Engineer

Partitionnement

Découpage d’une table en plusieurs parties physiques

Ce que c’est

Le partitionnement, aussi appelé partitionnement de table, stocke les lignes d’une table logique dans plusieurs tables ordinaires, chacune contenant les lignes dont la clé se situe à l’intérieur des bornes de cette table. Les requêtes et les écritures continuent d’utiliser le même nom qu’auparavant. Cette page suit PostgreSQL, qui appelle sa forme intégrée le partitionnement déclaratif : la table est déclarée partitionnée, avec une méthode et une clé de partitionnement, et chaque partition est créée comme une table distincte.

La table partitionnée ne contient elle-même aucune ligne. Une insertion est dirigée vers la partition dont les bornes correspondent à la clé, et une mise à jour qui change la clé déplace la ligne vers la partition à laquelle elle appartient désormais. Les partitions partagent les colonnes du parent, mais peuvent avoir leurs propres index et réglages de stockage, ainsi que leur propre tablespace ; une partition peut elle-même être partitionnée à nouveau.

Il existe trois méthodes. Le partitionnement par plage attribue à chaque partition un intervalle de valeurs, généralement des dates ; la borne inférieure est incluse et la borne supérieure exclue : avec les intervalles 1–10 et 10–20, la valeur 10 appartient donc au second. Le partitionnement par liste nomme les valeurs acceptées par chaque partition, par exemple un ensemble de codes de pays. Le partitionnement par hachage attribue à chaque partition un module et un reste : une ligne va là où le hachage de sa clé donne ce reste, ce qui répartit les lignes uniformément lorsqu’aucune plage naturelle ne convient. Une partition par défaut peut recueillir les valeurs qui ne correspondent à aucune autre, uniquement pour le partitionnement par plage ou par liste.

Le partitionnement se fait dans un même serveur de base de données. Répartir les données entre des serveurs distincts, chacun avec sa copie du schéma, est du sharding, une étape beaucoup plus importante. SQL Server et BigQuery ont leurs propres formes de partitionnement et leur propre syntaxe ; les idées ci-dessous restent valables même lorsque les commandes diffèrent.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que le partitionnement ? » 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 plupart des grandes tables grossissent par une extrémité. Commandes, relevés, journaux et événements arrivent chaque jour, restent des mois ou des années, et la plupart des requêtes portent sur la partie récente. Sans partitionnement, chaque requête travaille sur une table et un ensemble d'index qui ne cessent de grossir.

Avec une clé de partitionnement adaptée aux requêtes, le planificateur peut prouver que la plupart des partitions ne peuvent contenir aucune ligne correspondant à la requête et les exclure complètement du plan. PostgreSQL appelle cela l'élagage des partitions. Un rapport sur la semaine passée ne lit que la partition de cette semaine-là. La documentation ajoute un gain moins évident : les partitions reprenant le rôle des niveaux supérieurs d'un arbre d'index, les parties des index les plus utilisées ont davantage de chances de tenir en mémoire.

C'est la suppression des anciennes données qui change le plus. Supprimer un mois de lignes dans une seule table exige un long `DELETE`, qui laisse des lignes mortes à nettoyer par `VACUUM`. Avec une partition par mois, appliquer la même politique de conservation se résume à un `DROP TABLE` ou à un `DETACH PARTITION`, que la documentation décrit comme bien plus rapide qu'une opération de masse et qui évite entièrement le surcoût de `VACUUM`. Une partition détachée reste une

table ordinaire : il est donc possible de la sauvegarder ou d'en synthétiser les données avant sa suppression. Les partitions rarement lues peuvent être placées sur un stockage moins coûteux et plus lent.

Le gain a un prix, et la documentation dit clairement quand il est rentable. Chaque partition ajoute du travail de planification, et les bénéfices ne justifient généralement l'effort que pour une table qui serait autrement très grande ; une règle empirique veut qu'elle soit plus grande que la mémoire physique du serveur de base de données. Le partitionnement ne change pas non plus une chose : une ligne que se disputent de nombreuses transactions reste une seule ligne dans une seule partition, derrière un seul verrou.

À quoi cela ressemble en pratique

Une table `orders` découpée par mois sur `created_at` est le cas le plus simple. Une note sur la montée en charge d'une base avant le sharding le présente dans sa section consacrée au partitionnement des tables autour d'une ligne très sollicitée. La clé primaire est `(id, created_at)` , car elle doit inclure la colonne de partitionnement. Une requête indiquant une plage de dates ne lit que les partitions de cette plage, les index du mois courant restent assez petits pour tenir en mémoire, et la suppression des données dépassant la durée de conservation se fait avec un seul `DETACH PARTITION` au lieu d'un `DELETE` de millions de lignes. La limite est tout aussi nette : le partitionnement n'accélère pas la ligne soumise à contention, et une requête qui n'indique pas la clé lit toutes les partitions.

Un projet de performance PostgreSQL pour une application contenant des comptes utilisateurs, des organisations clientes, des données géographiques et des relevés de consommation quotidienne d'électricité a découpé la table de consommation électrique en plages temporelles, par jour et par mois, et utilisé le partitionnement par hachage pour les données géographiques. Cette intervention s'accompagnait d'index GiST et composites et de requêtes réécrites ; le résultat combiné a été une amélioration des temps de réponse d'un facteur 10 et une baisse de 40 % de l'utilisation des ressources.

À une échelle bien plus grande, le partitionnement était une technique parmi plusieurs pour au moins 100 bases de données PostgreSQL, SQL Server et BigQuery, avec une grande part de données géographiques et de séries temporelles. Le partitionnement temporel faisait partie de la conception des schémas PostgreSQL et SQL Server, tandis que BigQuery utilisait des tables

partitionnées dans le temps et organisées par clustering. Avec les index spatiaux, les vues matérialisées et la mise en cache, les temps de réponse des requêtes sur ces données ont diminué de 40 à 60 %.

Où les problèmes surviennent

Le partitionnement déplace une partie du travail du moment de la requête vers celui de la conception, et la plupart des problèmes viennent d'une décision de conception prise sans tenir compte des requêtes.

Les insertions échouent dès le premier jour d'un nouveau mois. Aucune partition ne couvre la nouvelle plage, et PostgreSQL refuse une insertion dans le parent qui ne correspond à aucune partition existante. Créez les partitions à l'avance, avec une tâche planifiée générant le DDL comme le suggère la documentation, et déclenchez une alerte lorsque la partition la plus récente approche de sa fin :

```
CREATE TABLE orders_2026_11
  PARTITION OF orders
  FOR VALUES FROM ('2026-11-01')
              TO ('2026-12-01');
```

Une partition par défaut évite ces erreurs, mais les lignes de tous les mois non couverts arrivent alors dans cette même table. Rattacher ensuite une nouvelle partition impose de parcourir la partition par défaut sous un verrou `ACCESS EXCLUSIVE`, sauf si une contrainte `CHECK` sur celle-ci exclut déjà la nouvelle plage. Considérez une partition par défaut non vide comme une alarme.

Les requêtes sont plus lentes après le partitionnement. Les requêtes courantes ne filtrent pas sur la clé de partitionnement, donc chacune lit toutes les partitions et leurs index. Une seconde cause est le nombre de partitions : la documentation indique que le planificateur gère bien jusqu'à quelques milliers de partitions, à condition que les requêtes habituelles les élaguent pour n'en conserver que quelques-unes. Au-delà, les besoins en temps de planification et en mémoire augmentent. Choisissez la clé à partir des clauses `WHERE` que l'application écrit déjà, vérifiez avec `EXPLAIN` que seules les partitions attendues apparaissent, et gardez un nombre proportionné. Une partition par client est un

piège typique, car le nombre de partitions augmente avec l'activité ; la documentation suggère plutôt un partitionnement par hachage avec un nombre raisonnable de partitions.

Impossible de créer une clé primaire ou une contrainte d'unicité. Sur une table partitionnée, une clé primaire ou unique doit inclure toutes les colonnes de la clé de partitionnement, et cette dernière ne doit contenir ni expression ni appel de fonction. L'index de chaque partition ne peut imposer l'unicité qu'à l'intérieur de celle-ci. Ajoutez la colonne de la clé à la contrainte, comme le fait la table `orders` de la note avec `(id, created_at)` .

La construction d'un index sur le parent bloque longtemps les écritures. Un simple `CREATE INDEX` sur le parent bloque les écritures jusqu'à l'indexation de toutes les partitions, et l'option `CONCURRENTLY` fonctionne par partition, pas sur le parent. Créez l'index avec `ON ONLY` sur le parent, ce qui le laisse marqué invalide, construisez l'index de chaque partition avec `CONCURRENTLY` , puis rattachez chacun avec `ALTER INDEX ... ATTACH PARTITION` . Une fois tous les index des partitions rattachés, l'index parent devient valide de lui-même.

Impossible de convertir directement une grande table existante. PostgreSQL ne transforme pas une table ordinaire en table partitionnée. Créez la nouvelle table partitionnée et rattachez l'ancienne comme l'une de ses partitions. Avant le rattachement, ajoutez une contrainte `CHECK` correspondant aux bornes de la partition pour éviter de parcourir toute la table tout en y maintenant un verrou `ACCESS EXCLUSIVE` .

Pour aller plus loin

- Partitionnement de table — le chapitre PostgreSQL : trois méthodes, élagage, maintenance, limites et bonnes pratiques.
- CREATE TABLE — `PARTITION BY` , `PARTITION OF` , bornes des partitions et partition par défaut.
- ALTER TABLE — `ATTACH PARTITION` et `DETACH PARTITION` , avec les verrous pris par chaque opération.
- CREATE INDEX — index sur les tables partitionnées, `ON ONLY` et construction avec `CONCURRENTLY` .

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 le partitionnement ? » 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/partitioning/