Engineer

consulta

Pregunta escrita a la que responde una base de datos

Qué es

Una consulta es una petición de datos escrita en un lenguaje que la base de datos entiende y que devuelve un resultado: normalmente un conjunto de filas y, a veces, un único número. El manual de PostgreSQL la define en una línea: obtener datos de una base de datos, o el comando que los obtiene, se denomina consulta, y en SQL ese comando es SELECT. El uso cotidiano amplía la palabra, y los ingenieros llaman consulta a casi cualquier instrucción que envían a una base de datos, incluidas las que modifican datos y devuelven algo. Ambos sentidos aparecen a continuación, y el plural, consultas, suele referirse a todo el tráfico que envía una aplicación.

La consulta que pregunta qué facturas llevan más tiempo pendientes de pago se lee casi como se formularía la pregunta en voz alta:

```
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;
```

El texto indica qué quiere y deja abierto el método. Entre recibirlo y devolver la primera fila, el planificador de PostgreSQL enumera las formas de obtener la respuesta, como leer toda la tabla de facturas o recorrer un índice sobre `due_on`, estima un coste para cada una y ejecuta la más barata. Esos costes se expresan en unidades arbitrarias, convencionalmente referidas a la lectura de una página del disco. Por eso, dos consultas que parecen iguales en pantalla pueden diferir en un factor de mil cuando aumenta el volumen de los datos con los que trabajan, y el plan es el único lugar donde se ve esa diferencia.

Las cláusulas no se ejecutan en el orden en que se escriben. El manual describe el procesamiento de SELECT así: primero WITH, después FROM, WHERE, la agrupación y HAVING, luego la lista de salida y, solo después, ORDER BY y LIMIT. Ese orden explica una regla que sorprende a casi todo el mundo alguna vez: un nombre asignado a una columna en la lista de selección funciona en ORDER BY y GROUP BY, mientras que WHERE se ejecuta antes de que exista el nombre, por lo que allí hay que volver a escribir la expresión completa. El resultado de cada consulta es a su vez una tabla; por eso las consultas pueden anidarse, con una subconsulta o una cláusula WITH que aporta sus filas a otra mayor.

Utilice el concepto en una decisión

Aplique «¿Qué es una consulta?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Detectar las consultas y los índices que faltan en un sistema cada año más pesado y corregir lo que realmente importa.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 30 días de datos eléctricos en 180 ms — 180 ms para agregar un mes de lecturas de contadores

30 días de datos eléctricos en 180 ms Póngase en contacto →

Por qué importa

En una aplicación respaldada por una base de datos relacional, las consultas determinan buena parte de la velocidad que percibe el usuario en una página y del hardware que necesita el sistema. Saber qué consultas importan requiere aritmética, no intuición: el coste de una consulta para el sistema es su duración multiplicada por la frecuencia con que se ejecuta. Una búsqueda de 2 ms ejecutada un millón de veces por hora suma más de media hora de tiempo de base de datos cada hora, mientras que un informe que tarda dos segundos y se ejecuta dos veces al día apenas se nota.

PostgreSQL lleva esa cuenta en la extensión pg_stat_statements. Agrupa en una sola entrada las instrucciones que solo difieren en sus valores literales y registra cuántas veces se ejecutó cada una y cuánto tiempo consumió en total. Ordenar por el total en vez de por la media es el primer hábito que da resultado al trabajar con consultas, porque señala la instrucción que merece corregirse en lugar de la que solo parece lenta.

Cada consulta también cuesta un viaje de ida y vuelta. La aplicación envía texto, el servidor lo analiza, lo planifica y lo ejecuta, y las filas regresan. En una búsqueda rápida mediante un índice, el viaje puede costar más que el trabajo.

Por eso, la forma de la conversación importa tanto como cada instrucción: un UPDATE que modifica cien filas hace un viaje, y un bucle que lee cien filas y actualiza cada una por separado hace ciento uno.

Después está la cuestión de quién escribe las consultas. Un ORM o un constructor de consultas las genera a partir del código de la aplicación, lo que resulta rápido para lecturas y escrituras sencillas y mantiene el código en un solo lenguaje. También oculta cuántas instrucciones envía una petición y qué aspecto tienen, hasta que una página se vuelve lenta. El SQL escrito a mano muestra ambas cosas, a cambio de que alguien se responsabilice de él como de cualquier otro código. Un reparto viable deja los registros rutinarios al ORM, escribe a mano las rutas críticas y lee el SQL generado de cualquier operación que suba a los primeros puestos de los totales.

Cómo se aplica en la práctica

El método que propone Optimice la base de datos antes de fragmentarla para una base de datos lenta empieza por las consultas, antes de que nadie calcule el precio de más máquinas. El primer paso pregunta a PostgreSQL qué instrucciones cuestan más en total, utilizando pg_stat_statements ordenado por tiempo total de ejecución. El segundo ejecuta EXPLAIN (ANALYZE, BUFFERS) sobre la primera instrucción de la lista para ver el plan que eligió la base de datos, las filas que devolvió realmente cada paso frente a las que esperaba el planificador y cuántas páginas leyó. La regla orientativa de la nota es que recorrer un millón de filas para devolver diez, o una estimación que se desvía en un factor de cien, suele explicar por sí solo la lentitud. También cuenta los viajes de ida y vuelta y traza la misma división que este artículo: lo que necesita acceder a los datos pasa a la base de datos, y las reglas de negocio permanecen en la aplicación.

Una consulta también puede hacer el trabajo de un bloqueo. En Vender la última entrada exactamente una vez, una venta relámpago falla de una de dos maneras. Leer las existencias, decidir en la aplicación y escribir el nuevo número permite que dos compradores se lleven ambos la última unidad; mantener un bloqueo de fila durante esa lógica de la aplicación hace que cada comprador espere detrás de él. La solución reúne la comprobación y el decremento en un UPDATE condicional que solo resta la cantidad donde quedan existencias suficientes y devuelve el nuevo nivel. Una fila devuelta significa que la unidad está reservada y cero filas significa que se ha agotado. Con READ COMMITTED, el nivel de aislamiento predeterminado de PostgreSQL, un segundo comprador que llega a la fila mientras la retiene el primero espera y después vuelve a evaluar la cláusula

WHERE sobre la fila confirmada, de modo que la última unidad va exactamente a uno de ellos. Una restricción CHECK que mantiene las existencias en cero o por encima respalda la consulta.

Dónde surgen los problemas

Una consulta barata domina la carga. La base de datos trabaja al límite, pero el registro de consultas lentas no muestra nada, porque cada llamada termina en unos milisegundos. La causa es la frecuencia: una consulta rápida llamada en cada petición, a menudo desde un bucle o desde un componente que aparece en todas las páginas. Ordene las instrucciones por tiempo total de ejecución en pg_stat_statements y después almacene el resultado en caché, agrupe las llamadas o abarate la consulta con un índice que coincida con su filtro.

Una página envía cientos de consultas. Cada consulta de la traza tarda un milisegundo y la página sigue tardando segundos. La causa habitual es un ORM que carga los registros relacionados de forma diferida, con una consulta por fila principal: el conocido patrón N+1. Cargue las filas relacionadas en una sola consulta, con un join o con `WHERE id = ANY($1)` y un array de claves, y vigile el número de instrucciones por petición con tanta atención como la duración de cada una.

La última unidad se vende dos veces. Dos peticiones leen el mismo nivel de existencias, ambas deciden que hay suficientes y ambas escriben. Leer y escribir en instrucciones separadas deja un hueco entre la comprobación y la acción, y el tráfico concurrente lo encuentra. Cierre ese hueco dentro de la consulta, con un UPDATE condicional y RETURNING como el anterior, o con SELECT … FOR UPDATE mantenido el menor tiempo posible y sin ninguna llamada de red entre el bloqueo y el commit.

Un diagnóstico modifica los datos. Alguien ejecuta EXPLAIN ANALYZE sobre un DELETE lento para ver su plan y las filas desaparecen. ANALYZE ejecuta la instrucción de verdad y descarta únicamente su salida, por lo que se producen todos los efectos secundarios. Encierre cualquier instrucción que modifique datos entre BEGIN y ROLLBACK antes de analizarla, o use EXPLAIN sin ANALYZE, que planifica sin ejecutar, cuando basten las estimaciones.

Lecturas adicionales

- SELECT — La sintaxis completa de una consulta en PostgreSQL, incluido el orden en que se procesan sus cláusulas.
- 7.1. Visión general — El inicio del capítulo del manual sobre consultas, con la definición de la palabra.
- 51.1. El recorrido de una consulta — El trayecto desde el texto del cliente, pasando por el analizador, el reescritor y el planificador, hasta el ejecutor.
- 14.1. Uso de EXPLAIN — Cómo leer un plan, qué significan las unidades de coste y por qué ANALYZE ejecuta la instrucción.
- F.32. pg_stat_statements: seguimiento de estadísticas de planificación y ejecución de SQL — La extensión que clasifica las consultas por su coste total.
- 13.2. Aislamiento de transacciones — Qué ve un segundo escritor con READ COMMITTED y por qué se vuelve a evaluar la cláusula WHERE.

Glosario

Elija según sus restricciones reales

Traiga un conjunto de datos representativo, las preguntas que debe responder y el cuello de botella actual. Podemos acordar cómo medir la calidad, la capacidad de procesamiento y la recuperación.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es una consulta?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/query/