Índices Columnares en SQL Server

¿Qué son los índices columnares?

Los **índices columnares** (columnstore indexes) en SQL Server son una estructura de almacenamiento orientada a columnas, diseñada específicamente para acelerar consultas analíticas, operaciones de agregación, filtrado masivo y exploración de grandes volúmenes de datos. A diferencia del almacenamiento tradicional **rowstore**, donde una fila completa se almacena junta, en un columnstore cada columna se almacena de forma independiente y comprimida.

Esta arquitectura permite: - Leer solo las columnas necesarias. - Aplicar diferentes tipos de compresión por columna. - Ejecutar consultas en modo batch (procesamiento vectorizado). - Eliminar segmentos completos durante un filtrado (“segment elimination”).

Como consecuencia, el rendimiento analítico aumenta de forma significativa, a la vez que disminuye el tamaño ocupado por los datos.

Ventajas y su fundamento técnico

1. Rendimiento superior en consultas analíticas

Por qué sucede: - En rowstore, SQL Server debe leer *toda la fila*, incluso si solo se necesitan 2 columnas. - En columnstore, solo se leen las columnas involucradas en la consulta. - La compresión permite leer una fracción del tamaño original. - El motor usa **Batch Mode**, procesando miles de filas por ciclo de CPU en lugar de filas individuales.

Esto se traduce en menor IO, menor CPU y menor tiempo total.

2. Reducción del tamaño de almacenamiento

Fundamento: - Cada columna se comprime de forma independiente. - Los datos suelen tener alta repetición o baja cardinalidad por columna. - Columnstore utiliza técnicas avanzadas como: - *Dictionary encoding* - *Run Length Encoding (RLE)* - *Value encoding*

Esta combinación logra compresiones típicas entre **5× y 15×**, reduciendo drásticamente el espacio físico y el IO necesario.

3. Lectura mucho más eficiente de columnas específicas

Por qué sucede: - En rowstore, aunque solo selecciones una columna, SQL Server debe leer las páginas de datos completas donde están mezcladas todas las columnas. - En columnstore, cada columna está en su propio segmento físico.

Por ejemplo, una consulta que pide solo `precio_unitario` y `cantidad` evita leer: - `fecha_venta` - `id_producto` - cualquier otra columna

Esto reduce IO entre **80% y 95%** en consultas reales.

4. Segment elimination

Cada columna se divide en **segmentos** (grupos de ~1 millón de filas).
Cada segmento guarda metadatos: *min* y *max* de la columna.

Cuando haces:

```
WHERE fecha_venta >= '2024-01-01'
```

SQL Server descarta automáticamente todos los segmentos cuyos valores no caen en ese rango.
Esto evita leer millones de filas completas.

5. Procesamiento vectorizado (Batch Mode)

En lugar de procesar fila por fila como en rowstore, un índice columnstore permite que SQL Server procese columnas completas en bloques vectoriales de miles de valores.

El CPU trabaja menos e instrucción por instrucción rinde más.

Desventajas y explicación técnica

1. Peor rendimiento en cargas pequeñas y operaciones OLTP

Por qué pasa: - Columnstore está optimizado para lecturas masivas y agregaciones. - Las actualizaciones y deletes afectan toda la compresión del segmento. - SQL Server mantiene un delta store para cambios pequeños, lo que añade sobrecarga.

2. Requiere más memoria para consultas complejas

Debido al procesamiento vectorial y la descompresión, ciertas consultas pueden usar más memoria aunque sean más rápidas.

3. No es adecuado para tablas altamente transaccionales

Las modificaciones frecuentes rompen la compresión eficiente.
Por eso columnstore es ideal para **tablas grandes, estables y analíticas**.

¿Cómo utilizar los índices columnares?

Existen dos tipos:

Nonclustered Columnstore Index (NCCI)

- Se agrega sobre una tabla rowstore existente.
- Mantiene el índice clustered original.
- Ideal para entornos mixtos OLTP/Analítico.

Ejemplo usado en nuestro proyecto:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX idx_ncc_ventas_big
ON ventas_big (id_producto, cantidad, precio_unitario, fecha_venta);
```

Clustered Columnstore Index (CCI)

Transforma la tabla completa en formato columnstore.
No se usa aquí porque la tabla ya tenía un índice clustered por su PK.

Aplicación práctica en el proyecto

Se generó una tabla nueva con un millón de registros:

Tabla creada:

```
ventas_big(
  id_venta INT IDENTITY PRIMARY KEY,
  id_producto INT,
  cantidad INT,
  precio_unitario DECIMAL(10,2),
  fecha_venta DATE
)
```

Inserción de 1.000.000 de filas usando datos reales de producto:

(Resumen del script real aplicado.)

Creación del índice columnar usado en pruebas

```
CREATE NONCLUSTERED COLUMNSTORE INDEX idx_ncc_ventas_big
ON ventas_big (id_producto, cantidad, precio_unitario, fecha_venta);
```

Este índice no afecta la PK ni genera inconsistencias.

Análisis de rendimiento – Comparación real obtenida

Se activaron las métricas:

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

Se comparó: - `ventas_big_rowstore` (sin columnstore)
- `ventas_big` (con columnstore)

Consultas ejecutadas:

1. Agregación por producto:

```
SELECT id_producto, SUM(cantidad)
FROM ventas_big
GROUP BY id_producto;
```

Resultados obtenidos:

Rowstore:

- logical reads: **4082**
- CPU: **110 ms**
- elapsed: **177 ms**
- filas devueltas: **100** (porque había GROUP BY id_producto)

Columnstore:

- logical reads: **0** (porque la mayoría fue segment elimination + lectura columnar)
- LOB logical reads: **335** (páginas columnstore)
- CPU: **0 ms**
- elapsed: **6 ms**
- segment reads: **1**

Interpretación:

- Reducción del IO del **91.7%**
- Reducción del tiempo de CPU del **100%**
- Reducción del tiempo de ejecución del **96.6%**

La tabla tenía efectivamente **1.000.000 de filas**.
El motor devolvió solo 100 resultados por el GROUP BY.

Conclusión

Los índices columnares en SQL Server ofrecen ventajas contundentes para consultas analíticas en tablas grandes, como la generada en el proyecto. Su rendimiento superior se debe a:

- Lectura orientada a columnas
- Alta compresión
- Eliminación de segmentos
- Procesamiento vectorizado en Batch Mode
- Menor IO y CPU

Las pruebas reales demuestran reducciones enormes en tiempos y lecturas, confirmando el impacto práctico y la aplicabilidad en escenarios de análisis de datos de gran escala.