Engineer

Databasfråga

▼ en skriven fråga som en databas besvarar

Vad det är

En databasfråga är en begäran om data, skriven i ett språk som databasen förstår, som kommer tillbaka med ett resultat: oftast en mängd rader, ibland ett enda tal. Handboken till PostgreSQL definierar den på en rad: att hämta data ur en databas, eller kommandot som gör hämtningen, kallas en query, och i SQL är det kommandot SELECT. I vardagligt tal tänjs ordet längre, och utvecklare kallar nästan varje sats de skickar till en databas för en SQL-fråga, även de som ändrar data och lämnar något tillbaka. Båda betydelserna förekommer nedan, och pluralformen, databasfrågor, betyder oftast all trafik som en applikation skickar.

En SQL-fråga om vilka fakturor som har väntat längst på betalning läses nästan som man skulle säga det högt:

```
SELECT i.number, c.name, i.amount, i.due_on
FROM invoices AS i
JOIN customers AS c ON c.id = i.customer_id
WHERE i.paid_on IS NULL
  AND i.due_on < current_date
ORDER BY i.due_on
LIMIT 20;
```

Texten anger vad den vill ha och lämnar metoden öppen. Mellan att PostgreSQL tar emot den och returnerar den första raden listar planeraren de vägar som kan ge svaret, t.ex. att läsa hela tabellen invoices eller att gå igenom ett index på `due_on`, uppskattar en kostnad för varje väg och kör den billigaste. Kostnaderna anges i godtyckliga enheter som av konvention utgår från läsningen av en sida från disk. Två databasfrågor som ser likadana ut på skärmen kan därför skilja sig med en faktor tusen när datan under dem växer, och planen är det enda ställe där skillnaden syns.

Satsdelarna körs inte i den ordning de är skrivna. Handboken beskriver behandlingen av SELECT med WITH först, sedan FROM, WHERE, gruppering och HAVING, därefter listan med utdata och först efter det ORDER BY och LIMIT. Den ordningen förklarar en regel som överraskar de flesta en gång: ett namn som ges åt en kolumn i urvalslistan fungerar i ORDER BY och GROUP BY, medan WHERE körs innan namnet finns, så där måste uttrycket skrivas ut igen. Resultatet av varje databasfråga är i sig en tabell, och därför går databasfrågor att nästla, med en inre SQL-fråga eller en WITH-sats som matar sina rader till en större.

Använd begreppet i ett beslut

Använd ”Vad är en databasfråga?” 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

I en applikation som vilar på en relationsdatabas avgör databasfrågorna en stor del av hur snabb en sida känns och hur mycket hårdvara systemet behöver. Vilka databasfrågor som spelar roll är aritmetik snarare än magkänsla: kostnaden för systemet är varaktigheten multiplicerad med hur ofta satsen körs. Ett uppslag på 2 ms som körs en miljon gånger i timmen blir mer än en halvtimmes databastid varje timme, medan en rapport som tar två sekunder och körs två gånger om dagen knappt märks.

PostgreSQL för den bokföringen i tillägget pg_stat_statements. Det slår ihop satser som bara skiljer sig i sina litterala värden till en post och registrerar hur många gånger varje sats har körts och den totala tid den har tagit. Att sortera på totalen i stället för på medelvärdet är den vana som lönar sig först i arbetet med databasfrågor, eftersom den pekar ut den sats som är värd att rätta i stället för den som bara ser långsam ut.

Varje databasfråga kostar också en tur och retur. Applikationen skickar text, servern tolkar, planerar och kör den, och raderna färdas tillbaka. För ett snabbt uppslag i ett index kan resan väga tyngre än arbetet. Därför spelar samtalets

form lika stor roll som varje enskild sats: en UPDATE som berör hundra rader gör en resa, och en loop som läser hundra rader och uppdaterar dem en i taget gör 101 resor.

Sedan återstår frågan om vem som skriver databasfrågorna. En ORM eller ett bibliotek som bygger SQL-frågor genererar dem ur applikationskoden, vilket går fort för enkla läsningar och skrivningar och håller koden i ett enda språk. Det döljer också hur många satser en begäran skickar och hur de ser ut, tills en sida blir långsam. Handskriven SQL visar båda, till priset att någon måste äga den som all annan kod. En fungerande uppdelning låter ORM:en sköta rutinposterna, skriver de prestandakritiska kodvägarna för hand och läser den genererade SQL-koden för allt som klättrar till toppen av totalerna.

Så ser det ut i praktiken

Metoden för en långsam databas i Optimera databasen innan du shardar den börjar med databasfrågorna, innan någon räknar på fler maskiner. Det första steget frågar PostgreSQL vilka satser som kostar mest totalt, med pg_stat_statements sorterat på total körtid. Det andra kör EXPLAIN (ANALYZE, BUFFERS) på satsen överst, för att se den plan som databasen valde, de rader varje steg verkligen returnerade jämfört med de rader planeraren väntade sig och hur många sidor den läste. Anteckningens tumregel är att en skanning av en miljon rader för att returnera tio, eller en uppskattning som slår fel med en faktor hundra, för det mesta förklarar långsamheten på egen hand. Den räknar också turer och returer och drar samma gräns som den här artikeln drar ovan: det som måste se datan flyttar in i databasen, och affärsreglerna stannar i applikationen.

En databasfråga kan också göra ett lås jobb. I Att sälja den sista biljetten exakt en gång går en blixtrea snett på ett av två sätt. Att läsa lagersaldot, besluta i applikationen och skriva tillbaka det nya talet låter två köpare båda ta den sista enheten. Att hålla ett radlås över den applikationslogiken får varje köpare att vänta på sin tur bakom det. Lösningen viker ihop kontrollen och minskningen till en enda villkorad UPDATE som drar av antalet bara där tillräckligt mycket finns kvar i lager och returnerar den nya nivån. En rad tillbaka betyder att enheten är reserverad, och noll rader betyder slutsålt. Med READ COMMITTED, standardnivån för isolering i PostgreSQL, väntar en andra köpare som når raden medan den första håller den, och prövar sedan WHERE-villkoret igen mot den bekräftade raden, så att den sista enheten går till exakt en av dem. Ett CHECK-villkor som håller saldot på noll eller mer står bakom databasfrågan.

Var det går fel

En billig databasfråga dominerar lasten. Databasen går varm, men loggen för långsamma databasfrågor förblir tyst, eftersom varje anrop blir klart på några millisekunder. Orsaken är frekvensen: en snabb SQL-fråga som anropas vid varje begäran, ofta från en loop eller från en komponent som finns på varje sida. Rangordna satserna efter total körtid i pg_stat_statements och cacha sedan resultatet, samla anropen i omgångar eller gör databasfrågan billigare med ett index som matchar dess filter.

En sida skickar hundratals databasfrågor. Varje SQL-fråga i spåret tar en millisekund, och sidan tar ändå flera sekunder. En ORM som läser in relaterade poster vid behov, en SQL-fråga per föräldrarad, är den vanliga orsaken, känd som N+1-mönstret. Läs in de relaterade raderna i en enda SQL-fråga, med en join eller med `WHERE id = ANY($1)` och en array med nycklar, och bevaka antalet satser per begäran lika noga som varaktigheten för var och en.

Den sista enheten säljs två gånger. Två begäranden läser samma lagersaldo, båda bedömer att det räcker och båda skriver. Att läsa och skriva i separata satser lämnar en lucka mellan kontrollen och handlingen, och samtidig trafik hittar den. Stäng luckan inne i databasfrågan, med en villkorad UPDATE och RETURNING som ovan, eller med SELECT … FOR UPDATE som hålls så kort som möjligt och utan något nätverksanrop mellan låset och commit.

En diagnos som ändrade datan. Någon kör EXPLAIN ANALYZE på en långsam DELETE för att titta på planen, och raderna är borta. ANALYZE kör satsen på riktigt och kastar bara dess utdata, så varje sidoeffekt inträffar. Lägg varje sats som ändrar data inom BEGIN och ROLLBACK innan den analyseras, eller använd vanlig EXPLAIN, som planerar utan att köra, när uppskattningarna räcker.

Läs mer

- SELECT — den fullständiga syntaxen för en databasfråga i PostgreSQL, inklusive den ordning i vilken satsdelarna behandlas.
- 7.1. Overview — början på handbokens kapitel om databasfrågor, med definitionen av ordet.
- 51.1. The Path of a Query — vägen från klientens text genom parser, omskrivare och planerare till exekveraren.
- 14.1. Using EXPLAIN — hur man läser en plan, vad kostnadsenheterna betyder och varför ANALYZE kör satsen.

- F.32. pg_stat_statements — track statistics of SQL planning and execution — tillägget som rangordnar databasfrågor efter vad de kostar totalt.
- 13.2. Transaction Isolation — vad en andra skrivande transaktion ser under READ COMMITTED och varför WHERE-villkoret prövas igen.

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 en databasfråga?” 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/query/