Engineer

Schema

Namensraum für Objekte innerhalb einer Datenbank

Was es ist

Ein Schema ist ein Namensraum innerhalb einer Datenbank: ein Behälter für Tabellen, Views, Datentypen, Funktionen und Operatoren. Derselbe Name kann deshalb konfliktfrei in zwei Schemas vorkommen. Das PostgreSQL-Handbuch vergleicht Schemas mit Verzeichnissen eines Dateisystems. Sie lassen sich allerdings nicht verschachteln: Eine Datenbank enthält Schemas, Schemas enthalten Objekte. Das Wort hat eine zweite, ebenso gebräuchliche, lockerere Bedeutung: Das Schema einer Datenbank ist ihre gesamte Struktur mit jeder Tabelle, Spalte, jedem Schlüssel und jeder Integritätsbedingung. Beide Bedeutungen kommen unten vor. Der Plural Schemas meint fast immer die Namensräume.

Der Abrechnungsbereich einer Anwendung kann in einem eigenen Schema liegen, und eine Sitzung kann wählen, in welchen Schemas sie sucht:

```
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
```

Ein Name mit vorangestelltem Schema wie `billing.invoices` meint immer dieselbe Tabelle. Ohne Schema wird er über search_path gesucht: eine Liste von Schemas, die der Server bis zum ersten Treffer der Reihe nach durchläuft. Die Standardliste lautet `"$user", public` : ein nach der aktuellen Rolle benanntes

Schema, falls vorhanden, danach das Schema public, mit dem jede neue Datenbank beginnt. Im ersten vorhandenen Schema des Pfads legt CREATE TABLE auch eine Tabelle ab, deren Name kein Schema enthält. Deshalb landen Objekte in public, wenn niemand etwas anderes vorgibt. Ein in keinem aufgeführten Schema gefundener Name verursacht einen Fehler, selbst wenn ein anderes Schema derselben Datenbank ihn enthält.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Schema?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Eine Struktur, die zur tatsächlichen Arbeitsweise Ihres Unternehmens passt, damit sich auch spätere Fragen noch beantworten lassen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Der erste Grund ist Ordnung. Eine Datenbank mit einigen Hundert Tabellen braucht eine Ebene über der Tabelle. Schemas schaffen sie: Abrechnung in einem, Konten in einem anderen, die Tabellen eines Drittanbieterwerkzeugs in einem dritten, ohne die Namen der anderen zu belegen. Eine Clientverbindung erreicht eine einzige Datenbank. Schemas sind aber nicht starr getrennt: Erlauben es die Rechte, kann eine Rolle Objekte aus jedem Schema der verbundenen Datenbank verwenden, und eine Abfrage kann Tabellen mehrerer Schemas verknüpfen.

Der zweite Grund ist Zugriffskontrolle. USAGE auf einem Schema lässt eine Rolle die enthaltenen Objekte nachschlagen, CREATE lässt sie neue hinzufügen. USAGE zu entziehen blockiert den Zugriff auf die Objekte. Ihre Namen bleiben allerdings aus den Systemkatalogen lesbar, sodass ein Schema die Daten schützt und nicht das Wissen darum, was existiert. `CREATE SCHEMA name AUTHORIZATION role` macht die angegebene Rolle zur Besitzerin eines neuen Schemas. Mit `pg_` beginnende Namen sind dem System vorbehalten.

Der dritte Grund ist Sicherheit, weil Schemas an der Namensauflösung beteiligt sind. Die meisten Namen in Abfragen stehen ohne Schema. Mit den Worten des Handbuchs vertraut man durch Hinzufügen eines Schemas zu search_path praktisch jeder Rolle, die dort Objekte anlegen darf. Eine solche Rolle kann eine

gewöhnliche Abfrage Funktionen ihrer Wahl ausführen lassen, als hätte die abfragende Person sie aufgerufen. Sichere Muster halten Schemas, in die andere schreiben dürfen, aus dem Suchpfad eines Nutzers heraus. Ein Beispiel ist ein privates Schema je Nutzer, angelegt mit `CREATE SCHEMA alice AUTHORIZATION alice`, das der Eintrag `"$user"` dann zuerst findet. PostgreSQL 15 führte neue Datenbanken näher an dieses Muster heran: Es entzog das allgemeine Recht, in public Objekte anzulegen, und machte den Datenbankbesitzer über die Rolle pg_database_owner zum Besitzer von public.

Portabilität ist der letzte Punkt. Der SQL-Standard kennt Schemas, aber kein Schema public. Das Handbuch merkt an, dass manche SQL-Systeme möglicherweise überhaupt keine Schemas implementieren. Qualifizierte Namen lassen sich besser übertragen als Code, der auf search_path baut.

Wie es in der Praxis aussieht

Das Datenmodell in 348 Tabellen in 34 Postgres-Schemas modelliert verwendet Schemas in der ersten Bedeutung als Ebene über der Tabelle. Die maritime Domäne mit Fachkräften, Unternehmen, Schiffen, Stellen und Bewertungen ist in normalisierten, nach Domäne geordneten PostgreSQL-Schemas aufgebaut: 348 Tabellen in 34 Schemas. Tabellennamen stehen ausnahmslos im Plural, Schemanamen im Singular. Das vermeidet zugleich viele Kollisionen mit reservierten Wörtern. Echte Fremdschlüssel und Constraints sichern die Beziehungen. Eine neue Funktionalität hinzuzufügen heißt meist, das Schema entlang seiner vorhandenen Struktur zu erweitern, statt dagegen zu arbeiten. Laut Seite ist das der einzige Grund, warum 34 Schemas eine Struktur und kein Wildwuchs sind.

Ein Schema kann auch zwei Aufgaben voneinander abgrenzen. In Stripe und In-App-Käufe mit Berechtigungen verbunden laufen Stripe-Kartenzahlungen und In-App-Käufe von Apple und Google in einem Zahlungsschema mit 9 Tabellen und 34 Funktionen zusammen. Ein getrenntes Zugriffsschema mit 11 Tabellen und 34 Funktionen beantwortet „Darf dieses Konto das?“, ohne zu wissen, über welchen Store bezahlt wurde. Die bei jeder Anfrage auf dem Server erneut geprüfte Antwort steuert den Zugang zu Unternehmensverzeichnis, Suche und Datenexport. Der Preis sind laut Seite zwei Schemas, wo ein kleineres Produkt eines bräuchte, sowie eine Berechtigungsabfrage bei Anfragen, die sonst ohne diesen Aufwand auskämen.

Die zweite Bedeutung des Worts behandelt 1.022 Migrationen hinter einer Prüfung auf Schemaabweichungen. Die meisten Projekte beschreiben ein Schema zweimal: im Anfangsaufbau, der es von Grund auf erzeugt, und in den Migrationen, die es wachsen ließen. Ohne Prüfung entwickeln sie sich auseinander. Hier ist die Datenbank als 1.022 Migrationen von 036 bis 1102 versioniert. Ein CI-Job baut sie bei jeder Änderung zweimal: einmal frisch und einmal mit allen der Reihe nach eingespielten Migrationen. Danach vergleicht er beides, die vorbefüllten Daten ebenso wie die Struktur. Eine Migration, die Daten in einer Nachschlagetabelle falsch nachträgt, richtet ebenso viel Schaden an wie eine, die eine Spalte vergisst. Nur eines davon erscheint im Schemavergleich. Jede Abweichung lässt den Build mit ausgegebenem Unterschied scheitern. Deshalb muss jede Migration beim Einspielen der gesamten Folge und beim Aufbau von Grund auf funktionieren.

Wo es schiefgeht

Eine eingeschleuste Funktion in einem gemeinsamen Schema. Die Anleitung zu CVE-2018-1058 im PostgreSQL-Wiki zeigt den Angriff: Ein Nutzer legt `lower(varchar)` in public an. Das eingebaute `lower` liegt in pg_catalog und nimmt text entgegen. Für eine varchar-Spalte passt die eingeschleuste Funktion daher genauer. Ein Kollege, der `SELECT lower(full_name) FROM a` ausführt, führt stattdessen den eingeschleusten Code aus. Qualifizieren Sie Aufrufe in sensiblem Code als `pg_catalog.lower`, entziehen Sie CREATE auf public den Rollen, die es nicht brauchen, und listen Sie mit `\df public.*` auf, was bereits vorhanden ist.

Ein Upgrade behält alte Rechte bei. Der sicherere Standard von PostgreSQL 15 gilt für neue Cluster und neu angelegte Datenbanken. Ein Clusterupgrade oder das Wiederherstellen eines Dumps behält die vorhandenen public-Rechte bei. Ein Server mit Version 15 oder später kann deshalb weiterhin jeder Rolle das Anlegen von Objekten dort erlauben. Laut Handbuch ist der Standardpfad mit CREATE auf public niemals ein sicheres Muster; zugleich ist er der Standard in Datenbanken, die von 14 oder früher aktualisiert wurden. Führen Sie dort `REVOKE CREATE ON SCHEMA public FROM PUBLIC` aus und prüfen Sie public anschließend auf Objekte mit Namen wie in pg_catalog.

Eine Tabelle entsteht am falschen Ort. Eine Migration erstellt `invoices` ohne Schema. Die Tabelle landet im ersten vorhandenen Schema des Suchpfads der ausführenden Rolle. Die Anwendung verbindet sich mit einem anderen Pfad und liest dann eine andere Tabelle oder erhält `relation "invoices" does not exist`. Qualifizieren Sie jeden Namen in Migrationen und prüfen Sie `SELECT current_schema()` in der ausführenden Sitzung.

Eine Löschung reicht über das Schema hinaus. `DROP SCHEMA billing` weigert sich, solange das Schema etwas enthält, weil RESTRICT der Standard ist. Mit CASCADE löscht es jedes Objekt darin und jedes davon abhängige Objekt. Das Handbuch warnt, dass dadurch auch Objekte in anderen Schemas als dem genannten verschwinden können. Es rät, DROP zuerst ohne CASCADE auszuführen und die DETAIL-Zeilen zu lesen, die die betroffenen Objekte aufzählen.

Ein Name, den das System zuerst auflöst. pg_catalog wird implizit vor den in search_path aufgeführten Schemas durchsucht, wenn es im Pfad fehlt. Ist es ausdrücklich genannt, legt seine Position die Suchreihenfolge fest. Eine eigene Tabelle, deren Name mit `pg_` beginnt, kann verdeckt werden, wenn eine spätere Version eine gleichnamige Systemtabelle ergänzt. Das Handbuch rät deshalb von solchen Namen ab. Reservierte Wörter wie `user` und `order` verursachen eine mildere Form des Problems: Eine so benannte Tabelle braucht in jeder Anweisung doppelte Anführungszeichen.

Weiterführende Lektüre

- 5.10. Schemas — Das Handbuchkapitel zu qualifizierten Namen, public, search_path, sicheren Nutzungsmustern und Portabilität.
- CREATE SCHEMA — Befehlssyntax, AUTHORIZATION-Klausel, reserviertes Präfix `pg_` und Abweichungen PostgreSQLs vom Standard.
- 5.8. Rechte — Was USAGE und CREATE bei einem Schema bedeuten, neben allen übrigen Rechten.
- E.20. Version 15 — Die Version, die CREATE auf public bei neuen Datenbanken entzog und bei aktualisierten beibehielt.
- CVE-2018-1058: Den Suchpfad schützen — Die Anleitung des Projekts zum Angriff mit eingeschleuster Funktion, samt Abhilfe und Prüfkommandos für eine Datenbank.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Schema?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/schema/