Daniel Granados Retana

Diego Granados Retana

Bases de Datos I

Prof. Rodrigo Núñez

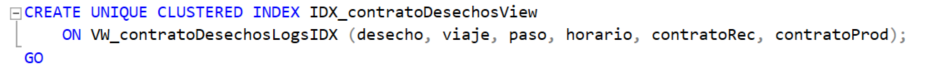
**Query con múltiples joins**

Para este ejercicio, utilizamos la siguiente query:

Text

Description automatically generated

El objetivo de este query es obtener el contrato de producción de cada desecho que se añade a una planta luego de su recolección, con el fin de saber en qué producción serán utilizados. Luego, creamos los views. El primer índice que se crea sobre una vista debe ser un unique clustered index. Cada vez que intentábamos crear el index, nos daba un error: había valores repetidos. Para solucionar esto, intentamos utilizar un distinct, pero tampoco se puede usarlo. Nos decía que podíamos usar un aggregate function o un COUNT\_BIG(\*), entonces también le pusimos el COUNT\_BIG(\*), que es una función que retorna la cantidad de elementos en un grupo. Ya con esto logramos crear el siguiente índice porque el COUNT\_BIG(\*) cuenta y agrupa los registros iguales y así los simplifica en uno solo:



Utilizamos estas columnas porque las posibles combinaciones únicas que se dan para cada desecho de cada viaje, por lo que accederlas rápidamente es muy ventajoso para el proceso.

De acuerdo con el execution plan, la instrucción que toma más tiempo es:

Table

Description automatically generated

Este hash match lo que está haciendo es crear un hash table con base en las columnas desechosPlantasLogs.desechoId, desechosPlantasLogs.viajeId, pasosRecoleccion.recHorarioId y recoleccionesPorProducto.prodContratoId. Luego de crear el índice, vimos que hubo una diferencia sustancial en el costo del query y un leve aumento en la velocidad.

Text

Description automatically generated with low confidence

Graphical user interface, text, application

Description automatically generated

Incluso, los resultados de la vista indexada estaban ordenados porque el clustered index los ordena.

Table

Description automatically generated

No obstante, luego intentamos crear un nonclustered index sobre cada campo del build residual, pero eso no afectó positivamente al rendimiento.

La razón por la cual disminuyó tanto el costo de la vista indexada es porque los resultados de la vista se almacenan como si fueran una tabla (Yaseen, 2019). En una vista normal dinámica, los resultados se calculan todas las veces con base en el query del view, por lo que no hay un gran beneficio en el rendimiento. Por lo tanto, acceder a la información con una vista indexada va a ser mucho más rápido, debido a que ya está calculada. Por otro lado, si los datos de la vista son actualizados frecuentemente o la vista no se llama mucho, es posible que el uso de espacio y el costo de actualizar los datos sea mayor al beneficio que trae la vista indexada. Es por esta razón que se utiliza la instrucción SCHEMABINDING, que hace que no se puedan aplicar cambios estructurales a las tablas involucradas en el índice.

Por lo tanto, como norma, si la tabla va a cambiar mucho, es mejor usar una vista dinámica. Si no, es mejor usar una vista indexada para facilitar el acceso a la información.

**Norma de optimización**

Por cada factura, la cual se compone de ítems de venta de productos, obtener la cantidad total de los productos vendidos, donde el tipo de producto no sea 2 ("colchón"), que fueron producidos en un proceso donde participó un productor (objectType = 7). Obtener el dinero correspondiente a cada productor en cada factura, con base en los porcentajes de ganancia y el monto de la venta. El dinero se presenta en la moneda base. Se excluyen los resultados correspondientes al productor 5 (GGGames).

Text

Description automatically generated

Graphical user interface, application, table

Description automatically generated



Rendimiento del query original: Se ejecutó en 442 ms.

Text, letter

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Unidad de workload | Explicación | Norma |
|  | El motor recorre la tabla de itemsProductos con base en el primary key PK\_ventasProductosLogs y a partir de eso extrae los registros cuyas fechas estén entre el intervalo establecido. Se relaciona con el S7, donde se recorre a partir de un índice individual y se filtra por un conjunctive selection. | - Ordenar la tabla de itemsProductos en cuanto a la fecha para que sea más fácil encontrar las fechas que están en un intervalo sin tener que revisar toda la tabla. Para cuando las fechas se salgan del intervalo.  - Cuando se hace un Clustered Index Scan y se filtra por una llave non-key, Revisar el campo non-key que está en el Predicate y agregar un NonClustered Index. Incluso, se pueden agregar las columnas que aparecen en el Output List, mientras no sean muchas, para extraer la información directamente. Se podría poner un nonclustered index en itemsProductos.fecha para ordenar los registros en cuanto a las fechas y sea más fácil acceder a ellos directamente. |

V2: Se aplicará la norma de crear un nonclustered index en itemsProductos.fecha. Como son muchas columnas, no se incluyen como included columns.

El rendimiento mejoró a 425 ms, una mejora de 17ms.

Table

Description automatically generatedText, letter

Description automatically generated

Sin embargo, este índice no contribuyó a disminuir el porcentaje en el costo de la operación ni la estrategia que utiliza.

Esto es porque el optimizador todavía determina que hacer el Clustered Index Scan es la forma más eficiente para realizar la operación. Debemos darle más herramientas para que lo haga más eficientemente.

Se intentó aplicar la norma de ordenar la tabla por fecha, pero al usar un EXCEPT y aggregate functions, el campo de itemsProductos.fecha debería estar en el output list y en el GROUP BY, lo cual retornaría resultados diferentes a lo que se busca con el query.

V3: En la misma operación anterior, se aplica la norma de agregar un nonclustered index con los campos que extrae en el output list.

Text

Description automatically generated

Table

Description automatically generatedEl rendimiento mejoró de 442ms a 407ms, una mejora de 35ms.

Esto también mejoró la estrategia y el costo de la operación en el plan de ejecución. Pasó de un Clustered Index Scan con un peso de 12%, a un Index Seek (NonClustered) con un peso de 8%.

Esto es porque puede usar la estructura del árbol B para llegar más directo al campo por donde se realiza la consulta. También, al almacenar las otras columnas en el índice, no tiene que realizar operaciones de I/O para extraer la información, por lo que lo hace más rápida la consulta

|  |  |  |
| --- | --- | --- |
| Unidad de workload | Explicación | Norma |
|  | Aquí hace una búsqueda lineal (S1) sobre toda la tabla de itemsFactura y selecciona los registros donde el tipoItemId=3. Retorna los campos de facturaId y itemId. | Cuando hay que hacer una búsqueda secuencial para buscar dónde se cumple la condición en una columna, indexar la columna problema con un nonclustered index para hacer una búsqueda S5 donde a partir de la condición, extrae todos los records que la cumplen. En este caso, pondríamos un nonclustered index en tipoItemId. Adicionalmente, las columnas que tiene que extraer esos registros pueden incluirse como non-key included columns en el índice para obtenerlas directamente con el índice y así reducir las operaciones de I/O. |

Se aplica la norma y agregamos un NonClustered Index con Included Columns en itemsFactura.tipoItemId y ponemos el output list (facturaId y itemId) como los included Columns.

Text

Description automatically generated

De la versión anterior a esta, mejoró de 407ms a 385ms.

Aquí, el método pasó de un Table Scan a un Index Seek (NonClustered)

Table

Description automatically generated El porcentaje pasó de un 12% a un 7%.

Diagrama de plan de ejecución:

Diagram, schematic

Description automatically generated

Explicación

Nuestro query contiene dos SELECTS, una para resolver los resultados generales y la otra para resolver los resultados que se deben excluir del primer SELECT por medio del EXCEPT. Por esta razón, el plan de ejecución se divide en dos ramas, las cuales son esencialmente iguales.

Norma

Si la consulta contiene un EXCEPT o INTERSECT que opera sobre consultas muy similares, intentar eliminar ese EXCEPT por medio de igualdades, desigualdades, o algún otro medio.

Schematic, timeline

Description automatically generatedResultados: Norma con el EXCEPT vs. Norma sin el EXCEPT

Text, letter

Description automatically generated

El rendimiento mejoró de 385ms a 341ms. El diagrama del plan de ejecución se redujo significativamente:

Diagram, schematic

Description automatically generated

Esto ocurre porque solo tiene que resolver un query. No tiene que repetir operaciones, ya que el filtro del productorId se hace en el Clustered Index Seek en la tabla de productores.

Text

Description automatically generatedTable

Description automatically generatedEsto se nota en las cantidades de lectura en las tablas. Se reducen a la mitad.

|  |  |  |
| --- | --- | --- |
| Unidad de workload | Explicación | Norma |
| Table  Description automatically generated | Aquí hace un Hash Match (J4) con base en el campo de facturaId en las tablas de itemsFactura y facturas. | Buscar que las columnas involucradas en la igualdad para el Hash Match estén ordenadas, ya sea por un Sort, un clustered index o un nonclustered index para intentar convertirlo en un Merge Join (J3) |

Se agregó un Clustered Index en las columnas de facturaId y itemId de la tabla itemsFactura. La tabla de facturas ya tiene un clustered index en facturaId.

A picture containing text

Description automatically generated

En comparación con la versión anterior, el query tiene un porcentaje de costo menor en general.

Table

Description automatically generatedDiagram

Description automatically generated with medium confidence

El método para el join pasó de un Hash Match a un Merge Join. Esto ocurrió porque las dos tablas están ordenadas con base en el clustered index en facturaId, por lo que puede utilizar el método J3.

No obstante, en cuanto al rendimiento, no observamos una mejoría en el tiempo de ejecución.

Sin índice:

Text

Description automatically generated

Con índice:

Text

Description automatically generated

Sin embargo, es posible que, con una mayor cantidad de datos, la eficiencia del Merge Join nos beneficie y haga la consulta más rápidamente.

En la siguiente comparación, se aprecia que con el índice se requieren menos operaciones de I/O en la tabla de itemsFactura, de 574 versus 771.

Text

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Unidad de workload | Explicación | Norma |
|  | Aquí hace un Clustered Index Scan (S7) sobre la tabla de lotesProduccionLogs usando el primary key PK\_productosFabricados. Luego usa los otros elementos de la condición conjuntiva para seleccionar registros específicos. | Revisar cuáles son las columnas involucradas en el predicate. Agregar un nonclustered index en las columnas del PROBE o igualdades para que las pueda filtrar más rápidamente. Poner columnas diferentes en el output list como included columns en el index mientras no sean muchas. |

Table

Description automatically generated

Aplicando la norma, se agrega un nonclustered index en las columnas de productoId y prodContratoId, las cuales están en las cláusulas de PROBE en el Predicate. Esto hace que pueda recorrer el índice para encontrar las filas que ocupa en lugar de recorrer todos los registros de la tabla. Por esta razón, el operador cambia a un Index Seek (Nonclustered) y lo hace más eficientemente.

Text

Description automatically generatedEsto hace por ejemplo, que las lecturas en la tabla de lotesProduccionLogs bajen de 246 a 47. El tiempo de ejecución también es menor.

|  |  |  |
| --- | --- | --- |
| Unidad de workload | Explicación | Norma |
|  | Aquí hace un Clustered Index Scan (S7) sobre la tabla de actoresContratoProd usando el primary key PK\_actoresContratoProd. Luego usa los otros elementos de la condición conjuntiva (si el objectTypeId = 7) para seleccionar registros específicos. | Revisar cuáles son las columnas involucradas en el predicate. Agregar un nonclustered index en las columnas del PROBE o que están en una condición y son non-key para que las pueda filtrar más rápidamente. Poner columnas diferentes en el output list como included columns en el index, mientras no sean muchas. |

Table

Description automatically generatedSe agregó un nonclustered index en la tabla actoresContratoProd sobre objectTypeId con prodContratoId, actorId y genericId como included columns. Esto convirtió el operador a un Index Seek (Nonclustered). Como en los casos anteriores, puede usar el índice directamente para encontrar los que tengan un objectTypeId = 7 y extraer los valores que ocupa directo del índice. Así, el peso del operador es menor.

Con este índice, pasa de realizar 84 lecturas lógicas a solo 10 lecturas en la tabla de actoresContratoProd. El tiempo de ejecución también es menor con el índice.

Text

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Unidad de workload | Explicación | Norma |
|  | Aquí hace un Clustered Index Scan (S7) sobre la tabla de porcentajesActores usando el primary key PK\_porcentajesActores. Luego extrae la columna de actorId como Probe para el Hash Match que va a hacer después. | Revisar cuáles son las columnas involucradas en el predicate. Agregar un nonclustered index en las columnas del PROBE o que están en una condición y son non-key para que las pueda filtrar más rápidamente. Poner columnas diferentes en el output list como included columns en el index, mientras no sean muchas. |

Table

Description automatically generatedSe agregó un nonclustered index en la tabla porcentajesActores sobre actorId con porcentaje y productoId como included columns. Esto cambió el operador a un Index Scan (NonClustered). No es un index seek porque no tenemos una condición de igualdad. No obstante, el resultado es una operación más rápida porque la búsqueda en el árbol B es más rápida. Adicionalmente, tras esta conversión, el Hash Match que venía después se transformó en un Merge Join debido a que el campo donde se hace el join, el cual es actorId, está ordenado en ambas tablas por medio de índices.

Text

Description automatically generatedEste índice permite que las lecturas de la tabla porcentajesActores bajaran de 129 a 31. El tiempo de ejecución también disminuyó.

**Diagram, schematic

Description automatically generated**Tras estos cambios, pasamos de una consulta cuyo plan de ejecución se veía así:

Diagram, schematic

Description automatically generatedA una consulta cuyo plan de ejecución se ve así:

Comparando los dos: tenemos que la primera versión tiene un costo 3 veces mayor que la segunda.

Timeline

Description automatically generated with low confidence

Timeline

Description automatically generated

En cuanto al tiempo de ejecución y operaciones de I/O, la segunda versión es más rápida y tiene muchas menos operaciones de I/O:

Text

Description automatically generated

Text

Description automatically generated

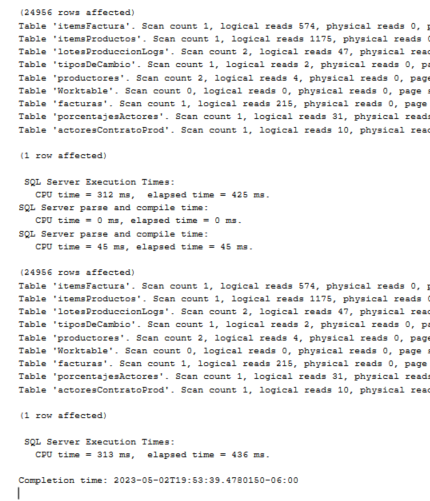
**Common Table Expression**

Encapsulando el query optimizado en un CTE:

Graphical user interface, text

Description automatically generated

No se percibe una mejora en rendimiento encapsulando la consulta en un CTE. Más bien, a veces el tiempo de ejecución es mayor, pero esto es muy variable. Las operaciones de lectura y escritura son iguales. Los porcentajes de costos son los mismos:

Timeline

Description automatically generated

🡨 Query sin CTE

🡨 Query con CTE

Query simplificada con CTEs:

Text

Description automatically generatedEn este query, realizamos el filtro antes de hacer los joins para trabajar directamente con la información correcta. Así se simplifica las conjunciones en el WHERE del SELECT principal.

En este caso, el costo del query con CTEs es ligeramente mayor:

Text

Description automatically generatedTimeline

Description automatically generated with medium confidence

La cantidad de operaciones de lectura en las tablas se mantiene igual.

No obstante, sí se percibe una ligera mejora en cuanto al tiempo de ejecución, por lo que sí es recomendado intentar simplificar el query por medio de CTEs. Adicionalmente, pueden ayudar a mejorar la legibilidad y comprensión del query:

**Resumen de la norma:**

1. Cuando se hace un Clustered Index Scan y se filtra por una llave non-key, revisar el campo non-key que está en el Predicate y agregar un NonClustered Index. Incluso, se pueden agregar las columnas que aparecen en el Output List, mientras no sean muchas, para extraer la información directamente.
2. Cuando hay que hacer una búsqueda secuencial para buscar dónde se cumple la condición en una columna, indexar la columna problema con un nonclustered index para hacer una búsqueda S5 donde a partir de la condición, extrae todos los records que la cumplen. Adicionalmente, las columnas que tiene que extraer esos registros pueden incluirse como non-key included columns en el índice para obtenerlas directamente con el índice y así reducir las operaciones de I/O.
3. Si la consulta contiene un EXCEPT o INTERSECT que opera sobre consultas muy similares, intentar eliminar ese EXCEPT por medio de igualdades, desigualdades, o algún otro medio.
4. Buscar que las columnas involucradas en la igualdad para el Hash Match estén ordenadas, ya sea por un Sort, un clustered index o un nonclustered index para intentar convertirlo en un Merge Join (J3)
5. Revisar cuáles son las columnas involucradas en el predicate. Agregar un nonclustered index en las columnas del PROBE o que están en una condición y son non-key para que las pueda filtrar más rápidamente. Poner columnas diferentes del output list como included columns en el index, mientras no sean muchas.
6. Intentar simplificar el query por medio de CTEs para obtener una ligera mejora en el rendimiento.
7. En cuanto a vistas, si la tabla va a cambiar mucho, es mejor usar una vista dinámica para recalcular el SELECT cada vez que se usa la vista. Si no ocupo consultar la misma versión de los datos frecuentemente, es mejor usar una vista indexada o estática para facilitar el acceso a la información, ya que esta se guarda.

**Bibliografía:**

Yaseen, A. (2019). SQL Server indexed views. SQL Shack - articles about database auditing, server performance, data recovery, and more. <https://www.sqlshack.com/sql-server-indexed-views/>