Tema 2: Optimización de consultas a través de índices

1. Tipos de Índices en SQL Server y sus Aplicaciones

Los índices son estructuras adicionales que aceleran la búsqueda de datos dentro de las tablas, de forma similar al índice de un libro.

Su uso adecuado mejora drásticamente el rendimiento de consultas, especialmente en bases grandes como tu sistema de riego.

A continuación, se explican todos los tipos de índices relevantes, con ejemplos y su aplicación práctica.

1.1 Clustered Index (Índice Agrupado)

Qué es

- Ordena físicamente los datos en disco según la columna del índice.
- Cada tabla **solo puede tener un índice clustered**.
- Es la estructura principal de la tabla.

Cuando usarlo

- Columnas por las que se consulta **frecuentemente por rangos**, como fechas.
- Columnas que definen el orden natural de los datos.
- Tablas muy grandes donde necesitas minimizar lecturas lógicas.

Ejemplo

```
CREATE CLUSTERED INDEX IX_LECTURAS_FECHA

ON LecturasSensor (fecha_lectura);
```

1.2 Nonclustered Index (Índice No Agrupado)

Qué es

- Es como un índice secundario, apunta a las filas reales en la tabla.
- No modifica el orden físico de los datos.
- Una tabla puede tener **varios índices nonclustered**.

Cuándo usarlo

- Búsquedas por igualdad o rangos

- Filtrado por columnas que no forman parte del índice clustered
- Para acelerar JOINs o WHERE frecuentes

Ejemplo

CREATE NONCLUSTERED INDEX IX_SENSORES_ESTADO

ON Sensores (estado_sensor);

1.3 Nonclustered Index con Columnas Incluidas (INCLUDE)

Qué es

Una variante del índice nonclustered que **almacena columnas adicionales** en el índice para evitar hacer "key lookups".

Cuándo usarlo

- Consultas SELECT que regresan muchas columnas
- Reportes
- Consultas de lectura intensiva

Ejemplo

```
CREATE NONCLUSTERED INDEX IX_LECTURAS_FECHA_INCL
ON LecturasSensor (fecha_lectura)
INCLUDE (humedad, temperatura, nivel_bateria);
```

1.4 Índices Únicos (UNIQUE INDEX)

Qué es

- Impiden valores duplicados en la columna.
- Garantizan integridad, además de mejorar rendimiento.

Cuándo usarlo

- Columnas que deben ser únicas (email, DNI, códigos exclusivos)
- Claves alternativas

Ejemplo

```
CREATE UNIQUE INDEX IX_ZONA_NOMBRE

ON ZonaRiego(nombre);
```

1.5 Índices Compuestos (Multicolumna)

Qué es

- Índices creados con 2 o más columnas.

Cuándo usarlo

- Consultas que filtran usando más de una columna.
- JOINs entre tablas por varias claves.**Ejemplo**

```
CREATE NONCLUSTERED INDEX IX_LECTURAS_SENSOR_FECHA

ON LecturasSensor (id_sensor_lectura, fecha_lectura);
```

1.6 Filtered Indexes (Índices Filtrados)

Qué es

Índices nonclustered que almacenan **solo las filas que cumplen una condición**.

Cuándo usarlo

- Tablas grandes pero con valores repetidos
- Para mejorar consultas sobre subconjuntos (ej.: solo sensores activos)

Ejemplo

```
CREATE NONCLUSTERED INDEX IX_SENSORES_ACTIVOS

ON Sensores(estado_sensor)

WHERE estado_sensor = 1;
```

1.7 Columnstore Index (Índice Columnar)

Qué es

- Índice orientado a columnas, ideal para análisis masivo de datos.
- Permite compresión muy alta.
- Usado en bases de datos de tipo BI o analíticas.

Cuándo usarlo

- Tablas de millones de registros
- Consultas de agregación o análisis

Ejemplo

CREATE CLUSTERED COLUMNSTORE INDEX IX_LECTURAS_COLUMNSTORE

ON LecturasSensor;

2. Descripción del Entorno

La base de datos corresponde a un sistema de riego que almacena información de sensores de humedad, temperatura, batería y su ubicación en zonas de riego. La tabla utilizada para el estudio fue LecturasSensor, que contiene millones de lecturas históricas y un campo fecha_lectura ideal para pruebas de filtrado por rango temporal.

3. Carga Masiva de Datos

Para simular un entorno de producción con grandes volúmenes de datos, se generó un millón de registros utilizando un script automatizado:

```
USE sistema_riego;
GO

DECLARE @minSensor INT, @maxSensor INT;
SELECT @minSensor = MIN(id_sensor), @maxSensor = MAX(id_sensor) FROM
Sensores;

;WITH Numbers AS (
    SELECT TOP (1000000) ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) AS
n
    FROM sys.objects AS a
    CROSS JOIN sys.objects AS b
    CROSS JOIN sys.objects AS c
)
INSERT INTO LecturasSensor (fecha_lectura, humedad, temperatura,
nivel_bateria, id_sensor_lectura)
SELECT
    DATEADD(DAY, -ABS(CHECKSUM(NEWID())) % 730, GETDATE()),
    CAST(ROUND(RAND(CHECKSUM(NEWID())) * 100, 2) AS FLOAT),
    CAST(ROUND(RAND(CHECKSUM(NEWID())) * 35 + 5, 2) AS FLOAT),
    ABS(CHECKSUM(NEWID())) % 100,
```

```
    FLOOR(RAND(CHECKSUM(NEWID())) * (@maxSensor - @minSensor + 1)) +
@minSensor
FROM Numbers;
```

4. Pruebas, Consultas y Planes de Ejecución

Se evaluaron tres escenarios: sin índices, con un índice clustered en fecha_lectura y con un índice nonclustered que incluye columnas adicionales. El objetivo fue comparar tiempos, lecturas lógicas y tipo de operación utilizado por SQL Server.

4.1 Escenario 1: Consulta sin Índices

En este escenario la consulta realiza un Table Scan, recorriendo toda la tabla debido a que no existe ningún índice que permita filtrar por fecha.

```
SET STATISTICS TIME ON;
SET STATISTICS IO ON;

SELECT *
FROM LecturasSensor
WHERE fecha_lectura BETWEEN DATEADD(DAY, -30, GETDATE()) AND GETDATE();
```

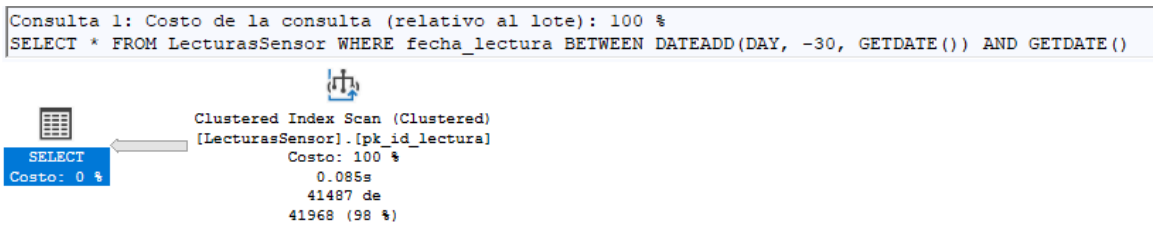
Resultados:

• Tiempo de CPU: 78 ms

• Tiempo total: 271 ms

• Lecturas lógicas: 5001

• Tipo de operación: Table Scan



4.2 Escenario 2: Índice Clustered en fecha_lectura

El índice clustered ordena físicamente la tabla usando la columna fecha_lectura. Para crearlo fue necesario convertir la clave primaria en NONCLUSTERED.

```
ALTER TABLE LecturasSensor
DROP CONSTRAINT pk_id_lectura;

ALTER TABLE LecturasSensor
ADD CONSTRAINT pk_id_lectura
PRIMARY KEY NONCLUSTERED (id_lectura, id_sensor_lectura);
```

```
CREATE CLUSTERED INDEX IX_LECTURAS_FECHA
ON LecturasSensor (fecha_lectura);
```

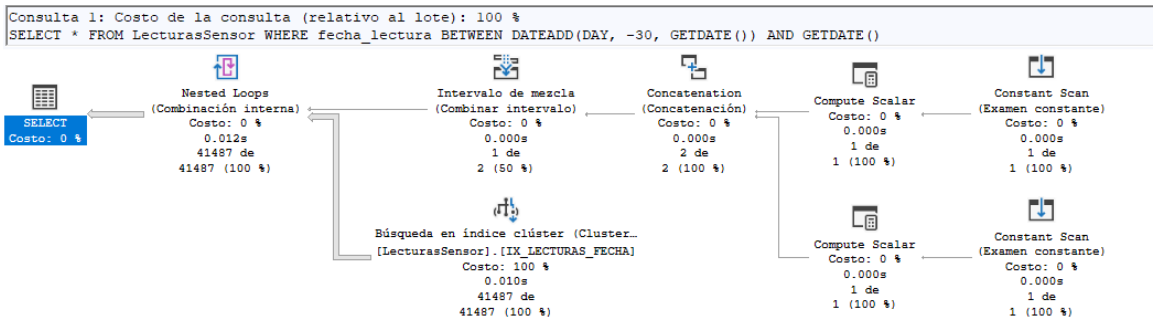
Resultados:

• Tiempo de CPU: 16 ms

• Tiempo total: 288 ms

• Lecturas lógicas: 251

• Tipo de operación: Clustered Index Seek / Range Scan



4.3 Escenario 3: Índice Nonclustered con Columnas Incluidas

Este índice optimiza aún más la consulta porque permite que SQL Server resuelva todo el SELECT directamente desde el índice, sin acceder a la tabla base, eliminando los Key Lookups.

```
CREATE NONCLUSTERED INDEX IX_LECTURAS_FECHA_INCL
ON LecturasSensor (fecha_lectura)
INCLUDE (humedad, temperatura, nivel_bateria);
```

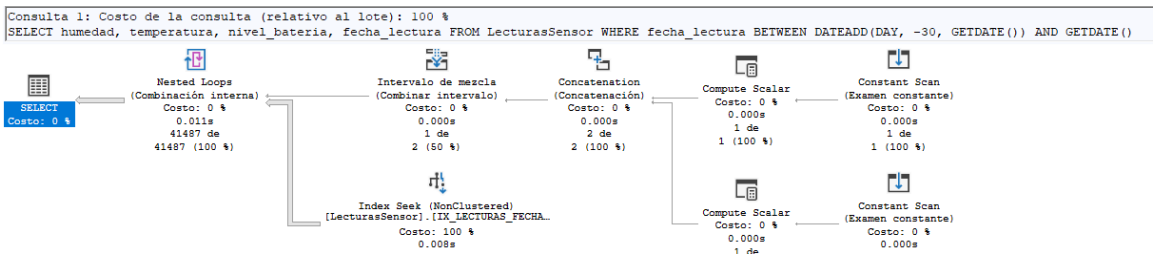
Resultados:

• Tiempo de CPU: 0 ms

• Tiempo total: 189 ms

• Lecturas lógicas: 196

• Tipo de operación: Nonclustered Index Seek (Range Scan).



5. Conclusiones

Las pruebas realizadas demuestran claramente el impacto de los índices en el rendimiento. El Table Scan inicial fue el más costoso en tiempo y lecturas. El índice clustered redujo drásticamente las lecturas, aunque incrementó el tiempo total debido al costo físico del índice. Finalmente, el índice nonclustered con columnas incluidas ofreció el mejor rendimiento global, minimizando el costo de lectura y acelerando la ejecución.