Engineer

CTE

consulta con nombre que una sentencia lee como una tabla

Qué es

Una CTE, abreviatura de common table expression o expresión de tabla común, es una consulta a la que se da nombre en una cláusula WITH al principio de una sentencia, para que el resto de esta pueda leer sus filas como si procedieran de una tabla. Una sentencia puede definir varias CTE, cada una capaz de leer las nombradas antes, y la consulta final las combina. El manual de PostgreSQL las describe como «tablas temporales que existen solo para una consulta», una imagen adecuada con una salvedad: no se almacena nada con ese nombre y la siguiente sentencia no puede verlo. Cada CTE puede ser un SELECT o, en PostgreSQL, un INSERT, UPDATE, DELETE o MERGE, y la sentencia a la que se añade la cláusula WITH también puede ser cualquiera de ellas.

Una pregunta como «qué contadores consumieron más del triple de su propio consumo habitual en algún día de los últimos 30 días» se divide de forma natural en pasos:

```
WITH daily AS (
  SELECT meter_id, read_at::date AS day, sum(kwh) AS kwh
  FROM readings
  WHERE read_at >= current_date - 30
  GROUP BY meter_id, read_at::date
),
typical AS (
  SELECT meter_id, avg(kwh) AS avg_kwh
  FROM daily
  GROUP BY meter_id
)
SELECT d.meter_id, d.day, d.kwh, t.avg_kwh
FROM daily AS d
JOIN typical AS t USING (meter_id)
WHERE d.kwh > 3 * t.avg_kwh
ORDER BY d.day, d.meter_id;
```

El primer paso suma las lecturas diarias de cada contador; el segundo calcula la media de esos totales por contador; y el último compara cada día con la media de su propio contador. La misma respuesta puede escribirse con subconsultas anidadas, pero los totales diarios aparecerían dos veces en el texto, una dentro de la otra, y el lector tendría que trabajar de dentro hacia fuera. Con WITH, la lógica se lee de arriba abajo y cada paso puede ejecutarse por separado mientras se comprueba.

Añadir RECURSIVE permite que una CTE lea su propia salida, algo que SQL sin esa construcción no tiene otra forma de expresar. Una CTE recursiva tiene una consulta inicial, después UNION o UNION ALL y después una consulta que vuelve a referirse a la CTE. PostgreSQL ejecuta la consulta inicial, guarda sus filas en una tabla de trabajo y sigue ejecutando la segunda sobre las filas más recientes hasta que una ronda no añade ninguna. Así recorre una sentencia un árbol de profundidad desconocida, como las categorías que dependen de un encabezado:

```
WITH RECURSIVE subtree AS (
  SELECT id, parent_id, name, 1 AS depth
  FROM categories
  WHERE id = 42
  UNION ALL
  SELECT c.id, c.parent_id, c.name, s.depth + 1
  FROM categories AS c
  JOIN subtree AS s ON c.parent_id = s.id
)
SELECT id, name, depth FROM subtree ORDER BY depth, name;
```

Utilice el concepto en una decisión

Aplique «¿Qué es una CTE?» 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

La primera razón es la legibilidad. Una consulta analítica de cien líneas con cuatro niveles de anidamiento es difícil de revisar y más difícil aún de cambiar. La misma lógica como cadena de pasos con nombre da a quien revisa una frase por paso,

da al autor un lugar para poner un comentario y permite ejecutar un paso aislado para ver si sus filas parecen correctas antes de confiar en el siguiente.

La segunda razón es cómo evalúa la base de datos los pasos, y aquí los detalles han cambiado con el tiempo. PostgreSQL evalúa normalmente una consulta WITH una vez por ejecución de la sentencia, aunque esta la mencione varias veces, por lo que un cálculo costoso necesario en dos lugares se ejecuta una sola vez. El coste de esa garantía es que el planificador no puede trasladar un filtro de la consulta externa a una CTE referenciada varias veces, porque el filtro debería aplicarse a uno de sus usos y no a los demás. La CTE se calcula completa y las filas se descartan después.

Antes de PostgreSQL 12, eso ocurría con todas las CTE: cada una se evaluaba antes que el resto de la consulta y nunca se integraba en ella. La versión 12 cambió el comportamiento predeterminado. Una CTE no recursiva, sin efectos secundarios y referenciada una sola vez se integra ahora en la consulta externa, de modo que el planificador optimiza juntos los dos niveles y puede llevar el filtro externo hasta un índice de la tabla base. MATERIALIZED fuerza el comportamiento antiguo y NOT MATERIALIZED fuerza la integración incluso para una CTE utilizada varias veces, con el riesgo de ejecutar su trabajo dos veces. SQLite acepta las mismas dos palabras clave como indicaciones no vinculantes y dice que tomó la idea de PostgreSQL. Otros motores deciden por su cuenta, y el plan es donde hay que comprobarlo.

La tercera razón es que una CTE puede modificar datos. Un DELETE con RETURNING en una cláusula WITH puede pasar su salida a un INSERT de la sentencia principal, lo que mueve filas de una tabla a otra en una sola sentencia que se ejecuta por completo o no se ejecuta. Las CTE que modifican datos se ejecutan exactamente una vez y siempre hasta terminar, se lea o no su salida, lo que las convierte en una herramienta para agrupar varias escrituras, no en una forma de describir un resultado.

Cómo se aplica en la práctica

El trabajo de acelerar 10 veces el rendimiento de PostgreSQL se realizó sobre una base de datos de cuentas de usuario, organizaciones cliente, ubicaciones y lecturas diarias de electricidad que se ralentizaba al crecer los datos. La solución tuvo tres partes: índices específicos y GiST, particionamiento temporal de la tabla de lecturas y reescritura de consultas. Las reescrituras utilizaban CTE y vistas materializadas para los datos consultados con frecuencia, a fin de evitar

recorridos completos de tablas, con EXPLAIN ANALYZE señalando las uniones y subconsultas que convenía reestructurar. La mejora de diez veces en los tiempos de respuesta y la caída del 40 % en el uso de recursos procedieron de las tres partes juntas, no solo de las CTE. Conviene señalar la combinación porque cumplen funciones distintas: una CTE da nombre a un resultado intermedio durante una sentencia, mientras que una vista materializada almacena un resultado que muchas sentencias pueden leer hasta que se actualiza.

La otra vertiente de las CTE, la legibilidad, sustentó los 500 informes de análisis de datos eléctricos y GIS. Todos los informes se escribieron en SQL puro para PostgreSQL y BigQuery sobre datos de contadores inteligentes con coordenadas, identificadores de contador y consumos con marca de tiempo, y los scripts utilizaban CTE junto con funciones de ventana, subconsultas y uniones geoespaciales. Cada informe se entregó con SQL anotado para que otros miembros del equipo pudieran reproducir y auditar el análisis, además de notas sobre problemas comunes de calidad de los datos, como coordenadas GPS ausentes y valores de contadores corruptos. Un SQL que se lee paso a paso se comparte bien: los informes se convirtieron en una referencia entre departamentos y ayudaron a mejorar un 10–15 % la eficiencia de la planificación operativa. Los pasos con nombre son lo que permite anotar un SQL de ese tipo: una nota encima de cada paso puede indicar qué elimina o qué da por supuesto.

Dónde surgen los problemas

Un filtro aplicado cuando todo el trabajo ya está hecho. Una CTE referenciada dos veces o marcada MATERIALIZED se calcula completa antes de que la consulta externa la filtre, y la copia que produce no tiene índices. Una consulta que debería tocar unas pocas filas lee una tabla entera. Lea el plan: si la CTE aparece como un paso independiente, con recuentos de filas muy superiores a lo que devuelve la consulta, pruebe NOT MATERIALIZED o reformule la consulta para referenciar la CTE una sola vez. Mantenga la forma materializada cuando la CTE llame a una función costosa, para que se ejecute una vez por fila y no una vez por referencia.

Una consulta recursiva que no termina. Unos datos con un ciclo, como dos categorías que figuren cada una como padre de la otra, hacen que una CTE recursiva con UNION ALL dé vueltas mientras el servidor lo permita. UNION solo ayuda cuando se repiten filas completas, algo que rara vez ocurre al añadir una columna de profundidad. Registre en un array el recorrido de claves visitadas y deténgase si reaparece una, o utilice la cláusula CYCLE añadida en PostgreSQL 14. Un LIMIT en la consulta externa sirve como truco de prueba, y el manual

desaconseja depender de él en producción. SQL Server limita la recursión de forma predeterminada a 100 niveles y cancela la sentencia al superarlos, así que el mismo error se detiene en lugar de continuar.

Cambios que el resto de la sentencia no ve. Una CTE que modifica datos y la consulta principal se ejecutan de forma concurrente sobre la misma instantánea, así que una consulta principal que lea directamente la tabla de destino ve las filas tal como estaban antes del cambio. El único canal entre ambas es RETURNING. Si dos partes de una sentencia modifican la misma fila, es peor: solo surte efecto una modificación y es difícil prever cuál. Lea las filas cambiadas de la salida RETURNING de la CTE y asegúrese de que cada fila solo la modifique una parte de la sentencia.

Un plan que cambió con una actualización. El código escrito para PostgreSQL 11 o anterior a veces dependía de que una CTE se evaluara primero, como barrera deliberada contra un plan que el autor no quería. Tras actualizar a 12 o posterior, la misma CTE se integra en la consulta externa y el planificador elige algo nuevo, normalmente más rápido y en ocasiones mucho más lento. Compare la salida de EXPLAIN de las sentencias lentas antes y después de la actualización y añada MATERIALIZED donde la barrera fuera intencionada.

Lecturas adicionales

- 7.8. Consultas WITH (expresiones de tabla comunes) — el capítulo del manual sobre CTE: recursión, detección de ciclos, materialización y sentencias que modifican datos.
- E.23. Versión 12 — la versión que empezó a integrar las CTE de un solo uso en la consulta externa.
- E.25. Versión 14 — la versión que añadió las cláusulas SEARCH y CYCLE del estándar SQL.
- La cláusula WITH — la variante de SQLite, incluidas sus indicaciones de materialización y de dónde tomó la idea.
- WITH common_table_expression (Transact-SQL) — las reglas de SQL Server para CTE, incluido el límite MAXRECURSION de recursión.

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 CTE?» 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/cte/