Engineer

Requête

▼ question écrite à laquelle répond une base de données

Ce que c’est

Une requête est une demande de données écrite dans un langage que la base comprend. Elle renvoie un résultat : généralement un ensemble de lignes, parfois un seul nombre. Le manuel de PostgreSQL la définit en une phrase : l’extraction de données d’une base, ou la commande qui réalise cette extraction, s’appelle une requête, et en SQL cette commande est SELECT. L’usage courant élargit ce sens : les ingénieurs appellent requête presque toute instruction envoyée à une base, y compris celles qui modifient des données et renvoient quelque chose. Les deux sens apparaissent ci-dessous, et le pluriel, requêtes, désigne généralement tout le trafic qu’une application envoie.

La requête demandant quelles factures attendent leur paiement depuis le plus longtemps se lit presque comme la question formulée à voix haute :

```
SELECT i.number, c.name, i.amount, i.due_on
FROM invoices AS i
JOIN customers AS c ON c.id = i.customer_id
WHERE i.paid_on IS NULL
  AND i.due_on < current_date
ORDER BY i.due_on
LIMIT 20;
```

Le texte indique ce qu’il veut et laisse la méthode ouverte. Entre sa réception et le renvoi de la première ligne, le planificateur de PostgreSQL recense les façons d’obtenir la réponse, par exemple parcourir toute la table des factures ou un index sur `due_on`, estime le coût de chacune et exécute la moins coûteuse. Ces coûts s’expriment en unités arbitraires, conventionnellement rapportées à la lecture d’une page sur disque. Deux requêtes qui se ressemblent à l’écran peuvent donc différer d’un facteur mille lorsque les données sous-jacentes augmentent, et le plan est le seul endroit où cette différence apparaît.

Les clauses ne s'exécutent pas dans leur ordre d'écriture. Le manuel décrit le traitement de SELECT dans cet ordre : WITH d'abord, puis FROM, WHERE, le regroupement et HAVING, puis la liste de sortie, et seulement ensuite ORDER BY et LIMIT. Cet ordre explique une règle qui surprend la plupart des gens une première fois : un nom donné à une colonne dans la liste de sélection fonctionne dans ORDER BY et GROUP BY, tandis que WHERE s'exécute avant que ce nom existe, ce qui oblige à y réécrire l'expression. Le résultat de toute requête est lui-même une table, ce qui permet d'imbriquer les requêtes, une sous-requête ou une clause WITH fournissant ses lignes à une requête plus vaste.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'une requête ? » 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

Dans une application reposant sur une base relationnelle, les requêtes déterminent en grande partie la réactivité perçue d'une page et les ressources matérielles nécessaires au système. Les requêtes importantes se repèrent par le calcul plutôt que par l'intuition : le coût d'une requête pour le système est sa durée multipliée par sa fréquence d'exécution. Une recherche de 2 ms exécutée un million de fois par heure représente plus d'une demi-heure de temps de base de données chaque heure, tandis qu'un rapport de deux secondes exécuté deux fois par jour compte à peine.

PostgreSQL tient ce décompte dans l'extension pg_stat_statements. Elle réunit en une seule entrée les instructions qui ne diffèrent que par leurs valeurs littérales et enregistre leur nombre d'exécutions ainsi que leur durée totale. Trier par le total plutôt que par la moyenne est la première habitude payante dans le travail sur les requêtes : elle désigne l'instruction qui mérite une correction plutôt que celle qui semble simplement lente.

Chaque requête coûte aussi un aller-retour. L'application envoie du texte, le serveur l'analyse, le planifie et l'exécute, puis les lignes reviennent. Pour une recherche rapide dans un index, le trajet peut coûter plus que le travail. La forme

de l'échange compte donc autant qu'une instruction isolée : un UPDATE qui touche cent lignes fait un aller-retour, tandis qu'une boucle qui lit cent lignes puis les met à jour une par une en fait cent un.

Reste à savoir qui écrit les requêtes. Un ORM ou un constructeur de requêtes les génère à partir du code applicatif, ce qui permet de produire rapidement les requêtes simples de lecture et d'écriture et conserve le code dans un seul langage. Cela masque aussi le nombre d'instructions qu'une requête applicative envoie et leur forme, jusqu'à ce qu'une page devienne lente. Le SQL écrit à la main rend ces deux aspects visibles, au prix d'une responsabilité de maintenance semblable à celle de tout autre code. Une répartition viable confie les enregistrements courants à l'ORM, écrit à la main les chemins les plus sollicités et examine le SQL généré dès qu'une instruction arrive en tête des totaux.

À quoi cela ressemble en pratique

La méthode proposée pour une base lente dans Optimiser la base avant de la répartir en shards commence par les requêtes, avant de chiffrer de nouvelles machines. La première étape demande à PostgreSQL quelles instructions coûtent le plus au total, avec pg_stat_statements trié par durée totale d'exécution. La deuxième exécute EXPLAIN (ANALYZE, BUFFERS) sur l'instruction en tête pour voir le plan choisi par la base, comparer les lignes réellement renvoyées à chaque étape aux prévisions du planificateur et compter les pages lues. La règle empirique de cette note est qu'un parcours d'un million de lignes pour en renvoyer dix, ou une estimation erronée d'un facteur cent, explique généralement à lui seul la lenteur. Elle compte aussi les allers-retours et trace la même frontière que cet article : ce qui doit voir les données va dans la base, et les règles métier restent dans l'application.

Une requête peut aussi remplir le rôle d'un verrou. Dans Vendre le dernier billet une seule fois, une vente flash échoue de l'une de deux façons. Lire le niveau du stock, décider dans l'application puis réécrire le nombre permet à deux acheteurs de prendre chacun la dernière unité ; conserver un verrou de ligne pendant cette logique applicative fait attendre tous les acheteurs derrière lui. La correction réunit la vérification et la décrémentation dans un seul UPDATE conditionnel, qui ne soustrait la quantité que si le stock restant suffit et renvoie le nouveau niveau. Une ligne renvoyée signifie que l'unité est réservée, et zéro ligne que le stock est épuisé. Avec READ COMMITTED, le niveau d'isolation par défaut de PostgreSQL, un deuxième acheteur qui atteint la ligne pendant que le premier la détient

attend, puis réévalue la clause WHERE sur la ligne validée : la dernière unité revient donc à un seul des deux. Une contrainte CHECK qui maintient le stock à zéro ou au-dessus complète la requête.

Où les problèmes surviennent

Une requête peu coûteuse domine la charge. La base est très sollicitée, mais le journal des requêtes lentes reste silencieux, car chaque appel finit en quelques millisecondes. La cause est la fréquence : une requête rapide appelée à chaque requête applicative, souvent dans une boucle ou un composant présent sur chaque page. Classez les instructions par durée totale d'exécution dans pg_stat_statements, puis mettez le résultat en cache, regroupez les appels ou réduisez le coût de la requête avec un index adapté à son filtre.

Une page envoie des centaines de requêtes. Chaque requête de la trace prend une milliseconde, et la page met tout de même plusieurs secondes. La cause habituelle est un ORM qui charge les enregistrements associés à la demande, à raison d'une requête par ligne parente : le motif N+1. Chargez les lignes associées en une seule requête, par une jointure ou avec `WHERE id = ANY($1)` et un tableau de clés, puis surveillez le nombre d'instructions par requête applicative aussi attentivement que la durée de chacune.

La dernière unité est vendue deux fois. Deux requêtes applicatives lisent le même stock, décident toutes deux qu'il suffit et écrivent toutes deux. Séparer lecture et écriture en instructions distinctes ouvre un intervalle entre vérification et action, que le trafic concurrent exploite. Fermez cet intervalle dans la requête, avec un UPDATE conditionnel et RETURNING comme ci-dessus, ou avec SELECT … FOR UPDATE conservé le moins longtemps possible et sans appel réseau entre le verrouillage et le commit.

Un diagnostic modifie les données. Quelqu'un exécute EXPLAIN ANALYZE sur un DELETE lent pour examiner son plan, et les lignes disparaissent. ANALYZE exécute réellement l'instruction et n'en écarte que la sortie : tous les effets de bord se produisent. Encadrez toute instruction modifiant les données par BEGIN et ROLLBACK avant de l'analyser, ou utilisez EXPLAIN seul, qui planifie sans exécuter, si des estimations suffisent.

Pour aller plus loin

- SELECT — la syntaxe complète d'une requête PostgreSQL, y compris l'ordre de traitement de ses clauses.
- 7.1. Vue d'ensemble — le début du chapitre du manuel consacré aux requêtes, avec la définition du mot.
- 51.1. Le parcours d'une requête — le trajet du texte du client à travers l'analyseur syntaxique, le réécriveur et le planificateur jusqu'à l'exécuteur.
- 14.1. Utiliser EXPLAIN — comment lire un plan, ce que signifient les unités de coût et pourquoi ANALYZE exécute l'instruction.
- F.32. pg_stat_statements — statistiques de planification et d'exécution SQL — l'extension qui classe les requêtes selon leur coût total.
- 13.2. Isolation des transactions — ce que voit un deuxième processus d'écriture avec READ COMMITTED et pourquoi la clause WHERE est réévaluée.

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 requête ? » 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/query/