Engineer

Vista materializada

Vista que guarda el resultado de su consulta como una tabla

Qué es

Una vista materializada es una consulta cuyo resultado conserva la base de datos: leerla devuelve filas calculadas anteriormente, en lugar de ejecutar de nuevo la consulta. En inglés británico se escribe materialised view. Se sitúa entre otros dos objetos más antiguos. Una vista normal solo guarda su consulta, que se ejecuta sobre las tablas subyacentes en cada lectura. Una tabla creada con `CREATE TABLE AS` guarda un resultado, pero olvida la consulta que lo produjo. Una vista materializada conserva ambos: las filas que se leen y la consulta, que puede ejecutarse de nuevo para sustituirlas.

PostgreSQL incorporó las vistas materializadas en la versión 9.3, publicada el 9 de septiembre de 2013; las notas de la versión atribuyen el trabajo a Kevin Grittner. La documentación explica que utilizan el sistema de reglas como las vistas normales, pero conservan el resultado con una estructura similar a una tabla. Para el analizador de consultas, una vista materializada es una relación como cualquier tabla o vista: un `SELECT` lee directamente las filas almacenadas; la consulta guardada solo importa al rellenar la vista. La vista no puede actualizarse directamente. Para sustituir su contenido por el resultado actualizado de una consulta, hay que volver a ejecutar la consulta guardada.

Dos órdenes realizan ese trabajo. `CREATE MATERIALIZED VIEW` define la vista y, salvo que se indique `WITH NO DATA`, ejecuta la consulta de inmediato para rellenarla. `REFRESH MATERIALIZED VIEW` vuelve a ejecutar la consulta guardada y sustituye todo el contenido por el nuevo resultado. La versión 9.4, publicada el 18 de diciembre de 2014, añadió `CONCURRENTLY`, que permite actualizar sin bloquear las sesiones que leen la vista. Ambas órdenes son extensiones de PostgreSQL y no forman parte del estándar SQL; otras bases de datos ofrecen la idea con su propia sintaxis y sus reglas.

Utilice el concepto en una decisión

Aplique «¿Qué es una vista materializada?» 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

Algunas consultas son costosas y se repiten mucho: un panel que suma por día un año de pedidos, un informe que une seis tablas o una página que clasifica productos por ventas. Sobre las tablas de origen, cada solicitud vuelve a pagar la exploración, la unión y la agregación completas, aunque la respuesta apenas cambie entre solicitudes. Una vista materializada paga ese coste una vez por actualización; cada lectura posterior explora una tabla pequeña calculada de antemano. La documentación de PostgreSQL indica que leer los datos almacenados es «a menudo mucho más rápido que acceder a las tablas subyacentes directamente o mediante una vista».

El precio es la actualidad de los datos. Entre actualizaciones, la vista muestra cómo estaban en la última; en PostgreSQL, nada la actualiza si no lo solicita alguien o una tarea programada. Muchos usos pueden aceptarlo. Las ventas de ayer no cambian hoy y un informe mensual no necesita los pedidos de este minuto. La vista materializada obliga a decidir explícitamente qué antigüedad puede tener la respuesta y quién debe impedir que la supere.

También proporciona un lugar para los índices: el resultado almacenado puede tener índices propios, diseñados para las consultas que lo leen. La documentación de PostgreSQL muestra otro uso: los datos traídos de otro sistema mediante un adaptador de datos externos, que quizá no admita índices, pueden copiarse a una vista materializada que sí los admita.

Para un equipo que mantiene normalizado su esquema, este suele ser el lugar adecuado para la estructura aplanada. La normalización almacena cada hecho una vez y responde a la mayoría de las preguntas mediante uniones. Una vista materializada puede conservar el resultado de las uniones y las sumas que necesitan los informes, mientras las tablas de origen mantienen una sola copia de cada hecho y todas las escrituras siguen yendo a un único lugar.

Cómo se aplica en la práctica

Pensemos en las tablas de pedidos de una tienda: `orders`, con una fila por pedido y la fecha en que se hizo, y `order_lines`, con el SKU, la cantidad y el precio unitario de cada línea. Un panel de ventas necesita unidades e ingresos por producto y día. La consulta es breve, pero con varios años de pedidos lee todas las líneas:

```
CREATE MATERIALIZED VIEW daily_sales AS
SELECT o.ordered_on                  AS day,
       l.sku,
       sum(l.quantity)               AS units,
       sum(l.quantity * l.unit_price) AS revenue
FROM   orders o
JOIN   order_lines l USING (order_id)
WHERE  o.ordered_on < CURRENT_DATE
GROUP  BY o.ordered_on, l.sku;
```

La cláusula `WHERE` excluye el día actual, cuyas cifras aún cambian: el panel solo muestra días completos. El ejemplo de la documentación de PostgreSQL trata del mismo modo un día de ventas aún sin terminar. La orden ejecuta la consulta una vez y guarda el resultado. Después se crea un índice:

```
CREATE UNIQUE INDEX daily_sales_sku_day ON daily_sales (sku, day);
```

El índice cumple dos funciones. Permite al panel encontrar los días de un producto sin leer toda la vista y hace posible una actualización concurrente posterior. La consulta agrupa por día y SKU: cada pareja aparece una vez y el índice puede ser único. A partir de aquí, el panel lee la vista como una tabla:

```
SELECT day, units, revenue
FROM   daily_sales
WHERE  sku = 'TEA-250'
ORDER  BY day;
```

Cada noche, poco después de medianoche, una tarea programada la pone al día:

```
REFRESH MATERIALIZED VIEW CONCURRENTLY daily_sales;
```

Sin `CONCURRENTLY`, una actualización que afecta a muchas filas tiende a consumir menos recursos y terminar antes, pero podría bloquear las lecturas del panel hasta acabar. Con esa opción, las lecturas continúan durante la actualización; la documentación señala que incluso puede ser más rápida si cambian pocas filas, como aquí, donde cada noche se añade un día. La opción exige dos condiciones. La vista debe contener ya datos y tener al menos un índice único sobre nombres de columnas simples que abarque todas las filas: no sirve un índice de expresiones ni uno con cláusula `WHERE`. El índice anterior satisface la segunda; crear la vista con sus datos satisfizo la primera.

Otras bases de datos fijan límites distintos. En BigQuery, las vistas materializadas se mantienen en segundo plano: los cambios de las tablas de base se añaden de forma incremental sin solicitarlos, una consulta a la vista incorpora los cambios recientes para mantener actualizado el resultado y la optimización inteligente puede redirigir hacia la vista una consulta escrita sobre las tablas de base. Esto se aplica a las vistas incrementales, que admiten un subconjunto limitado del lenguaje SQL. Una vista definida con `allow_non_incremental_definition = true` admite la mayoría de las consultas, pero cada actualización ejecuta la consulta completa y sus resultados no están siempre actualizados.

SQL Server logra un resultado similar con vistas indexadas. Añadir un índice agrupado que imponga unicidad a una vista, que debe crearse con `WITH SCHEMABINDING` y tener una definición determinista, almacena su resultado con la misma organización que una tabla indexada; el optimizador puede usarlo incluso en una consulta que no nombre la vista. SQL Server actualiza la vista indexada con cada inserción, actualización y borrado en sus tablas de origen: no hay actualización periódica que programar. El coste se traslada a las escrituras y la documentación de Microsoft advierte de que su rendimiento puede degradarse de forma considerable.

La misma contrapartida aparece en el trabajo de producción. En un puesto anterior, la fundadora construyó flujos de procesamiento ETL y ELT en PL/pgSQL que utilizaban tablas particionadas y vistas materializadas para acelerar las lecturas en grandes conjuntos de datos, junto con restricciones de claves primarias y externas para la integridad referencial e índices únicos y compuestos para las uniones.

Dónde surgen los problemas

Cifras antiguas que parecen actuales. Una tarea de actualización detenida hace una semana deja que la vista responda a todas las consultas deprisa y de forma incorrecta, sin que el resultado indique su antigüedad. Haga que la tarea registre la hora de cada actualización correcta donde el panel pueda mostrarla y emita una alerta cuando el tiempo transcurrido supere el máximo aceptado por los lectores.

Una actualización bloquea a los lectores. Un `REFRESH MATERIALIZED VIEW` normal sobre una vista grande podría bloquear todas las sesiones que intenten leerla hasta terminar. Añada un índice único y use `CONCURRENTLY`, o programe las actualizaciones normales en horas sin lectores. Si cambia la mayoría de las filas en cada ejecución, la actualización normal puede seguir siendo la más barata.

Se rechaza `CONCURRENTLY`. La opción necesita un índice único sobre columnas simples que abarque todas las filas. No sirve un índice único sobre `lower(email)` ni uno limitado por una cláusula `WHERE`. También necesita una vista con datos: falla si se creó con `WITH NO DATA`, y ambas opciones no pueden combinarse. Añada un índice válido y rellene la vista una vez con una actualización normal.

Una vista que no se puede consultar. `WITH NO DATA` crea la vista sin ejecutar su consulta y la marca como no disponible para lectura; cualquier consulta falla hasta la primera actualización. Puede ser lo adecuado en una migración que no deba esperar una consulta larga, pero entonces hay que planificar esa primera actualización y no olvidarla.

Un cambio de definición que nunca llegó. `CREATE MATERIALIZED VIEW IF NOT EXISTS` en una migración no hace nada más que emitir un aviso si ya existe una vista con ese nombre; la documentación no garantiza que la existente se parezca a la nueva definición. Cambiar la consulta de esa forma deja la anterior intacta. Elimine la vista con `DROP MATERIALIZED VIEW` y vuelva a crearla en la misma migración.

Tareas de actualización que se solapan. Una vista materializada solo puede tener una actualización en curso, incluso con `CONCURRENTLY`; programarla con más frecuencia de lo que tarda solo hace que la vista se retrase más. Mida la duración y programe en función de ella.

Se confía en el orden de las filas. Si la consulta que define la vista incluye `ORDER BY`, el contenido inicial sale en ese orden, pero una actualización no promete conservarlo. Incluya `ORDER BY` en las consultas que lean la vista.

Funciones que pierden sus tablas. Al ejecutar la consulta que define la vista, durante la creación o la actualización, PostgreSQL establece temporalmente `search_path` en `pg_catalog, pg_temp` y restringe lo que puede hacer: una función que cree tablas temporales falla. Una función que busque sus tablas mediante nombres sin esquema puede superar todas las pruebas manuales y fallar solo durante la actualización. Califique con el esquema los nombres de tabla de toda función invocada por la vista.

La tarea se ejecuta con el rol equivocado. Actualizar requiere el privilegio `MAINTAIN` sobre la vista materializada. Una tarea ejecutada con un rol de informes que puede leerla, pero no mantenerla, falla cada noche. Conceda ese privilegio al rol que ejecute la actualización.

Se guarda una copia cuando bastaría un índice. Si una consulta es lenta porque falta un índice, una vista materializada oculta el problema y añade datos antiguos. Lea primero el plan de consulta. Si la consulta solo necesita un nombre, basta una vista normal, que siempre está actualizada; reserve las materializadas para resultados costosos de calcular y que puedan leerse ligeramente desactualizados sin riesgo.

Lecturas adicionales

- Vistas materializadas — el capítulo de PostgreSQL sobre su relación con vistas y tablas, con un resumen de ventas actualizado cada noche y un ejemplo de adaptador de datos externos.
- CREATE MATERIALIZED VIEW — la referencia de la orden, incluidos `WITH NO DATA`, `IF NOT EXISTS` y el `search_path` restringido durante la consulta.
- REFRESH MATERIALIZED VIEW — la orden de actualización, las condiciones de `CONCURRENTLY` y el privilegio `MAINTAIN` requerido.
- Notas de PostgreSQL 9.3 — la versión de septiembre de 2013 que incorporó las vistas materializadas.
- Introducción a las vistas materializadas — las vistas incrementales y no incrementales de BigQuery, actualización automática y optimización inteligente.
- Crear vistas indexadas — las vistas indexadas de SQL Server, sus requisitos y el coste para las escrituras.

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 vista materializada?» 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/materialized-view/