Engineer

Partitionering

att dela upp en tabell i fysiska delar

Vad det är

Partitionering, eller tabellpartitionering, lagrar raderna i en logisk tabell i flera vanliga tabeller, där var och en håller de rader vars nyckel faller inom just dess gränser, medan läsningar och skrivningar fortsätter att använda det enda namn de alltid har använt. PostgreSQL, som den här sidan följer, kallar sin inbyggda form deklarativ partitionering: tabellen deklareras som partitionerad, med en metod och en partitionsnyckel, och varje partition skapas som en egen tabell.

Själva den partitionerade tabellen håller inga rader. En insättning dirigeras till den partition vars gränser passar nyckeln, och en uppdatering som ändrar nyckeln flyttar raden till den partition där den nu hör hemma. Partitionerna delar föräldertabellens kolumner men kan ha egna index, egna lagringsinställningar och ett eget tabellutrymme, och en partition kan i sin tur partitioneras igen.

Det finns tre metoder. Intervallpartitionering ger varje partition ett spann av värden, oftast datum; den nedre gränsen ingår och den övre gör det inte, så med spannen 1–10 och 10–20 hör värdet 10 till det andra. Listpartitionering räknar upp vilka värden varje partition tar emot, till exempel en uppsättning landskoder. Hashpartitionering ger varje partition en modulus och en rest, och en rad hamnar där hashvärdet av dess nyckel lämnar just den resten, vilket sprider raderna jämnt när det saknas ett naturligt intervall att skära efter. En standardpartition kan fånga upp de värden som inte passar någon annan, men bara vid intervall- och listpartitionering.

Partitionering sker inuti en och samma databasserver. Att dela upp data över separata servrar, var och en med sin egen kopia av databasschemat, är sharding, och det är ett betydligt större steg. SQL Server och BigQuery har egna former av partitionering, med egen syntax, och idéerna nedan gäller även där kommandona skiljer sig.

Använd begreppet i ett beslut

Använd ”Vad är partitionering?” 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

De flesta stora tabeller växer i ena änden. Ordrar, mätvärden, loggar och händelser kommer in varje dag, ligger kvar i månader eller år, och de flesta databasfrågor gäller den senaste delen. Utan partitionering arbetar varje fråga mot en enda tabell och en enda uppsättning index som hela tiden blir större.

Med en partitionsnyckel som passar frågorna kan planeraren bevisa att de flesta partitioner inte kan innehålla någon matchande rad och lämna dem helt utanför planen. PostgreSQL kallar det partition pruning. En rapport om förra veckan läser förra veckans partition och inget annat. Dokumentationen lägger till en mindre uppenbar vinst: eftersom partitionerna tar över det jobb som de övre nivåerna i ett indexträd annars gör, är det mer sannolikt att de mest använda delarna av indexen ryms i minnet.

Störst förändring blir det när gammal data ska bort. Att radera en månad av rader ur en enda tabell är en lång `DELETE` som lämnar döda rader efter sig som `VACUUM` måste städa upp. Med en partition per månad blir samma gallring en enda `DROP TABLE` eller en enda `DETACH PARTITION`, något som dokumentationen beskriver som betydligt snabbare än en massoperation och som helt slipper merkostnaden för `VACUUM`. En frånkopplad partition är fortfarande en vanlig tabell, så den kan säkerhetskopieras eller sammanfattas innan den tas bort, och partitioner som sällan läses kan ligga på billigare och långsammare lagring.

Vinsten har ett pris, och dokumentationen säger rakt ut när den lönar sig. Varje partition lägger till arbete i planeringen, och fördelarna är normalt värda det bara när tabellen annars skulle vara mycket stor, med tumregeln att tabellen ska vara större än databasserverns fysiska minne. Partitionering lämnar också en sak orörd: en enskild rad som många transaktioner slåss om är fortfarande en rad i en partition, bakom ett lås.

Så ser det ut i praktiken

Partitionering är här alltid ett steg bland flera, bredvid index, omskrivna frågor och cachning, och aldrig hela svaret.

En anteckning om att skala en databas innan den shardas har ett avsnitt om att partitionera tabellerna runt den heta raden. Där partitioneras tabellen `orders` per månad på `created_at`, med primärnyckeln `(id, created_at)` eftersom nyckeln måste innehålla partitionskolumnen. En fråga som anger ett datumintervall läser bara partitionerna inom det, den här månadens index förblir små nog att ligga i minnet, och gallringen blir en enda `DETACH PARTITION` i stället för en `DELETE` av miljontals rader. Anteckningen är också tydlig med gränsen: partitionering gör inte den omstridda raden snabbare, och en fråga som inte anger nyckeln läser varje partition.

Ett prestandaprojekt i PostgreSQL för en applikation med användarkonton, tenants, geografiska data och daglig elförbrukning delade upp eltabellen i tidsintervall per dag och månad och använde hashpartitionering för de geografiska data. Det skedde tillsammans med GiST-index, sammansatta index och omskrivna frågor, och det samlade resultatet var 10 gånger bättre svarstider och 40 % lägre resursanvändning.

En tredje sida beskriver minst 100 databaser i PostgreSQL, SQL Server och BigQuery, till stor del med geografiska data och tidsseriedata. Tidsseriepartitionering var en del av schemadesignen i PostgreSQL och SQL Server, och i BigQuery användes tidspartitionerade och klustrade tabeller. Tillsammans med spatiala index, materialiserade vyer och cachning minskade svarstiderna för de data med 40–60 %.

Var det går fel

Partitionering flyttar en del av arbetet från frågetid till designtid, och de flesta problem kommer från ett designbeslut som fattades utan frågorna framför sig.

Insättningar börjar misslyckas den första dagen i en ny månad. Det finns ingen partition som täcker det nya intervallet, och PostgreSQL vägrar en insättning i föräldertabellen som inte hör till någon befintlig partition. Skapa partitioner i förväg, från ett schemalagt jobb som genererar DDL-satserna, som dokumentationen föreslår, och larma när den nyaste partitionen närmar sig sitt slut:

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

En standardpartition stoppar felen, men då hamnar rader för varje månad som saknar en egen partition tillsammans i den enda tabellen, och att koppla in en ny partition senare skannar standardpartitionen under ett `ACCESS EXCLUSIVE` -lås, om inte en `CHECK` -begränsning på standardpartitionen redan utesluter det nya intervallet. Se en standardpartition som innehåller rader som en varning.

Frågor blir långsammare efter partitioneringen än före. De vanliga frågorna filtrerar inte på partitionsnyckeln, så var och en läser varje partition och dess index. En andra orsak är antalet partitioner: enligt dokumentationen klarar planeraren upp till några tusen bra, förutsatt att typiska frågor sållar bort alla utom ett fåtal, och planeringstid och minne växer bortom det. Välj nyckeln utifrån de `WHERE` -villkor som applikationen redan skriver, kontrollera med `EXPLAIN` att bara de förväntade partitionerna dyker upp och håll antalet i proportion. En partition per kund är en typisk fälla, eftersom antalet växer med verksamheten; dokumentationen föreslår i stället hashpartitionering med ett rimligt antal partitioner.

En primärnyckel eller unikhetsbegränsning går inte att skapa. På en partitionerad tabell måste en unik nyckel eller primärnyckel innehålla varje kolumn i partitionsnyckeln, och partitionsnyckeln får inte innehålla uttryck eller funktionsanrop. Indexet i varje partition kan bara upprätthålla unikhet inom den partitionen. Lägg till nyckelkolumnen i begränsningen, så som anteckningens tabell `orders` gör med `(id, created_at)` .

Att bygga ett index på föräldertabellen blockerar skrivningar länge. En vanlig `CREATE INDEX` på föräldertabellen stänger ute skrivningar tills varje partition har indexerats, och alternativet `CONCURRENTLY` fungerar per partition snarare än på föräldern. Skapa indexet med `ON ONLY` på föräldertabellen, vilket lämnar det markerat som ogiltigt, bygg indexet på varje partition med `CONCURRENTLY` och koppla in vart och ett med `ALTER INDEX ... ATTACH PARTITION` . När indexet för varje partition är inkopplat blir föräldertabellens index giltigt av sig självt.

En befintlig stor tabell går inte att ställa om på plats. PostgreSQL gör inte om en vanlig tabell till en partitionerad. Skapa den nya partitionerade tabellen och koppla in den gamla tabellen som en av dess partitioner. Lägg innan dess till en `CHECK` -begränsning som motsvarar partitionens gränser, så att inkopplingen slipper skanna hela tabellen medan den håller ett `ACCESS EXCLUSIVE` -lås på den.

Läs mer

- Table Partitioning — kapitlet i dokumentationen för PostgreSQL: de tre metoderna, pruning, underhåll, begränsningar och bästa praxis.
- CREATE TABLE — `PARTITION BY` , `PARTITION OF` , partitionsgränser och standardpartitionen.
- ALTER TABLE — `ATTACH PARTITION` och `DETACH PARTITION` , och vad var och en låser.
- CREATE INDEX — index på partitionerade tabeller, `ON ONLY` och bygge med `CONCURRENTLY` .

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 partitionering?” 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/partitioning/