**Engineer**

# Databaseforespørgsel

▼ et nedskrevet spørgsmål en database svarer på

## Hvad det er

En databaseforespørgsel, i daglig tale bare en forespørgsel, er en anmodning om data, skrevet i et sprog databasen forstår, som kommer tilbage med et resultat: som regel et sæt rækker, af og til et enkelt tal. PostgreSQL's manual definerer begrebet på én linje: Det at hente data fra en database, eller den kommando der henter dem, kaldes en query, og i SQL er den kommando SELECT. Mange danske udviklere bruger det engelske ord direkte, og i daglig tale glider betydningen, så næsten enhver sætning, der sendes til en database, kaldes en forespørgsel, også dem der ændrer data og sender noget retur. Begge betydninger bruges nedenfor: forespørgslen er den enkelte sætning, og forespørgsler i flertal er som regel al den trafik, en applikation sender.

Spørger man, hvilke fakturaer der har ventet længst på betaling, ligger forespørgslen tæt på, hvordan man ville sige det højt:

```
SELECT f.nummer, k.navn, f.total, f.forfald
FROM fakturaer AS f
JOIN kunder AS k ON k.id = f.kunde_id
WHERE f.betalt_den IS NULL
  AND f.forfald < current_date
ORDER BY f.forfald
LIMIT 20;
```

Teksten siger, hvad den vil have, og lader metoden være åben. Mellem at modtage den og returnere den første række opregner PostgreSQL's planlægger de måder, den kan finde svaret på – for eksempel at læse hele fakturatabellen eller at gå gennem et indeks på `forfald` – anslår en omkostning for hver og kører den billigste. Omkostningerne er i vilkårlige enheder, traditionelt sat i

forhold til at læse én side fra disken. To forespørgsler, der ligner hinanden på skærmen, kan derfor variere med en faktor tusind, når data under dem vokser, og planen er det eneste sted, forskellen kan ses.

Klausulerne udføres i en anden rækkefølge, end de er skrevet i. Manualen beskriver behandlingen af SELECT sådan: først WITH, så FROM, WHERE, gruppering og HAVING, derefter outputlisten og først til sidst ORDER BY og LIMIT. Den rækkefølge forklarer en regel, de fleste bliver overraskede over én gang: Et navn, man giver en kolonne i select-listen, virker i ORDER BY og GROUP BY, men WHERE kører, før navnet findes, så dér må udtrykket skrives ud igen. Resultatet af enhver forespørgsel er selv en tabel, og derfor kan forespørgsler indlejres, så en delforespørgsel eller en WITH-klausul fodrer en større med sine rækker.

Brug begrebet i en beslutning

Brug »Hvad er en databaseforespørgsel?« 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

I en applikation bygget på en relationel database afgør forespørgslerne meget af, hvor hurtig en side føles, og hvor meget hardware systemet kræver. Hvilke forespørgsler der tæller, er et regnestykke snarere end en fornemmelse: En forespørgsels pris for systemet er dens varighed ganget med, hvor ofte den kører. Et opslag på 2 ms, der kører en million gange i timen, bliver til over en halv times databasetid hver time, mens en rapport på to sekunder, der kører to gange om dagen, knap kan ses.

PostgreSQL fører det regnskab i udvidelsen pg_stat_statements. Den samler sætninger, der kun adskiller sig i deres konkrete værdier, i én post og noterer, hvor mange gange hver kørte, og hvor lang tid den brugte i alt. At sortere efter total i stedet for gennemsnit er den vane, der betaler sig først i arbejdet med forespørgsler, fordi den peger på den sætning, der er værd at rette, frem for den, der bare ser langsom ud.

Hver forespørgsel koster også en rundtur. Applikationen sender tekst, serveren parser, planlægger og kører den, og rækkerne rejser retur. For et hurtigt opslag i et indeks kan rundturen veje tungere end selve arbejdet. Derfor betyder samtalens form lige så meget som den enkelte sætning: Én UPDATE, der rører hundrede rækker, er én rundtur, mens en løkke, der læser hundrede rækker og opdaterer dem én ad gangen, er hundrede og én.

Så er der spørgsmålet om, hvem der skriver forespørgslerne. En ORM eller en query builder genererer dem ud fra applikationskoden, hvilket går hurtigt for enkle læsninger og skrivninger og holder koden i ét sprog. Den skjuler også, hvor mange sætninger et kald sender, og hvordan de ser ud, indtil en side bliver langsom. Håndskrevet SQL viser begge dele, til gengæld skal nogen eje den som al anden kode. En brugbar fordeling lader ORM'en klare de rutinemæssige poster, skriver de ydelseskritiske kodestier i hånden og læser den genererede SQL for alt, der kravler op i toppen af totalerne.

## Sådan ser det ud i praksis

Metoden for en langsom database i Optimér databasen, før du sharder den begynder med forespørgslerne, før nogen prissætter flere maskiner. Første skridt spørger PostgreSQL, hvilke sætninger der koster mest i alt, med pg_stat_statements sorteret efter samlet køretid. Andet skridt kører EXPLAIN (ANALYZE, BUFFERS) på sætningen øverst for at se den plan, databasen valgte, de rækker hvert trin faktisk returnerede over for dem, planlæggeren forventede, og hvor mange sider den læste. Notens tommelfingerregel er, at en scanning af en million rækker for at returnere ti, eller et estimat, der rammer hundrede gange ved siden af, som regel forklarer langsomheden alene. Den tæller også rundture og trækker samme grænse som ovenfor: Det, der skal se data, flytter ind i databasen, og forretningsreglerne bliver i applikationen.

En forespørgsel kan også gøre en låses arbejde. I At sælge den sidste billet præcis én gang kan et flash-salg gå galt på to måder. Læser applikationen lageret, beslutter og skriver det nye tal tilbage, kan to købere begge tage den sidste enhed; holder den en rækkelås hen over den logik, må alle købere vente på den. Løsningen samler kontrollen og nedskrivningen i én betinget UPDATE, der kun trækker antallet fra, hvis der er nok på lager, og returnerer det nye antal. Én række tilbage betyder, at enheden er reserveret, og nul rækker betyder udsolgt. Ved READ COMMITTED, PostgreSQL's standardisolationsniveau, venter en anden køber, der når rækken, mens den første holder den, og evaluerer derefter

WHERE-klausulen igen mod den committede række, så den sidste enhed går til præcis én af dem. En CHECK-begrænsning, der holder lageret på nul eller derover, står bag forespørgslen.

## Hvor det går galt

**En billig forespørgsel, der dominerer belastningen.** Databasen kører varm, men loggen over langsomme forespørgsler forbliver stille, fordi hvert kald er færdigt på få millisekunder. Årsagen er hyppighed: én hurtig forespørgsel, der kører ved hvert kald til serveren, ofte fra en løkke eller fra en komponent, der står på hver side. Sorter sætningerne efter samlet køretid i pg_stat_statements, og cache så resultatet, saml kaldene, eller gør forespørgslen billigere med et indeks, der passer til dens filter.

**En side, der sender hundredvis af forespørgsler.** Hver forespørgsel i sporet tager et millisekund, og siden tager stadig sekunder. En ORM, der indlæser relaterede poster efter behov, én forespørgsel pr. overordnet række, er den sædvanlige årsag, kendt som N+1-mønstret. Hent de relaterede rækker i én forespørgsel, med et join eller med `WHERE id = ANY($1)` og et array af nøgler, og hold lige så godt øje med antallet af sætninger pr. kald som med varigheden af hver enkelt.

**Den sidste enhed solgt to gange.** To kald læser det samme lager, begge konkluderer, at der er nok, og begge skriver. Når læsning og skrivning sker i hver sin sætning, opstår der et hul mellem kontrollen og handlingen, og samtidig trafik finder det. Luk hullet inde i forespørgslen, med en betinget UPDATE og RETURNING som ovenfor eller med SELECT … FOR UPDATE holdt så kort som muligt og uden netværkskald mellem låsen og committet.

**En diagnose, der ændrede data.** Nogen kører EXPLAIN ANALYZE på en langsom DELETE for at se planen, og rækkerne er væk. ANALYZE udfører sætningen for alvor og kasserer kun dens output, så alle bivirkninger sker. Pak enhver dataændrende sætning ind i BEGIN og ROLLBACK, før man analyserer den, eller brug almindelig EXPLAIN, der planlægger uden at udføre, når estimaterne er nok.

## Læs mere

- SELECT — den fulde syntaks for en forespørgsel i PostgreSQL, inklusive den rækkefølge, klausulerne behandles i.

- 7.1. Overview — begyndelsen på manualens kapitel om forespørgsler, med definitionen af ordet.
- 51.1. The Path of a Query — vejen fra klientens tekst gennem parser, omskrivning og planlægger til executor.
- 14.1. Using EXPLAIN — hvordan man læser en plan, hvad omkostningsenhederne betyder, og hvorfor ANALYZE kører sætningen.
- F.32. pg_stat_statements — track statistics of SQL planning and execution — udvidelsen, der rangerer forespørgsler efter, hvad de koster i alt.
- 13.2. Transaction Isolation — hvad en anden skrivende transaktion ser under READ COMMITTED, og hvorfor WHERE-klausulen tjekkes igen.

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 en databaseforespørgsel?« 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/query/**