Introducción a índices filtrados en SQL Server

En este artículo se pretende proporcionar información acerca de los índices filtrados de SQL Server y sus repercusiones en el rendimiento.

Introducción

Los índices son las estructuras de datos especiales que ayudan a mejorar el rendimiento de las consultas en SQL Server. Frente a este gran beneficio de los índices, ocupan espacio en los discos duros y pueden ralentizar el rendimiento de las operaciones de modificación de datos (actualizar, insertar, eliminar). Cuando una consulta modifica los datos de una tabla, el motor de base de datos necesita actualizar todos los índices relacionados en los que han cambiado los datos. En algunos casos, para minimizar estas desventajas de los índices, el uso de índices filtrados de SQL Server podría ser el enfoque adecuado. Supongamos que con frecuencia consultamos un pequeño subconjunto de una tabla con las mismas condiciones y el resto de la tabla contiene demasiadas filas. En este escenario, podemos usar un índice filtrado de SQL Server para tener acceso a este pequeño conjunto de datos más rápido para que podamos reducir los costos de almacenamiento y mantenimiento de índices.

¿Qué es el índice filtrado de SQL Server?

Un índice filtrado de SQL Server es un índice ***non-clustered*** que nos permite aplicar condiciones específicas para cubrir un subconjunto de filas de la tabla. Por ejemplo, tenemos una tabla de materiales y la aplicación web solo solicita los datos de material activos. En este caso, crear un índice para las únicas filas de materiales activos es muy razonable que crear un índice para filas enteras de la tabla. Un índice filtrado bien diseñado puede proporcionar algunas ventajas, y abordemos estas ventajas en las siguientes secciones.

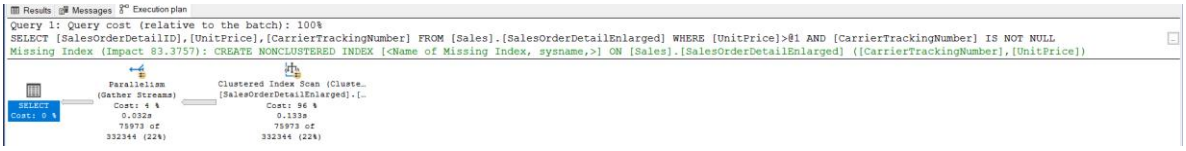
Pre-requisitos:

En este artículo, usaremos la base de datos AdventureWorks y también usaremos el script de ampliación para obtener una copia más grande de esta base de datos.

Rendimiento filtrado de índices y consultas de SQL Server

Como dijimos, los índices son uno de los objetos de base de datos importantes para mejorar el rendimiento de las consultas si se diseñan de manera efectiva. Además de esto, SQL Server puede sugerir índices que faltan después de procesar una consulta y piensa que el índice sugerido puede ayudar a mejorar el rendimiento de la consulta. Ahora, habilitemos el plan de ejecución real y ejecutemos la siguiente consulta.

```
SELECT  SalesOrderDetailID,UnitPrice,CarrierTrackingNumber
FROM Sales.SalesOrderDetailEnlarged
WHERE  UnitPrice >1500 AND CarrierTrackingNumber IS  NOT NULL
```

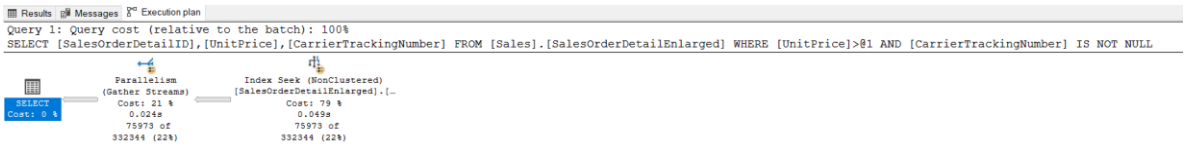


Como podemos ver, SQL Server sugiere un índice faltante y afirma que el índice sugerido puede mejorar el rendimiento de la consulta en aproximadamente un 83%. No debemos aplicar estas recomendaciones de índice de forma incontrolable porque SQL Server no tiene en cuenta nuestras cargas de trabajo u otros directorios al crear estas recomendaciones. Por lo tanto, antes de aplicar los índices que faltan, debemos considerar los pros y los contras. Sin embargo, para esta consulta, el índice sugerido parece muy razonable porque cubre las columnas CarrierTrackingNumber, UnitPrice para que todas las columnas requeridas estén involucradas en el índice. Ahora, crearemos el índice sugerido por SQL Server, pero este índice cubre todas las filas de la tabla.

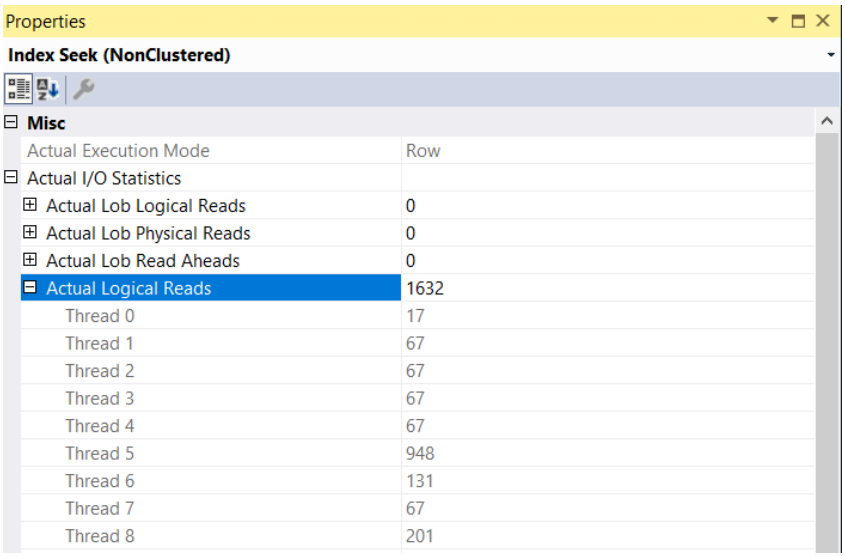
```
CREATE NONCLUSTERED INDEX IndexNonFiltered
ON [Sales].[SalesOrderDetailEnlarged] ([UnitPrice],[CarrierTrackingNumber])
```

Después de crear el índice, volveremos a ejecutar la consulta y analizaremos su plan de ejecución.

```
SELECT SalesOrderDetailID  ,UnitPrice,CarrierTrackingNumber FROM Sales.SalesOrderDetailEnlarged
WHERE  UnitPrice >1500 AND CarrierTrackingNumber IS  NOT NULL
```

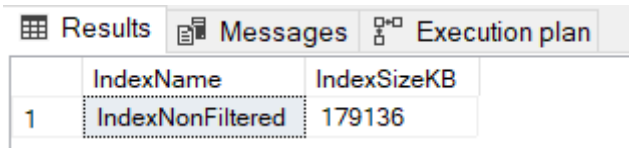


El operador NonClustered Seek, que aparece en el plan de ejecución de la consulta, indica que el motor de almacenamiento busca filas coincidentes mediante la estructura de árbol b del índice no agrupado. El atributo Lecturas lógicas reales del operador de búsqueda no agrupado muestra el número de páginas leídas de la memoria caché de datos.



Los índices ocupan un tamaño en las unidades de disco duro según los tipos de datos de columna indizados y el número de filas de la tabla. La siguiente consulta muestra los tamaños de índice y nuestro tamaño de índice es 179.136 MB.

```
SELECT
  i.name              AS IndexName,
  SUM(s.used_page_count) * 8  AS IndexSizeKB
FROM sys.dm_db_partition_stats  AS s
JOIN sys.indexes            AS i
ON s.[object_id] = i.[object_id] AND s.index_id = i.index_id
WHERE s.[object_id] = object_id('Sales.SalesOrderDetailEnlarged ')
and i.name= 'IndexNonFiltered'
GROUP BY i.name
```



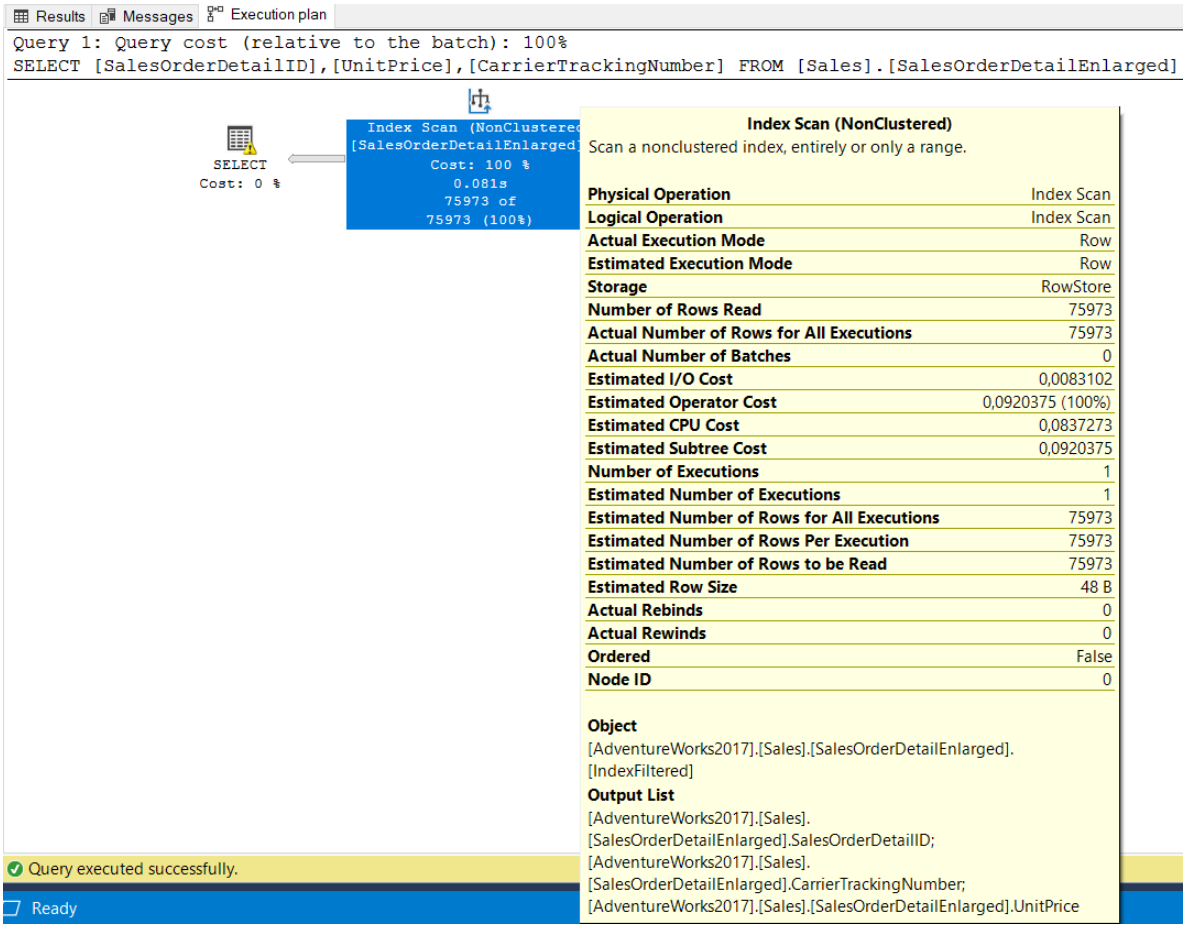
Ahora crearemos un índice filtrado de SQL Server para las filas que son los valores de columna UnitPrice son mayores que 1500 y los valores de columna CarrierTrackingNumber no son nulos, es decir, crearemos un índice personalizado para la cláusula WHERE de nuestra consulta.

```
IF EXISTS (SELECT name FROM sys.indexes
  WHERE name = N'IndexNonFiltered'
  AND object_id = OBJECT_ID (N'Sales.SalesOrderDetailEnlarged'))
          BEGIN
DROP INDEX IndexNonFiltered
  ON Sales.SalesOrderDetailEnlarged
END

---Create Filtered Index

CREATE NONCLUSTERED INDEX IndexFiltered
```

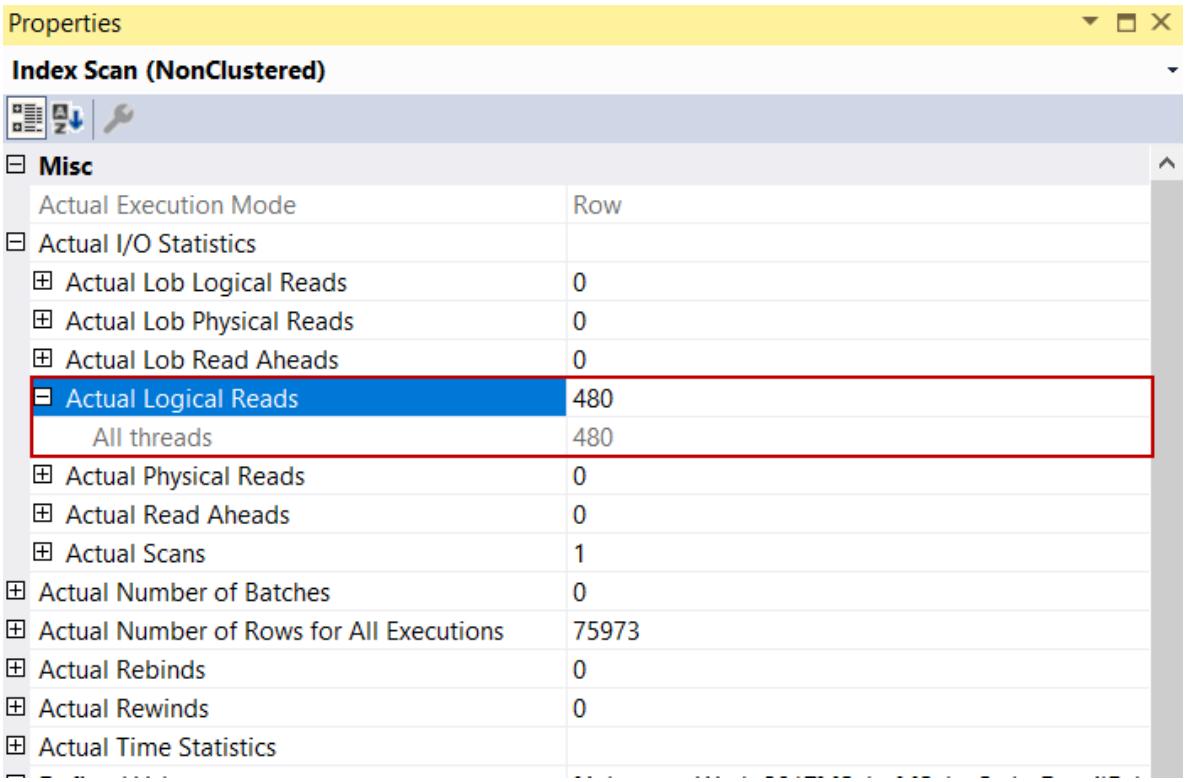
```
ON [Sales].[SalesOrderDetailEnlarged] (UnitPrice , CarrierTrackingNumber)
WHERE UnitPrice >1500 AND
 CarrierTrackingNumber IS  NOT NULL
GO
```

Ahora, ejecutaremos la misma consulta nuevamente analizando el plan de ejecución.

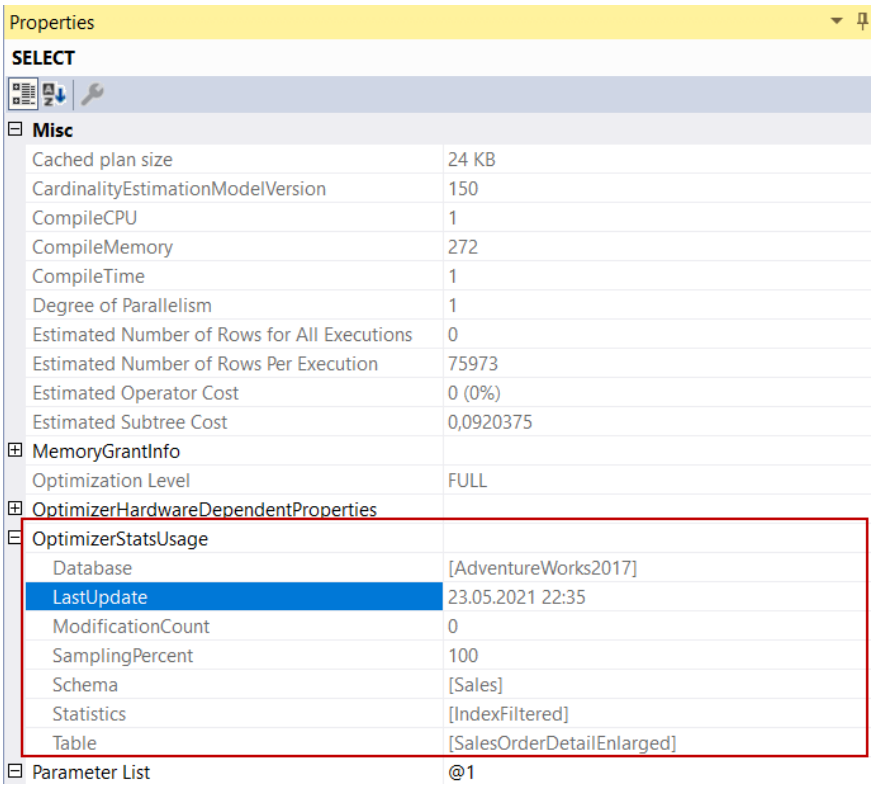
```
SELECT SalesOrderDetailID ,UnitPrice,
CarrierTrackingNumber FROM Sales.SalesOrderDetailEnlarged
WHERE UnitPrice >1500 AND
  CarrierTrackingNumber IS NOT NULL
```



El plan de ejecución nos muestra que, el optimizador de consultas usó el índice filtrado de SQL Server para ejecutar la consulta. Sin embargo, estamos viendo una señal de advertencia en el operador SELECT, pero este es un error relacionado con SQL Server por esta razón, podemos ignorar esta señal de advertencia para esta consulta. Cuando revisamos las estadísticas de I/O de esta consulta, nos muestra que la consulta solo lee 483 páginas de datos de la caché del búfer. Este valor es muy menor que el índice de tabla completa.

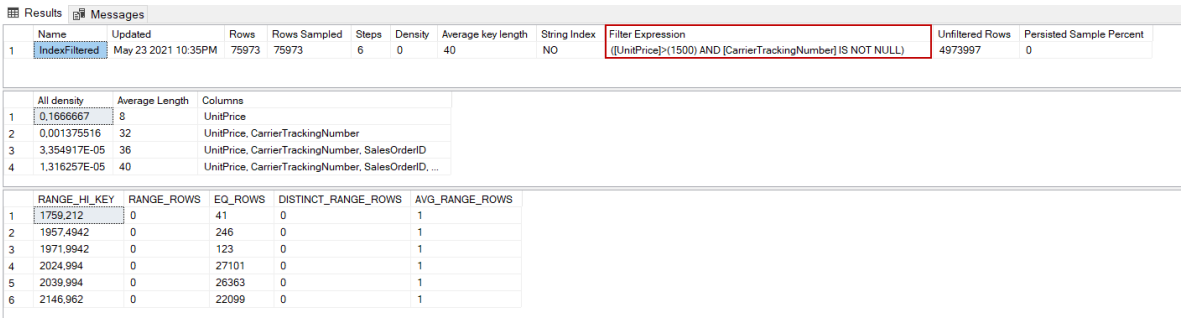


Statistics almacena la distribución de los valores de columna y desempeña un papel clave en el rendimiento de la consulta porque el optimizador de consultas utiliza datos estadísticos para estimar cuántas filas se pueden devolver desde la consulta en ejecución. Cuando creamos un índice, se generan automáticamente estadísticas para almacenar la distribución de los valores de columna. En el operador SELECT, podemos averiguar qué estadísticas utiliza el optimizador de consultas. El atributo OptimizerStatUsage nos muestra los nombres de las estadísticas utilizadas y otros detalles sobre las estadísticas utilizadas durante la ejecución de la consulta.



El subatributo LastUpdate muestra cuándo se actualizaron las estadísticas por última vez y el subatributo Recuento de modificaciones muestra cuántas veces se modifica la estadística después de actualizar las estadísticas. El comando DBCC SHOW_STATISTICS se utiliza para obtener detalles sobre las estadísticas. Con la ayuda de la siguiente consulta, podemos obtener información sobre el índice filtrado que creamos.

```
DBCC SHOW_STATISTICS('Sales.SalesOrderDetailEnlarged','IndexFiltered')
```



La imagen de arriba nos muestra que el histograma (representación estadística de sus datos) se crea solo para los valores filtrados.

Al mismo tiempo, el espacio ocupado por el índice filtrado en el disco duro es menor que el índice de tabla completa.

```
SELECT
  i.name          AS IndexName,
```

```
3   SUM(s.used_page_count) * 8  AS IndexSizeKB
4FROM sys.dm_db_partition_stats  AS s
5JOIN sys.indexes              AS i
6ON s.[object_id] = i.[object_id] AND s.index_id = i.index_id
7WHERE s.[object_id] = object_id('Sales.SalesOrderDetailEnlarged ')
8and i.name= 'IndexFiltered'
9GROUP BY i.name
```

Results | Messages

	IndexName	IndexSizeKB
1	IndexFiltered	3864

Como resultado, el uso del índice nos ha proporcionado 2 beneficios significativos:

- Mejorar el rendimiento de las consultas y la calidad del plan
- Reducir el costo de almacenamiento de índices

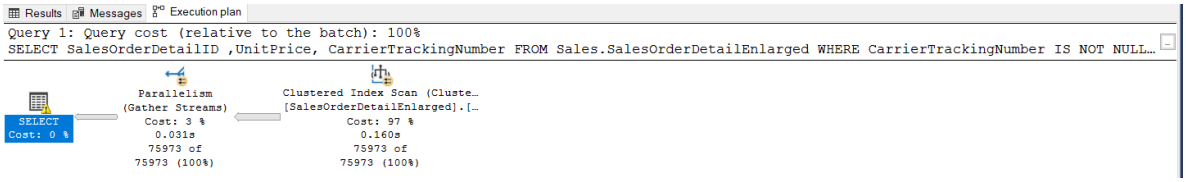
Desventajas del índice filtrado de SQL Server

Al igual que con la mayoría de las características de SQL Server, los índices filtrados tienen ventajas y desventajas. En esta sección, analizaremos estas desventajas de los índices filtrados. Los procedimientos almacenados son las consultas SQL preparadas para invocar y pueden tomar entradas como parámetros. Ahora, crearemos un procedimiento almacenado muy simple que acepta un parámetro. Este nombre de parámetro es @PUnitPrice y su tipo de datos es money.

```
1CREATE OR ALTER PROCEDURE TestFilterIndex
2@PUnitPrice AS MONEY
3AS
4SELECT SalesOrderDetailID  ,UnitPrice,
5CarrierTrackingNumber FROM Sales.SalesOrderDetailEnlarged
6WHERE CarrierTrackingNumber IS NOT NULL
7AND UnitPrice >@PUnitPrice
```

Después de crear el procedimiento almacenado, establecemos el parámetro @PUnitPrice en el valor 1500 e invocamos el procedimiento. En este caso, esperamos que la consulta use el índice filtrado. Analicemos el plan de ejecución del procedimiento almacenado.

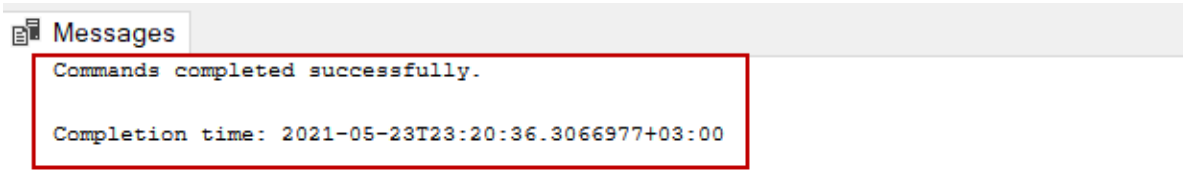
```
1EXECUTE TestFilterIndex 1500
```



El plan de consulta de procedimiento almacenado no utilizó el índice filtrado que se creó y utilizó el operador de detección de índice agrupado en lugar de él. ¿Cuál es la razón de este comportamiento del optimizador de consultas? En realidad, la respuesta es muy básica porque los planes de consulta de

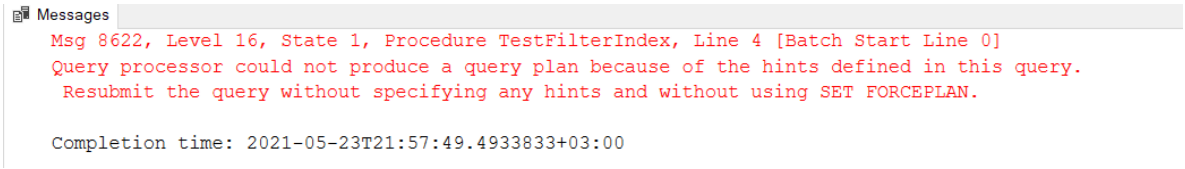
procedimientos almacenados se almacenan en la memoria caché del plan de consulta para que el optimizador de consultas pueda reutilizar los planes de consulta almacenados en caché. Sin embargo, si invocamos el procedimiento almacenado para los diferentes parámetros, el índice filtrado y sus estadísticas no saben nada sobre estos valores. Por lo tanto, el optimizador de consultas no utiliza un índice filtrado para las consultas parametrizadas. Tal vez, podemos pensar en usar una sugerencia de índice porque esta sugerencia obliga al optimizador de consultas a usar el índice especificado en la consulta. Ahora modificaremos el procedimiento TestFilterIndex y lo forzaremos a usar el índice filtrado con la ayuda de la sugerencia de índice. SQL Server no devuelve un error durante la modificación del procedimiento almacenado.

```
ALTER  PROCEDURE [dbo].[TestFilterIndex]
@PUnitPrice AS MONEY
AS
SELECT SalesOrderDetailID  ,UnitPrice,
CarrierTrackingNumber FROM Sales.SalesOrderDetailEnlarged
WITH (INDEX(IndexFiltered))
WHERE CarrierTrackingNumber IS NOT NULL
AND UnitPrice >@PUnitPrice
```



Como segundo paso, intentamos invocar el procedimiento almacenado.

```
EXECUTE TestFilterIndex 1500
```

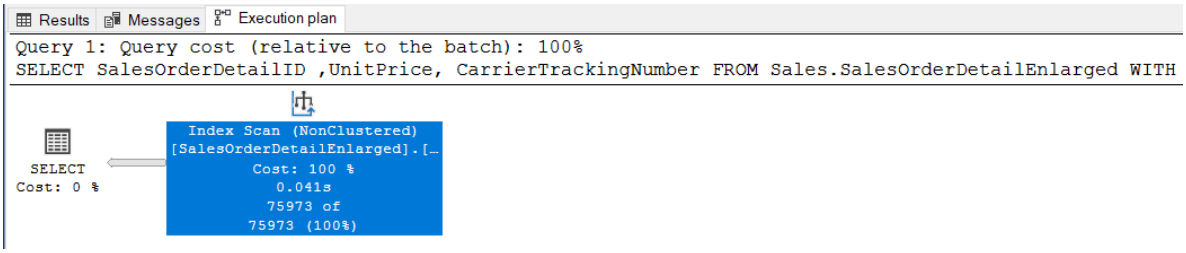


Como podemos ver, durante la ejecución del procedimiento almacenado, SQL Server devuelve un error sobre el uso de esta sugerencia de índice. La opción RECOMPILAR puede resolver este problema sólo para los parámetros que proporcionan expresión de índice de filtro.

```
ALTER  PROCEDURE [dbo].[TestFilterIndex]
@PUnitPrice AS MONEY
AS
SELECT SalesOrderDetailID  ,UnitPrice,
CarrierTrackingNumber FROM Sales.SalesOrderDetailEnlarged
WITH (INDEX(IndexFiltered))
WHERE CarrierTrackingNumber IS NOT NULL
AND UnitPrice >@PUnitPrice
OPTION (RECOMPILE)
```

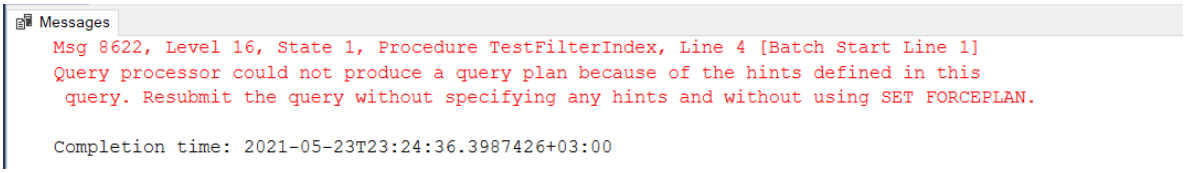
Cuando ejecutamos el procedimiento almacenado con el mismo parámetro, se ejecutará correctamente.

```
EXECUTE TestFilterIndex 1500
```



Sin embargo, si invocamos el procedimiento para los siguientes parámetros, SQL Server sigue devolviendo un error.

```
EXECUTE TestFilterIndex 1000
```



Conclusion

En este artículo, nos centramos en el índice filtrado de SQL Server. Cuando el índice de filtro se usa en enfoques apropiados, nos da una ventaja en el rendimiento de las consultas.