Engineer

Partisjonering

å dele én tabell opp i fysiske biter

Hva det er

Partisjonering, også kalt tabellpartisjonering, lagrer radene i én logisk tabell i flere vanlige tabeller, der hver tar vare på radene med en nøkkel innenfor sine egne grenser, mens spørringer og skriving fortsatt bruker det ene navnet de alltid har brukt. PostgreSQL, som denne siden tar utgangspunkt i, kaller sin innebygde variant deklarativ partisjonering: tabellen deklareres som partisjonert, med en metode og en partisjonsnøkkel, og hver partisjon opprettes som en egen tabell.

Den partisjonerte tabellen har ingen rader selv. En innsetting rutes til partisjonen med grenser som passer nøkkelen, og en oppdatering som endrer nøkkelen, flytter raden til partisjonen der den nå hører hjemme. Partisjonene deler kolonnene til foreldretabellen, men kan ha egne indekser, egne lagringsinnstillinger og eget tabellområde (tablespace), og en partisjon kan selv partisjoneres videre.

Det finnes tre metoder. Intervallpartisjonering (range) gir hver partisjon et spenn av verdier, som regel datoer; den nedre grensen er med og den øvre ikke, så med spennene 1–10 og 10–20 hører verdien 10 til det andre. Listepartisjonering (list) navngir verdiene hver partisjon tar imot, f.eks. et sett landkoder. Hash-partisjonering gir hver partisjon en modulus og en rest, og en rad havner der hashen av nøkkelen gir nettopp den resten, noe som fordeler radene jevnt når det ikke finnes noe naturlig intervall å skjære langs. En standardpartisjon (default) kan fange opp verdiene som ikke passer noe annet sted, men bare ved intervall- og listepartisjonering.

Partisjonering skjer inne i én databaseserver. Å dele dataene på separate servere, hver med sin egen kopi av databaseskjemaet, er sharding, og det er et mye større steg. SQL Server og BigQuery har egne former for partisjonering, med egen

syntaks, og tankene nedenfor gjelder også der kommandoene er andre.

Bruk begrepet i en beslutning

Bruk «Hva er partisjonering?» 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

De fleste store tabeller vokser i den ene enden. Ordrer, måleravlesninger, logger og hendelser kommer inn hver dag, blir liggende i måneder eller år, og de fleste spørringene gjelder den nyeste delen. Uten partisjonering arbeider hver spørring mot én tabell og ett sett indekser som bare fortsetter å vokse.

Med en partisjonsnøkkel som passer spørringene, kan planleggeren bevise at de fleste partisjonene ikke kan inneholde noen rad som stemmer, og utelate dem helt fra planen. PostgreSQL kaller dette partition pruning, beskjæring av partisjoner. En rapport om forrige uke leser forrige ukes partisjon og ingenting annet. Dokumentasjonen nevner også en gevinst som er mindre opplagt: fordi partisjonene overtar jobben til de øverste nivåene i et indekstre, er det større sjanse for at de mest brukte delene av indeksene får plass i minnet.

Aller størst er endringen når gamle data skal bort. Å slette en måned med rader fra én enkelt tabell er en lang `DELETE` som etterlater døde rader `VACUUM` må rydde opp i. Med én partisjon per måned blir den samme oppbevaringsregelen én `DROP TABLE` eller én `DETACH PARTITION`, som dokumentasjonen kaller langt raskere enn en masseoperasjon, og som unngår kostnaden ved `VACUUM` helt. En frakoblet partisjon er fortsatt en vanlig tabell, så det kan tas sikkerhetskopi av den, eller den kan oppsummeres, før den slettes, og partisjoner som sjelden leses, kan ligge på billigere og tregere lagring.

Gevinsten har en pris, og dokumentasjonen sier rett ut når den lønner seg. Hver partisjon gir planleggingen mer arbeid, og fordelene er normalt bare verdt det når tabellen ellers ville vært svært stor. Tommelfingerregelen er at tabellen bør

være større enn det fysiske minnet på databaseserveren. Én ting endrer partisjonering heller ikke: en rad som mange transaksjoner slåss om, er fortsatt én rad i én partisjon, bak én lås.

Slik ser det ut i praksis

Et notat om å skalere en database før den shardes, har et avsnitt om å partisjonere tabellene rundt den hete raden. Tabellen `orders` der er partisjonert per måned på `created_at`, med primærnøkkelen `(id, created_at)`, fordi nøkkelen må inneholde partisjonskolonnen. En spørring som angir et datointervall, leser bare partisjonene innenfor det, denne månedens indekser holder seg små nok til å ligge i minnet, og oppbevaringsregelen blir én enkelt `DETACH PARTITION` i stedet for en `DELETE` av millioner av rader. Notatet er like tydelig på grensen: partisjonering gjør ikke raden det kjempes om, raskere, og en spørring som ikke angir nøkkelen, leser hver eneste partisjon.

Et ytelsesprosjekt i PostgreSQL for en applikasjon med brukerkontoer, leietakere, geografiske data og daglig strømforbruk delte strømtabellen inn i tidsintervaller per dag og måned, og brukte hash-partisjonering for de geografiske dataene. Det kom sammen med GiST-indekser, sammensatte indekser og omskrevne spørringer, og samlet ble svartidene for spørringer 10 ganger bedre, mens ressursbruken falt med 40 %.

En tredje side handler om minst 100 databaser i PostgreSQL, SQL Server og BigQuery, mye av det med geografiske data og tidsseriedata. Tidsseriepartisjonering var en del av skjemadesignet i PostgreSQL og SQL Server, og i BigQuery ble det brukt tidspartisjonerte og klyngede tabeller. Sammen med romlige indekser, materialiserte visninger og caching falt svartidene for spørringer mot disse dataene med 40–60 %.

Hvor det går galt

Partisjonering flytter en del av arbeidet fra det øyeblikket spørringen kjøres, til det øyeblikket tabellen designes, og de fleste problemene kommer av en designbeslutning som ble tatt uten at spørringene var i syne.

Innsettinger begynner å feile første dag i en ny måned. Ingen partisjon dekker det nye intervallet, og PostgreSQL avviser en innsetting i foreldretabellen som ikke hører hjemme i noen eksisterende partisjon. Opprett partisjonene på

forhånd, fra en planlagt jobb som genererer DDL-en, slik dokumentasjonen foreslår, og varsle når den nyeste partisjonen nærmer seg slutten:

```
CREATE TABLE orders_2026_11
  PARTITION OF orders
  FOR VALUES FROM ('2026-11-01')
              TO ('2026-12-01');
```

En standardpartisjon stopper feilene, men da havner radene for hver måned som ikke er dekket, sammen i den ene tabellen, og å koble til en ny partisjon senere skanner standardpartisjonen under en `ACCESS EXCLUSIVE` -lås, med mindre en `CHECK` -begrensning på standardpartisjonen allerede utelukker det nye intervallet. Behandle hver rad som havner i standardpartisjonen, som en alarm.

Spørringene er tregere etter partisjoneringen enn før. De vanlige spørringene filtrerer ikke på partisjonsnøkkelen, så hver av dem leser alle partisjonene og indeksene deres. En annen årsak er antallet partisjoner: dokumentasjonen sier at planleggeren takler opptil noen få tusen godt, forutsatt at typiske spørringer beskjærer bort alle unntatt noen få, og at planleggingstid og minnebruk vokser utover det. Velg nøkkelen ut fra `WHERE` -leddene applikasjonen allerede skriver, sjekk med `EXPLAIN` at bare de ventede partisjonene dukker opp, og hold antallet i rimelige proporsjoner. Én partisjon per kunde er en typisk felle, siden antallet vokser med virksomheten; dokumentasjonen foreslår heller hash-partisjonering med et fornuftig antall partisjoner.

En primærnøkkel eller unik begrensning lar seg ikke opprette. På en partisjonert tabell må en unik nøkkel eller primærnøkkel inneholde hver kolonne i partisjonsnøkkelen, og partisjonsnøkkelen kan ikke inneholde uttrykk eller funksjonskall. Indeksen i hver partisjon kan bare håndheve unikhet innenfor den partisjonen. Legg nøkkelkolonnen til begrensningen, slik notatets `orders` -tabell gjør med `(id, created_at)` .

Å bygge en indeks på foreldretabellen blokkerer skriving lenge. En vanlig `CREATE INDEX` på foreldretabellen stenger ute skriving til hver partisjon er indeksert, og valget `CONCURRENTLY` virker per partisjon, ikke på foreldretabellen. Opprett indeksen med `ON ONLY` på foreldretabellen, som lar den stå merket som ugyldig, bygg indeksen på hver partisjon med `CONCURRENTLY` , og koble hver av dem til med `ALTER INDEX ... ATTACH PARTITION` . Når indeksen i hver partisjon er koblet til, blir indeksen på foreldretabellen gyldig av seg selv.

En eksisterende stor tabell kan ikke gjøres om der den står. PostgreSQL gjør ikke en vanlig tabell om til en partisjonert. Opprett den nye partisjonerte tabellen, og koble den gamle tabellen til som én av partisjonene. Før tilkoblingen legger du til en `CHECK` -begrensning som stemmer med partisjonens grenser, så tilkoblingen ikke skanner hele tabellen mens den holder en `ACCESS EXCLUSIVE` -lås på den.

Les mer

- Table Partitioning — kapitlet i PostgreSQL-dokumentasjonen: de tre metodene, beskjæring, vedlikehold, begrensninger og beste praksis.
- CREATE TABLE — `PARTITION BY` , `PARTITION OF` , partisjonsgrenser og standardpartisjonen.
- ALTER TABLE — `ATTACH PARTITION` og `DETACH PARTITION` , med hvilken lås hver av dem tar.
- CREATE INDEX — indekser på partisjonerte tabeller, `ON ONLY` og bygging med `CONCURRENTLY` .

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 partisjonering?» 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/partitioning/