Engineer

PostgreSQL

🐘 databasen det meste av dette arbeidet ble bygd på

Hva det er

PostgreSQL er en relasjonsdatabaseserver med fri og åpen kildekode. Den holder data i tabeller, svarer på spørsmål skrevet i SQL og gjør hver transaksjon til alt eller ingenting. De fleste som jobber med den, kaller den Postgres, og prosjektet regner begge navnene som sine. Spør du en ingeniør som bruker den daglig, «Hva er PostgreSQL?», får du et nøkternt svar: det er den generelle databasen et nytt system starter på når ingen har en konkret grunn til å velge en annen.

Den er eldre enn det meste av programvaren den nå ligger under. POSTGRES-prosjektet startet ved University of California i Berkeley under professor Michael Stonebraker, og implementeringen begynte i 1986. I 1994 la Andrew Yu og Jolly Chen til en SQL-tolk og ga ut resultatet som Postgres95. To år senere fikk det navnet PostgreSQL, og versjonsnumrene startet på nytt på 6.0 for å fortsette rekken Berkeley hadde begynt på. Prosjektet beskriver selv systemet som objektrelasjonelt: utover tabeller og joins kan en bruker legge til nye datatyper, funksjoner, operatorer, aggregater, indeksmetoder og prosedyrespråk, og serveren behandler dem som sine egne.

Arkitektonisk er den en klient og en server. Én serverprosess eier datafilene, og hver klienttilkobling får sin egen prosess, forket for akkurat den tilkoblingen. Applikasjoner sender SQL over tilkoblingen gjennom en driver for språket de er skrevet i. Samtidigheten bygger på MVCC (multiversion concurrency control): en oppdatering skriver en ny versjon av raden i stedet for å overskrive den gamle, så hver setning leser et konsistent øyeblikksbilde. Lesere venter aldri på skrivere, og skrivere venter aldri på lesere.

Hvor står den blant alternativene? Ved siden av MySQL og MariaDB som de andre relasjonsserverne med åpen kildekode, ved siden av Oracle og Microsoft SQL Server som de kommersielle, ved siden av SQLite som den innebygde databasen

som bor inne i ett program, og opp mot dokumentlagre som MongoDB, som holder poster uten en fast struktur.

Bruk begrepet i en beslutning

Bruk «Hva er PostgreSQL?» 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

Valget av database er en beslutning et team lever med i årevis, fordi dataene lever lenger enn koden som er skrevet mot dem. PostgreSQL er standardvalget man sjelden angrer på, av tre grunner en kjøper kan sjekke selv.

Den første er kostnad og innlåsing. Den er utgitt under PostgreSQL License, en kort og tillatende lisens i samme familie som BSD og MIT, så det finnes ingen avgift per prosessorkjerne, ingen lisensrevisjon og ingen leverandør som kan endre vilkårene. Amazon, Microsoft og Google selger den hver for seg som administrert tjeneste, og den samme databasen kjører på en bærbar maskin. Å flytte mellom leverandører blir dermed en flytting av data, ikke en omskriving av applikasjonen.

Den andre er bredde. Én server dekker relasjonstabeller, JSON-dokumenter i `jsonb` -kolonner, fulltekstsøk, geospatiale spørringer gjennom utvidelsen PostGIS og planlagte jobber gjennom `pg_cron` . Hver av dem er et eget produkt et team slipper å drifte, sikre, ta sikkerhetskopi av og ansette folk til. For de fleste produkter avgjøres PostgreSQL mot MySQL her: til vanlige lesinger og skrivinger fungerer begge godt, og forskjellen viser seg når arbeidet trenger rikere typer, strengere regler eller en utvidelse.

Den tredje er levetid. Prosjektet gir ut én hovedversjon i året og støtter hver av dem i fem år, så en oppgradering kan planlegges i stedet for å bli tvunget fram av en krise. PostgreSQL 18, utgitt i september 2025, fikk asynkron I/O og en innebygd funksjon for tidsordnede UUID-er, og SQL skrevet for utgaver som er ti år eldre, kjører som regel uendret på den.

Avveiningene er reelle. En prosess per tilkobling gjør tilkoblinger dyre, så en travel applikasjon trenger en tilkoblingspooler. Skalerbarheten kommer lettest fra én større maskin, ikke fra flere: å spre skrivinger over mange servere betyr sharding, replikering eller en utvidelse, og hver av dem gir mer å drifte. Og standardinnstillingene er tilpasset en liten maskin, så en seriøs installasjon må konfigureres med vilje.

Slik ser det ut i praksis

Mange av prosjektene på dette nettstedet kjører på PostgreSQL, og tre av dem viser tre forskjellige måter å lene seg på den.

På én plattform ble databasen hele datalaget. Hver lesing og skriving går gjennom én av 1 275 PostgreSQL-funksjoner fordelt på 34 skjemaer, og API-et i Go har ingen direkte tilgang til noen tabell. Rollen API-et logger inn som, `mariner`, har EXECUTE på funksjonene og USAGE på skjemaene og ingenting annet, omtrent 4 000 eksplisitte tildelinger. Funksjonene kjører som SECURITY DEFINER under en egen eierrolle uten innlogging og med fast `search_path`. Tilgangsgrensen håndheves av Postgres selv; prisen er at et nytt felt betyr en migrering og en funksjonsendring.

I en applikasjon med leietakere, steder og daglige strømmålinger handlet arbeidet om justering, ikke design, og oppdraget var 10 ganger raskere uten å bygge om arkitekturen. GiST-indekser for de romlige dataene, sammensatte indekser på leietaker og datointervall, intervallpartisjonering av strømtabellen per dag og måned, og spørringer som ble skrevet om etter at planene deres var lest med `EXPLAIN ANALYZE`, ga 10 ganger bedre svartider og 40 % lavere ressursbruk.

Og på en plattform med tregt bakgrunnsarbeid bor jobbkøen i den samme databasen. River holder køen sin i PostgreSQL, så en jobb og radene den berører, committes eller rulles tilbake sammen. Bak den står 15 worker-moduler, 8 planlagte oppgaver og 20 `pg_cron`-jobber som rydder i partisjoner, roterer salter og oppdaterer aggregater. Siden sier grensen rett ut: en kø i Postgres slutter å være riktig svar ved et visst volum, og én bevegelig del mindre var verdt mer enn spillerom ingen kom til å bruke.

Hvor det går galt

PostgreSQL mister sjelden data. Den overrasker likevel folk, og overraskelsene kommer i noen få kjente former.

Tabellen vokser mens antallet rader står stille. En `UPDATE` eller `DELETE` etterlater den gamle radversjonen til `VACUUM` fjerner den, fordi en annen transaksjon fortsatt kan lese den. En transaksjon som står åpen i timevis, holder igjen oppryddingen for hele databasen, og standardinnstillingene for autovacuum passer små tabeller. Lukk transaksjoner som står i tomgang, juster autovacuum per tabell for de store, og partisjoner data som går ut på dato, slik at gamle måneder droppes i stedet for å slettes. Vacuum har også en annen jobb: transaksjons-ID-er er 32 bit, hver tabell må vacuumes minst én gang per to milliarder transaksjoner, og en server som kommer for langt etter, nekter nye skrivinger for å beskytte dataene.

Minnet tar slutt ved den første trafikktoppen. Hver tilkobling er en serverprosess med eget minne, og standardgrensen er 100 tilkoblinger. Tjue applikasjonsinstanser med en pool på femti hver ber om tusen prosesser. Sett en pooler som PgBouncer foran, og dimensjoner poolen til hver applikasjon etter arbeidet, som oftest en håndfull tilkoblinger per instans.

En spørring som tok millisekunder, begynner å ta sekunder. Planleggeren valgte en annen plan etter hvert som dataene vokste, ofte fordi statistikken var blitt utdatert eller indeksene var bygd for fjorårets spørringer. Les planen før du endrer noe, oppdater statistikken med `ANALYZE`, og indekser det predikatet spørringen faktisk filtrerer på:

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT id, total FROM orders
WHERE customer_id = 42 AND placed_at >= '2026-01-01';
```

Innsettinger går tregere etter hvert som tabellen blir stor. Tilfeldige UUID-er som primærnøkler havner overalt i B-tre-indeksen, så hver innsetting treffer en ny side, og indeksen fragmenteres. Tidsordnede identifikatorer unngår dette: én plattform gjorde UUID v7 til standardnøkkel med `uuidv7()` i PostgreSQL 18, og nye rader sorteres inn i indeksen i stedet for å bli strødd utover den.

Les mer

- What Is PostgreSQL? — prosjektets egen definisjon, med egenskapene og utvidelsespunktene den nevner.
- A Brief History of PostgreSQL — Berkeley, Postgres95 og navnebyttet, fortalt av dem som gjorde det.
- Concurrency Control: Introduction — hvordan øyeblikksbilder lar lesing og skriving gå uten at de blokkerer hverandre.
- Routine Vacuuming — døde radversjoner, autovacuum og omløp av transaksjons-ID-er, grundig forklart.
- Using EXPLAIN — hvordan du leser en spørringsplan, med gjennomgåtte eksempler.
- Versioning Policy — den årlige hovedversjonen og de fem årene med støtte bak hver av dem.

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 PostgreSQL?» 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/postgresql/