Engineer

Запит до бази даних

▼ записане питання, на яке відповідає база даних

Що це таке

Запит до бази даних — це прохання видати дані, записане мовою, яку ця база розуміє; у відповідь вона повертає результат, здебільшого набір рядків, іноді одне число. Посібник PostgreSQL визначає це одним реченням: отримання даних із бази, або команду, яка їх отримує, називають запитом (query), і в SQL така команда — SELECT. У щоденному вжитку слово ширшає, і інженери називають запитом майже будь-яку інструкцію до бази, разом із тими, що змінюють дані й щось повертають. Нижче трапляються обидва значення. Текст запиту, план запиту, кількість запитів на одну сторінку — усе це про запити, які застосунок надсилає базі, і саме від них залежить, як вона поводиться під навантаженням.

Питання про те, які рахунки-фактури найдовше чекають на оплату, звучить у SQL майже так, як його поставили б уголос:

```
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;
```

Текст називає, що потрібно, а спосіб лишає базі. Між отриманням тексту й поверненням першого рядка планувальник PostgreSQL перебирає способи дістати відповідь — скажімо, прочитати всю таблицю invoices або пройти індексом за `due_on`, — оцінює вартість кожного й виконує найдешевший. Вартість вимірюють в умовних одиницях, які традиційно прив’язують до

читання однієї сторінки з диска. Тому два запити, схожі на екрані, можуть відрізнятися у тисячу разів, коли дані під ними зростуть, і побачити цю різницю можна лише в плані.

Частини запиту виконуються в іншому порядку, ніж їх записано. Посібник описує обробку SELECT так: спершу WITH, далі FROM, WHERE, групування й HAVING, потім список виведення і лише наприкінці ORDER BY та LIMIT. Цей порядок пояснює правило, яке більшість людей колись дивує: ім’я, дане стовпцю в списку SELECT, працює в ORDER BY і GROUP BY, але WHERE виконується раніше, ніж це ім’я з’являється, тож там вираз доводиться писати повністю. Результат будь-якого запиту сам є таблицею, тому запити вкладаються один в одний: підзапит або конструкція WITH передає свої рядки більшому запиту.

Застосуйте поняття до рішення

Застосуйте «Що таке запит до бази даних?» 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

У застосунку з реляційною базою даних саме запити значною мірою визначають, наскільки швидкою здається сторінка і скільки заліза потрібно системі. Які запити важать найбільше — питання арифметики, а не чуття: вартість запиту для системи — це його тривалість, помножена на частоту. Запит на 2 мс, що виконується мільйон разів на годину, дає понад пів години роботи бази щогодини, а звіт на дві секунди, який запускають двічі на день, майже непомітний.

PostgreSQL веде цей облік у розширенні pg_stat_statements. Воно об’єднує інструкції, що різняться лише конкретними значеннями, в один запис і зберігає, скільки разів кожна виконувалася і скільки часу забрала загалом. Сортування за сумарним часом, а не за середнім, — звичка, яка в роботі із запитами окупається першою: вона вказує на інструкцію, яку варто виправити, а не на ту, що лише здається повільною.

Кожен запит коштує ще й звернення туди й назад. Застосунок надсилає текст, сервер розбирає, планує й виконує його, а рядки повертаються мережею. Для швидкого пошуку за індексом саме звернення може важити більше за роботу. Тож форма розмови з базою важить не менше, ніж окрема інструкція: один UPDATE, що зачіпає сто рядків, — це одне звернення, а цикл, який читає сто рядків і оновлює їх по одному, — сто одне.

Лишається питання, хто пише запити. ORM або конструктор запитів (query builder) генерує їх із коду застосунку: це швидко для простих читань і записів і тримає код однією мовою. Водночас він ховає, скільки інструкцій надсилає один виклик і як вони виглядають, доки сторінка не стане повільною. Написаний вручну SQL показує і те, і те, але потребує власника, як будь-який інший код. Робочий поділ віддає ORM рутинні операції із записами, гарячі шляхи пише вручну і читає згенерований SQL усього, що піднімається на верх рейтингу за сумарним часом.

Як це виглядає на практиці

Метод для повільної бази в нотатці Оптимізуйте базу, перш ніж її шардувати починається із запитів — раніше, ніж хтось оцінить вартість нових машин. Перший крок питає PostgreSQL, які інструкції коштують найбільше сумарно, через pg_stat_statements, відсортований за загальним часом виконання. Другий запускає EXPLAIN (ANALYZE, BUFFERS) для інструкції на першому місці, щоб побачити обраний базою план, скільки рядків кожен крок повернув насправді проти очікуваного планувальником і скільки сторінок прочитав. Практичне правило нотатки: сканування мільйона рядків заради десяти або оцінка, що помиляється в сто разів, зазвичай самі пояснюють повільність. Нотатка враховує й звернення до бази та проводить ту саму межу, що й ця стаття вище: усе, що має бачити дані, переходить у базу, а бізнес-правила лишаються в застосунку.

Запит може виконувати й роботу блокування. У нотатці Продати останній квиток рівно один раз флеш-розпродаж ламається двома способами. Якщо застосунок читає залишок, ухвалює рішення і записує нове число назад, двоє покупців можуть узяти останню одиницю; якщо ж він тримає блокування рядка весь цей час, усі інші покупці чекають, доки його знімуть. Виправлення зливає перевірку й зменшення залишку в один умовний UPDATE, який віднімає кількість лише тоді, коли залишку вистачає, і повертає нове значення. Повернувся один рядок — одиницю зарезервовано, нуль рядків — розпродано. На рівні READ COMMITTED, типовому для PostgreSQL, другий покупець, що дістався рядка, поки його тримає перший, чекає, а потім знову

перевіряє умову WHERE на вже зафіксованому рядку, тож остання одиниця дістається рівно одному з них. За запитом стоїть ще й обмеження CHECK, яке тримає залишок на рівні нуля або вище.

Де виникають проблеми

Дешевий запит, що забирає все навантаження. База працює на межі, а журнал повільних запитів мовчить, бо кожен виклик завершується за кілька мілісекунд. Причина — частота: один швидкий запит, що виконується на кожне звернення до сервера, часто з циклу або з компонента, присутнього на кожній сторінці. Відсортуйте інструкції за сумарним часом у pg_stat_statements, а тоді кешуйте результат, об'єднуйте виклики в пакети або здешевіть запит індексом, що відповідає його фільтру.

Сторінка, що надсилає сотні запитів. Кожен запит у трасуванні триває мілісекунду, а сторінка все одно вантажиться секундами. Звична причина — ORM, що ліниво підвантажує пов'язані записи, по запиту на кожен батьківський рядок; цей шаблон називають N+1. Отримуйте пов'язані рядки одним запитом, через з'єднання (JOIN) або через `WHERE id = ANY($1)` з масивом ключів, і стежте за кількістю інструкцій на один виклик так само уважно, як за тривалістю кожної.

Остання одиниця, продана двічі. Два виклики читають той самий залишок, обидва вирішують, що його вистачає, і обидва записують. Коли читання й запис ідуть окремими інструкціями, між перевіркою та дією лишається щілина, і паралельний трафік її знаходить. Закрийте щілину всередині запиту: умовним UPDATE з RETURNING, як вище, або SELECT … FOR UPDATE, що тримає блокування якомога коротше і без мережевих викликів між блокуванням і фіксацією транзакції.

Діагностика, що змінила дані. Хтось запускає EXPLAIN ANALYZE для повільного DELETE, щоб подивитися план, і рядки зникають. ANALYZE справді виконує інструкцію й відкидає лише її вивід, тож усі побічні ефекти відбуваються. Обгортайте будь-яку інструкцію, що змінює дані, у BEGIN і ROLLBACK перед аналізом або користуйтеся звичайним EXPLAIN, який лише планує, коли вистачає оцінок.

Що почитати далі

- SELECT — повний синтаксис запиту в PostgreSQL разом із порядком, у якому обробляються його частини.
- 7.1. Overview — початок розділу посібника про запити з визначенням самого слова.
- 51.1. The Path of a Query — шлях тексту від клієнта через парсер, переписування й планувальник до виконавця.
- 14.1. Using EXPLAIN — як читати план, що означають одиниці вартості і чому ANALYZE виконує інструкцію.
- F.32. pg_stat_statements — track statistics of SQL planning and execution — розширення, що впорядковує запити за їхньою сумарною вартістю.
- 13.2. Transaction Isolation — що бачить другий записувач на рівні READ COMMITTED і чому умову WHERE перевіряють знову.

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 запит до бази даних?» 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/query/