velger noe nytt, som regel raskere og av og til mye tregere. Sammenlign utdataene fra EXPLAIN for de trege setningene før og etter oppgraderingen, og legg til MATERIALIZED der sperren var tilsiktet.

Les mer

- 7.8. WITH Queries (Common Table Expressions) — kapittelet i manualen om CTE-er: rekursjon, oppdagelse av sykluser, materialisering og setninger som endrer data.
- E.23. Release 12 — versjonen som begynte å folde CTE-er som brukes én gang, inn i den ytre spørringen.
- E.25. Release 14 — versjonen som la til klausulene SEARCH og CYCLE fra SQL-standarden.
- The WITH Clause — utgaven til SQLite, med hintene for materialisering og hvor de er hentet fra.
- WITH common_table_expression (Transact-SQL) — reglene SQL Server har for CTE-er, blant dem grensen MAXRECURSION for rekursjon.

Ordliste

Velg ut fra deres faktiske rammer

Ta med et representativt datasett, spørsmålene det skal besvare og dagens flaskehals. Vi kan avtale målinger av kvalitet, gjennomstrømning og gjenoppretting.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er CTE?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/cte/