Engineer

# Spørring

▼ et skrevet spørsmål en database svarer på

## Hva det er

En spørring er en forespørsel om data, skrevet i et språk databasen forstår, som kommer tilbake med et resultat: som regel et sett rader, noen ganger ett enkelt tall. Håndboken til PostgreSQL definerer ordet på én linje: å hente data fra en database, eller kommandoen som henter dem, kalles en spørring (query), og i SQL er den kommandoen SELECT. I dagligtale strekkes ordet lenger, og utviklere kaller nesten hver setning de sender til en database, for en spørring, også de som endrer data og sender noe tilbake. Begge betydningene brukes nedenfor, og flertallet, spørringer, betyr gjerne all trafikken en applikasjon sender.

Å spørre hvilke fakturaer som har ventet lengst på betaling, leses nesten slik spørsmålet ville blitt stilt høyt:

```
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;
```

Teksten sier hva den vil ha, og lar metoden stå åpen. Fra teksten mottas til første rad returneres, lister planleggeren i PostgreSQL opp måtene den kan finne svaret på, f.eks. å lese hele fakturatabellen eller å gå gjennom en indeks på `due_on`, anslår en kostnad for hver og kjører den billigste. Kostnadene er i vilkårlige enheter, som etter konvensjon er forankret i å lese én side fra disk. To spørringer som ser like ut på skjermen, kan derfor skille seg med en faktor tusen når dataene under dem vokser, og planen er det eneste stedet den forskjellen synes.

Leddene kjøres i en annen rekkefølge enn de er skrevet i. Håndboken beskriver behandlingen av SELECT med WITH først, så FROM, WHERE, gruppering og HAVING, deretter SELECT-listen, og først etter det ORDER BY og LIMIT. Den rekkefølgen forklarer en regel som overrasker de fleste én gang: et navn som gis til en kolonne i SELECT-listen, virker i ORDER BY og GROUP BY, mens WHERE kjøres før navnet finnes, så der må uttrykket skrives ut på nytt. Resultatet av hver spørring er selv en tabell, og derfor kan spørringer nøstes, med en underspørring eller et WITH-ledd som mater radene sine inn i en større spørring.

Bruk begrepet i en beslutning

Bruk «Hva er en spørring?» 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

I en applikasjon med en relasjonsdatabase bak avgjør spørringene mye av hvor rask en side føles, og hvor mye maskinvare systemet trenger. Hvilke spørringer som betyr noe, er regning og ikke magefølelse: det en spørring koster systemet, er varigheten ganget med hvor ofte den kjøres. Et oppslag på 2 ms som kjøres en million ganger i timen, blir til mer enn en halvtime databasetid hver time, mens en rapport som tar to sekunder og kjøres to ganger om dagen, knapt synes.

PostgreSQL fører det regnskapet i utvidelsen pg_stat_statements. Den slår sammen setninger som bare skiller seg i konstantene de inneholder, til én oppføring, og registrerer hvor mange ganger hver ble kjørt, og hvor lang tid de brukte til sammen. Å sortere etter total i stedet for gjennomsnitt er vanen som lønner seg først i arbeid med spørringer, fordi den peker på setningen som er verdt å rette, og ikke på den som bare ser treg ut.

Hver spørring koster også en rundtur. Applikasjonen sender tekst, serveren parser, planlegger og kjører den, og radene reiser tilbake. For et raskt oppslag i en indeks kan rundturen veie mer enn selve arbeidet. Formen på samtalen betyr derfor like mye som hver enkelt setning: én UPDATE som rører hundre rader, er én rundtur, og en løkke som leser hundre rader og oppdaterer dem én etter én, er hundre og én.

Så er det spørsmålet om hvem som skriver spørringene. En ORM eller en spørringsbygger genererer dem fra applikasjonskoden, noe som går raskt for enkel lesing og skriving og holder koden i ett språk. Men det skjuler også hvor mange setninger en forespørsel sender, og hvordan de ser ut, helt til en side blir treg. Håndskrevet SQL viser begge deler, mot at noen eier den som all annen kode. En deling som fungerer, lar ORM-en ta seg av rutinepostene, skriver de ytelseskritiske kodestiene for hånd og leser den genererte SQL-en for alt som klatrer til toppen av totalene.

## Slik ser det ut i praksis

Metoden for en treg database i Optimaliser databasen før du sharder den begynner med spørringene, før noen priser flere maskiner. Første steg spør PostgreSQL hvilke setninger som koster mest totalt, med pg_stat_statements sortert etter samlet kjøretid. Andre steg kjører EXPLAIN (ANALYZE, BUFFERS) på setningen øverst for å se planen databasen valgte, radene hvert steg faktisk returnerte mot radene planleggeren ventet, og hvor mange sider den leste. Tommelfingerregelen i notatet er at en skanning av en million rader for å returnere ti, eller et estimat som bommer med en faktor hundre, som regel forklarer tregheten alene. Notatet teller rundturer også, og trekker den samme grensen som denne artikkelen trekker over: det som må se dataene, flytter inn i databasen, og forretningsregler blir i applikasjonen.

En spørring kan også gjøre jobben til en lås. I Å selge den siste billetten nøyaktig én gang går et lynsalg galt på én av to måter. Å lese lagerbeholdningen, avgjøre i applikasjonen og skrive det nye tallet tilbake lar to kjøpere ta den siste enheten begge to; å holde en radlås over den applikasjonslogikken får hver kjøper til å vente bak den. Løsningen samler sjekken og nedtellingen i én betinget UPDATE som trekker fra antallet bare der det er nok igjen på lager, og returnerer det nye nivået. Én rad tilbake betyr at enheten er reservert, og null rader betyr utsolgt. På READ COMMITTED, standardnivået for isolasjon i PostgreSQL, venter en annen kjøper som når raden mens den første holder den, og evaluerer så WHERE-leddet på nytt mot den committede raden, så den siste enheten går til nøyaktig én av dem. En CHECK-skranke som holder beholdningen på null eller over, står bak spørringen.

## Hvor det går galt

**En billig spørring som dominerer lasten.** Databasen er hardt belastet, men loggen over trege spørringer er stille, fordi hvert kall er ferdig på noen få millisekunder. Årsaken er hyppighet: én rask spørring som kalles ved hver

forespørsel, ofte fra en løkke eller fra en komponent som finnes på hver side. Ranger setningene etter samlet kjøretid i pg_stat_statements, og bruk så caching på resultatet, samle kallene i batcher, eller gjør spørringen billigere med en indeks som passer filteret.

**En side som sender hundrevis av spørringer.** Hver spørring i sporingen tar et millisekund, og siden bruker likevel sekunder. En ORM som laster relaterte poster etter behov (lazy loading), med én spørring per foreldrerad, er den vanlige årsaken, kjent som N+1-mønsteret. Last de relaterte radene i én spørring, med en join eller med `WHERE id = ANY($1)` og en matrise med nøkler, og følg med på antallet setninger per forespørsel like nøye som varigheten til hver enkelt.

**Den siste enheten solgt to ganger.** To forespørsler leser den samme lagerbeholdningen, begge avgjør at det er nok, og begge skriver. Lesing og skriving i separate setninger lar det stå en åpning mellom sjekken og handlingen, og samtidig trafikk finner den. Lukk åpningen inne i spørringen, med en betinget UPDATE og RETURNING som over, eller med SELECT … FOR UPDATE holdt så kort som mulig og uten noe nettverkskall mellom låsen og commiten.

**En diagnose som endret dataene.** Noen kjører EXPLAIN ANALYZE på en treg DELETE for å se planen, og radene er borte. ANALYZE utfører setningen på ordentlig og kaster bare utdataene, så hver sideeffekt skjer. Pakk enhver setning som endrer data, inn i BEGIN og ROLLBACK før du analyserer den, eller bruk vanlig EXPLAIN, som planlegger uten å kjøre, når estimatene holder.

## Les mer

- SELECT — hele syntaksen for en spørring i PostgreSQL, med rekkefølgen leddene behandles i.
- 7.1. Overview — starten på kapittelet om spørringer i håndboken, med definisjonen av ordet.
- 51.1. The Path of a Query — veien fra klientens tekst gjennom parser, omskriver og planlegger til utføreren.
- 14.1. Using EXPLAIN — hvordan du leser en plan, hva kostnadsenhetene betyr, og hvorfor ANALYZE kjører setningen.
- F.32. pg_stat_statements — track statistics of SQL planning and execution — utvidelsen som rangerer spørringer etter hva de koster totalt.
- 13.2. Transaction Isolation — hva en annen skrivende transaksjon ser under READ COMMITTED, og hvorfor WHERE-leddet sjekkes på nytt.

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 en spørring?» 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/query/**