Engineer

Партиціонування

поділ однієї таблиці на фізичні частини

Що це таке

Партиціонування, або партиціонування таблиць (table partitioning), зберігає рядки однієї логічної таблиці в кількох звичайних таблицях, кожна з яких містить рядки з ключем у межах власних границь, а запити й записи і далі звертаються до того самого імені, що й завжди. PostgreSQL, на який спирається ця сторінка, називає свою вбудовану форму декларативним партиціонуванням (declarative partitioning): таблицю оголошують партиціонованою, задаючи метод і ключ партиціонування, а кожну партицію створюють як окрему таблицю.

Сама партиціонована таблиця рядків не зберігає. Вставка спрямовується в ту партицію, чиї границі відповідають ключу, а оновлення, що змінює ключ, переносить рядок у партицію, якій він тепер належить. Усі партиції мають ті самі стовпці, що й батьківська таблиця, але кожна може мати власні індекси, параметри зберігання й табличний простір (tablespace), а партицію можна знову партиціонувати.

Методів три. Партиціонування за діапазонами (range) дає кожній партиції проміжок значень, найчастіше дат; нижня границя входить у проміжок, а верхня ні, тож за проміжків 1–10 і 10–20 значення 10 належить другому. Партиціонування за списком (list) перелічує значення, які приймає кожна партиція, наприклад групу кодів країн. Хеш-партиціонування (hash) задає кожній партиції модуль і остачу, і рядок потрапляє туди, де хеш його ключа дає саме цю остачу, що рівномірно розподіляє рядки, коли природного діапазону для поділу немає. Партиція за замовчуванням (default partition) може приймати значення, що не підходять до інших партицій, для range і list, але не для hash.

Партиціонування відбувається в межах одного сервера бази даних. Розподіл даних між окремими серверами, кожен зі своєю копією схеми, — це шардування, і це набагато більший крок. SQL Server і BigQuery мають власні форми партиціонування з власним синтаксисом, і викладені нижче ідеї діють і там, навіть коли команди інші.

Застосуйте поняття до рішення

Застосуйте «Що таке партиціонування?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Пошук запитів і відсутніх індексів у системі, що рік за роком важчає, та виправлення тих, що справді мають значення.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. 30 днів даних про електроенергію за 180 мс — 180 ms на агрегацію місяця показань лічильників

30 днів даних про електроенергію за 180 мс Зв'язатися з нами →

Чому це важливо

Більшість великих таблиць росте з одного кінця. Замовлення, показники, логи й події надходять щодня, зберігаються місяцями чи роками, а більшість запитів стосується найсвіжішої частини. Без партиціонування кожен запит працює з однією таблицею й одним набором індексів, які постійно більшають.

Коли ключ партиціонування відповідає запитам, планувальник може довести, що більшість партицій не здатна містити потрібний рядок, і повністю виключити їх із плану. PostgreSQL називає це відсіканням партицій (partition pruning). Звіт за минулий тиждень читає партицію минулого тижня і більше нічого. Документація називає й менш очевидну перевагу: оскільки партиції беруть на себе роль верхніх рівнів дерева індексу, найуживаніші частини індексів імовірніше вмістяться в пам’яті.

Найбільше змінюється видалення старих даних. Видалити місяць рядків з однієї суцільної таблиці — це довгий `DELETE`, після якого лишаються мертві рядки для `VACUUM`. Якщо на кожен місяць є своя партиція, те саме очищення — це один `DROP TABLE` або один `DETACH PARTITION`, що, за документацією, набагато швидше за масову операцію і повністю обходиться без навантаження від `VACUUM`. Від’єднана партиція лишається звичайною таблицею, тож перед видаленням її можна зберегти в резервній копії чи підсумувати, а партиції, які рідко читають, можна тримати на дешевшому й повільнішому сховищі.

За цю перевагу доводиться платити, і документація прямо каже, коли вона окупається. Кожна партиція додає роботи плануванню, і користь зазвичай вартує зусиль лише тоді, коли таблиця інакше була б дуже великою; емпіричне правило — таблиця має перевищувати фізичну пам'ять сервера бази даних. А одного партиціонування не змінює: рядок, за який змагається багато транзакцій, і далі лишається одним рядком в одній партиції за одним блокуванням.

Як це виглядає на практиці

Перш ніж шардувати, варто розділити на партиції таблиці навколо гарячого рядка: так радить нотатка про масштабування бази даних, де цьому кроку відведено окремий розділ. Її таблицю `orders` поділено за місяцями по `created_at`, з первинним ключем `(id, created_at)`, бо ключ мусить містити стовпець партиціонування. Запит, що називає діапазон дат, читає лише партиції в його межах, індекси цього місяця лишаються досить малими, щоб жити в пам'яті, а очищення стає одним `DETACH PARTITION` замість `DELETE` мільйонів рядків. Нотатка так само прямо називає межу: партиціонування не пришвидшує рядок, за який конкурують транзакції, а запит, що не називає ключа, читає всі партиції.

Проєкт із продуктивності PostgreSQL для застосунку з обліковими записами користувачів, даними орендарів, географічними даними й щоденним споживанням електроенергії розділив таблицю електроенергетичних даних на часові діапазони за днями й місяцями, а для географічних даних застосував хеш-партиціонування. Це поєднувалося з індексами GiST і композитними індексами та переписаними запитами, і загальний результат — час відповіді на запити зменшився в 10 разів, а використання ресурсів знизилося на 40%.

Найбільша за обсягом робота охоплює щонайменше 100 баз даних у PostgreSQL, SQL Server і BigQuery, переважно з GIS-даними й часовими рядами. Партиціонування часових рядів увійшло в дизайн схем для PostgreSQL і SQL Server, а в BigQuery використано таблиці з розбиттям за часом і кластеризацією. Разом із просторовою індексацією, матеріалізованими поданнями й кешуванням час відповіді на запити до цих даних скоротився на 40–60%.

Де виникають проблеми

Партиціонування переносить частину роботи з часу виконання запитів на час проєктування, і більшість проблем походить від рішення, ухваленого без огляду на запити.

Вставки починають повертати помилки першого ж дня нового місяця. Нового діапазону не покриває жодна партиція, а PostgreSQL відхиляє вставку в батьківську таблицю, якщо рядок не відповідає наявним партиціям. Створюйте партиції заздалегідь запланованим завданням, яке генерує DDL, як і радить документація, і надсилайте сповіщення, коли найновіша партиція наближається до свого кінця:

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

Партиція за замовчуванням зупиняє помилки, але тоді рядки всіх непокритих місяців збираються разом в одній таблиці, а коли згодом приєднують нову партицію, партицію за замовчуванням сканують під блокуванням `ACCESS EXCLUSIVE`, якщо обмеження `CHECK` на ній ще не виключає новий діапазон. Вважайте непорожню партицію за замовчуванням тривожним сигналом.

Після партиціонування запити повільніші, ніж до нього. Поширені запити не фільтрують за ключем партиціонування, тож кожен читає всі партиції та їхні індекси. Друга причина — кількість партицій: документація каже, що планувальник досить добре справляється з кількома тисячами, якщо типові запити відсікають усі, крім кількох, а понад це зростають час планування й споживання пам’яті. Обирайте ключ за умовами `WHERE`, які застосунок уже пише, перевіряйте через `EXPLAIN`, що в плані лише очікувані партиції, і тримайте їхню кількість у розумних межах. Окрема партиція на кожного клієнта — типова пастка, бо кількість росте разом із бізнесом; документація натомість пропонує хеш-партиціонування з розумною кількістю партицій.

Первинний ключ чи унікальне обмеження не вдається створити. На партиціонованій таблиці унікальний або первинний ключ мусить містити всі стовпці ключа партиціонування, а ключ партиціонування не може містити

виразів чи викликів функцій. Індекс кожної партиції забезпечує унікальність лише в межах своєї партиції. Додайте стовпець ключа до обмеження, як це зроблено в таблиці `orders` з нотатки, де ключ — `(id, created_at)`.

Побудова індексу на батьківській таблиці надовго блокує записи. Звичайний `CREATE INDEX` на батьківській таблиці блокує записи, доки не проіндексовано кожну партицію, а `CONCURRENTLY` працює для кожної партиції окремо, а не для батьківської таблиці. Створіть індекс з `ON ONLY` на батьківській таблиці, і його буде позначено як недійсний, побудуйте індекс на кожній партиції з `CONCURRENTLY` і приєднайте кожен через `ALTER INDEX ... ATTACH PARTITION`. Щойно індекси всіх партицій приєднано, батьківський індекс сам стає дійсним.

Наявну велику таблицю не вдається перетворити на місці. PostgreSQL не робить зі звичайної таблиці партиціоновану. Створіть нову партиціоновану таблицю й приєднайте стару як одну з її партицій. Перед приєднанням додайте обмеження `CHECK`, що відповідає границям партиції, щоб приєднання не сканувало всю таблицю, тримаючи на ній блокування `ACCESS EXCLUSIVE`.

Що почитати далі

- Table Partitioning — розділ документації PostgreSQL: три методи, відсікання партицій, обслуговування, обмеження й найкращі практики.
- CREATE TABLE — `PARTITION BY`, `PARTITION OF`, границі партицій і партиція за замовчуванням.
- ALTER TABLE — `ATTACH PARTITION` і `DETACH PARTITION` та які блокування бере кожна з них.
- CREATE INDEX — індекси на партиціонованих таблицях, `ON ONLY` і побудова з `CONCURRENTLY`.

Словник

Обирайте за фактичними умовами

Надайте репрезентативний набір даних, запитання до нього та поточне вузьке місце. Узгодимо вимірювання якості, пропускної здатності та відновлення.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке партиціонування?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/partitioning/