



Universidad Nacional del Nordeste



Facultad de Ciencias Exactas y Naturales y Agrimensura

Carrera: Licenciatura en Sistemas de Información

Cátedra: Base de Datos I

Tema: Optimización de consultas a través de índices

Año: 2024

Integrantes:

Almirón, Pedro Augusto.

DNI: 42.791.957 **LU:** 54251

Indice

• RESUMEN.....	3
• OBJETIVO PRINCIPAL.....	3
• <i>ENTENDER CONCEPTUALMENTE EL SIGNIFICADO.</i>	3
• <i>ENTENDER LOS DIFERENTES TIPOS DE ÍNDICES</i>	3
• <i>ENTENDER LA DIFERENCIA EN EL ACCESO A LOS DATOS A TRAVÉS DE ÍNDICES Y SIN ELLOS</i>	3
• CAPÍTULO I: INTRODUCCIÓN	
• <i>IMPORTANCIA DE LA OPTIMIZACIÓN DE BÚSQUEDAS EN BASES DE DATOS.</i>	3
• <i>PAPEL CRUCIAL DE LOS ÍNDICES EN LA OPTIMIZACIÓN DEL RENDIMIENTO DE CONSULTAS</i>	3
• CAPÍTULO II: MARCO CONCEPTUAL O REFERENCIAL.....	4
• <i>INTRODUCCIÓN A LOS ÍNDICES EN LOS MOTORES DE BASES DE DATOS</i>	4
• <i>FUNDAMENTOS DE LOS ÍNDICES</i>	4
• <i>TIPOS DE ÍNDICES EN SQL SERVER</i>	4
• <i>ÍNDICES CLUSTERED Y NON-CLUSTERED</i>	5
• <i>DIFERENCIAS ENTRE ÍNDICES CLUSTERED Y NON-CLUSTERED</i>	5
• <i>ESTRATEGIAS DE USO Y CREACIÓN DE ÍNDICES</i>	5
• CAPÍTULO III: METODOLOGÍA SEGUIDA.....	6
• <i>DESCRIPCIÓN DE CÓMO SE REALIZÓ EL TRABAJO PRÁCTICO</i>	6
• <i>HERRAMIENTAS (INSTRUMENTOS Y PROCEDIMIENTOS)</i>	6
• <i>INSERCIÓN MASIVA DE DATOS</i>	7
• <i>GENERACIÓN DE CONSULTAS Y CREACIÓN DE ÍNDICES</i>	8
• CAPÍTULO IV: DESARROLLO DEL TEMA/RESULTADOS.....	11
• <i>RESULTADOS DE LAS PRUEBAS DE RENDIMIENTO CON DIFERENTES ÍNDICES</i>	11
• <i>COMPARACIÓN DE TIEMPOS DE RESPUESTA ENTRE CONSULTAS CON Y SIN ÍNDICES.</i>	
• <i>IMPACTO DE LOS ÍNDICES EN EL RENDIMIENTO DEL SISTEMA</i>	11
• CAPÍTULO V: CONCLUSIÓN.....	12
• <i>RESUMEN DE LOS HALLAZGOS Y CONCLUSIONES DE LAS PRUEBAS</i>	12
• <i>IMPORTANCIA DE LA IMPLEMENTACIÓN ADECUADA DE ÍNDICES EN LA OPTIMIZACIÓN DE BASES DE DATOS</i>	12
• CAPÍTULO VI: BIBLIOGRAFÍA.....	12
• <i>FUENTES CONSULTADAS PARA LA INVESTIGACIÓN Y DESARROLLO DEL TRABAJO.</i>	

Resumen:

Los índices son una herramienta clave en el mundo del rendimiento de las bases de datos en general, permite hacer de forma eficiente consultas que a grande rasgos, si una base de datos no tiene una buena gestión de índices podría significar una perdida enorme de rendimiento y en algunos casos causar que la aplicación o sistema ni siquiera funcione.

Objetivo Principal:

- Entender conceptualmente el significado.
- Entender los diferentes tipos de índices.
- Entender la diferencia en el acceso a los datos a través de índices y sin ellos.

CAPÍTULO I: INTRODUCCIÓN

En el vasto mundo de las bases de datos, la optimización de búsquedas es un tema fundamental que influye directamente en el rendimiento y la eficiencia de los sistemas de gestión de datos. La capacidad de recuperar información de manera rápida y precisa es esencial en cualquier aplicación o sistema que maneje grandes volúmenes de datos. Los índices juegan un papel crucial en este proceso, permitiendo acelerar las consultas al proporcionar acceso directo a los registros relevantes. Existen varios tipos de índices, en este trabajo en específico se tendrá especial énfasis en los tipos de índices **Clúster y No Clustered**

a. **Tema.** El tema de este trabajo se centra en la optimización de búsqueda a través de índices. Este enfoque apunta a realizar consultas a determinadas tablas de manera eficiente, es decir requiriendo el menor costo posible del cpu (Unidad central del procesamiento)

b. **Definición o planteamiento del problema.** En este trabajo integrador exploraremos en detalle el concepto de optimización de búsquedas a través de índices en bases de datos. Analizaremos cómo los índices optimizan las consultas al mejorar la velocidad de acceso a los datos y reducir la carga en los sistemas. Además, examinaremos los diferentes tipos de índices disponibles en los sistemas de gestión de bases de datos, así como las estrategias y técnicas utilizadas para diseñar, implementar y mantener índices eficientes.

c. **Objetivo del Trabajo Práctico.** Proporcionar una comprensión integral de los índices de búsqueda, su definición, sus tipos y sus

aplicaciones para poder entender la importancia de trabajar con ellos. Para ello planteamos los siguientes objetivos:

i. Objetivos Generales:

- Comprender a fondo la administración de índices de búsqueda en bases de datos y su impacto en el rendimiento de los datos.

ii. Objetivos Específicos:

- Identificar los conceptos básicos relacionados con los índices, tipos, diferencias en una base de datos.
- Investigar cómo la gestión de índices contribuye al rendimiento y eficacia de las consultas en la base de datos.
- Analizar la diferencia entre los índices clúster y no clustered y la diferencia de trabajar con y sin ellos.

CAPITULO II: MARCO CONCEPTUAL o REFERENCIAL

Introducción a los índices en los motores de bases de datos

Los índices juegan un papel fundamental en la optimización del rendimiento de las consultas en SQL Server. Proporcionan una forma eficiente de acceder a los datos almacenados en tablas, permitiendo una recuperación rápida y efectiva de la información. En este trabajo, exploraremos en detalle el concepto de índices en SQL Server, su funcionamiento, tipos y estrategias de uso para mejorar el rendimiento de las consultas.

Fundamentos de los índices

Los índices se basan en estructuras de datos optimizadas, como árboles Binarios, que permiten una búsqueda rápida y eficiente de datos en función de los valores de las columnas indexadas. Al acceder a una tabla mediante un índice, el motor de base de datos puede evitar la necesidad de explorar todas las filas de la tabla en busca de los datos requeridos, lo que resulta en tiempos de respuesta más rápidos y un rendimiento general mejorado de las consultas.

Tipos de índices en Sql Server:

Los índices en SQL Server son estructuras que permiten una recuperación rápida de datos en una tabla, lo que mejora significativamente el rendimiento de las consultas. Existen varios tipos de índices, pero los más comunes son los índices clustered y non-clustered.

Índices Clustered:

Los índices clustered reorganizan físicamente los datos en la tabla según el orden de las claves de índice.

Cada tabla puede tener solo un índice clustered, ya que determina el orden físico de los datos.

Al ser el orden físico de los datos, el índice clustered también almacena los datos de la tabla, lo que significa que la clave del índice es la clave primaria de la tabla o, si no hay clave primaria, la primera columna definida en el índice.

Los índices clustered son especialmente eficientes para consultas que recuperan rangos de datos consecutivos o que devuelven una gran cantidad de columnas de la tabla.

Índices Non-Clustered:

Los índices non-clustered son una estructura de índice separada de los datos de la tabla.

Cada tabla puede tener varios índices non-clustered.

Los índices non-clustered contienen las claves de índice y un puntero a la ubicación física de los datos en la tabla.

Aunque los índices non-clustered son más versátiles que los índices clustered, ya que permiten múltiples índices y no alteran la organización física de los datos, pueden ser menos eficientes para consultas que recuperan grandes cantidades de datos o realizan operaciones de ordenación, ya que requieren acceso adicional a los datos de la tabla.

Diferencias entre Índices Clustered y Non-Clustered:

La principal diferencia entre los índices clustered y non-clustered radica en la forma en que se organizan físicamente los datos.

Los índices clustered reorganizan los datos en la tabla según el orden de las claves de índice, mientras que los índices non-clustered mantienen una estructura de índice separada.

Los índices clustered son más eficientes para consultas que recuperan rangos de datos consecutivos o devuelven una gran cantidad de columnas de la tabla, mientras que los índices non-clustered son más versátiles pero pueden ser menos eficientes para consultas que recuperan grandes cantidades de datos.

Estrategias de Uso y Creación de Índices:

Crear índices de manera efectiva es crucial para mejorar el rendimiento de las consultas en SQL Server. Aquí hay algunas estrategias básicas y consideraciones clave:

Identificación de Escenarios de Uso:

Analiza las consultas más frecuentes y las operaciones de búsqueda que se realizan con mayor regularidad en la base de datos.

Identifica las consultas que podrían beneficiarse de índices para mejorar su rendimiento.

Selección de Columnas a Indexar:

Indexa las columnas que se utilizan frecuentemente en las cláusulas WHERE, JOIN y ORDER BY de las consultas.

Considera indexar columnas que tienen alta cardinalidad (valores únicos) y que se utilizan para filtrar datos con frecuencia.

Evita indexar columnas que tienen baja selectividad o que se actualizan con frecuencia, ya que los índices pueden afectar el rendimiento de las operaciones de escritura.

Casos de Uso y Ejemplos Prácticos:

Estudio de Casos: Análisis de casos reales donde se aplicaron estrategias de índices en SQL Server para mejorar el rendimiento de consultas en entornos de producción.

Ejemplos Prácticos: Demostración de la creación y utilización de índices en SQL Server a través de ejemplos de código y escenarios de consulta comunes.

CAPÍTULO III: METODOLOGÍA SEGUIDA.

Las acciones llevadas a cabo para realizar este trabajo fueron:

a. Descripción de cómo se realizó el Trabajo Práctico.

Una vez que recibí el tema del Trabajo Práctico, asignado por el profesor Dario Villegas de la cátedra de Base de Datos I, empecé a investigar el tema de consulta a través de índices. El primer trabajo estuvo bastante incompleto, no porque estuviese mal sino porque las pruebas no se podían ver los resultados debido a que el lote de datos no era demasiado grande, en esta ocasión el profesor mencionado anteriormente me recomendó generar un script (demostración a posteriori) para realizar una inserción masiva de 1 millón de registros en la tabla gastos, donde se puede ver mejor el resultado y el impacto en de los índices en la base de datos.

Una de las dificultades que encontré durante este proyecto estuvo relacionada con la naturaleza de los índices, es decir existen muchos tipos y no son gratis utilizarlos, en términos de almacenamiento. También crear índices sin discriminación no garantiza un mejor rendimiento, muchas veces hace que el sistema sea más complejo innecesariamente y dificulta el escalado y el mantenimiento de la base de datos.

Para resolver esto, utilice muchas bibliografías, páginas de internet y también con ayuda de inteligencia artificial, donde recomiendan que índices utilizar, en que columnas implementar y sobre todo cuando vale la pena utilizar.

En este trabajo se realizaron pruebas independientes generando 2 bases de datos diferente para realizar las consultas con los índices mencionados.

b. Herramientas (Instrumentos y procedimientos).

Es importante recordar, que estas consultas se debe hacer en un entorno local o en una base de pruebas para verificar que funciona, para luego asi poder implementarla en la producción

1. Abrimos el SQL Server Management Studio.
2. Generamos una nueva consulta, donde realizaremos la prueba.

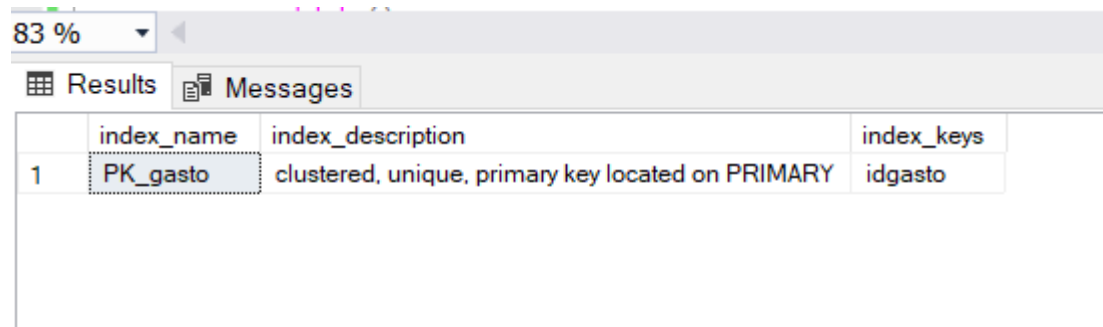
Importante : Las claves primarias (Primary key) generan un índice cluster por defecto en la mayoría de los motores de las bases de datos

```
--proyecto bases de datos 1 2024 - ALMIRON PEDRO AUGUSTO, OPTIMIZACION DE BUSQUEDA MEDIANTE INDICES
use consorcio
--VERIFICAMOS QUE NO EXISTE NINGUN INDEX EN LA TABLA CONSORCIO MAS QUE LA PK QUE SE GENERA POR DEFECTO
execute sp_helpindex 'gasto'

--activamos estadísticas de tiempo y de lectura
SET STATISTICS TIME ON;
SET STATISTICS IO ON;
```

3. En este trabajo se generara 3 bases de datos distintas entre si con distintos índices en la tabla gasto, para poder comparar así los resultados entre las 3 y así descubrir cual de las 3 es mas optima para una determinada consulta

4. El comando “execute sp_helpindex” nos devolverá en forma de registros cualquier tipo de índice que este creado en la tabla en este caso la tabla gasto:



	index_name	index_description	index_keys
1	PK_gasto	clustered, unique, primary key located on PRIMARY	idgasto

Como era de esperarse la tabla gasto tiene un índice cluster generado por la PK_gasto

i. Creamos la base de datos prueba 1

Cremos la base de datos prueba 1, luego ejecutamos el modelo de datos ‘consorcio’ con su respectivo lote de datos, al final insertamos 1.000.000 de registros en la tabla gasto, se adjunta el script correspondiente

```
create database prueba1
use prueba1
--cargamos el modelo de datos consorcio e insertamos el lote de datos correspondiente

--insertamos 1.000.000 de registros en la tabla gastos para que tenga sentido los indices
DECLARE @TotalRows INT = 1000000; -- Total de registros a insertar
DECLARE @Counter INT = 1;
DECLARE @FechaPago datetime = GETDATE(); -- Fecha de pago constante, se toma la fecha actual
DECLARE @Periodo INT = 1; -- Periodo fijo para todos los registros

WHILE @Counter <= @TotalRows
BEGIN
    INSERT INTO gasto (idprovincia, idlocalidad, idconsorcio, periodo, fechapago, idtipogasto, importe)
    VALUES (1,
            1,
            1,
            2,
            getdate(),
            1,
            1000); -- Mismo valor para el importe en cada registro

    SET @Counter = @Counter + 1;
END;
```

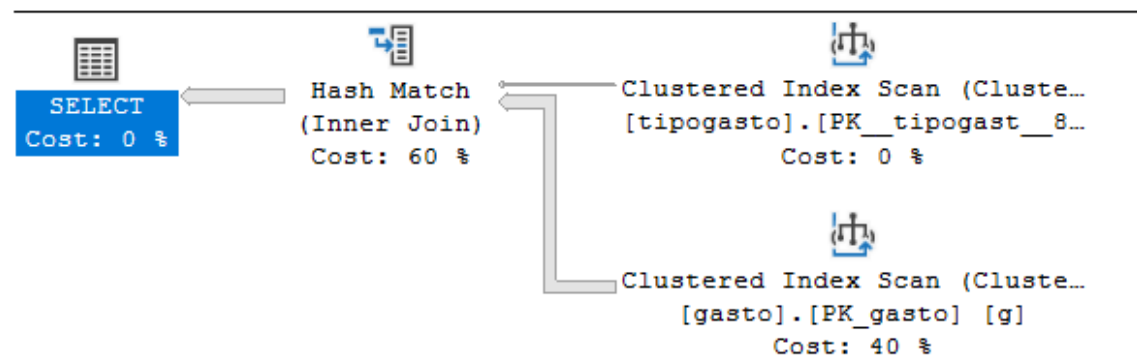
Es un script que contiene lógica básica de programación simplemente es iterar el numero de veces necesaria la ejecución de la misma instrucción (en este caso el insert), los valores son exactamente los mismos para cada uno de los registros excepto obviamente la clave primaria, se puede modificar el script para obtener valores aleatorios por cada registro, pero eso podría afectar significativamente el tiempo de ejecución de la inserción, situación la cual no es el objetivo de este trabajo en particular.

j. Generamos una consulta a la tabla con el lote insertado para la base de datos prueba 1:

A continuación, se generará una consulta a la tabla gasto con los registros insertados anteriormente y verificaremos el rendimiento de la misma con el plan de ejecución de Sql server

```
/*Realizar una búsqueda por periodo,
seleccionando los campos: fecha de pago y tipo de gasto (con la descripción correspondiente)*/
SELECT g.fechapago, tg.descripcion FROM gasto g
inner join tipogasto tg on tg.idtipogasto = g.idtipogasto
where g.periodo = 2
```

Plan de ejecución



Aquí están los costos estimados de lo que necesita la consulta para ser ejecutada

Clustered Index Scan (Clustered)		Hash Match	
Scanning a clustered index, entirely or only a range.		Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.	
Physical Operation	Clustered Index Scan	Physical Operation	Hash Match
Logical Operation	Clustered Index Scan	Logical Operation	Inner Join
Estimated Execution Mode	Row	Estimated Execution Mode	Row
Storage	RowStore	Estimated Operator Cost	8.1292125 (60%)
Estimated Operator Cost	5.3543 (40%)	Estimated I/O Cost	0
Estimated I/O Cost	4.24535	Estimated Subtree Cost	13.4868
Estimated Subtree Cost	5.3543	Estimated CPU Cost	7.64532
Estimated CPU Cost	1.10896	Estimated Number of Executions	1
Estimated Number of Executions	1	Estimated Number of Rows Per Execution	999660
Estimated Number of Rows to be Read	1008000	Estimated Number of Rows for All Executions	999660
Estimated Number of Rows for All Executions	999660	Estimated Row Size	44 B
Estimated Number of Rows Per Execution	999660	Node ID	0
Estimated Row Size	23 B	Output List	
Ordered	False	[prueba1].[dbo].[gasto].fechapago, [prueba1].[dbo].[tipogasto].descripcion	
Node ID	2	Hash Keys Probe	
Predicate		[prueba1].[dbo].[gasto].idtipogasto	
[prueba1].[dbo].[gasto].[periodo] as [g].[periodo]=(2)		Probe Residual	
Object		[prueba1].[dbo].[gasto].[idtipogasto] as [g].[idtipogasto]=[prueba1].[dbo].[tipogasto].[idtipogasto] as [tg].[idtipogasto]	
Output List			
[prueba1].[dbo].[gasto].fechapago, [prueba1].[dbo].[gasto].idtipogasto			

Como se ve en el plan, la instruccion que tiene mas costo es el inner join.

Ejecutamos la consulta

	fechapago	descripcion
1	2013-02-19 00:00:00.000	Aportes
2	2013-02-24 00:00:00.000	Limpieza
3	2013-02-03 00:00:00.000	Sueldos
4	2013-02-20 00:00:00.000	Servicios
5	2013-02-17 00:00:00.000	OTROS
6	2013-02-04 00:00:00.000	Limpieza
7	2013-02-15 00:00:00.000	Sueldos
8	2013-02-07 00:00:00.000	Aportes
9	2013-02-05 00:00:00.000	Aportes
10	2013-02-01 00:00:00.000	Limpieza
11	2013-02-13 00:00:00.000	OTROS
12	2013-02-11 00:00:00.000	Limpieza
13	2013-02-23 00:00:00.000	Servicios
14	2013-02-15 00:00:00.000	Sueldos
15	2013-02-20 00:00:00.000	Sueldos
16	2013-02-27 00:00:00.000	Limpieza
17	2013-02-05 00:00:00.000	Aportes

```
SQL Server parse and compile time:
    CPU time = 0 ms, elapsed time = 0 ms.

(1000771 rows affected)
Table 'Workfile'. Scan count 0, logical reads 0, physical re
Table 'Worktable'. Scan count 0, logical reads 0, physical r
Table 'gasto'. Scan count 1, logical reads 5750, physical re
Table 'tipogasto'. Scan count 1, logical reads 2, physical r

SQL Server Execution Times:
    CPU time = 1140 ms,  elapsed time = 4210 ms.

Completion time: 2024-02-27T18:29:17.9389530-03:00
```

Como se puede apreciar la consulta arroja un resultado de 1.404.543 registros con un tiempo de respuesta de 4210 ms lo que equivale a 4,21 segundos de respuesta

Conclusión de prueba 1

La consulta ejecutada utilizo el índice clúster PK_gasto para realizar las funciones correspondientes y mostrar la tabla de gastos con sus respectivas fecha y descripciones de gasto con una latencia de **4210ms**

k. Creamos bases de datos prueba 2

Creamos la base de datos prueba 2, y hacemos los mismos pasos realizados para la prueba 1

Creemos índice no cluster entre fechapago, idtipo gasto para experimentar si el costo de la consulta es diferente a la de la primera prueba, recordemos que la consulta es

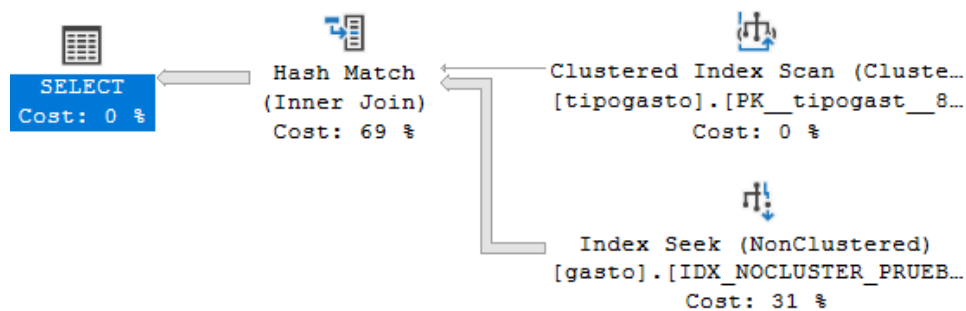
```
USE [prueba2]
GO
CREATE NONCLUSTERED INDEX [IDX_NOCLUSTER_PRUEBA1]
ON [dbo].[gasto] ([periodo])
INCLUDE ([fechapago],[idtipogasto])
sp_helpindex 'gasto' --verificamos que el indice se creo correctamente
exactamente la misma
```

El índice se crea con éxito, veamos el plan de ejecución de la consulta definida al principio

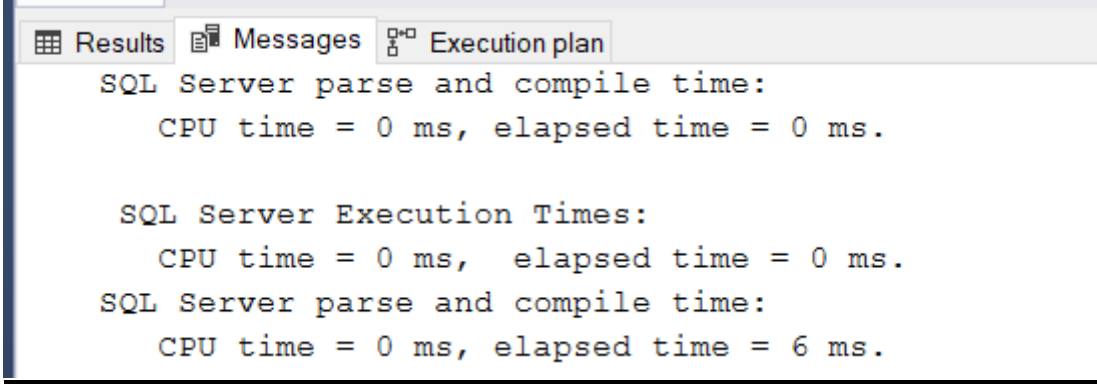
Query:

```
SELECT g.fechapago, tg.descripcion FROM gasto g
inner join tipogasto tg on tg.idtipogasto = g.idtipogasto
where g.periodo = 2
```

Plan de ejecucion



Resultados de la consulta con el índice no cluster IDX_GASTO TIPO GASTO

A screenshot of the SQL Server Enterprise Manager interface. At the top, there are three tabs: 'Results' (selected), 'Messages', and 'Execution plan'. The 'Results' tab displays the following text:

```
SQL Server parse and compile time:
    CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
    CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
    CPU time = 0 ms, elapsed time = 6 ms.
```

COMPAREMOS LOS RESULTADOS

QUERY CON INDICE CLUSTER PK_GASTO: 5.815 seg de respuesta

QUERY CON INDICE NO CLUSTER: 6 ms de respuesta

Es decir, en este caso, la creación del índice fue muy conveniente reduciendo de manera significativa los costos de eficiencia para la misma consulta

CAPÍTULO IV: DESARROLLO DEL TEMA/RESULTADOS.

QUERY CON INDICE NO CLUSTER PK_GASTO: 6ms de respuesta

QUERY CON INDICE CLUSTER: 5.815 seg de respuesta

Durante nuestras pruebas de rendimiento en la base de datos del proyecto, hemos evaluado el impacto de diferentes tipos de índices en el tiempo de respuesta de las consultas.

Al analizar los resultados, podemos observar que la prueba 2 con el índice no clúster fue mucho más eficiente que la prueba que solo cuenta con el índice cluster PK_gasto.

Hemos demostrado que, para la misma consulta, utilizando índices hemos podido ahorrarnos casi el 95% de costo y tiempo de respuesta. Tengamos presente que no siempre se puede obtener una mejora tan significativa, pero en un lote grande de datos cualquier mejor por pequeña que sea puede afectar significativamente el rendimiento de la base de datos

CAPÍTULO IV: Conclusión.

En resumen, nuestras pruebas de rendimiento han demostrado la significativa influencia que los índices ejercen en el tiempo de respuesta de las consultas en SQL Server. Aunque las diferencias entre el uso de índices clustered y no clustered fueron significativas en este caso específico, queda claro que la implementación adecuada de índices puede conducir a mejoras palpables en el rendimiento del sistema. Estos resultados subrayan la importancia de considerar cuidadosamente la creación de índices y su mantenimiento como parte integral de la optimización de bases de datos, especialmente en entornos donde la eficiencia y la capacidad de respuesta son críticas para el éxito del proyecto.

En segundo lugar, es esencial reconocer que, aunque los índices pueden proporcionar mejoras en el tiempo de respuesta de las consultas, su diseño y uso deben ser evaluados en función de las necesidades específicas del sistema. Esto implica una comprensión profunda de las consultas frecuentes, el patrón de acceso a los datos y la carga de trabajo esperada. Al equilibrar cuidadosamente estos factores y aplicar estrategias efectivas de creación y mantenimiento de índices, podemos aprovechar al máximo el potencial de los índices para optimizar el rendimiento de las bases de datos SQL Server.

CAPÍTULO IV: Bibliografía.

- <https://www.sqlshack.com/es/vision-general-y-estrategias-de-indices-sql/>
- <https://use-the-index-luke.com/es/sql/%C3%ADndice-anatom%C3%ADa>
- [https://es.wikipedia.org/wiki/%C3%8Dndice_\(base_de_datos\)](https://es.wikipedia.org/wiki/%C3%8Dndice_(base_de_datos))
- <https://www.ibm.com/docs/es/mam/7.6.0.8?topic=databases-database-indexing>