Engineer

Schéma

Espace de noms pour les objets d’une base de données

Ce que c’est

Un schéma est un espace de noms dans une base de données : un conteneur pour des tables, des vues, des types de données, des fonctions et des opérateurs, qui permet d’utiliser le même nom dans deux schémas sans conflit. Le manuel de PostgreSQL compare les schémas aux répertoires d’un système de fichiers, à ceci près qu’ils ne peuvent pas être imbriqués : une base contient des schémas, et les schémas contiennent des objets. Le mot a un second sens, plus large et tout aussi courant : le schéma d’une base de données désigne sa structure entière, chaque table, colonne, clé et contrainte. Les deux sens apparaissent ci-dessous, et le pluriel, schémas, désigne presque toujours les espaces de noms.

La partie facturation d’une application peut disposer de son propre schéma, et une session peut choisir dans quels schémas chercher :

```
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
```

Un nom accompagné de son schéma, comme `billing.invoices`, désigne toujours la même table. Un nom écrit sans schéma est recherché dans search_path, une liste de schémas que le serveur parcourt dans l’ordre jusqu’à trouver une correspondance. La liste par défaut est `"$user", public` : un schéma portant le nom du rôle courant, s’il existe, puis le schéma public avec lequel toute nouvelle

base commence. Le premier schéma existant du chemin est aussi celui où CREATE TABLE place une table dont le nom ne précise pas de schéma ; c'est pourquoi les objets arrivent dans public, sauf indication contraire. Un nom absent de tous les schémas de la liste provoque une erreur, même si un autre schéma de la même base le contient.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un schéma ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Une structure fidèle au fonctionnement de l'entreprise, qui permet encore de poser demain les questions dont vous aurez besoin.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 6 pipelines ETL/ELT conçus et exploités — 6 pipelines ETL sur trois moteurs de bases de données

6 pipelines ETL/ELT conçus et exploités Contactez-nous →

Pourquoi c'est important

La première raison est l'organisation. Une base de quelques centaines de tables a besoin d'un niveau au-dessus de la table, et les schémas le fournissent : la facturation dans l'un, les comptes dans un autre, les tables d'un outil tiers dans un troisième, à l'écart des noms de tous les autres. Une connexion cliente atteint une seule base, mais les schémas ne sont pas strictement isolés : un rôle peut utiliser des objets de n'importe quel schéma de la base à laquelle il est connecté, si ses privilèges le permettent, et une seule requête peut joindre des tables de plusieurs schémas.

La deuxième raison est le contrôle d'accès. USAGE sur un schéma permet à un rôle d'y rechercher les objets, et CREATE lui permet d'en ajouter. Retirer USAGE bloque l'accès aux objets, mais leurs noms restent lisibles dans les catalogues système : un schéma protège donc les données, sans cacher ce qui existe. `CREATE SCHEMA name AUTHORIZATION role` attribue la propriété d'un nouveau schéma au rôle indiqué, et les noms commençant par `pg_` sont réservés au système.

La troisième raison est la sécurité, car les schémas participent à la résolution des noms. La plupart des noms dans les requêtes sont écrits sans schéma : selon les termes du manuel, ajouter un schéma à search_path revient donc à faire confiance à tous les rôles qui peuvent y créer des objets. Un tel rôle peut amener une requête ordinaire à exécuter des fonctions de son choix, comme si la personne qui lance la requête les avait appelées. Les pratiques sûres excluent du chemin d'un utilisateur les schémas dans lesquels d'autres peuvent écrire : par

exemple, un schéma privé par utilisateur, créé avec `CREATE SCHEMA alice AUTHORIZATION alice`, que l'entrée `"$user"` trouve alors en premier. PostgreSQL 15 a rapproché les nouvelles bases de ce modèle : il a retiré à l'ensemble des rôles le droit de créer des objets dans public et a fait du propriétaire de la base, par l'intermédiaire du rôle pg_database_owner, le propriétaire de public.

La portabilité est le dernier point. Le standard SQL prévoit des schémas, mais pas de schéma public, et le manuel note que certains systèmes SQL pourraient ne pas implémenter les schémas du tout. Les noms qualifiés se transportent mieux que du code qui s'appuie sur search_path.

À quoi cela ressemble en pratique

Le modèle de données présenté dans 348 tables réparties dans 34 schémas Postgres utilise les schémas au premier sens, comme niveau au-dessus de la table. Le domaine maritime, avec ses professionnels, entreprises, navires, emplois et avis, est réparti dans des schémas PostgreSQL normalisés organisés par domaine : 348 tables dans 34 schémas. Les noms de tables sont au pluriel, dans des schémas nommés au singulier, sans exception, ce qui évite aussi de nombreuses collisions avec des mots réservés ; de véritables clés étrangères et contraintes maintiennent les relations entre les tables. Ajouter une fonctionnalité revient généralement à prolonger la structure existante plutôt qu'à travailler contre elle : la page présente cela comme la seule raison pour laquelle 34 schémas constituent une structure plutôt qu'une prolifération désordonnée.

Un schéma peut aussi délimiter deux responsabilités. Dans la mise en place des droits d'accès liés à Stripe et aux achats intégrés, les paiements par carte Stripe et les achats intégrés Apple et Google convergent tous vers un schéma de paiements de 9 tables et 34 fonctions. Un schéma d'accès distinct, de 11 tables et 34 fonctions, répond à la question « ce compte peut-il faire cela ? » sans connaître la boutique par laquelle le paiement a été effectué. Cette réponse, revérifiée sur le serveur à chaque requête, contrôle l'accès à l'annuaire des entreprises, à la recherche et à l'export de données. Le coût, comme l'explique la page, est d'avoir deux schémas là où un produit plus petit n'en voudrait qu'un, ainsi qu'une recherche de droits d'accès sur des requêtes qui n'auraient autrement pas eu ce coût.

Le second sens du mot fait l'objet de 1 022 migrations derrière un contrôle bloquant de dérive du schéma. La plupart des projets décrivent un schéma deux fois : dans l'installation initiale qui le crée de zéro, puis dans les migrations qui l'ont fait évoluer ; les deux divergent si rien ne les compare. Ici, la base est versionnée sous la forme de 1 022 migrations, numérotées de 036 à 1102, et une tâche de CI la construit deux fois à chaque modification, une fois à neuf et une fois en rejouant toutes les migrations dans l'ordre, puis compare les deux : les données initiales autant que la structure. Une migration qui complète mal après coup les données d'une table de référence cause autant de dégâts qu'une migration qui oublie une colonne, et une seule des deux erreurs apparaît dans un diff de schéma. Tout désaccord fait échouer la construction en affichant la différence : chaque migration doit donc fonctionner aussi bien lorsqu'elle est rejouée que lors d'une installation de zéro.

Où les problèmes surviennent

Une fonction piégée dans un schéma partagé. Le guide du wiki PostgreSQL consacré à CVE-2018-1058 montre l'attaque : un utilisateur crée `lower(varchar)` dans public. La fonction intégrée `lower` se trouve dans pg_catalog et prend du texte ; pour une colonne varchar, la fonction piégée correspond donc mieux, et un collègue qui lance `SELECT lower(full_name) FROM a` exécute le code piégé à la place. Qualifiez les appels dans le code sensible avec `pg_catalog.lower` , retirez CREATE sur public aux rôles qui n'en ont pas besoin et dressez l'inventaire de ce qui s'y trouve déjà avec `\df public.*` .

Une mise à niveau qui conserve les anciennes permissions. La configuration par défaut plus sûre de PostgreSQL 15 s'applique aux nouveaux clusters et aux bases nouvellement créées. Mettre un cluster à niveau ou restaurer un dump conserve les permissions que public possédait déjà : un serveur en version 15 ou ultérieure peut donc encore laisser tous les rôles y créer des objets. Le manuel indique que le chemin par défaut avec CREATE accordé sur public n'est jamais une pratique sûre, et note qu'il s'agit de la configuration par défaut dans les bases mises à niveau depuis la version 14 ou antérieure. Dans une telle base, lancez `REVOKE CREATE ON SCHEMA public FROM PUBLIC` , puis recherchez dans public les objets portant le même nom que ceux de pg_catalog.

Une table créée au mauvais endroit. Une migration crée `invoices` sans schéma, et la table arrive dans le premier schéma existant du chemin du rôle qui l'exécute. L'application, connectée avec un autre chemin, lit alors une autre table ou reçoit `relation "invoices" does not exist` . Qualifiez chaque nom dans les migrations et vérifiez `SELECT current_schema()` dans la session qui les exécute.

Une suppression qui dépasse le schéma. `DROP SCHEMA billing` refuse de s'exécuter tant que le schéma contient quelque chose, car RESTRICT est le comportement par défaut. Avec CASCADE, la commande supprime tous les objets qu'il contient et tous ceux qui en dépendent, et le manuel avertit que cela peut supprimer des objets dans d'autres schémas que celui désigné. Il conseille d'exécuter d'abord DROP sans CASCADE et de lire les lignes DETAIL, qui énumèrent ce qui disparaîtrait.

Un nom que le système résout en premier. pg_catalog est implicitement parcouru avant les schémas énumérés dans search_path lorsque ce chemin l'omet. S'il est nommé explicitement, sa position dans le chemin détermine l'ordre de recherche. Une table que vous créez et dont le nom commence par `pg_` peut être masquée lorsqu'une version ultérieure ajoute une table système du même nom ; le manuel déconseille donc ces noms. Des mots réservés comme `user` et `order` provoquent une version plus légère du problème : une table ainsi nommée exige des guillemets doubles dans chaque instruction.

Pour aller plus loin

- 5.10. Schémas — Le chapitre du manuel : noms qualifiés, schéma public, search_path, pratiques d'utilisation sûres et portabilité.
- CREATE SCHEMA — La syntaxe de la commande, la clause AUTHORIZATION, le préfixe réservé `pg_` et les écarts de PostgreSQL par rapport au standard.
- 5.8. Privilèges — Le sens de USAGE et CREATE sur un schéma, avec tous les autres privilèges.
- E.20. Version 15 — La version qui a retiré CREATE sur public des nouvelles bases et l'a conservé dans celles mises à niveau.
- Guide de CVE-2018-1058 : protéger votre chemin de recherche — L'explication par le projet de l'attaque par fonction piégée, avec les corrections et les commandes pour auditer une base.

Glossaire

Choisissez selon vos contraintes réelles

Présentez un jeu de données représentatif, les questions auxquelles il doit répondre et le goulot d'étranglement actuel. Nous pouvons convenir de la manière de mesurer la qualité, le débit et la reprise.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un schéma ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/schema/