Engineer

Databasschema

en namngiven grupp tabeller inuti en databas

Vad det är

Ett databasschema, i dagligt tal bara schema, är en namnrymd inuti en databas: en behållare för tabeller, vyer, datatyper, funktioner och operatorer, så att samma namn kan användas i två scheman utan att de krockar. PostgreSQL-manualen jämför scheman med kataloger i ett filsystem, med skillnaden att de inte kan nästlas: en databas rymmer scheman, och scheman rymmer objekt. Ordet har också en andra, lösare betydelse som är minst lika vanlig: en databas schema är hela dess struktur, varje tabell, kolumn, nyckel och villkor. Båda betydelserna förekommer nedan, och pluralformen scheman syftar nästan alltid på namnrymderna.

En faktureringsdel av en applikation kan bo i ett eget schema, och en session kan välja vilka scheman den letar 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;     -- pekar nu på billing.invoices
```

Ett namn som skrivs med sitt schema, som `billing.invoices`, betyder alltid samma tabell. Ett namn utan schema slås upp via search_path, en lista med scheman som servern går igenom i ordning tills den hittar en träff. Standardlistan är `"$user", public`: först ett schema med samma namn som den roll sessionen kör som, om ett sådant finns, sedan schemat public som varje ny databas börjar

med. Det första existerande schemat i sökvägen är också där CREATE TABLE lägger en tabell vars namn saknar schema, och därför hamnar objekt i public om ingen säger något annat. Ett namn som inte finns i något av de listade schemana ger ett fel, även om ett annat schema i samma databas har det.

Använd begreppet i ett beslut

Använd ”Vad är ett databasschema?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. En struktur som speglar hur verksamheten faktiskt fungerar, så att de frågor du behöver svar på senare fortfarande går att ställa.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Utformade och drev 6 ETL/ELT-pipelines — 6 ETL-pipelines över tre databasmotorer

Utformade och drev 6 ETL/ELT-pipelines Hör av er →

Varför det spelar roll

Ordning är det första skälet. En databas med några hundra tabeller behöver en nivå ovanför tabellen, och scheman ger den: fakturering i ett, konton i ett annat, ett tredjepartsverktygs tabeller i ett tredje, på behörigt avstånd från alla andras namn. En klientanslutning når en enda databas, men scheman är inte strikt åtskilda: en roll kan använda objekt i vilket schema som helst i den databas den är ansluten till, om privilegierna tillåter det, och en SQL-fråga kan joina tabeller från flera av dem.

Åtkomstkontroll är det andra. USAGE på ett schema låter en roll slå upp objekten i det, och CREATE låter den lägga till nya. Tas USAGE bort spärras åtkomsten till objekten, men deras namn går fortfarande att läsa i systemkatalogerna, så ett schema skyddar datan men inte vetskapen om vad som finns. `CREATE SCHEMA name AUTHORIZATION role` lämnar över ett nytt schema till en roll, och namn som börjar med `pg_` är reserverade för systemet.

Det tredje skälet är säkerhet, eftersom scheman deltar i namnuppslagningen. De flesta namn i SQL-frågor skrivs utan schema, och därför innebär det, med manualens ord, att lägga till ett schema i search_path i praktiken att lita på varje roll som kan skapa objekt i det. En sådan roll kan få en helt vanlig SQL-fråga att köra funktioner som rollen själv har valt, som om den som körde frågan hade anropat dem. De säkra mönstren håller scheman som andra kan skriva i borta från användarens sökväg, t.ex. ett privat schema per användare, skapat med `CREATE SCHEMA alice AUTHORIZATION alice`, som posten `"$user"` sedan hittar först.

PostgreSQL 15 flyttade nya databaser åt det hållet: versionen tog bort rätten för varje roll att skapa objekt i public och gjorde databasägaren, via rollen pg_database_owner, till ägare av public.

Portabilitet är den sista punkten. SQL-standarden har scheman men inget public-schema, och manualen påpekar att vissa SQL-system kanske inte implementerar scheman alls. Kvalificerade namn flyttar lättare mellan system än kod som lutar sig mot search_path.

Så ser det ut i praktiken

Datamodellen bakom 348 tabeller i 34 Postgres-scheman använder scheman i den första betydelsen, som nivån ovanför tabellen. Sjöfartsdomänen, med yrkesverksamma, företag, fartyg, jobb och recensioner, är uppbyggd som normaliserade PostgreSQL-scheman organiserade per domän: 348 tabeller fördelade på 34 av dem. Tabellnamnen står i plural inuti scheman med namn i singular, utan undantag, vilket också undviker en mängd kollisioner med reserverade ord, och riktiga främmande nycklar och villkor håller ihop relationerna. Att lägga till en funktion innebär oftast att schemat byggs ut längs de befintliga linjerna i stället för mot dem, och enligt sidan är det enda skälet till att 34 scheman blir en struktur och inte ett virrvarr.

Ett schema kan också vara gränsen mellan två uppgifter. I åtkomsträttigheter via Stripe och appköp strålar kortbetalningar via Stripe och köp i appen hos Apple och Google samman i ett betalningsschema med 9 tabeller och 34 funktioner. Ett separat åtkomstschema med 11 tabeller och 34 funktioner svarar på ”får det här kontot göra det här?” utan att veta vilken försäljningskanal som betalade. Servern hämtar svaret ur åtkomstschemat på nytt för varje anrop, och det styr åtkomsten till företagskatalogen, sökningen och dataexporten. Kostnaden är enligt sidan två scheman där en mindre produkt skulle nöja sig med ett, plus ett behörighetsuppslag på anrop som annars hade varit gratis.

Den andra betydelsen av ordet är ämnet för 1 022 migreringar bakom en grind mot schemadrift. De flesta projekt beskriver ett schema två gånger, i den initiala uppsättningen som bygger det från grunden och i de databasmigreringar som har fått det att växa, och de två glider isär om inget kontrollerar dem. Här har schemat vuxit migrering för migrering, från nummer 036 till 1102, och ett CI-jobb bygger det två gånger vid varje ändring, en gång från grunden och en gång med alla migreringar körda i ordning, och jämför sedan de två: grunddatan lika väl som strukturen. En migrering som i efterhand fyller i en uppslagstabell fel gör lika

mycket skada som en som glömmer en kolumn, och bara den ena syns i en schemadiff. Varje avvikelse fäller bygget med skillnaden utskriven, så varje migrering måste fungera både när den spelas upp i ordning och från noll.

Var det går fel

En funktion planterad i ett delat schema. PostgreSQL-wikins guide till CVE-2018-1058 visar attacken: en användare skapar `lower(varchar)` i public. Den inbyggda `lower` ligger i pg_catalog och tar text, så för en varchar-kolumn är den planterade funktionen den närmare träffen, och en kollega som kör `SELECT lower(full_name) FROM a` kör den planterade koden i stället. Kvalificera anrop i känslig kod som `pg_catalog.lower`, ta bort CREATE på public från de roller som inte behöver det och lista det som redan finns där med `\df public.*`.

En uppgradering som behöll de gamla rättigheterna. Den säkrare standarden i PostgreSQL 15 gäller nya kluster och nyskapade databaser. En uppgradering av ett kluster eller en återställd dump bevarar de rättigheter som public redan hade, så en server på version 15 eller senare kan fortfarande låta varje roll skapa objekt där. Manualen kallar standardsökvägen med CREATE beviljat på public för ett mönster som aldrig är säkert, och påpekar att det är standard i databaser som har uppgraderats från 14 eller tidigare. Kör i en sådan databas `REVOKE CREATE ON SCHEMA public FROM PUBLIC` och granska sedan public efter objekt med samma namn som de i pg_catalog.

En tabell som skapas på fel ställe. En databasmigrering skapar `invoices` utan schema, och tabellen hamnar i det första existerande schemat på sökvägen för den roll som kör den. Applikationen, som ansluter med en annan sökväg, läser sedan en annan tabell eller får `relation "invoices" does not exist`. Kvalificera varje namn i migreringar och kontrollera `SELECT current_schema()` i den session som kör dem.

En DROP som når förbi schemat. `DROP SCHEMA billing` vägrar så länge schemat innehåller något, eftersom RESTRICT är standard. Med CASCADE tas varje objekt i schemat bort, liksom varje objekt som beror på dem, och manualen varnar för att det kan ta bort objekt i andra scheman än det som nämndes. Rådet är att först köra DROP utan CASCADE och läsa DETAIL-raderna, som listar vad som skulle försvinna.

Ett namn som systemet slår upp först. När pg_catalog inte anges i search_path söker systemet implicit där före schemana som räknas upp i sökvägen. Om det anges uttryckligen bestämmer dess plats i sökvägen sökordningen. En egen tabell vars namn börjar med `pg_` kan skuggas när en senare version lägger till en systemtabell med samma namn, så manualen avråder från sådana namn. Reserverade ord som `user` och `order` ger en mildare variant av samma problem: en tabell med ett sådant namn behöver dubbla citattecken i varje sats.

Läs mer

- 5.10. Schemas — manualens kapitel: kvalificerade namn, schemat public, search_path, säkra användningsmönster och portabilitet.
- CREATE SCHEMA — kommandots syntax, AUTHORIZATION-satsen, det reserverade prefixet `pg_` och var PostgreSQL avviker från standarden.
- 5.8. Privileges — vad USAGE och CREATE betyder på ett schema, bredvid alla andra privilegier.
- E.20. Release 15 — versionen som tog bort CREATE på public från nya databaser och behöll det för uppgraderade.
- A Guide to CVE-2018-1058: Protect Your Search Path — projektets genomgång av attacken med planterade funktioner, med åtgärderna och kommandona för att granska en databas.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med ett representativt dataset, frågorna det ska besvara och dagens flaskhals. Vi kan avtala mått på kvalitet, genomströmning och återställning.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är ett databasschema?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/schema/