Engineer

Partitionering

at dele én tabel i fysiske stykker

Hvad det er

Partitionering, også kaldet tabelpartitionering, gemmer rækkerne fra én logisk tabel i flere almindelige tabeller, der hver rummer de rækker, hvis nøgle falder inden for dens egne grænser, mens databaseforespørgsler og skrivninger fortsat bruger det ene navn, de altid har brugt. PostgreSQL, som denne side følger, kalder sin indbyggede form deklarativ partitionering, på engelsk declarative partitioning: tabellen erklæres som partitioneret med en metode og en partitionsnøgle, og hver partition oprettes som en tabel for sig.

Den partitionerede tabel rummer selv ingen rækker. En indsættelse sendes videre til den partition, hvis grænser passer til nøglen, og en opdatering, der ændrer nøglen, flytter rækken til den partition, hvor den nu hører hjemme. Alle partitioner har forældretabellens kolonner, men hver kan have sine egne indeks, lagringsindstillinger og sit eget tablespace, og en partition kan selv partitioneres igen.

Der er tre metoder. Range-partitionering giver hver partition et interval af værdier, oftest datoer; den nedre grænse er med, og den øvre er ikke, så med intervallerne 1–10 og 10–20 hører værdien 10 til det andet. List-partitionering opregner de værdier, hver partition tager imod, for eksempel en gruppe landekoder. Hash-partitionering giver hver partition et modulus og en rest, og en række havner der, hvor hashen af dens nøgle giver netop den rest, hvilket fordeler rækkerne jævnt, når der ikke er et naturligt interval at skære efter. En standardpartition, på engelsk default partition, kan fange de værdier, der ikke passer i nogen anden, ved range og list, men ikke ved hash.

Partitionering foregår inden for én databaseserver. At fordele data over separate servere, hver med sin egen kopi af skemaet, er sharding, og det er et langt større skridt. SQL Server og BigQuery har deres egne former for partitionering med

deres egen syntaks, og idéerne nedenfor gælder også der, selv hvor kommandoerne er andre.

Brug begrebet i en beslutning

Brug »Hvad er partitionering?« 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

De fleste store tabeller vokser i den ene ende. Ordrer, målinger, logs og hændelser kommer ind hver dag, bliver liggende i måneder eller år, og de fleste forespørgsler spørger til den nyeste del. Uden partitionering arbejder hver forespørgsel mod én tabel og ét sæt indeks, der bliver ved med at vokse.

Med en partitionsnøgle, der passer til forespørgslerne, kan planneren bevise, at de fleste partitioner umuligt kan rumme en matchende række, og lade dem helt ude af planen. PostgreSQL kalder det partition pruning. En rapport over sidste uge læser sidste uges partition og intet andet. Dokumentationen nævner også en mindre oplagt gevinst: fordi partitionerne overtager rollen fra de øverste niveauer i et indekstræ, er der større chance for, at de mest brugte dele af indeksene kan ligge i hukommelsen.

Mest ændrer det sig, når gamle data skal væk. At slette en måneds rækker fra én samlet tabel er en lang `DELETE`, der efterlader døde rækker, som `VACUUM` skal rydde op efter. Med én partition pr. måned er den samme oprydning én `DROP TABLE` eller én `DETACH PARTITION`, som dokumentationen kalder langt hurtigere end en masseoperation, og som helt undgår arbejdet med `VACUUM`. En frakoblet partition er stadig en almindelig tabel, så den kan sikkerhedskopieres eller opsummeres, før den slettes, og partitioner, der sjældent læses, kan ligge på billigere og langsommere lager.

Gevinsten har en pris, og dokumentationen siger ligeud, hvornår den betaler sig. Hver partition giver planlægningen mere arbejde, og fordelene er normalt kun umagen værd, når tabellen ellers ville være meget stor; tommelfingerreglen er,

at tabellen bør være større end databaseserverens fysiske hukommelse. Én ting rører partitionering heller ikke ved: en enkelt række, som mange transaktioner kæmper om, er stadig én række i én partition bag én lås.

Sådan ser det ud i praksis

En note om at skalere en database, før den shardes, har et afsnit om at partitionere tabellerne omkring den varme række. Dens `orders` -tabel er partitioneret pr. måned på `created_at` , med primærnøglen `(id, created_at)` , fordi nøglen skal indeholde partitionskolonnen. En forespørgsel, der nævner et datointerval, læser kun partitionerne inden for det, denne måneds indekser forbliver små nok til at ligge i hukommelsen, og oprydning bliver én `DETACH PARTITION` i stedet for en `DELETE` af millioner af rækker. Noten siger også ligeud, hvor grænsen går: partitionering gør ikke den omstridte række hurtigere, og en forespørgsel, der ikke nævner nøglen, læser hver eneste partition.

Et PostgreSQL-projekt for en applikation med brugerkonti, tenant-oplysninger, geografiske data og dagligt elforbrug delte eltabellen op i tidsintervaller pr. dag og måned og brugte hash-partitionering til de geografiske data. Det skete sammen med GiST- og composite-indeks og omskrevne forespørgsler, og det samlede resultat var svartider, der blev 10 gange bedre, og et ressourceforbrug, der faldt med 40 %.

Partitionering kan også bruges i flere databasemotorer side om side. Med mindst 100 databaser på tværs af PostgreSQL, SQL Server og BigQuery, primært med GIS-data og tidsseriedata, indgik tidsseriepartitionering i skemadesignet til PostgreSQL og SQL Server, mens BigQuery brugte tidspartitionerede og clustrede tabeller. Sammen med spatial indeksering, materialized views og caching faldt svartiderne for de data med 40–60 %.

Hvor det går galt

Partitionering flytter noget af arbejdet fra forespørgselstid til designtid, og de fleste problemer kommer af en designbeslutning, der blev truffet uden forespørgslerne for øje.

Indsættelser begynder at fejle den første dag i en ny måned. Ingen partition dækker det nye interval, og PostgreSQL afviser en indsættelse i forældretabellen, der ikke passer til nogen eksisterende partition. Opret

partitionerne i god tid fra et planlagt job, der genererer DDL'en, som dokumentationen foreslår, og send en alarm, når den nyeste partition nærmer sig sin slutning:

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

En standardpartition stopper fejlene, men så havner rækkerne for alle udækkede måneder sammen i den ene tabel, og når en ny partition senere tilkobles, scannes standardpartitionen under en `ACCESS EXCLUSIVE` -lås, medmindre en `CHECK` -constraint på standardpartitionen allerede udelukker det nye interval. Behandl en standardpartition med rækker i som en alarm.

Forespørgsler er langsommere efter partitioneringen end før. De almindelige forespørgsler filtrerer ikke på partitionsnøglen, så hver af dem læser alle partitioner og deres indeks. En anden årsag er antallet af partitioner: dokumentationen siger, at planneren klarer op til et par tusind ret godt, forudsat at typiske forespørgsler skærer alle undtagen nogle få fra, og at planlægningstid og hukommelsesforbrug vokser derudover. Vælg nøglen ud fra de `WHERE` -betingelser, programmet allerede skriver, tjek med `EXPLAIN` , at kun de forventede partitioner optræder, og hold antallet i et rimeligt forhold. Én partition pr. kunde er en typisk fælde, fordi antallet vokser med forretningen; dokumentationen foreslår i stedet hash-partitionering med et rimeligt antal partitioner.

En primærnøgle eller en unik constraint lader sig ikke oprette. På en partitioneret tabel skal en unik nøgle eller primærnøgle indeholde alle partitionsnøglens kolonner, og partitionsnøglen må ikke indeholde udtryk eller funktionskald. Hver partitions indeks kan kun håndhæve unikhed inden for sin egen partition. Tilføj nøglekolonnen til constrainten, som notens `orders` -tabel gør med `(id, created_at)` .

En indeksbygning på forældretabellen blokerer skrivninger længe. Et almindeligt `CREATE INDEX` på forældretabellen spærrer for skrivninger, indtil hver partition er indekseret, og `CONCURRENTLY` virker pr. partition og ikke på forældretabellen. Opret indekset med `ON ONLY` på forældretabellen, så det

markeres som ugyldigt, byg indekset på hver partition med `CONCURRENTLY`, og tilkobl hvert af dem med `ALTER INDEX ... ATTACH PARTITION`. Når alle partitionernes indeks er tilkoblet, bliver forældreindekset gyldigt af sig selv.

En eksisterende stor tabel lader sig ikke lægge om på stedet. PostgreSQL gør ikke en almindelig tabel til en partitioneret. Opret den nye partitionerede tabel, og tilkobl den gamle tabel som en af dens partitioner. Tilføj før tilkoblingen en `CHECK`-constraint, der svarer til partitionens grænser, så tilkoblingen ikke scanner hele tabellen, mens den holder en `ACCESS EXCLUSIVE`-lås på den.

Læs mere

- Table Partitioning — PostgreSQL-kapitlet: de tre metoder, pruning, vedligeholdelse, begrænsninger og best practices.
- CREATE TABLE — `PARTITION BY`, `PARTITION OF`, partitionsgrænser og standardpartitionen.
- ALTER TABLE — `ATTACH PARTITION` og `DETACH PARTITION`, og hvad hver af dem låser.
- CREATE INDEX — indeks på partitionerede tabeller, `ON ONLY` og bygning med `CONCURRENTLY`.

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 partitionering?« 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/partitioning/