Engineer

Database

et datalager ordnet slik at programmer kan spørre i det

Hva det er

En database er data holdt i en fast struktur, med regler for hva som får komme inn, slik at programvare kan finne, endre og kontrollere innholdet. Ordet, ofte forkortet DB, brukes også løst om programmet som tar vare på dataene; det presise navnet på det er databasehåndteringssystem, eller DBMS. NISTs ordliste gjengir to definisjoner, og den videste av dem beskriver et lager av informasjon eller data som kan være et tradisjonelt relasjonsdatabasesystem, men som ikke trenger å være det.

Ordet gjør tre jobber, og en samtale går i ring når to personer mener hver sin:

- **Dataene selv:** kundene, ordrene og sensoravlesningene en virksomhet tar vare på.
- **Motoren som lagrer dem:** PostgreSQL, SQL Server, SQLite.
- **Én beholder blant flere inne i en server.** PostgreSQL-veiledningen stiller opp nivåene: rader med navngitte, typede kolonner utgjør tabeller, tabeller samles i databaser, og databasene én serverinstans forvalter sammen, utgjør en databaseklynge. SQL Server trekker det samme skillet mellom en instans og databasene den huser.

«Databasen er treg» kan altså bety én tabell, én beholder eller en hel maskin, og det lønner seg å spørre hvilken før noen begynner å feilsøke.

De fleste motorene i daglig bruk er relasjonelle: de lagrer data som relasjoner, det matematiske navnet på tabeller, og tar imot spørsmål på SQL. PostgreSQL-veiledningen nevner også andre ordninger. Filene og katalogene i et Unix-lignende system danner en hierarkisk database, og objektorienterte databaser

kom til senere. Redis kaller seg en data structure server: i stedet for tabeller tilbyr den strenger, hasher, lister, mengder og andre innebygde typer, og den brukes til caching, køer og hendelsesbehandling.

Motorene skiller seg også i hvor de kjører. En klient/server-motor er en prosess som går hele tiden, og som mange programmer når over et nettverk. En innebygd motor er et bibliotek inne i ett program. SQLite holder en hel database, med tabeller, indekser, triggere og visninger, i én vanlig fil, og prosjektet sier selv at det konkurrerer med `fopen()` og ikke med serverdatabaser. Forskjellen synes i hvordan et program kobler seg til:

```
# klient/server: koble til en tjeneste som allerede kjører
psql "host=db.internal dbname=shop user=app"

# innebygd: programmet åpner en fil, og motoren kjører inne i det
sqlite3 shop.db
```

En tredje familie er bygd for analyse. BigQuery lagrer hver kolonne for seg, der mange tradisjonelle systemer holder feltene i en rad samlet på disk. Radlagring passer når du slår opp én post; kolonnelagring passer når du summerer ett felt over millioner av rader, fordi spørringen leser den ene kolonnen og lar resten ligge.

Bruk begrepet i en beslutning

Bruk «Hva er en database?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. En struktur som samsvarer med hvordan virksomheten faktisk fungerer, slik at spørsmålene du trenger svar på senere, fortsatt er mulige å stille.

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. Designet og driftet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tvers av tre databasemotorer

Designet og driftet 6 ETL/ELT-pipelines Ta kontakt →

Hvorfor det er viktig

Filer holder også data. Det en database legger til, er løfter om de dataene i det øyeblikket mange programmer bruker dem samtidig.

Det første løftet er et konsistent bilde under samtidig bruk. PostgreSQL gir hver setning et øyeblikksbilde av dataene slik de sto på et bestemt tidspunkt, en utforming som kalles multiversion concurrency control, eller MVCC. Lesing blokkerer dermed aldri skriving, og skriving blokkerer aldri lesing. SQLite tillater så mange lesere du vil samtidig og én skriver om gangen, mens de andre skriverne venter på tur. De to passer for ulikt antall skrivere, og nettopp det antallet gjør mye av valget for deg.

Det andre løftet er at dataene beholder formen sin. Et skjema sier hvilke kolonner en tabell har og hvilken type hver av dem holder; nøkler og begrensninger sier hvilke rader som får finnes. Når databasen håndhever reglene, følger hvert program som skriver til den, dem. Når reglene bare bor i applikasjonskoden, etterlater det første programmet som hopper over dem rader som hver senere spørring må forsvare seg mot.

Det tredje er et felles språk, med dialekter. SQL er standarden ISO/IEC 9075, sist revidert som SQL:2023, men vedlegget om samsvar i PostgreSQL-dokumentasjonen påpeker at intet system i dag hevder fullt samsvar med kjernen, Core. Et skjema og spørringene mot det hører til motoren sin, og å flytte dem til en annen er en migrering med egen testing.

SQLite-dokumentasjonen gjør valget av motor til en kort sjekkliste. Data som ligger over et nettverk fra koden som sender SQL-en, mange prosesser som skriver i samme øyeblikk, eller innhold på vei mot terabyteområdet peker hver for seg mot klient/server; ellers er SQLite det enkleste svaret. Rapportering i stor skala legger til ett spørsmål til: om tunge lesinger hører hjemme på en egen analysekopi, fylt av ETL, slik at en lang rapport aldri konkurrerer med spørringene kundene venter på.

Slik ser det ut i praksis

Databasene i arbeidet her spenner fra en samling på over 100 til én enkelt fil.

I den store enden opprettet, forvaltet og optimaliserte en rolle som databaseingeniør over 100 databaser i PostgreSQL, Microsoft SQL Server og Google BigQuery i et raskt voksende teknologiselskap. De fleste lagret GIS-data og tidsseriedata som sensoravlesninger, logger og sanntidsmetrikker. Arbeidet ga normaliserte og denormaliserte skjemaer for PostgreSQL og SQL Server, med romlig indeksering (PostGIS på PostgreSQL-siden) og tidsseriepartisjonering, mens BigQuery bar store mengder tidsseriedata i tidspartisjonerte, klyngede

tabeller. Med indeksering, materialiserte visninger og caching på plass gikk svartidene for spørringer mot GIS- og tidsseriedata ned med 40–60 %, og automatisert overvåking reduserte nedetiden med 50 %.

På et dokumentert stadium i utviklingen lagret et mindre system faktaene bak en CV, en referanseliste, en portefølje og et søknadsbrev i én SQLite-fil på tre språk. Databasen hadde tjuetre tabeller, og én kommando bygde den fra en tom fil gjennom nitten ordnede trinn som hvert var trygt å gjenta. Førtien Python-moduler, 10 580 linjer til sammen, gjenga seks utdataformater fra den. En profil valgte hvilke resultater som ble vist og i hvilken rekkefølge, så et dokument rettet mot én type leser var, med sidens egne ord, «en spørring og ikke en kopi». Prisen var at hvert dokument deretter måtte lages gjennom denne databasen. Disse tallene beskriver det dokumenterte stadiet i utviklingen, ikke dagens kodelager eller språkene det støtter.

Mellom de to størrelsene ligger spørsmålet hver travel database før eller siden møter, og notatet om å optimalisere før du sharder svarer med en rekkefølge: se, stram inn, ta lesingene bort fra primærnoden, partisjoner tabellene rundt det hete punktet, og shard først når én maskin er full av godt arbeid. Å se betyr å spørre PostgreSQL selv: `pg_stat_statements` for setningene som koster mest totalt, deretter `EXPLAIN (ANALYZE, BUFFERS)` for hva den dyreste av dem faktisk gjør.

Hvor det går galt

Det meste av databasetrøbbel starter i en beslutning tatt tidlig, da dataene var små og beslutningen så gratis ut.

To kopier av samme faktum glir fra hverandre. CV-generatoren over finnes fordi fakta holdt i fire dokumenter blir rettet fire ganger, helt til to av dem sier forskjellige ting og ingenting forteller hvilket som gjelder. Det samme skjer med en tabell som eksporteres til et regneark og redigeres der, eller med en tjeneste nummer to som har sin egen kundeliste. Gi hvert faktum ett lager som eier det, og la alle andre kopier si hvor de kom fra og når. Cachen i notatet følger denne regelen strengt: den kan fortelle en kjøper at en vare er fullt reservert, men hvert ja kommer fortsatt fra databasen.

Flere maskiner kommer før noen har lest planene. Å dele data over flere servere sprer de trege setningene over mer maskinvare uten å gjøre noen av dem billigere, og det legger en varig regning på alt annet. Slik notatet lister det

opp, kjøres hver migrering, sikkerhetskopi, gjenoppretting og rapport én gang per shard, og en join på tvers av shards blir applikasjonskode. Les `pg_stat_statements` og planene først, og ta vare på det de viser som grunnlinje.

En lesing havner på en kopi som henger etter. En strømmende replika tar lesinger bort fra primærnoden og ligger litt bak den. En bruker som nettopp har skrevet noe og så leser det tilbake fra replikaen, kan få se den gamle verdien. Les en brukers egne ferske skrivinger fra primærnoden.

Rapporter og kunder deler én motor. En rapport som skanner to års historikk, holder på minne og disk som butikkens egne spørringer trenger, og det merker de som handler. Flytt tung analyse til en kopi i et datavarehus, og la den operative databasen få konsentrere seg om de mange små lesingene og skrivingene den er der for.

Les mer

- NIST glossary: database — to korte definisjoner fra NIST-publikasjoner, der den ene åpner for databaser som ikke er relasjonelle.
- PostgreSQL tutorial: Concepts — tabeller, rader, kolonner, databaser og databaseklyngen, samlet på én side.
- PostgreSQL: Introduction to MVCC — hvordan øyeblikksbilder lar lesere og skrivere arbeide uten å blokkere hverandre.
- PostgreSQL: SQL Conformance — delene og versjonene av ISO/IEC 9075, og hva samsvar med Core innebærer.
- Appropriate Uses For SQLite — hvor en innebygd motor passer, hvor en klient/server-motor passer bedre, og en sjekkliste for å velge.
- Overview of BigQuery storage — rad- og kolonneorientert lagring, og hvorfor analyse favoriserer den siste.

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 database?» 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/database/