Optimización de Consultas en un Sistema de Gestión Hospitalario mediante la Implementación de Índices

¿Qué es un Índice?

Un índice es simplemente un “atajo” para encontrar información dentro de un gran conjunto de datos, por lo general, dichos índices se referencian con caracteres simples o números.

El ejemplo perfecto o más cotidiano es un índice alfabético al final de un libro de texto. En vez de buscar página por página alguna palabra clave de la cual queremos saber, tenemos dos posibles caminos:

1. Buscar página por página (Método Lento o "Full Table Scan").
2. Utilizar el índice alfabético para encontrar dicha palabra y luego dirigirnos a la página exacta (Método Rápido o "Index Seek").

Aplicado a las bases de datos, un índice es una estructura de datos especial, separada de una tabla principal, que almacena una copia de una o más columnas de forma ordenada. Específicamente, funciona como un **puntero** (como el número de página) que le dice al motor de base de datos dónde se encuentra la fila completa.

Tipos de Índices en SQL Server

1. Índice Agrupado (Clustered Index)

Es aquel que define el **orden físico** en que las filas de datos se almacenan en el disco. No es un objeto separado de la tabla; la tabla misma se convierte en el índice. Debido a que los datos solo pueden estar almacenados físicamente en un único orden, una tabla sólo puede tener **un índice agrupado**.

- **Estructura (Árbol-B):**
  - Los nodos **raíz e intermedios** del árbol contienen valores de la clave del índice que actúan como "señales" para guiar la búsqueda.
  - El nivel más bajo del árbol, las **hojas (leaf nodes)**, no contienen punteros. Las hojas del índice agrupado son las **páginas de datos reales** de la tabla.
  - Esto significa que los datos de la tabla (ej. `paciente`) están organizados y ordenados secuencialmente en el disco según la clave del índice agrupado (ej. `id_paciente`).
- **Ventajas:**
  - **Rendimiento de Lectura Insuperable:** Para búsquedas exactas (`WHERE id_paciente = 123`) o búsquedas por rango (`WHERE id_paciente BETWEEN 100 AND 200`), es el más rápido, ya que los datos están físicamente contiguos.
  - **Sin Paso Adicional (No "Key Lookup"):** Cuando la búsqueda encuentra la fila en el índice, ha encontrado la fila de datos completa.
- **Limitaciones:**
  - **Solo UNO por Tabla:** Esta es su mayor limitación. Se debe elegir cuidadosamente qué columna (ej. `id_paciente`) será la clave agrupada.

    - **Costoso en Escrituras (`INSERT`, `UPDATE`):**
        - **`INSERT`:** SQL Server debe colocar la fila físicamente en su lugar ordenado. Si la página de datos está llena, ocurre una **"División de Página" (Page Split)**, que es una operación muy costosa y genera fragmentación.
        - **`UPDATE`:** Si se actualiza el valor de la clave agrupada, el motor debe borrar y mover físicamente la fila completa a su nueva ubicación.

2. Índice No Agrupado (Non-Clustered Index)

Es una estructura de datos **completamente separada** de la tabla. Es un "mini-libro" auxiliar que existe independientemente del orden físico de los datos. Como son estructuras separadas, una tabla puede tener **muchos** índices no agrupados.

- **Estructura (Árbol-B):**
    - Utiliza una estructura de Árbol-B similar, pero la gran diferencia está en el nivel de las hojas.
    - Las **hojas (leaf nodes)** de un índice no agrupado NO contienen la fila de datos completa.
    - En su lugar, las hojas contienen dos cosas:
        1. El **valor de la clave** del índice (ej. el `dni` 45.527.225).
        2. Un **"Localizador de Fila" (Row Locator)**. Este puntero le dice a la BD dónde encontrar la fila completa (generalmente, es la clave del índice agrupado, ej. `id_paciente = 1`).
- **Ventajas:**
    - **Múltiples Índices por Tabla:** Permite optimizar muchos tipos de consultas (uno para `dni`, otro para `apellido`, etc.).
    - **Escrituras (`INSERT`) más Rápidas:** Generalmente es más rápido que un `INSERT` en un índice agrupado, ya que la fila de datos principal se añade y el índice solo actualiza su propia estructura separada.
    - **Flexibilidad:** Son la herramienta principal para el "tuning" (afinamiento) de consultas.
- **Limitaciones:**
    - **Costo de "Key Lookup" (Búsqueda de Clave):** Es su principal desventaja. Para `SELECT * FROM Paciente WHERE dni = ...`, el motor primero busca en el índice no agrupado (rápido), obtiene el puntero (`id_paciente`), y luego debe hacer un **segundo salto** (el "Key Lookup") al índice agrupado (la tabla) para buscar el resto de las columnas.
    - **Costo de Almacenamiento:** Cada índice no agrupado es un objeto nuevo que ocupa espacio adicional.
    - **Sobrecarga de Escritura (Overhead):** Si una tabla tiene 5 índices, un solo `INSERT` provoca 6 operaciones de escritura (una en la tabla y una en cada índice), lo que puede llevar a una "Sobre-indexación".

Metodología y Pruebas

Para simular un entorno hospitalario de alta carga, se replicó la metodología de la práctica de la asignatura:

1. **Creación del Entorno:** Se crearon dos tablas: `Internaciones_SinIndice` e `Internaciones_ConIndice`.
2. **Carga Masiva:** Se insertaron **1 millón de registros** de internaciones ficticias en ambas tablas, conteniendo `id_internacion`, `id_paciente`, `fecha_ingreso` y `diagnostico`.
3. **Habilitación de Estadísticas:** Se utilizaron los comandos `SET STATISTICS TIME ON` y `SET STATISTICS IO ON` para medir con precisión el tiempo de CPU, el tiempo transcurrido y las lecturas lógicas.
4. **Consulta de Prueba:** Se ejecutó una consulta analítica común: buscar todas las internaciones ocurridas en un mes específico.

Análisis de los Resultados:

- **Escenario 1 (Sin Índice):** Es el peor caso. El motor debe leer la tabla entera (3109 lecturas lógicas) para encontrar las filas. Esto es inaceptable en un hospital, representando una espera de casi 2 segundos por una simple consulta.
- **Escenario 2 (Índice Agrupado):** Al ordenar físicamente la tabla por fecha, el motor solo lee las páginas de datos relevantes. El resultado es una **reducción del 88% en lecturas lógicas** y una mejora del 67% en el tiempo de respuesta. La desventaja es que se tuvo que eliminar la PK agrupada por defecto, recreándola como no agrupada.
- **Escenario 3 (Índice No Agrupado con `INCLUDE`):** Este fue el escenario más eficiente. Al crear un "Covering Index", la consulta obtuvo la `fecha_ingreso` (del índice) y el resto de columnas (`id_internacion`, etc.) de la cláusula `INCLUDE`. El motor **ni siquiera tuvo que tocar la tabla principal**, evitando el "Key Lookup" y logrando la menor cantidad de lecturas lógicas.

Conclusión

La hipótesis se confirma: la implementación de índices es una estrategia fundamental y efectiva para optimizar un SGH.

- **Sin índices**, un sistema es funcional a pequeña escala, pero colapsará bajo una carga de datos real, resultando en escaneos completos de tabla que degradan el rendimiento.
- Un **Índice Agrupado** ofrece una mejora drástica, especialmente para consultas por rango (como fechas). Su limitación (solo uno por tabla) exige una decisión de diseño estratégica: ¿es más importante para el hospital buscar por `id_paciente` (PK) o por `fecha_ingreso`?

- Un **Índice No Agrupado** (especialmente uno de cobertura o *Covering Index*) demostró ser la solución más eficiente y flexible. Permite mantener el índice agrupado en la Clave Primaria (para eficiencia operativa diaria) mientras se crean índices adicionales altamente optimizados para consultas específicas de reportes.

Para un SGH, se recomienda una estrategia híbrida: mantener el **Índice Agrupado en la Clave Primaria** (`id_paciente`, `id_internacion`) y diseñar **Índices No Agrupados con columnas incluidas** (`INCLUDE`) para optimizar las consultas analíticas y de reportes más frecuentes.