Universidad Nacional del Nordeste Facultad de Ciencias Exactas y Naturales y Agrimensura

BASE DE DATOS I

“Índices columnares en SQL server”

Autores:

Blanco, Fabricio Miguel

Candia, Jose Maria

Coronas Almada, Jezabel

Cristaldo, Gonzalo

Profesor: Villegas, Dario Oscar

Año: 2023

ÍNDICE

CAPÍTULO I: INTRODUCCIÓN

Tema

El tema que será investigado en el presente informe es el de “Índices Columnares en SQL server”, en el que se busca explorar y comprender los conceptos básicos, además aplicando lo investigado en un desarrollo de un caso práctico, en el que se busca realizar un análisis comparativo del rendimiento de consultas en SQL Server: Índices Columnares vs Índices Tradicionales

Definición o planteamiento del problema

Cómo manejar datos es algo imprescindible en estos tiempos, es bueno conocer y saber cómo implementar las alternativas que tenemos en las formas en las que dichos datos se almacenan y cómo se accede a ellos, ya que los tiempos de acceso pueden perjudicar o mejorar el rendimiento y la eficiencia en el manejo y acceso a una base de datos. Es acá, donde los “Índices columnares en SQL server” son de utilidad, ya que, si bien son una herramienta extremadamente útil, el implementarla puede ser complejo. Es así que partimos de preguntas como ¿Cómo se implementan? ¿Cómo se comparan los índices columnares con los índices tradicionales en términos de rendimiento de las consultas? ¿Cuál es el impacto de los índices columnares en el costo y el tiempo de respuesta de las consultas?

Objetivo del Trabajo Práctico

Aquí se busca investigar, entender , afianzar y utilizar los Índices columnares en un caso práctico, es así que se lo verá tanto teóricamente, como con una aplicación práctica particular.

Objetivos Generales

Como objetivo general, se toma la postura de proporcionar una comprensión clara desde nuestro lugar de aprendizaje del sustento teórico, como del uso práctico de los índices columnares en SQL server, tal como se indicó anteriormente.

Objetivos Específicos

Los objetivos específicos deseados son: entender la teoría que sustenta a los índices columnares, aprender cuando es conveniente su uso, como usarlo de manera correcta, y con el caso práctico, comparando una situación en la que no se lo utilice, en contraparte con su implementación efectiva práctica, más precisamente medir el rendimiento, tiempo, costos de ejecución , etc, para así demostrar su impacto en el rendimiento de una consulta sobre una base de datos.

CAPÍTULO II: MARCO CONCEPTUAL O REFERENCIAL

Los índices son una herramienta poderosa que permite a los sistemas de bases de datos encontrar rápidamente los datos que necesitan para responder a una consulta. Pero no todos los índices son iguales. Al igual que las herramientas en una caja de herramientas, diferentes índices son buenos para diferentes trabajos.

En este trabajo, nos centraremos en un tipo específico de índice conocido como índice columnar y cómo se implementa en SQL Server. A través de este estudio, esperamos mostrar cómo los índices columnares pueden mejorar el rendimiento y la eficiencia de las consultas en SQL Server. A continuación, presentaremos los conceptos clave que forman la base de este estudio.

Índices en Bases de Datos

Los índices son estructuras de datos que mejoran la velocidad de las operaciones de datos en una base de datos. Funcionan de manera similar a los índices en los libros, permitiendo al sistema de base de datos encontrar datos sin tener que buscar cada fila en una tabla de base de datos cada vez que se realiza una consulta.

Índices Agrupados

Los índices agrupados en SQL Server definen el orden en el cual los datos son físicamente almacenados en una tabla. En otras palabras, un índice agrupado determina el orden de las filas de datos en la tabla. Esto significa que los datos de las tablas pueden ser ordenados sólo de una forma, por lo tanto, **sólo puede haber un índice agrupado por tabla.** En SQL Server, la restricción de llave primaria crea automáticamente un índice agrupado en esa columna en particular.Los índices agrupados son organizados como páginas de 8 KB usando estructuras B-Árbol, para permitir al Motor SQL Server encontrar las filas requeridas asociadas con los valores de índice clave rápidamente. Cada página en la estructura de índice B-Árbol es considerada como un nodo de índice. El nodo de nivel superior es llamado el nodo Raíz y los nodos de nivel inferior son llamados nodos Hoja, donde las páginas de tablas de información son almacenadas y clasificadas basadas en los valores de índice clave.

Índices No Agrupados

Los índices no agrupados, por otro lado, tienen una estructura separada de las filas de datos. Un índice no agrupado contiene los valores de clave de índice no agrupado y cada entrada de valor de clave tiene un puntero a la fila de datos que contiene el valor clave. El puntero desde una fila de índice no agrupado hacia una fila de datos se denomina localizador de fila.En los índices no agrupados, una estructura no es común a las filas de datos. Esta estructura es independiente y comprende valores clave no agrupados. Los

índices no agrupados facilitan a los usuarios la inclusión de columnas no clave en el nivel principal.

Índices Columnares

Los índices columnares son un tipo de índice que almacena datos por columnas en lugar de filas. Esto es especialmente útil para las consultas OLAP (Procesamiento Analítico en Línea), que a menudo requieren sumar o realizar cálculos en un solo campo en muchas filas. Los índices columnares pueden mejorar significativamente el rendimiento de estas consultas.

Índices Columnares en SQL Server

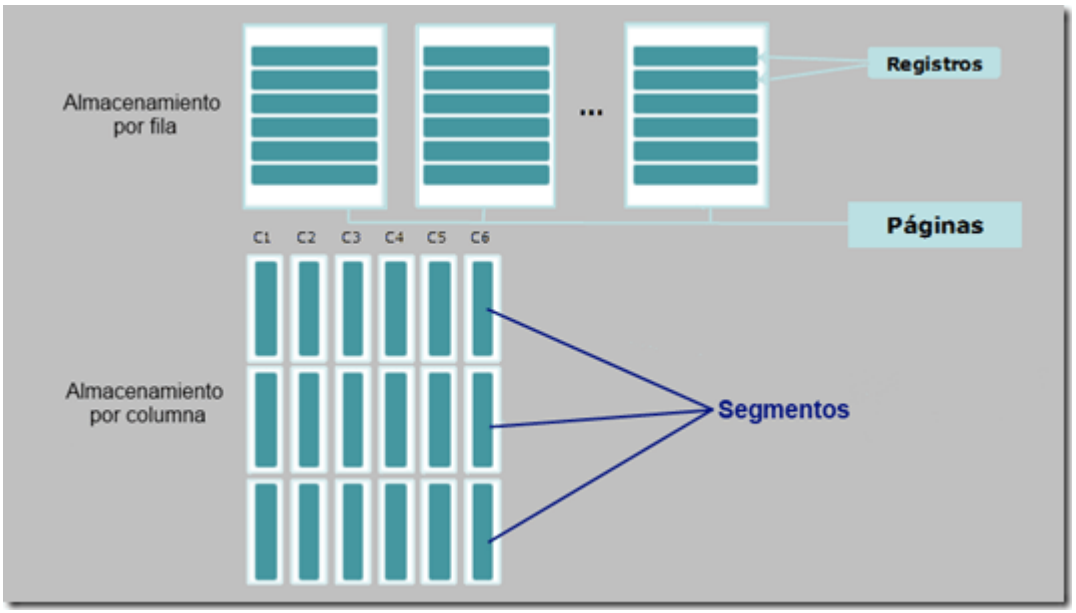
SQL Server es un sistema de gestión de bases de datos relacional desarrollado por Microsoft. A partir de la versión 2012, SQL Server introdujo el soporte para índices columnares, llamados Columnstore Indexes. Los índices de almacén de columnas son el estándar para almacenar y consultar las tablas de hechos de almacenamiento de datos de gran tamaño. Este índice usa el almacenamiento de datos basado en columnas y el procesamiento de consultas para lograr ganancias de hasta 10 veces el rendimiento de las consultas en el almacenamiento de datos sobre el almacenamiento tradicional orientado a filas. También puede lograr ganancias de hasta 10 veces la compresión de datos sobre el tamaño de los datos sin comprimir.

Un almacén de filas son datos organizados lógicamente como una tabla con filas y columnas, y **almacenados físicamente en un formato de filas.** Este formato es la forma tradicional de almacenar los datos de una tabla relacional..

Con los índices columnares los datos son organizados lógicamente como una tabla con filas y columnas, y almacenados **físicamente en un formato de columnas**.Los índices de almacén de columnas en SQL Server proporcionan una alta compresión de datos, lo que reduce el costo de almacenamiento y mejora el rendimiento de las consultas. Son ideales para análisis y almacenamiento de datos, y desde SQL Server 2016, también permiten análisis operativos en tiempo real.

Estos índices son rápidos debido a varias razones:

- Las columnas suelen tener valores similares, lo que resulta en altas tasas de compresión y reduce la E/S del sistema.
- La alta compresión mejora el rendimiento de las consultas al utilizar menos memoria.
- La ejecución por lotes procesa varias filas juntas, mejorando aún más el rendimiento.
- Las consultas suelen seleccionar solo unas pocas columnas de una tabla, reduciendo la E/S total desde los medios físicos.



Representación del almacenamiento con filas y columnas

Los índices de almacén de filas son más eficientes para las consultas que buscan datos específicos o un rango limitado de valores, por lo que son ideales para cargas de trabajo transaccionales.

Por otro lado, los índices de almacén de columnas son más eficientes para consultas analíticas que involucran grandes cantidades de datos, especialmente en tablas grandes. Son útiles para cargas de trabajo de análisis y almacenamiento de datos. Estos índices son especialmente útiles cuando las consultas realizan análisis que escanean grandes rangos de valores en lugar de buscar valores específicos.

Plan de Ejecución en SQL Server

El plan de ejecución es un conjunto de pasos generados por el optimizador de consultas de SQL Server para recuperar datos. Estos planes son la clave para entender cómo SQL Server interpreta una consulta y, en última instancia, determina la forma más eficiente de ejecutarla.

Hay dos tipos de planes de ejecución: estimados y reales. Los planes de ejecución estimados se generan sin ejecutar la consulta, mientras que los planes de ejecución reales se generan después de que se ejecuta la consulta. Los planes de ejecución reales contienen estadísticas adicionales, como el tiempo de CPU, el tiempo de lectura/escritura y el número de filas procesadas.

Los planes de ejecución se representan gráficamente en SQL Server Management Studio (SSMS), lo que facilita su interpretación. Cada operador (como un escaneo de tabla, un join o una ordenación) se representa como un icono en el plan de ejecución. Al pasar el cursor sobre un operador, se muestra información detallada sobre ese paso del plan..

CAPÍTULO III: METODOLOGÍA SEGUIDA

Descripción de Cómo se Realizó el Trabajo Práctico

En la realización de este proyecto de investigación sobre "Índices columnares en SQL Server", hemos seguido un enfoque colaborativo que aprovecha las ventajas de las tecnologías de comunicación y colaboración. Nuestra metodología se ha basado en la coordinación y el trabajo en equipo, junto con herramientas específicas para lograr nuestros objetivos de investigación. A continuación, se describen los principales pasos seguidos:

Planificación del Trabajo en Equipo

Inicialmente, definimos un plan de trabajo en equipo que incluía la asignación de tareas y secciones específicas del proyecto a cada miembro del grupo.

Coordinación de Reuniones y Comunicación

Utilizamos Google Meet para coordinar nuestras reuniones de equipo. A través de estas reuniones, debatimos y compartimos el progreso individual y discutimos cualquier pregunta o inquietud que surgiera durante el proceso de investigación.

Investigación y elaboración de Secciones

Cada miembro del equipo se enfocó en investigar y elaborar una sección específica del proyecto. Utilizamos Google Docs para trabajar en tiempo real en el documento del proyecto, lo que nos permitió editar, agregar y comentar sobre las secciones de manera colaborativa.

Herramientas, Instrumentos y Procedimientos

Herramientas de Comunicación

Empleamos Google Meet para llevar a cabo nuestras reuniones en línea, lo que facilitó la interacción y el intercambio de ideas entre los miembros del equipo, a pesar de la distancia.

Entorno de Desarrollo y Gestión de Datos

Utilizamos SQL Server Management Studio (SSMS) para la implementación práctica y ejecución de scripts relacionados con índices columnares en SQL Server. Esta herramienta nos permitió llevar a cabo nuestras investigaciones y experimentos de manera efectiva. También utilizamos GIT y Github para el desarrollo del script, para de esta manera llevar un registro de las modificaciones que se realizan en los archivos y poder compartir nuestro proyecto de manera pública.

Colaboración en Documentos:

Google Docs fue la plataforma central para la colaboración en tiempo real y la elaboración de cada sección del proyecto. Esta herramienta nos facilitó la edición, adición y revisión conjunta del contenido.

CAPÍTULO IV: DESARROLLO DEL TEMA / PRESENTACIÓN DE RESULTADOS

En este capítulo, presentaremos los hallazgos de nuestro estudio sobre el rendimiento de las consultas en SQL Server utilizando índices columnares en comparación con los índices tradicionales. Nuestro análisis se centrará en varios aspectos clave, incluyendo el tiempo de respuesta de las consultas, el costo de las consultas y las circunstancias bajo las cuales los índices columnares son más eficaces que los índices tradicionales.

Para las pruebas se utilizó la base de datos 'consorcio' proporcionada por la asignatura, en donde se realizaron varios cambios, lo primero fue crear una nueva tabla llamada gastonew utilizando como modelo la tabla gasto, en donde se añadió un **índice columnar no agrupado** sobre la tabla gastonew, ya que como se indicó en el marco conceptual **sólo puede haber un índice agrupado por tabla**, además se modificó el lote de datos, se añadió un script para insertar al menos 1 millón de registros aleatorios en las tablas gasto y gastonew, esto era necesario ya que para lograr un análisis correcto las tablas deberían de tener millones de registros.Diseñamos 2 consultas, 1 para cada tabla, expondremos los primeros resultados hallados al ejecutar estas consultas sobre las tablas mencionadas anteriormente.

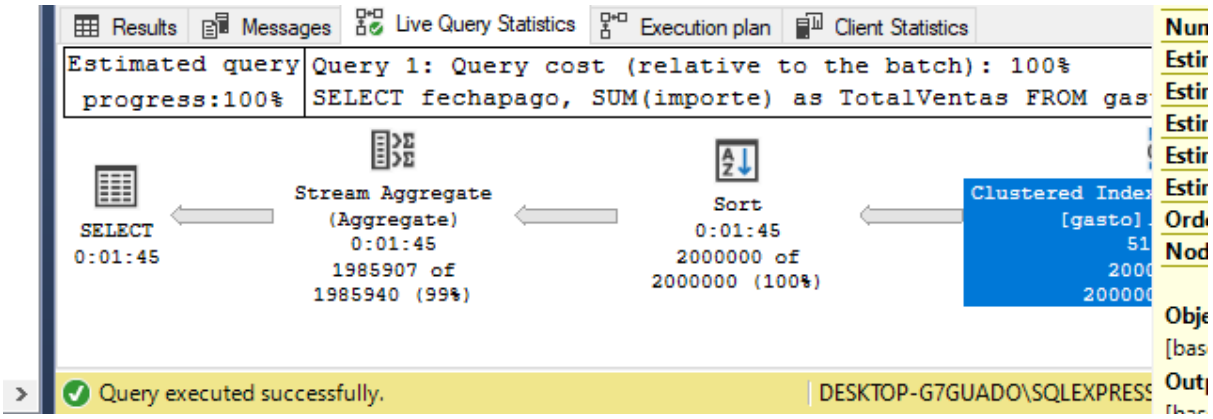
Tiempo de Respuesta de las Consultas

Aquí, presentaremos datos que muestran cómo el tiempo de respuesta de las consultas varía entre las consultas que utilizan índices columnares y las que no. Utilizaremos capturas de pantalla sobre los resultados arrojados por el sql server management studio.

La siguiente consulta realizada selecciona la fecha de pago y la suma total de los importes para cada fecha única, agrupa los resultados por fecha de pago y ordena los resultados por fecha de pago.

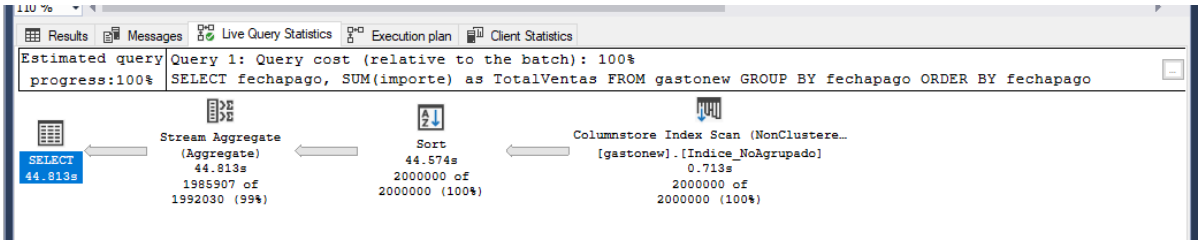
```sql
SELECT fechapago, SUM(importe) as TotalVentas
FROM gasto
GROUP BY fechapago
ORDER BY fechapago;


SELECT fechapago, SUM(importe) as TotalVentas
FROM gastonew
GROUP BY fechapago
ORDER BY fechapago;
```

Los resultados de la consulta sobre la tabla gasto (no contiene el índice columnar) son los siguientes:



Podemos observar que el tiempo que tardó en ejecutarse la consulta fue de 01:45m

Los resultados de la consulta sobre la tabla gastonew (contiene el índice columnar no agrupado) son los siguientes:



Podemos observar que el tiempo que tardó en ejecutarse la consulta fue de 44.813s

Respuesta de la consulta en la tabla gasto: **01:45m**

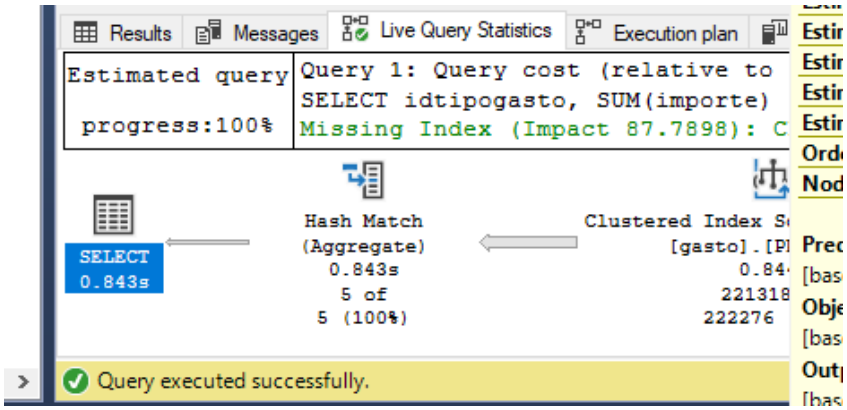
Respuesta de la consulta en la tabla gastonew: **44.813s**

Calculando el resultado obtenemos un porcentaje de mejora aproximado del 57.32%. Esto significa que la segunda consulta fue aproximadamente un 57.32% más rápida que la primera consulta.

La siguiente consulta suma los gastos (importe) de cada tipo de gasto (idtipogasto) en el período 3 (periodo = 3) de la tabla gasto. Cada tipo de gasto tendrá una suma total.

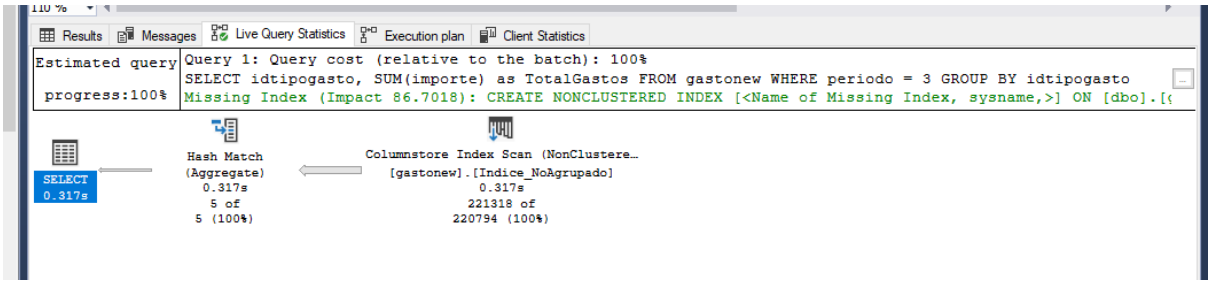
```sql
SELECT idtipogasto, SUM(importe) as TotalGastos
FROM gasto
WHERE periodo = 3
GROUP BY idtipogasto;

SELECT idtipogasto, SUM(importe) as TotalGastos
FROM gastonew
WHERE periodo = 3
GROUP BY idtipogasto;
```

Los resultados de la consulta sobre la tabla gasto (no contiene el índice columnar) son los siguientes:



Podemos observar que el tiempo que tardó en ejecutarse la consulta fue de 0.843s

Los resultados de la consulta sobre la tabla gastonew (contiene el índice columnar no agrupado) son los siguientes:



Podemos observar que el tiempo que tardó en ejecutarse la consulta fue de 0.317s

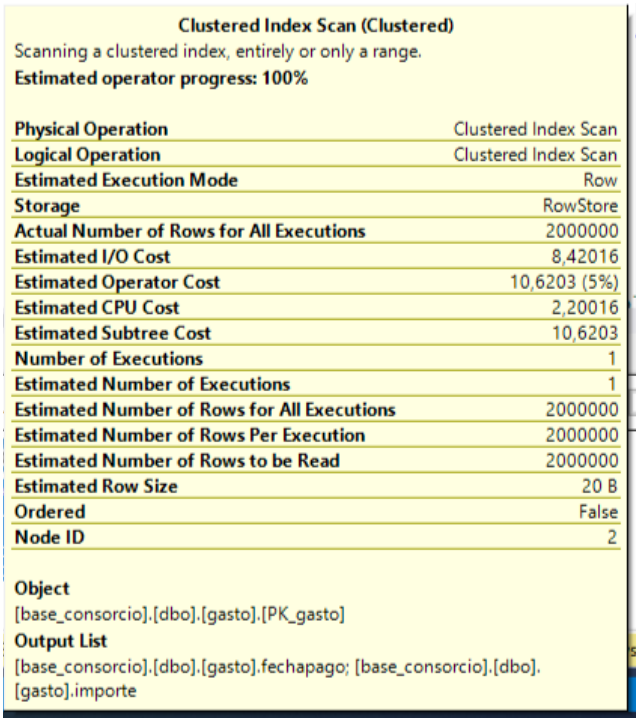
Respuesta de la consulta en la tabla gasto: **0.843s**

Respuesta de la consulta en la tabla gastonew: **0.317s**

Calculando el resultado obtenemos un porcentaje de mejora aproximado del 62.39%. Esto significa que la segunda consulta fue aproximadamente un 62.39% más rápida que la primera consulta.

Costo de las Consultas

En esta sección, se analizará el costo asociado con la ejecución de consultas utilizando índices columnares en comparación con los índices tradicionales. Nuevamente, utilizaremos capturas de pantalla para presentar estos datos de una manera fácil de entender.



Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows for All Executions	2000000
Estimated I/O Cost	8,42016
Estimated Operator Cost	10,6203 (5%)
Estimated CPU Cost	2,20016
Estimated Subtree Cost	10,6203
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2000000
Estimated Number of Rows Per Execution	2000000
Estimated Number of Rows to be Read	2000000
Estimated Row Size	20 B
Ordered	False
Node ID	2

Object
[base_consorcio].[dbo].[gasto].[PK_gasto]
Output List
[base_consorcio].[dbo].[gasto].fechapago; [base_consorcio].[dbo].[gasto].importe

Aquí se presentan los siguientes resultados:

Con Clustered Index Scan (consulta sobre la tabla gasto), arroja lo siguiente:

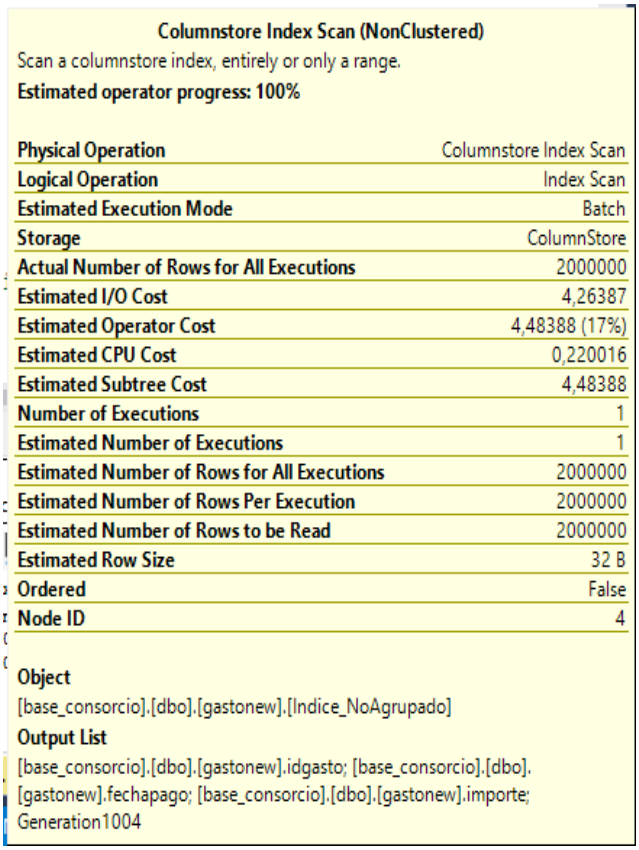
Almacenamiento: “Almacenamiento en Fila” (RowStore)

Costo estimado de E/S: 8.4

Costo estimado de operador: 10.62

Costo de CPU: 2.20

Costo estimado de subárbol: 10.62



Columnstore Index Scan (NonClustered)

Scan a columnstore index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Columnstore Index Scan
Logical Operation	Index Scan
Estimated Execution Mode	Batch
Storage	ColumnStore
Actual Number of Rows for All Executions	2000000
Estimated I/O Cost	4,26387
Estimated Operator Cost	4,48388 (17%)
Estimated CPU Cost	0,220016
Estimated Subtree Cost	4,48388
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2000000
Estimated Number of Rows Per Execution	2000000
Estimated Number of Rows to be Read	2000000
Estimated Row Size	32 B
Ordered	False
Node ID	4

Object
[base_consorcio].[dbo].[gastonew].[Indice_NoAgrupado]
Output List
[base_consorcio].[dbo].[gastonew].idgasto; [base_consorcio].[dbo].[gastonew].fechapago; [base_consorcio].[dbo].[gastonew].importe; Generation1004

Con Columnstore Index Scan (consulta sobre la tabla gastoNew), arroja lo siguiente:

Almacenamiento: “Almacenamiento en columna” (ColumnStore)

Costo estimado de E/S: 4.26 (49,29% de mejora con respecto al anterior)

Costo estimado de operador: 4.48 (57,81% de mejora con respecto al anterior)

Costo de CPU: 0.22 (90% de mejora con respecto al anterior.

Costo estimado de subárbol: 4.48 (57,81% de mejora con respecto al anterior)

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows for All Executions	221318
Estimated I/O Cost	8,42016
Estimated Operator Cost	10,6203 (84%)
Estimated CPU Cost	2,20016
Estimated Subtree Cost	10,6203
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	222276
Estimated Number of Rows Per Execution	222276
Estimated Number of Rows to be Read	2000000
Estimated Row Size	20 B
Ordered	False
Node ID	1

Predicate

[base_consorcio].[dbo].[gasto].[periodo]=(3)

Object

[base_consorcio].[dbo].[gasto].[PK_gasto]

Output List

[base_consorcio].[dbo].[gasto].idtipogasto; [base_consorcio].[dbo].[gasto].importe

Aquí se presentan los siguientes resultados:

Con Clustered Index Scan (otra consulta sobre la tabla gasto), arroja lo siguiente:

Almacenamiento: “Almacenamiento en Fila” (RowStore)

Costo estimado de E/S: 8.4

Costo estimado de operador: 10.62

Costo de CPU: 2.20

Costo estimado de subárbol: 10.62

Columnstore Index Scan (NonClustered)

Scan a columnstore index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Columnstore Index Scan
Logical Operation	Index Scan
Estimated Execution Mode	Batch
Storage	ColumnStore
Actual Number of Rows for All Executions	221318
Estimated I/O Cost	4,02387
Estimated Operator Cost	4,24388 (95%)
Estimated CPU Cost	0,220016
Estimated Subtree Cost	4,24388
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	220794
Estimated Number of Rows to be Read	2000000
Estimated Number of Rows for All Executions	220794
Estimated Row Size	32 B
Ordered	False
Node ID	3

Predicate

[base_consorcio].[dbo].[gastonew].[periodo]=(3)

Object

[base_consorcio].[dbo].[gastonew].[Indice_NoAgrupado]

Output List

[base_consorcio].[dbo].[gastonew].idgasto; [base_consorcio].[dbo].[gastonew].idtipogasto; [base_consorcio].[dbo].[gastonew].importe; Generation1004

Con Columnstore Index Scan (consulta sobre la tabla gastoNew), arroja lo siguiente:

Almacenamiento: “Almacenamiento en columna” (ColumnStore)

Costo estimado de E/S: 4.02 (Mejora del 52,14% con respecto al anterior)

Costo estimado de operador: 4.24 (Mejora de 60.08% con respecto al anterior)

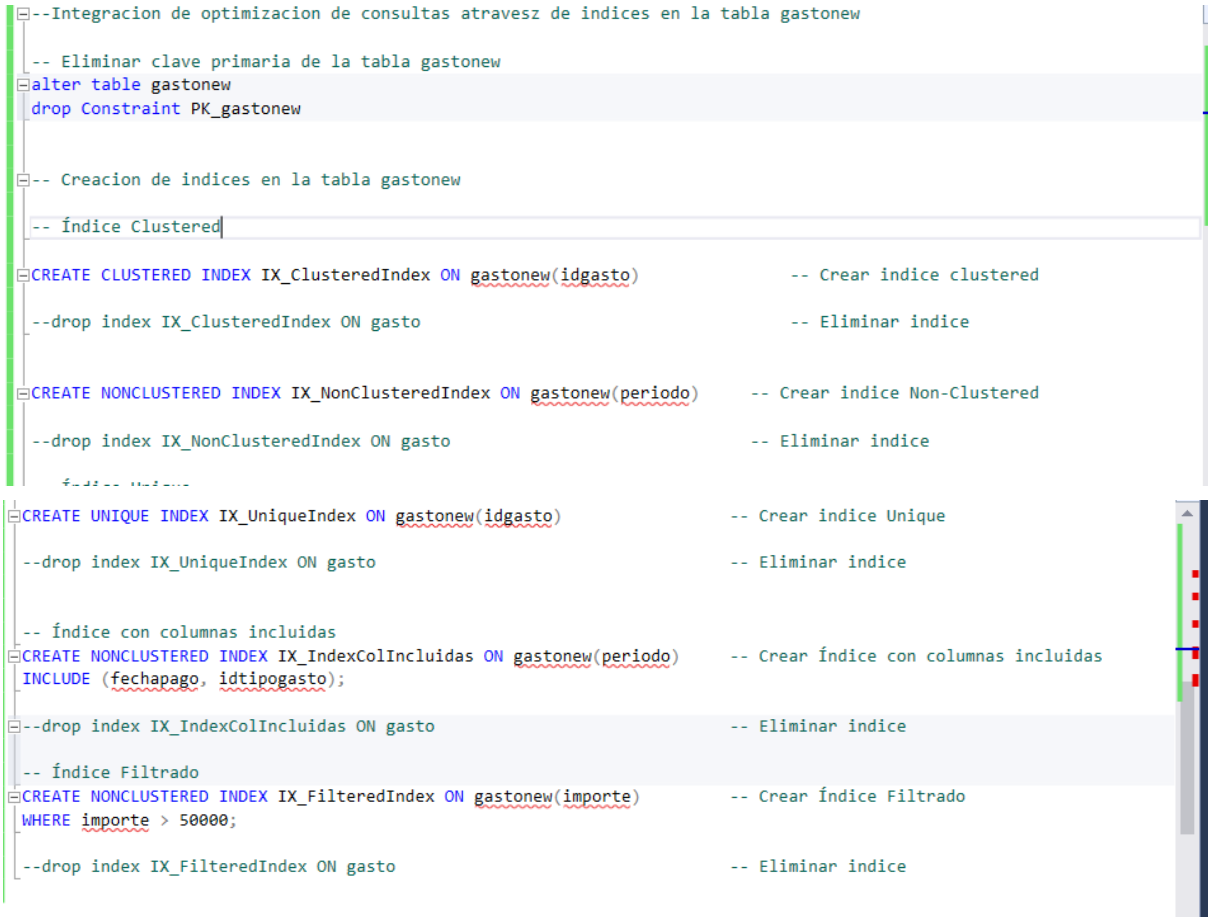
Costo de CPU: 0.22 (Mejora del 90% con respecto al anterior)

Costo estimado de subárbol: 4.24 (Mejora del 60.08% con respecto al anterior)

CAPÍTULO V: Integración de temas

Uno de los temas a aplicar en nuestro trabajo fue la optimización de consultas a través de índices. Observando el trabajo de nuestros compañeros e investigando al respecto, pudimos agregar varios índices en la tabla gastonew. Se crearon índices del tipo:

Índice Clustered, Índice Unique, Índice con columnas incluidas, Índice Filtrado



```
--Integracion de optimizacion de consultas atravesz de indices en la tabla gastonew

-- Eliminar clave primaria de la tabla gastonew
alter table gastonew
drop Constraint PK_gastonew


-- Creacion de indices en la tabla gastonew

-- Índice Clustered
CREATE CLUSTERED INDEX IX_ClusteredIndex ON gastonew(idgasto)              -- Crear indice clustered

--drop index IX_ClusteredIndex ON gasto                                   -- Eliminar indice


CREATE NONCLUSTERED INDEX IX_NonClusteredIndex ON gastonew(periodo)     -- Crear indice Non-Clustered

--drop index IX_NonClusteredIndex ON gasto                               -- Eliminar indice

CREATE UNIQUE INDEX IX_UniqueIndex ON gastonew(idgasto)                 -- Crear indice Unique

--drop index IX_UniqueIndex ON gasto                                     -- Eliminar indice


-- Índice con columnas incluidas
CREATE NONCLUSTERED INDEX IX_IndexColIncluidas ON gastonew(periodo)     -- Crear Índice con columnas incluidas
INCLUDE (fechapago, idtipogasto);

--drop index IX_IndexColIncluidas ON gasto                               -- Eliminar indice

-- Índice Filtrado
CREATE NONCLUSTERED INDEX IX_FilteredIndex ON gastonew(importe)         -- Crear Índice Filtrado
WHERE importe > 50000;

--drop index IX_FilteredIndex ON gasto                                   -- Eliminar indice
```

Otro de los temas que nos tocó integrar, fue el de triggers, lo que hicimos fue crear una tabla llamada auditoría, donde guardaremos todos los cambios, y los triggers que lo aplicamos a la tabla conserje, en sus 3 casos, insert, update y delete, se busca que cada vez que se realice una acción de carácter insertar, eliminación o edición los datos relacionados se almacenen en la tabla auditoría, es decir que, cada vez que se inserte un conserje nuevo podremos visualizarlo en la tabla auditoría.Esta tabla resulta muy útil en situaciones donde, por ejemplo, se requiere saber en qué fecha se creó un usuario o se necesita deshacer un dato editado o borrado.

```sql
--La tabla en cuestión sirve para dos propósitos principales: proporcionar una vista rápida de los datos alterados en
CREATE TABLE auditoria (
    id_auditoria int identity primary key,
    tabla_afectada varchar(100),
    columna_afectada varchar(100),
    accion varchar(10),
    fecha_hora datetime,
    usuario varchar(50),
    valor_anterior varchar(max),
    valor_actual varchar(max),
);

GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- Trigger para INSERT en la tabla conserje
CREATE TRIGGER dbo.trconserjeInsert
   ON  dbo.conserje
   AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, ac
```

```sql
-- Trigger para DELETE en la tabla conserje
CREATE TRIGGER dbo.trconserjeDelete
   ON  dbo.conserje
   AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
    SELECT 'dbo.conserje','idconserje', deleted.idconserje, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;

    INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
    SELECT 'dbo.conserje','apeynom', deleted.apeynom, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;

    INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
    SELECT 'dbo.conserje','tel', deleted.tel, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;

    INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
    SELECT 'dbo.conserje','fechnac', deleted.fechnac, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;

    INSERT INTO dbo.auditoria(tabla_afectada, columna_afectada, valor_Anterior, valor_actual, usuario, fecha_hora, accion)
    SELECT 'dbo.conserje','estciv', deleted.estciv, NULL, SYSTEM_USER, CURRENT_TIMESTAMP, 'Delete' FROM deleted;
END
GO
```

CAPÍTULO VI: CONCLUSIONES

En conclusión, y en concordancia con lo indicado tanto por la teoría como por la práctica, se han obtenido resultados determinantes que confirman la utilidad de los índices columnares en la reducción del uso de recursos y en la mejora de la eficiencia y el tiempo utilizados. Los índices columnares, al almacenar los datos en un formato basado en columnas, permiten un acceso más rápido y eficiente a los datos. Esto se traduce en una disminución significativa en las operaciones de E/S y, por ende, en una mejora notable en el rendimiento de las consultas.

Nuestra investigación demostró que la implementación de índices columnares en SQL Server puede conducir a una mejora significativa en el rendimiento de las consultas. Las consultas que utilizan índices columnares en nuestra base de datos de prueba mostraron

tiempos de respuesta considerablemente más cortos en comparación con las mismas consultas en la base de datos sin índices columnares. Por ejemplo, en una consulta, observamos una mejora de aproximadamente el 62.39% en el tiempo de respuesta. Además de una mejora en el rendimiento, notamos una disminución en el costo de ejecución de las consultas al utilizar índices columnares. Los planes de ejecución estimados revelaron que el costo de E/S y de CPU disminuyó significativamente hasta en un 90% en las consultas que involucran índices columnares. Esto puede traducirse en un ahorro de recursos y costos en entornos de producción con una alta carga de consultas.

Finalmente, concluimos que nuestros hallazgos respaldan la idea de que los índices columnares son especialmente efectivos en situaciones que involucran grandes conjuntos de datos y consultas analíticas que escanean amplios rangos de valores. Su capacidad para comprimir datos y reducir la E/S del sistema es particularmente beneficiosa en tales escenarios.

CAPÍTULO VII: BIBLIOGRAFÍA

1. **Microsoft Docs (2023).** “Guía Índices.” Microsoft Docs.
2. **Camuña Rodríguez, J. F. (2015)**. “Lenguajes de definición y modificación de datos SQL”. Antequera, Málaga, Spain: IC Editorial.