Engineer

PostgreSQL

La base relationnelle de la plupart des travaux présentés

Ce que c’est

PostgreSQL est un serveur de bases de données relationnelles, libre et gratuit, qui conserve les données dans des tables, répond à des requêtes écrites en SQL et rend chaque transaction indivisible : elle aboutit entièrement ou pas du tout. La plupart des personnes qui l’utilisent l’appellent Postgres, et le projet reconnaît les deux noms. À la question « qu’est-ce que PostgreSQL ? », un ingénieur de terrain répondrait simplement : c’est la base de données généraliste sur laquelle démarrer un nouveau système quand aucune raison précise ne justifie d’en choisir une autre.

Il est plus ancien que la plupart des logiciels qui reposent aujourd’hui sur lui. Le projet POSTGRES a commencé à l’université de Californie à Berkeley, sous la direction du professeur Michael Stonebraker, et son implémentation a débuté en 1986. Andrew Yu et Jolly Chen lui ont ajouté un interpréteur SQL en 1994 et ont publié le résultat sous le nom Postgres95. Deux ans plus tard, il a été renommé PostgreSQL, et la numérotation des versions a repris à 6.0 pour poursuivre la série commencée à Berkeley. Le projet décrit lui-même le système comme objet-relationnel : au-delà des tables et des jointures, l’utilisateur peut ajouter des types de données, des fonctions, des opérateurs, des agrégats, des méthodes d’indexation et des langages procéduraux, que le serveur traite comme ses propres éléments.

Son architecture comprend un client et un serveur. Un processus serveur détient les fichiers de données, et chaque connexion cliente reçoit son propre processus, créé par duplication pour cette seule connexion. Les applications lui envoient du SQL par cette connexion, au moyen d’un pilote adapté à leur langage. La concurrence repose sur le contrôle de concurrence multiversion : une mise à jour écrit une nouvelle version d’une ligne au lieu d’écraser l’ancienne. Chaque instruction lit ainsi un instantané cohérent, les lecteurs n’attendent jamais les écrivains et les écrivains n’attendent jamais les lecteurs.

Où se situe-t-il parmi les autres solutions ? Aux côtés de MySQL et MariaDB parmi les serveurs relationnels libres, d'Oracle et de Microsoft SQL Server parmi les solutions commerciales, de SQLite comme base embarquée dans un programme, et face aux bases documentaires comme MongoDB, qui conservent les enregistrements sans schéma fixe.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que PostgreSQL ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Identifier les requêtes et les index manquants dans un système alourdi au fil des ans, puis corriger ce qui compte.

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. 30 jours de données électriques traités en 180 ms — 180 ms pour agréger un mois de relevés de compteurs

30 jours de données électriques traités en 180 ms Contactez-nous →

Pourquoi c'est important

Choisir une base de données engage une équipe pour des années, car les données survivent au code qui les exploite. PostgreSQL est un choix par défaut que l'on risque peu de regretter, pour trois raisons qu'un acheteur peut vérifier.

La première est le coût et la dépendance au fournisseur. Il est distribué sous la PostgreSQL License, une courte licence permissive de la famille BSD et MIT : aucun droit à payer par cœur, aucun audit, aucun fournisseur susceptible de changer les conditions. Amazon, Microsoft et Google le proposent chacun comme service géré, et la même base fonctionne sur un ordinateur portable. Changer d'hébergeur revient donc à migrer les données plutôt qu'à réécrire l'application.

La deuxième est l'étendue de ses possibilités. Un seul serveur prend en charge les tables relationnelles, les documents JSON dans des colonnes `jsonb`, la recherche en texte intégral, les requêtes géospatiales avec l'extension PostGIS et les tâches planifiées avec `pg_cron`. Chacune de ces fonctions évite à l'équipe d'exploiter, de sécuriser et de sauvegarder un produit distinct, ainsi que de recruter pour s'en occuper. Pour la plupart des produits, c'est ici que se joue le choix entre PostgreSQL et MySQL : les deux conviennent bien aux lectures et écritures simples, et la différence apparaît quand il faut des types plus riches, des contraintes plus strictes ou une extension.

La troisième est la pérennité. Le projet publie une version majeure par an et assure le support de chacune pendant cinq ans. Une mise à niveau peut donc être planifiée plutôt qu'imposée par une urgence. PostgreSQL 18, publié en septembre 2025, a ajouté les entrées-sorties asynchrones et une fonction native produisant des UUID ordonnés dans le temps. Le SQL écrit pour des versions antérieures de dix ans y fonctionne généralement sans modification.

Les contreparties sont réelles. Un processus par connexion rend les connexions coûteuses : une application très sollicitée a donc besoin d'un gestionnaire de pool. Il est plus facile de le faire évoluer sur une machine plus puissante que sur plusieurs machines : répartir les écritures entre plusieurs serveurs exige du partitionnement, de la réplication ou une extension, qui alourdissent chacun l'exploitation. Ses réglages par défaut sont par ailleurs adaptés à une petite machine : une installation conséquente doit donc être configurée délibérément.

À quoi cela ressemble en pratique

De nombreux projets de ce site reposent sur PostgreSQL, et trois d'entre eux montrent trois façons différentes de s'appuyer sur lui.

Sur une plateforme, la base de données est devenue toute la couche de données. Toutes les lectures et toutes les écritures passent par l'une des 1 275 fonctions PostgreSQL, réparties dans 34 schémas, et l'API Go n'a aucun accès direct aux tables. Le rôle avec lequel l'API se connecte, `mariner`, détient EXECUTE sur les fonctions et USAGE sur les schémas, et rien d'autre : environ 4 000 autorisations explicites. Les fonctions s'exécutent en SECURITY DEFINER sous un propriétaire distinct sans droit de connexion, avec un `search_path` fixé. Postgres lui-même impose la limite d'accès ; la contrepartie est qu'ajouter un champ exige une migration et la modification d'une fonction.

Sur une application conservant les organisations clientes, les lieux et les relevés quotidiens d'électricité, il s'agissait d'optimisation plutôt que de conception, avec un objectif de gain de performance de 10 fois sans refonte de l'architecture. Des index GiST pour les données spatiales, des index composites sur l'organisation cliente et la plage de dates, le partitionnement par plage de la table d'électricité, par jour et par mois, ainsi que des requêtes réécrites après lecture de leurs plans avec `EXPLAIN ANALYZE`, ont amélioré les temps de réponse des requêtes d'un facteur 10 et réduit l'utilisation des ressources de 40 %.

Sur une plateforme comportant des traitements lents en arrière-plan, la file de tâches réside dans la même base. River conserve sa file dans PostgreSQL : une tâche et les lignes qu'elle touche sont donc validées ou annulées ensemble. Elle s'appuie sur 15 modules de traitement, 8 tâches planifiées et 20 tâches `pg_cron` qui élaguent les partitions, renouvellent les sels et actualisent les agrégats. La page énonce clairement la limite : au-delà d'un certain volume, une file dans Postgres cesse d'être le bon choix, mais supprimer un composant à exploiter valait davantage qu'une réserve de capacité dont personne n'allait se servir.

Où les problèmes surviennent

PostgreSQL perd rarement des données. Il réserve néanmoins des surprises, qui prennent quelques formes bien connues.

La table grossit alors que le nombre de lignes reste stable. Un `UPDATE` ou un `DELETE` laisse l'ancienne version de la ligne en place jusqu'à ce que `VACUUM` la supprime, car une autre transaction peut encore être en train de la lire. Une transaction laissée ouverte pendant des heures retarde ce nettoyage pour toute la base, et les réglages par défaut d'autovacuum conviennent aux petites tables. Fermez les transactions inactives, réglez autovacuum table par table pour les grandes tables et partitionnez les données qui expirent afin de supprimer les anciennes partitions mensuelles plutôt que leurs lignes. Vacuum remplit aussi un second rôle : les identifiants de transaction sont codés sur 32 bits, chaque table doit être nettoyée par vacuum au moins une fois tous les deux milliards de transactions, et un serveur qui prend trop de retard refuse les nouvelles écritures pour protéger les données.

La mémoire s'épuise dès le premier pic de trafic. Chaque connexion est un processus serveur doté de sa propre mémoire, et la limite par défaut est de 100 connexions. Vingt instances applicatives possédant chacune un pool de cinquante connexions demandent mille processus. Placez devant le serveur un gestionnaire de pool comme PgBouncer, et dimensionnez le pool de chaque application selon le travail à effectuer : il suffit généralement de quelques connexions par instance.

Une requête qui prenait des millisecondes se met à prendre des secondes. L'optimiseur a choisi un autre plan à mesure que les données ont augmenté, souvent parce que ses statistiques sont devenues obsolètes ou que les index

avaient été construits pour les requêtes d'il y a un an. Lisez le plan avant de changer quoi que ce soit, actualisez les statistiques avec `ANALYZE` et indexez le prédicat sur lequel la requête filtre réellement :

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

Les insertions ralentissent quand la table grossit. Des UUID aléatoires utilisés comme clés primaires se répartissent dans tout l'index B-tree : chaque insertion touche donc une page différente et l'index se fragmente. Des identifiants ordonnés dans le temps évitent cela : une plateforme a fait d'UUID v7 sa clé par défaut avec `uuidv7()` de PostgreSQL 18, et les nouvelles lignes s'insèrent dans l'ordre de l'index au lieu de s'y disperser.

Pour aller plus loin

- Qu'est-ce que PostgreSQL ? — la définition du projet lui-même, avec les fonctionnalités et les points d'extension qu'il cite.
- Bref historique de PostgreSQL — Berkeley, Postgres95 et le changement de nom, racontés par les personnes qui y ont participé.
- Introduction au contrôle de concurrence — comment les instantanés permettent aux lectures et aux écritures de se poursuivre sans se bloquer mutuellement.
- Nettoyage régulier par vacuum — les versions mortes des lignes, autovacuum et le rebouclage des identifiants de transaction, en détail.
- Utiliser EXPLAIN — comment lire un plan de requête, avec des exemples détaillés.
- Politique de gestion des versions — la version majeure annuelle et les cinq années de support de chacune.

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 que PostgreSQL ? » 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/postgresql/