Engineer

CTE

sous-requête nommée qu'une requête lit comme une table

Ce que c’est

Une CTE, abréviation de common table expression, est une requête nommée dans une clause WITH au début d’une instruction. Le reste de l’instruction peut alors lire ses lignes comme si elles venaient d’une table. Une instruction peut définir plusieurs CTE, chacune pouvant lire celles nommées avant elle, et la requête finale, au bas de l’instruction, les rassemble. Le manuel PostgreSQL les décrit comme des « tables temporaires qui n’existent que pour une requête ». L’image est juste, avec une réserve : rien n’est stocké sous ce nom, et l’instruction suivante ne peut pas y accéder. Chaque CTE peut être un SELECT ou, dans PostgreSQL, un INSERT, UPDATE, DELETE ou MERGE ; l’instruction à laquelle la clause WITH est attachée peut aussi être de l’un de ces types.

Une question comme « quels compteurs ont enregistré une consommation supérieure à trois fois la normale sur une journée des 30 derniers jours » se divise naturellement en étapes :

```
WITH daily AS (
  SELECT meter_id, read_at::date AS day, sum(kwh) AS kwh
  FROM readings
  WHERE read_at >= current_date - 30
  GROUP BY meter_id, read_at::date
),
typical AS (
  SELECT meter_id, avg(kwh) AS avg_kwh
  FROM daily
  GROUP BY meter_id
)
SELECT d.meter_id, d.day, d.kwh, t.avg_kwh
FROM daily AS d
JOIN typical AS t USING (meter_id)
WHERE d.kwh > 3 * t.avg_kwh
ORDER BY d.day, d.meter_id;
```

La première étape totalise les relevés quotidiens de chaque compteur, la deuxième fait la moyenne de ces totaux par compteur, et la dernière compare chaque journée à la moyenne de son propre compteur. On peut obtenir la même réponse avec des sous-requêtes imbriquées, mais le calcul des totaux quotidiens apparaîtrait alors deux fois dans le texte, l'une des occurrences étant imbriquée dans l'autre, et la lecture devrait se faire de l'intérieur vers l'extérieur. Avec WITH, la logique se lit de haut en bas, et chaque étape peut être exécutée seule pendant sa vérification.

L'ajout de RECURSIVE permet à une CTE de lire sa propre sortie, ce que le SQL ordinaire n'a pas d'autre moyen d'exprimer. Une CTE récursive contient une requête initiale, puis UNION ou UNION ALL, puis une requête qui fait référence à la CTE. PostgreSQL exécute la requête initiale, conserve ses lignes dans une table de travail, puis exécute à nouveau la seconde requête sur les lignes les plus récentes jusqu'à ce qu'un tour n'en ajoute aucune. C'est ainsi qu'une instruction parcourt un arbre de profondeur inconnue, comme les catégories sous une rubrique :

```
WITH RECURSIVE subtree AS (
  SELECT id, parent_id, name, 1 AS depth
  FROM categories
  WHERE id = 42
  UNION ALL
  SELECT c.id, c.parent_id, c.name, s.depth + 1
  FROM categories AS c
  JOIN subtree AS s ON c.parent_id = s.id
)
SELECT id, name, depth FROM subtree ORDER BY depth, name;
```


Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'une CTE ? » 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

La première raison est la lisibilité. Une requête analytique de cent lignes imbriquée sur quatre niveaux est difficile à relire et encore plus à modifier. La même logique sous forme de chaîne d’étapes nommées donne à la personne qui la relit une phrase par étape, à son auteur un endroit pour un commentaire, et à chacun le moyen d’exécuter une étape seule pour vérifier ses lignes avant de faire confiance à la suivante.

La deuxième raison est la façon dont la base évalue les étapes, et les détails ont changé avec le temps. PostgreSQL évalue normalement une requête WITH une fois par exécution de l’instruction, même si celle-ci y fait référence plusieurs fois. Un calcul coûteux nécessaire à deux endroits ne s’exécute donc qu’une fois. Le coût de cette garantie est que le planificateur ne peut pas faire descendre un filtre de la requête externe dans une CTE référencée plusieurs fois, car ce filtre devrait s’appliquer à un usage et pas aux autres. La CTE est calculée entièrement, et le filtrage n’écarte les lignes qu’après coup.

Avant PostgreSQL 12, cela valait pour toute CTE : chacune était évaluée avant le reste de la requête, sans jamais y être fusionnée. La version 12 a changé le comportement appliqué sans indication contraire. Une CTE non récursive, sans effet de bord et référencée une seule fois est désormais intégrée à la requête externe. Le planificateur optimise les deux niveaux ensemble et peut faire descendre le filtre externe jusqu’à un index de la table de base. MATERIALIZED impose l’ancien comportement, et NOT MATERIALIZED impose l’intégration même pour une CTE utilisée plusieurs fois, au risque d’exécuter son travail deux fois. SQLite accepte les deux mêmes mots-clés comme indications non contraignantes et dit avoir repris l’idée à PostgreSQL. Les autres moteurs décident eux-mêmes ; c’est dans le plan qu’il faut vérifier.

La troisième raison est qu’une CTE peut modifier des données. Un DELETE avec RETURNING dans une clause WITH peut passer sa sortie à un INSERT dans l’instruction principale. Cela déplace des lignes d’une table vers une autre dans une instruction unique qui s’effectue entièrement ou pas du tout. Les CTE qui modifient des données s’exécutent exactement une fois et toujours jusqu’au bout, que leur sortie soit lue ou non. Elles servent ainsi à faire plusieurs écritures ensemble, plutôt qu’à décrire un résultat.

À quoi cela ressemble en pratique

Le travail décrit dans la multiplication par 10 des performances PostgreSQL concernait une base de comptes utilisateurs, d'organisations clientes, de lieux et de relevés quotidiens d'électricité qui ralentissait avec la croissance des données. La correction avait trois volets : des index ciblés et GiST, un partitionnement temporel de la table des relevés et la réécriture de requêtes. Les réécritures utilisaient des CTE et des vues matérialisées pour les données souvent consultées afin d'éviter les parcours complets de table. EXPLAIN ANALYZE indiquait les jointures et sous-requêtes à restructurer. La division par dix des temps de réponse et la baisse de 40 % des ressources consommées venaient des trois volets ensemble, pas des seules CTE. L'association mérite attention, car les deux outils font des choses différentes : une CTE nomme un résultat intermédiaire le temps d'une instruction ; une vue matérialisée stocke un résultat que plusieurs instructions peuvent lire jusqu'à son actualisation.

L'autre apport des CTE, la lisibilité, a soutenu les 500 rapports d'analyse de données électriques et SIG. Chaque rapport était écrit en SQL pur pour PostgreSQL et BigQuery, sur des données de compteurs intelligents avec coordonnées, identifiants de compteurs et consommations horodatées. Les scripts associaient les CTE à des fonctions de fenêtrage, des sous-requêtes et des jointures géospatiales. Chaque rapport était livré avec du SQL annoté pour permettre aux collègues de reproduire et d'auditer l'analyse, ainsi que des notes sur des problèmes courants de qualité, comme les coordonnées GPS manquantes et les valeurs de compteurs corrompues. Un SQL lisible étape par étape se partage bien : les rapports sont devenus une référence commune aux différents services et ont contribué à améliorer de 10–15 % l'efficacité de la planification opérationnelle. Les étapes nommées rendent ce SQL annotable : une note au-dessus de chacune peut expliquer ce qu'elle retire ou suppose.

Où les problèmes surviennent

Le filtre arrive une fois tout le travail terminé. Une CTE référencée deux fois ou marquée MATERIALIZED est calculée entièrement avant que la requête externe la filtre, et la copie produite n'a pas d'index. Une requête qui ne devrait toucher que quelques lignes lit toute une table. Lisez le plan : si la CTE apparaît comme une étape distincte, avec beaucoup plus de lignes que la requête n'en renvoie, essayez NOT MATERIALIZED ou reformulez la requête pour ne référencer la CTE qu'une fois. Gardez la forme matérialisée quand la CTE appelle une fonction coûteuse, afin que celle-ci s'exécute une fois par ligne plutôt qu'une fois par référence.

Une requête récursive ne s'arrête pas. Des données cycliques, comme deux catégories dont chacune désigne l'autre comme parent, font tourner une CTE avec UNION ALL aussi longtemps que le serveur le permet. UNION n'aide que si des lignes entières se répètent, ce qui est rare une fois une colonne de profondeur ajoutée. Suivez le chemin des clés visitées dans un tableau et arrêtez-vous quand une clé réapparaît, ou utilisez la clause CYCLE ajoutée par PostgreSQL 14. Un LIMIT sur la requête externe est une astuce de test ; le manuel déconseille de s'y fier en production. SQL Server limite la récursion à 100 niveaux sans autre réglage et annule l'instruction au-delà ; le même défaut s'arrête donc au lieu de continuer.

Le reste de l'instruction ne voit pas les changements. Une CTE qui modifie des données et la requête principale s'exécutent concurremment sur le même instantané ; une requête principale lisant directement la table cible voit donc les lignes d'avant le changement. Leur seul canal est RETURNING. Deux parties d'une même instruction modifiant la même ligne font pire : une seule modification prend effet, et il est difficile de prévoir laquelle. Lisez les lignes changées dans la sortie RETURNING de la CTE et veillez à ce qu'une seule partie de l'instruction modifie chaque ligne.

Le plan change avec une mise à niveau. Du code écrit pour PostgreSQL 11 ou antérieur reposait parfois sur l'évaluation préalable d'une CTE comme barrière délibérée contre un plan indésirable. Après une mise à niveau vers 12 ou ultérieur, la même CTE est intégrée à la requête externe et le planificateur choisit autre chose, généralement plus rapide et parfois beaucoup plus lent. Comparez les sorties EXPLAIN des instructions lentes avant et après la mise à niveau, puis ajoutez MATERIALIZED là où la barrière était voulue.

Pour aller plus loin

- 7.8. Requêtes WITH (expressions de table communes) — le chapitre du manuel sur les CTE : récursion, détection des cycles, matérialisation et instructions de modification des données.
- E.23. Version 12 — la version qui a commencé à intégrer les CTE utilisées une seule fois à la requête externe.
- E.25. Version 14 — la version qui a ajouté les clauses SEARCH et CYCLE de la norme SQL.
- La clause WITH — la version de SQLite, ses indications de matérialisation et leur provenance.
- WITH common_table_expression (Transact-SQL) — les règles de SQL Server pour les CTE, dont la limite de récursion MAXRECURSION.

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'une CTE ? » 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/cte/