Daniel Granados Retana

Diego Granados Retana

Bases de Datos I

Prof. Rodrigo Núñez

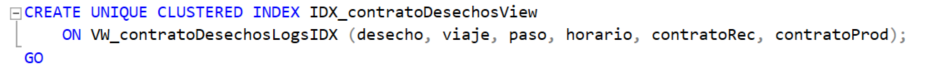
**Query con múltiples joins**

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 views. El primer indíce que se cree sobre una vista debe ser un unique clustered index. Cada vez que intentábamos crear el index, nos daba un error, como que 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 COUNTBIG(\*), 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:



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 una 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 se mayor al beneficio que trae la vista. 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, exceptuando el tipo de producto 2 "colchón", que fueron producidos en un proceso donde participó un productor. 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 cuyo Id es 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. Establece el itemsProductos.loteId como el probe para el Hash Match que viene después. 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.  -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. Incluso, se pueden agregar las columnas que aparecen en el Output List, mientras no sean muchas, para extraer la información 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.

V3: Se aplica la norma de agregar un nonclustered index con los campos que extrae en el output list.

Text

Description automatically generated

El 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. El itemId es el probe para el Hash Match join del siguiente paso. 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 como 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.

Table

Description automatically generated

|  |  |  |
| --- | --- | --- |
| 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 o un clustered 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.

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

A picture containing text

Description automatically generated

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.

**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/