Engineer

Databaseskjema

en navngitt gruppe tabeller i én database

Hva det er

Et databaseskjema er et navnerom inne i en database. Det rommer tabeller, visninger, datatyper, funksjoner og operatorer, slik at samme navn kan brukes i to skjemaer uten at de kolliderer. PostgreSQL-manualen sammenligner skjemaer med kataloger i et filsystem, bortsett fra at de ikke kan nøstes: en database holder skjemaer, og skjemaene holder objekter. Ordet har også en annen og løsere betydning, som er like vanlig: skjemaet til en database er hele strukturen, med hver tabell, kolonne, nøkkel og skranke. Begge betydningene dukker opp nedenfor, og flertallet, skjemaer, betyr nesten alltid navnerommene. På engelsk heter det schema, og mange norske utviklere sier det slik.

En faktureringsdel av en applikasjon kan bo i sitt eget skjema, og en økt kan velge hvilke skjemaer den leter i:

```
CREATE SCHEMA billing;

CREATE TABLE billing.invoices (
  id     bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  total  numeric(12, 2) NOT NULL
);

SHOW search_path;                  -- "$user", public
SET search_path = billing, public;
SELECT count(*) FROM invoices;     -- now resolves to billing.invoices
```

Et navn skrevet med skjemaet foran, som `billing.invoices`, betyr alltid den samme tabellen. Et navn uten skjema slås opp gjennom search_path, en liste over skjemaer serveren går gjennom i rekkefølge til den finner et treff. Standardlisten er `"$user", public`: først et skjema med samme navn som den gjeldende rollen, hvis det finnes, deretter public, som hver ny database starter med. Det første

eksisterende skjemaet i listen er også der CREATE TABLE legger en tabell som oppgis uten skjema, og det er derfor objekter havner i public med mindre noen sier noe annet. Et navn som ikke finnes i noen av skjemaene i listen, gir en feil, selv om et annet skjema i den samme databasen har det.

Bruk begrepet i en beslutning

Bruk «Hva er et databaseskjema?» 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

Den første grunnen er orden. En database med noen hundre tabeller trenger et nivå over tabellen, og skjemaene gir det: fakturering i ett, kontoer i et annet, tabellene til et tredjepartsverktøy i et tredje, unna alle andres navn. En klienttilkobling når bare én database, men skjemaene er ikke vanntette rom: en rolle kan bruke objekter i alle skjemaene i databasen den er koblet til, så sant privilegiene tillater det, og én spørring kan koble sammen tabeller fra flere av dem.

Den andre grunnen er tilgangskontroll. USAGE på et skjema lar en rolle slå opp objektene i det, og CREATE lar den legge til nye. Tar du bort USAGE, stenges tilgangen til objektene, men navnene deres kan fortsatt leses fra systemkatalogene. Et skjema verner altså dataene, men skjuler ikke hva som finnes. `CREATE SCHEMA name AUTHORIZATION role` gir et nytt skjema til en rolle, og navn som begynner på `pg_`, er reservert for systemet.

Den tredje grunnen er sikkerhet, fordi skjemaene er med på å avgjøre hva et navn betyr. De fleste navn i spørringer skrives uten skjema, og med manualens ord betyr det å legge et skjema inn i search_path i praksis at du stoler på hver rolle som kan opprette objekter der. En slik rolle kan få en helt vanlig spørring til å kjøre funksjoner den selv har valgt, som om den som kjørte spørringen, hadde kalt dem. De sikre mønstrene holder skjemaer som andre kan skrive til, utenfor søkestien til en bruker, f.eks. med ett privat skjema per bruker, opprettet med `CREATE SCHEMA alice AUTHORIZATION alice`, som `"$user"` i listen da finner først.

PostgreSQL 15 flyttet nye databaser i den retningen: versjonen tok fra hver rolle retten til å opprette objekter i public og gjorde databaseeieren, gjennom rollen pg_database_owner, til eier av public.

Til slutt kommer flyttbarhet. SQL-standarden har skjemaer, men ikke noe public, og manualen nevner at enkelte SQL-systemer kanskje ikke har skjemaer i det hele tatt. Fullt kvalifiserte navn tåler en flytting bedre enn kode som lener seg på search_path.

Slik ser det ut i praksis

Datamodellen bak 348 tabeller fordelt på 34 Postgres-skjemaer bruker skjemaer i den første betydningen, som nivået over tabellen. Det maritime domenet, med fagfolk, selskaper, skip, stillinger og omtaler, er lagt opp som normaliserte PostgreSQL-skjemaer ordnet etter domene: 348 tabeller fordelt på 34 av dem. Tabellnavnene står i flertall inne i skjemaer med navn i entall, uten unntak, og det unngår på kjøpet mange kollisjoner med reserverte ord. Ekte fremmednøkler og skranker holder relasjonene mellom dem oppe. Å legge til en funksjon betyr som regel å utvide skjemaet langs de linjene som allerede finnes, og siden sier at nettopp det er grunnen til at 34 skjemaer er en struktur og ikke et villniss.

Et skjema kan også være grensen mellom to oppgaver. I laget for betaling og tilgangsrettigheter møtes kortbetalinger via Stripe og kjøp i appen hos Apple og Google i et betalingsskjema med 9 tabeller og 34 funksjoner. Et eget tilgangsskjema med 11 tabeller og 34 funksjoner svarer på «har denne kontoen lov til å gjøre dette?» uten å vite hvilken salgskanal som betalte. For hver forespørsel slår serveren opp i tilgangsskjemaet på nytt, og det oppslaget alene avgjør om kontoen slipper til i bedriftskatalogen, søket og dataeksporten. Prisen er, som siden sier, to skjemaer der et mindre produkt ville klart seg med ett, pluss et rettighetsoppslag på forespørsler som ellers ville vært gratis.

Den andre betydningen av ordet er temaet for 1 022 migreringer bak en port for skjemaavvik. De fleste prosjekter beskriver et skjema to ganger, i oppsettet som bygger det fra bunnen av og i migreringene som fikk det til å vokse, og de to glir fra hverandre med mindre noe sjekker dem. Her har skjemaet vokst gjennom 1 022 migreringer, fra nummer 036 til 1102, og en CI-jobb bygger databasen to ganger ved hver endring, én gang fra start og én gang med alle migreringene spilt av i rekkefølge, og sammenligner så de to: startdataene i tillegg til strukturen. En migrering som fyller en oppslagstabell feil, gjør like mye skade

som en som glemmer en kolonne, og bare én av dem synes i en skjemadiff. Ethvert avvik stopper byggingen med forskjellen skrevet ut, så hver migrering må virke både når den spilles av på nytt og fra kald start.

Hvor det går galt

En funksjon plantet i et delt skjema. PostgreSQL-wikiens gjennomgang av CVE-2018-1058 viser angrepet: én bruker oppretter `lower(varchar)` i public. Den innebygde `lower` ligger i pg_catalog og tar text, så for en varchar-kolonne er den plantede funksjonen det nærmeste treffet, og en kollega som kjører `SELECT lower(full_name) FROM a`, kjører den plantede koden i stedet. Kvalifiser kall i følsom kode som `pg_catalog.lower`, ta CREATE på public fra rollene som klarer seg uten, og list opp det som allerede ligger der med `\df public.*`.

En oppgradering som tok med seg de gamle rettighetene. Den tryggere standarden i PostgreSQL 15 gjelder nye klynger og nyopprettede databaser. En oppgradert klynge eller en gjenopprettet dump beholder rettighetene public allerede hadde, så en server på versjon 15 eller nyere kan fortsatt la hver rolle opprette objekter der. Manualen kaller standardstien med CREATE tildelt på public et mønster som aldri er sikkert, og nevner at det er standarden i databaser oppgradert fra 14 eller eldre. Kjør i en slik database `REVOKE CREATE ON SCHEMA public FROM PUBLIC`, og gå deretter gjennom public etter objekter med samme navn som dem i pg_catalog.

En tabell opprettet på feil sted. En migrering oppretter `invoices` uten skjema, og tabellen havner i det første eksisterende skjemaet i søkestien til rollen som kjører den. Applikasjonen, som kobler til med en annen søkesti, leser da en annen tabell eller får `relation "invoices" does not exist`. Kvalifiser hvert navn i migreringene, og sjekk `SELECT current_schema()` i økten som kjører dem.

En DROP som når forbi skjemaet. `DROP SCHEMA billing` nekter så lenge skjemaet inneholder noe, fordi RESTRICT er standard. Med CASCADE fjernes hvert objekt i skjemaet og hvert objekt som avhenger av dem, og manualen advarer om at dette kan fjerne objekter i andre skjemaer enn det som ble nevnt. Rådet er å kjøre DROP uten CASCADE først og lese DETAIL-linjene, som lister opp hva som ville forsvinne.

Et navn systemet finner først. Når pg_catalog er utelatt fra search_path, søker systemet implisitt der før det søker i skjemaene som står i stien. Når det er nevnt eksplisitt, bestemmer plasseringen i stien søkerekkefølgen. En egen tabell med

et navn som begynner på `pg_` , kan bli skygget når en senere versjon legger til en systemtabell med samme navn, og manualen fraråder derfor slike navn. Reserverte ord som `user` og `order` gir en mildere variant av samme bry: en tabell med et slikt navn trenger doble anførselstegn i hver eneste setning.

Les mer

- 5.10. Schemas — manualens kapittel: kvalifiserte navn, public, search_path, sikre bruksmønstre og flyttbarhet.
- CREATE SCHEMA — syntaksen, AUTHORIZATION-leddet, det reserverte prefikset `pg_` og hvor PostgreSQL avviker fra standarden.
- 5.8. Privileges — hva USAGE og CREATE betyr på et skjema, side om side med alle de andre privilegiene.
- E.20. Release 15 — versjonen som tok CREATE på public fra nye databaser og lot den bli igjen i oppgraderte.
- A Guide to CVE-2018-1058: Protect Your Search Path — prosjektets egen gjennomgang av angrepet med plantede funksjoner, med rettelsene og kommandoene for å gå gjennom en database.

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 et databaseskjema?» 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/schema/